msdn® training

Module 8: Using ADO .NET

Contents

This course is based on the prerelease version (Beta 2) of Microsoft® Visual Studio® .NET Enterprise Edition. Content in the final release of the course may be different from the content included in this prerelease version. All labs in the course are to be completed with the Beta 2 version of Visual Studio .NET Enterprise Edition.

Microsoft

Instructor Notes

Presentation:
120 Minutes

Lab:
60 Minutes

This module provides students with the knowledge needed to create high-performance data access applications in Microsoft® Visual Basic® .NET.

It begins with a discussion about the need for ADO .NET and the benefits it brings to the development environment. Students will then learn about ADO .NET providers and the objects they supply.

Students will then learn about the **DataSet** object. Students will learn how to add, update, and delete data by using it. Students will then learn how to use data designers and data binding in Microsoft Windows® Forms and Web Forms. Finally, students learn how Extensible Markup Language (XML) integrates with ADO .NET.

After completing this module, students will be able to:

- List the benefits of ADO .NET.
- Create applications by using ADO .NET.
- List the main ADO .NET objects and their functions.
- Use Microsoft Visual Studio® .NET data designers and data binding.
- Explain how XML integrates with ADO .NET.

Materials and Preparation

This section provides the materials and preparation tasks that you need to teach this module.

Required Materials

To teach this module, you need the following materials:

- Microsoft PowerPoint® file 2373A_08.ppt
- Module 8, “ Using ADO .NET”
- Lab 8.1, Creating Applications That Use ADO .NET

Preparation Tasks

To prepare for this module:

- Read all of the materials for this module.
- Read the instructor notes and the margin notes for the module.
- Practice the demonstrations.
- Complete the practice.
- Complete the lab.

Demonstrations

This section provides demonstration procedures that will not fit in the margin notes or are not appropriate for the student notes.

Retrieving Data Using ADO .NET

To prepare for the demonstration

1. Open the DataReaderDemo.sln solution in the *install folder*\DemoCode\Mod08\DataReader\Starter folder.
2. View the code behind the **Display** button, and explain the variable declarations.

To add code to the form

1. Locate the comment " Add code to open connection," and add the following lines of code:

```
conSQL.ConnectionString = "data source=localhost;initial
    catalog=cargo;integrated security=true;"
conSQL.Open( )
```

2. Locate the comment " Add code to set CommandText and Connection properties of the command," and add the following lines of code:

```
comSQL.CommandText = "Select FirstName, LastName from
Customers"
comSQL.Connection = conSQL
```

3. Locate the comment " Add code to execute the command," and add the following line of code:

```
readerSQL = comSQL.ExecuteReader( )
```

4. Locate the comment " Add code to loop through the records, add each to the listbox," and add the following lines of code:

```
Do While readerSQL.Read
lstCustomers.Items.Add
(readerSQL.Item("FirstName").ToString & " " &
readerSQL.Item("LastName").ToString)
Loop
```

To test the code

1. Run the application, and click **Display**. You should see a list of customers in the Cargo database.
2. Quit the application, and quit Visual Studio .NET.

Using the Data Form Wizard

To run the wizard

1. Open Visual Studio .NET, and create a new Windows application project in the *install folder*\DemoCode\Mod08\DataFormWizard folder.
2. In the Solution Explorer, click WindowsApplication1. On the **Project** menu, click **Add New Item**, select **Data Form Wizard**, and click **Open**.
3. Run the wizard, using the information provided in the following table. Where no details are specified, use the defaults.

Setting	Value
New dataset name	dsCargo
Which connection should the wizard use?	New Connection
Server name	localhost
Log on	Integrated security
Database name	Cargo
Which items do you want to access?	Customers Invoices
Relationship name	CustInvRel
Parent table	Customers
Key	CustomerID
Child table	Invoices
Key	CustomerID
How do you want to display your data?	Single record in individual controls

4. Save the project.

To test the form

1. In the project Properties window, change the **Startup object** to **DataForm1**.
2. Run the application.
3. Click **Load** to load the data into the form.
4. Change the **CompanyName** field to Microsoft, and then click **Update**.
5. Click **Add**, and enter some sample data (use **2222** for the CustomerID). Move away from that record, and verify that there are now 22 records in the form.
6. Move back to your new record, and click **Delete**. Verify that there are now 21 records in the form.
7. Quit the application.

Using XML Schema

Delivery Tip
Ensure that you add the item to the project, not to the solution, or the demonstration will not function as documented.

To add the XML file to the project

1. Open the XMLDataDemo.sln solution in the *install folder*\DemoCode\Mod08\XMLData\Starter folder.
2. In the Solution Explorer, click **XMLDataDemo**. On the **Project** menu, click **Add Existing Item**. Change the **Files of type** to **Data Files (*.xsd, *.xml)**, select **Customers.xml**, and then click Open.
3. Open Customers.xml and review the data extracted from the Cargo database.

To generate a schema

1. On the **XML** menu, click **Create Schema**.
2. Point out that an xmlns attribute has been added to the root element identifying the schema.
3. Review both the Schema view and XML view of Customers.xsd.
4. In the Schema view, change the data type for the ID attribute to **integer**. Show the corresponding change in the XML view.

To validate data

1. In the XML view of Customers.xml, rename the <Pword> element belonging to the first customer to <Password>. Remember to also change the closing tag.
2. On the **XML** menu, click **Validate XML Data**.
3. Review the tasks created in the Task List and explain the meaning of each one.
4. Change the element name back to <Pword>.
5. On the **XML** menu, click **Validate XML Data**. Point out that the message in the status bar stating that no validation errors were found.

Module Strategy

Use the following strategy to present this module:

- ADO .NET Overview

  This lesson provides an overview of ADO .NET, including the advantages it provides. Some students who are familiar with the Microsoft .NET Framework will be aware of these benefits, but others will need to know why ADO .NET is worth learning.

- .NET Data Providers

  This lesson introduces the four main objects within the data providers. The first two objects this lesson describes, **Connection** and **Command**, are similar to their Microsoft ActiveX® Data Objects (ADO) counterparts, but be sure to point out the differences needed in the code.

  The third object, the **DataReader**, is similar to the default ADO cursor, the firehose cursor. Students often find this similarity helpful when they are learning about the **DataReader**.

  The fourth object, the **DataAdapter**, is primarily used to populate **DataSet**s and to update the data source from **DataSet**s. Simply point out its syntax here because you will provide a thorough explanation of its uses in the next lesson.

- The **DataSet** Object

  This lesson begins with a review of disconnected data. Again, if students seem to be aware of the issues, then only explain the topic briefly.

  This lesson then describes how to use the main objects within a **DataSet**: **DataTable** objects and **DataRelation** objects. Students might try to equate a DataSet with an ADO Recordset, and you need to ensure that they understand that a DataSet contains a collection of tables that are sometimes related.

  Finally, you will explain how to update data in the DataSet and how to propagate those changes to the data source. Students need to be aware that the data is disconnected, which means that any changes the students make are local and that they need to explicitly pass the changes back to the source.

- Data Designers and Data Binding

  This lesson covers some of the new graphical user interface (GUI) features for data access in Visual Basic .NET. Remind students that all of these features simply automate the coding that they have learned so far, and that they can create things by using the tools and then customize the code for their own purposes.

  The lesson also covers how to use data binding in both Windows Forms and Web Forms. These techniques are distinctly different from those of Visual Basic 6.0, so ensure that you cover the topics adequately.

- XML Integration

  This lesson relies on the students' knowledge of XML. Ensure that they have some understanding of it before embarking on the topics, or you may confuse them. The main things they need to know are how an XML document is hierarchical, what it looks like, and why XML is so important for developers.

  The topics about schemas should be relatively easy because students will either be aware of database or XML schemas, and the importance of validation. The final topic is only designed to explain the links between DataSets and XmlDataDocument. Do not get involved in a long discussion about XmlDataDocument, because it is not necessary to be able to use ADO .NET.

Overview

Topic Objective
To provide an overview of the module topics and objectives.

Lead-in
In this module, you will learn how to use ADO .NET from Visual Basic .NET.

- **ADO .NET Overview**
- **.NET Data Providers**
- **The DataSet Object**
- **Data Designers and Data Binding**
- **XML Integration**

In this module, you will learn how to use ADO .NET from Microsoft® Visual Basic® .NET version 7.0. You will learn about the Microsoft .NET providers included in the .NET Framework and about how to use the **DataSet** object. You also will learn how to use the Microsoft Visual Studio® .NET data designers and how to bind data to Microsoft Windows® Forms and Web Forms. Finally, you will learn about the integration of Extensible Markup Language (XML) with ADO .NET.

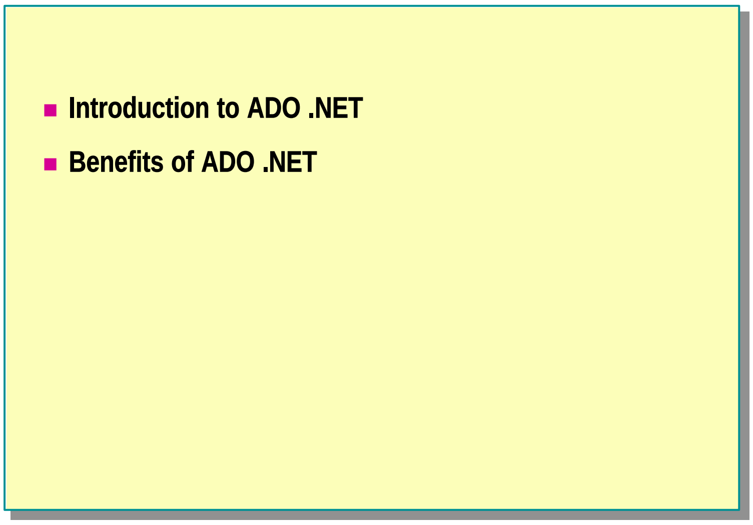
After completing this module, you will be able to:

- List the benefits of ADO .NET.
- Create applications using ADO .NET.
- List the main ADO .NET objects and their functions.
- Use Visual Studio .NET data designers and data binding.
- Explain how XML integrates with ADO .NET.

◆ ADO .NET Overview

Topic Objective
To provide an overview of the topics covered in this lesson.

Lead-in
ADO .NET provides many enhancements upon ADO.



- **Introduction to ADO .NET**
- **Benefits of ADO .NET**

ActiveX® Data Objects for the .NET Framework (ADO .NET) provide many enhancements for accessing data in a disconnected environment. ADO .NET contains objects that are similar to those of ADO, allowing you to update your skills easily.

In this lesson, you will learn where ADO .NET is within the .NET Framework, and about the benefits ADO .NET provides.

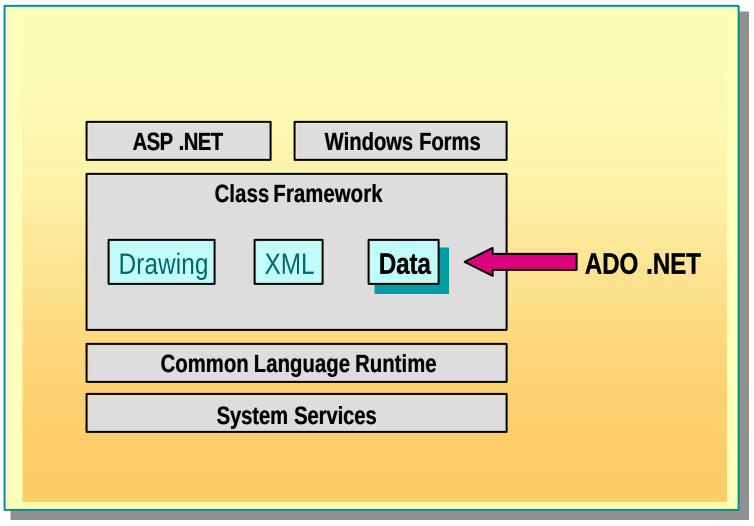
After completing this lesson, you will be able to:

- Describe the role of ADO .NET in the .NET Framework.
- List the major benefits of ADO .NET.

Introduction to ADO .NET

Topic Objective
To discuss ADO .NET and its place in the .NET Framework.

Lead-in
ADO .NET is a component of the .NET Framework.



ADO .NET is a set of classes that allow .NET -based applications to read and update information in databases and other data stores. You can access these classes through the **System.Data** namespace provided by the .NET Framework.

ADO .NET provides consistent access to a wide variety of data sources, including Microsoft SQL Server™ databases, OLE DB–compliant databases, non-relational sources such as Microsoft Exchange Server, and XML documents.

Earlier data access methods, such as Data Access Object (DAO), concentrate on tightly coupled, connected data environments. One of the main purposes of ADO .NET is to enhance the disconnected data capabilities. Many of the common ADO objects that you have worked with correlate to ADO .NET objects, although there are also many new classes to enhance the data access model.

ADO .NET uses .NET data providers to link your applications to data sources. .NET data providers are similar to the OLE DB providers used in ADO, although they are primarily concerned with moving data into and out of a database rather than providing interfaces over all of a database's functionality.

ADO .NET includes two .NET data providers:

- .NET data provider for SQL Server

  For use with SQL Server 7.0 and later.
- .NET data provider for OLE DB

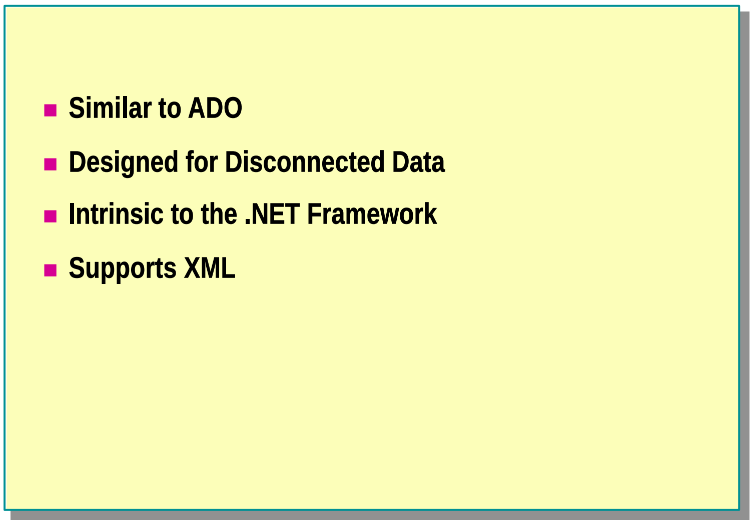
  For use with data sources exposed by OLE DB.

The ADO .NET data providers contain tools to allow you to read, update, add, and delete data in multitier environments. Most of the objects in the two libraries are similar and are identified by the prefix on their name. For example, **SqlDataReader** and **OleDbDataReader** both provide a stream of records from a data source.

Benefits of ADO .NET

Topic Objective
To discuss the benefits of using ADO .NET.

Lead-in
ADO .NET provides many benefits to the Visual Basic .NET developer.



- **Similar to ADO**
- **Designed for Disconnected Data**
- **Intrinsic to the .NET Framework**
- **Supports XML**

ADO .NET provides many benefits to experienced Visual Basic developers, including:

- Similar programming model to that of ADO

  This makes it easy for Visual Basic developers who are familiar with ADO to update their skills. You can still use ADO in Visual Basic .NET, so you can keep existing code, but use the features of ADO .NET in new projects.

- Designed for disconnected data

  ADO .NET is designed for working with disconnected data in a multitier environment. It uses XML as the format for transmitting disconnected data, which makes it easier to communicate with client applications that are not based on Windows.

Delivery Tip
You may find that you need to expand on the problems that were faced when transmitting data as COM objects, such as dependency on a Windows-based client over a local area network.

- Intrinsic to the .NET Framework

  Because ADO .NET is intrinsic to the .NET Framework, you have all the advantages of using the .NET Framework, including ease of cross-language development.

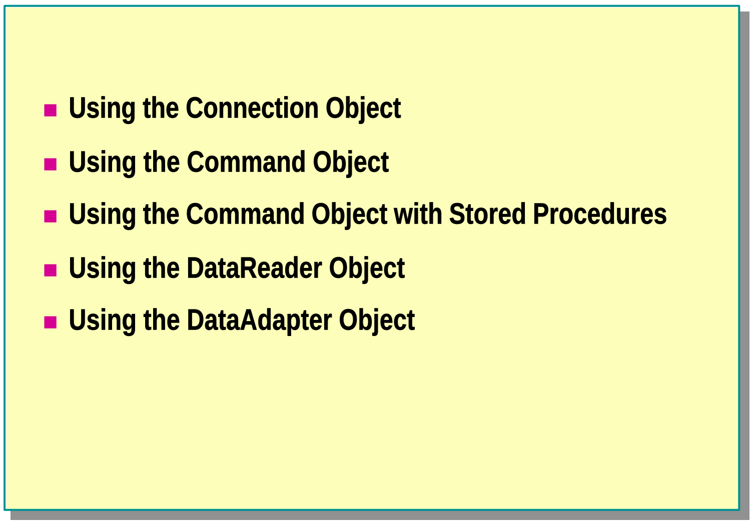
- Supports XML

  ADO and XML have previously been incompatible: ADO was based on relational data, and XML is based on hierarchical data. ADO .NET brings together these two data access techniques and allows you to integrate hierarchical and relational data, as well as alternate between XML and relational programming models.

◆ .NET Data Providers

Topic Objective
To provide an overview of the topics covered in this lesson.

Lead-in
The .NET Data Providers allow access to specific types of data sources.



- Using the Connection Object
- Using the Command Object
- Using the Command Object with Stored Procedures
- Using the DataReader Object
- Using the DataAdapter Object

The .NET data providers allow access to specific types of data sources. You can use the **System.Data.SQLClient** namespace to access SQL Server 7.0 and later databases, and the **System.Data.OLEDB** namespace to access any data source exposed through OLE DB.

Each of these providers contains four main objects that you can use to connect to a data source, read the data, and manipulate the data prior to updating the source.

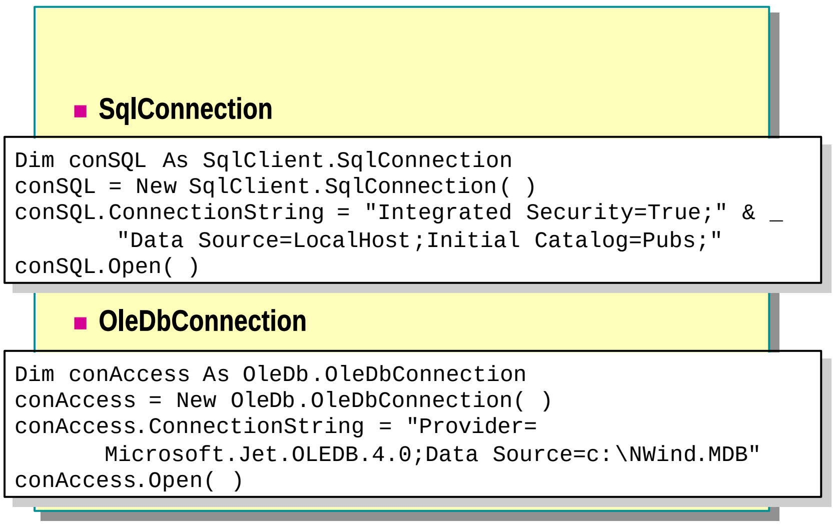
After completing this lesson, you will be able to:

- Use the **Connection** object to connect to a database.
- Use the **Command** object to execute commands and, optionally, to return data from a data source.
- Use the **DataReader** object to create a read-only data stream.
- Use the **DataAdapter** object to exchange data between a data source and a **DataSet**.

Using the Connection Object

Topic Objective
To introduce the ADO .NET **Connection** object.

Lead-in
You can use the **Connection** object to connect to a specific data source.



■ **SqlConnection**

```
Dim conSQL As SqlClient.SqlConnection
conSQL = New SqlClient.SqlConnection( )
conSQL.ConnectionString = "Integrated Security=True;" & _
        "Data Source=LocalHost;Initial Catalog=Pubs;"
conSQL.Open( )
```

■ **OleDbConnection**

```
Dim conAccess As OleDb.OleDbConnection
conAccess = New OleDb.OleDbConnection( )
conAccess.ConnectionString = "Provider=
       Microsoft.Jet.OLEDB.4.0;Data Source=c:\NWind.MDB"
conAccess.Open( )
```

To connect to a database, you set the connection type, specify the data source, and connect to the data source. When you are finished working with the data, you close the connection.

1. Set the connection type.

   You can use the **Connection** object to connect to a specific data source. You can use either the **SqlConnection** object to connect to SQL Server databases or the **OleDbConnection** object to connect to other types of data sources.

2. Specify the data source.

   After you set the connection type, you use a **ConnectionString** to specify the source database and other information used to establish the connection. The format of these strings differs slightly between the **SqlClient** namespace and the **OleDb** namespace.

3. Connect to the data source.

   Each object supports an **Open** method that opens the connection after the connection properties have been set, and a **Close** method that closes the connection to the database after all transactions have cleared.

SqlConnection

The **SqlConnection** object is optimized for SQL Server 7.0 and later databases by bypassing the OLE DB layer. It is recommended that you use this object, not **OleDbConnection**, when working with these types of data sources.

The SQL Client .NET Data Provider supports a **ConnectionString** format that is similar to ADO connection strings. This consists of name-value pairs providing the information required when connecting to the data source. The following table lists the most commonly used pairs.

Keyword name	Description	Default value
Connection Timeout (or **Connect Timeout**)	Length of time to wait for a connection to succeed before returning an error	**15 seconds**
Initial Catalog	Name of the database	None
User ID	SQL Server logon account (if using SQL Server security)	None
Password (or **Pwd**)	SQL Server password (if using SQL Server security)	None
Data Source (or **Server** or **Address** or **Addr** or **Network Address**)	Name or network address of SQL Server	None
Integrated Security (or **Trusted_Connection**)	Whether the connection is a secure connection	**False**

The following example shows how to connect to a SQL Server database by using the SQL Client .NET Data Provider:

```
Dim conSQL As SqlClient.SqlConnection
conSQL = New SqlClient.SqlConnection( )
conSQL.ConnectionString = "Integrated Security=True;" & _
"Data Source=LocalHost;Initial Catalog=Pubs;"
conSQL.Open( )
```

OleDbConnection

The **OleDbConnection** object exposes methods similar to those of the **SqlConnection** object, but certain data sources will not support all the available methods of the **OleDbConnection** class.

The OLE DB .NET Data Provider uses a **ConnectionString** that is identical to that of ADO, except that the **Provider** keyword is now required, and the **URL**, **Remote Provider**, and **Remote Server** keywords are no longer supported.

The following example shows how to connect to a SQL Server database by using the OLE DB .NET Data Provider:

Delivery Tip
If you demonstrate this code, ensure that NWind.mdb is in the specified path.

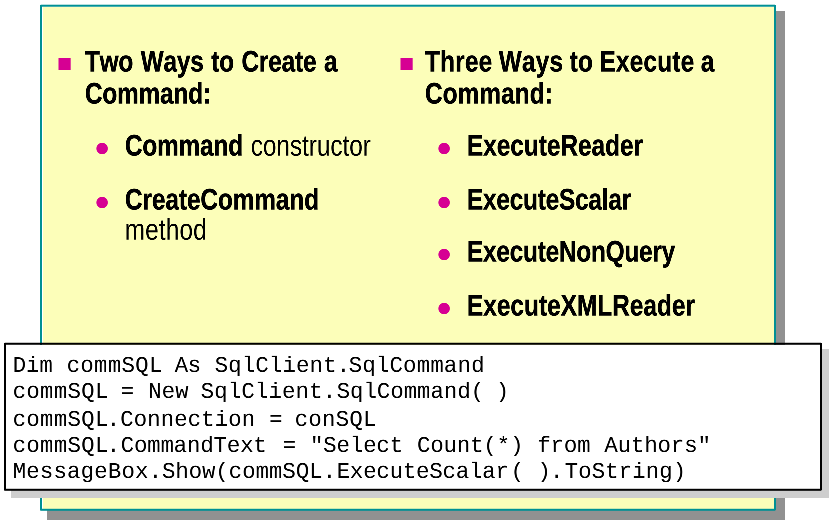
```
Dim conAccess As OleDb.OleDbConnection
conAccess = New OleDb.OleDbConnection( )
conAccess.ConnectionString =
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\NWind.MDB"
conAccess.Open( )
```

Note The examples in the remainder of this module use the **SQL Client** namespace. For more information about the **OLE DB** namespace, search for "OleDBConnection" in the Visual Basic .NET documentation.

Using the Command Object

Topic Objective
To introduce the ADO .NET **Command** object.

Lead-in
Once you have established a connection to your data source, you can use the **Command** object to execute SQL statements.



- **Two Ways to Create a Command:**
  - **Command** constructor
  - **CreateCommand** method
- **Three Ways to Execute a Command:**
  - **ExecuteReader**
  - **ExecuteScalar**
  - **ExecuteNonQuery**
  - **ExecuteXMLReader**

```
Dim commSQL As SqlClient.SqlCommand
commSQL = New SqlClient.SqlCommand( )
commSQL.Connection = conSQL
commSQL.CommandText = "Select Count(*) from Authors"
MessageBox.Show(commSQL.ExecuteScalar( ).ToString)
```

You can use the ADO .NET **Command** object to execute commands and, optionally, to return data from a data source. You can use the **SqlCommand** with SQL Server databases and the **OleDbCommand** with all other types of data sources.

Creating Commands

You can create a command in one of two ways:

- Use the **Command** constructor, passing the **Connection** name as an argument.
- Use the **CreateCommand** method of the **Connection** object.

You can use the **CommandText** property of the **Command** object to set and retrieve the SQL statement being executed. You can use any valid SQL statement with the specified data source, including data manipulation, definition, and control statements.

Executing Commands

You can only execute a **Command** within a valid and open connection. The **Command** object provides three methods that you can use to execute commands:

- **ExecuteReader**

  Use this method when the query will return a stream of data such as a **Select** statement returning a set of records. This method returns the records in a **SqlDataReader** or **OleDbDataReader** object.

- **ExecuteScalar**

  Use this method when the query will return a singleton value; for example, a **Select** statement returning an aggregate value. It executes the query and returns the first column of the first row in the result set, ignoring any other data that is returned. This method requires less code than using the **ExecuteReader** method and accessing a single value from the **SqlDataReader** object.

- **ExecuteNonQuery**

  Use this method when the query will not return a result; for example, an **Insert** statement.

- **ExecuteXMLReader**

  Use this method when the query includes a valid FOR XML clause. This is only valid when using the **SQLCommand** object.

Delivery Tip
Do not go into detail about the **DataReader** object at this time because it will be covered later in this lesson.

The following example shows how to use the **Command** object to query a database and retrieve data:

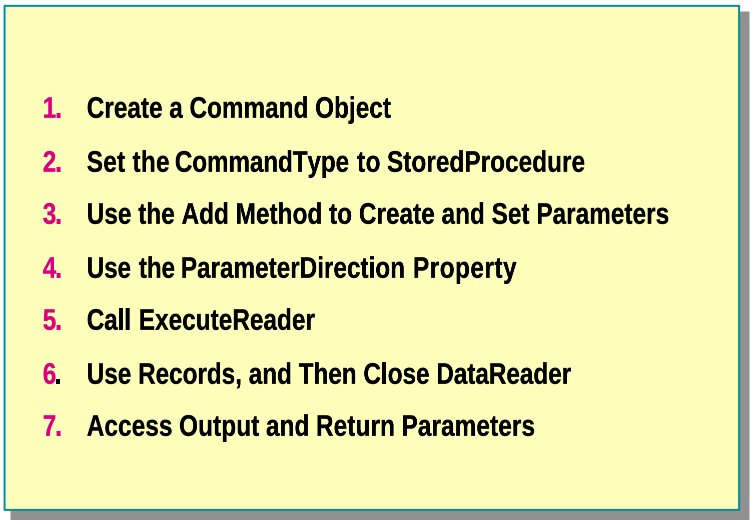
```
Dim conSQL As SqlClient.SqlConnection
conSQL = New SqlClient.SqlConnection( )
conSQL.ConnectionString = "Integrated Security=True;" & _
"Data Source=LocalHost;Initial Catalog=Pubs;"
conSQL.Open( )

Dim commSQL As SqlClient.SqlCommand
commSQL = New SqlClient.SqlCommand( )
commSQL.Connection = conSQL
commSQL.CommandText = "Select Count(*) from Authors"
MessageBox.Show(commSQL.ExecuteScalar( ).ToString)
```

This code determines how many rows are present in the Authors table of the Pubs database and displays the result.

Using the Command Object with Stored Procedures

Topic Objective
To discuss how to use the **Command** object to execute stored procedures.

Lead-in
Many databases will contain stored procedures that you want to execute.



1. Create a Command Object
2. Set the CommandType to StoredProcedure
3. Use the Add Method to Create and Set Parameters
4. Use the ParameterDirection Property
5. Call ExecuteReader
6. Use Records, and Then Close DataReader
7. Access Output and Return Parameters

You can also use the Command object to execute stored procedures in a database. You may need to perform some additional steps when preparing the **Command** to allow for the use of parameters in the stored procedure.

Use the following steps to execute a stored procedure with the **Command** object:

Delivery Tip
Talk through these steps with respect to the code example below.

1. Create a **Command** object.
2. Set the **CommandType** property to **StoredProcedure**.
3. Use the **Add** method to create and set any parameters.
4. Use the **ParameterDirection** property to set parameter type.
5. Call the **ExecuteReader** method.
6. Use the **DataReader** object to view or manipulate the records, and close it when finished.
7. Access any output and return parameters.

The following example shows how to execute a stored procedure using ADO .NET.

```
Imports System.Data.SqlClient

Private Sub Button1_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button1.Click

  Dim conSQL As SqlClient.SqlConnection
  conSQL = New SqlClient.SqlConnection( )
  conSQL.ConnectionString = "Integrated Security=True;" & _
  "Data Source=LocalHost;Initial Catalog=Pubs;"
  conSQL.Open( )

  Dim commSQL As SqlClient.SqlCommand = New SqlCommand( )
  commSQL.Connection = conSQL
  commSQL.CommandType = CommandType.StoredProcedure
  commSQL.CommandText = "byroyalty"

  Dim paramSQL As New SqlClient.sqlParameter( _
      "@percentage", SqlDbType.Int)
  paramSQL.Direction = ParameterDirection.Input
  paramSQL.Value = "30"
  commSQL.Parameters.Add(paramSQL)

  Dim datRead As SqlClient.SqlDataReader
  datRead = commSQL.ExecuteReader( )
  Do While datRead.Read( )
      MessageBox.Show(datRead(0).ToString)
  Loop
  datRead.Close( )
End Sub
```

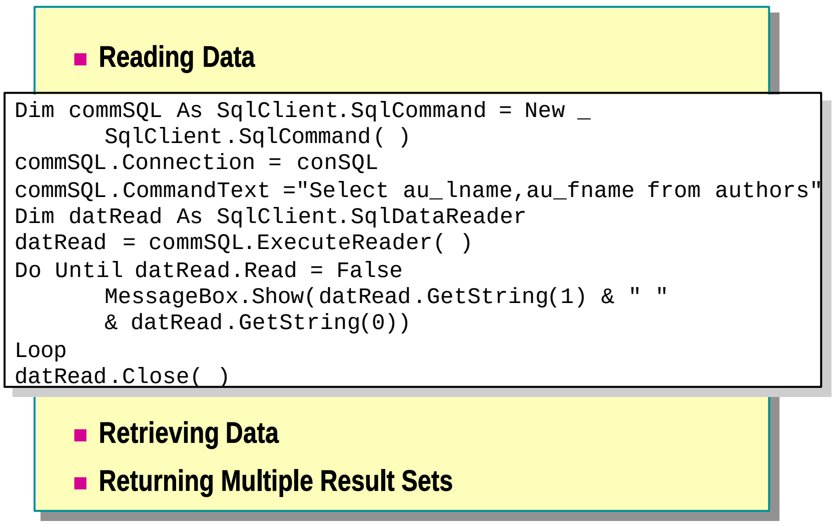
Delivery Tip
Do not go into detail about the **DataReader** object at this point because it will be covered later in this lesson.

Tip If you are running a query that will only return one row, you can improve the performance of your application by returning this data as output parameters from a stored procedure.

Using the DataReader Object

Topic Objective
To discuss how to use the **DataReader** object.

Lead-in
The **DataReader** object gives you fast, efficient access to a stream of data.



- Reading Data

```
Dim commSQL As SqlClient.SqlCommand = New _
       SqlClient.SqlCommand( )
commSQL.Connection = conSQL
commSQL.CommandText ="Select au_lname,au_fname from authors"
Dim datRead As SqlClient.SqlDataReader
datRead = commSQL.ExecuteReader( )
Do Until datRead.Read = False
       MessageBox.Show(datRead.GetString(1) & " "
       & datRead.GetString(0))
Loop
datRead.Close( )
```

- Retrieving Data
- Returning Multiple Result Sets

You can use the **DataReader** object to create a read-only, forward-only stream of data. This is an efficient method for accessing data that you only need to read through once. You can improve application performance by using this object because it holds only a single row of data at a time in memory instead of caching the entire set of records.

There are two versions of this object:

- **SqlDataReader** for SQL Server databases
- **OleDbDataReader** for other data sources

The **SqlDataReader** object contains some methods that are not available to the **OleDbDataReader**. These are **GetSQL***type* methods that you can use to retrieve SQL Server–specific data type columns from the data source.

Reading Data

You can instantiate the **DataReader** object by using the **ExecuteReader** method of the **Command** object. After you create the **DataReader**, you can call the **Read** method to obtain data in the rows. You can access the columns by name, ordinal number, or native type in conjunction with ordinal number.

You must ensure that you use the **Close** method of the **DataReader** object before accessing any output or return parameters from a stored procedure.

The following example shows how to retrieve data by using the **DataReader** object:

```
Dim conSQL As SqlClient.SqlConnection
conSQL = New SqlClient.SqlConnection( )
conSQL.ConnectionString = "Integrated Security=True;" & _
"Data Source=LocalHost;Initial Catalog=Pubs;"
conSQL.Open( )

Dim commSQL As SqlClient.SqlCommand = New _
SqlClient.SqlCommand( )
commSQL.Connection = conSQL
commSQL.CommandText = "Select au_lname, au_fname from authors"

Dim datRead As SqlClient.SqlDataReader
datRead = commSQL.ExecuteReader( )
Do Until datRead.Read = False
MessageBox.Show(datRead(1).ToString & " " &
datRead(0).ToString)
Loop
datRead.Close( )
```

Retrieving Data

Because you will often know the data types of your return data, you can use the **Get** methods to retrieve data in columns by specifying their data type. This approach can improve application performance because no type conversion is required, but your data and output types must be identical.

The following example shows how to use the **GetString** method to retrieve data. With **GetString**, you no longer need the **ToString** method shown in the preceding example.

```
Do Until datRead.Read = False
MessageBox.Show(datRead.GetString(1) & " " &
datRead.GetString(0))
Loop
```

Returning Multiple Result Sets

Sometimes you will issue commands that return more than one result set. By default, the **DataReader** will only read the first result set. You can use the **NextResult** method of the **DataReader** to retrieve the next result set into the **DataReader** object. If there are no more result sets, this method returns **False**.

The following example shows how to create a stored procedure that returns two result sets from a SQL Server database:

```
CREATE PROCEDURE MultiResult AS
Select * from authors
Select * from titles
Return 0
GO
```

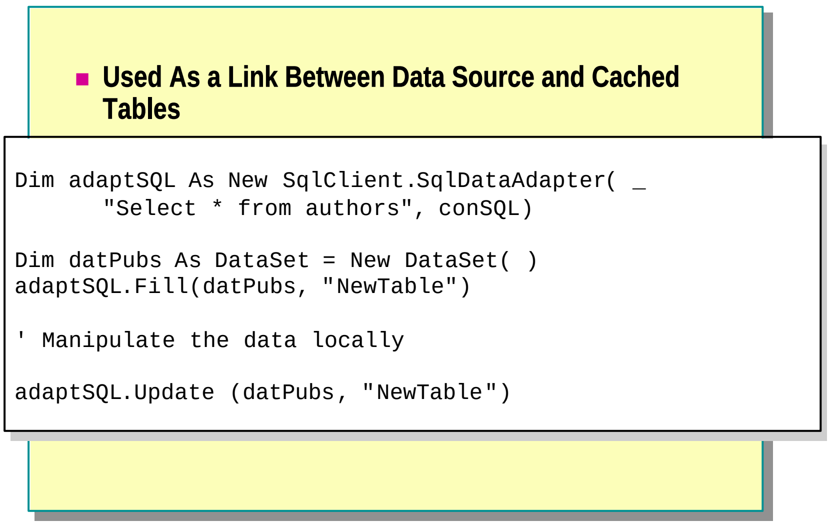
The following example shows how to execute the stored procedure **MultiResult** and access the information contained in each result set:

```
Dim conSQL As SqlClient.SqlConnection
conSQL = New SqlClient.SqlConnection( )
conSQL.ConnectionString = "Integrated Security=True;" & _
"Data Source=LocalHost;Initial Catalog=Pubs;"
conSQL.Open( )

Dim commSQL As SqlClient.SqlCommand = New _
SqlClient.SqlCommand( )
commSQL.Connection = conSQL
commSQL.CommandType = CommandType.StoredProcedure
commSQL.CommandText = "MultiResult"

Dim datRead As SqlClient.SqlDataReader
datRead = commSQL.ExecuteReader( )
Do
  Do Until datRead.Read = False
      MessageBox.Show(datRead.GetString(1))
  Loop
Loop While datRead.NextResult

datRead.Close( )
```

Using the DataAdapter Object

Topic Objective
To discuss how to use the **DataAdapter** object.

Lead-in
The **DataAdapter** serves as the link between a data source and a **DataSet**. It is used to retrieve data and insert it into **DataTable** objects for disconnected use.



- **Used As a Link Between Data Source and Cached Tables**

```
Dim adaptSQL As New SqlClient.SqlDataAdapter( _
      "Select * from authors", conSQL)

Dim datPubs As DataSet = New DataSet( )
adaptSQL.Fill(datPubs, "NewTable")

' Manipulate the data locally

adaptSQL.Update (datPubs, "NewTable")
```

You can use the **DataAdapter** object to exchange data between a data source and a **DataSet**. You can use it to retrieve appropriate data and insert it into **DataTable** objects within a **DataSet**, and to update changes from the **DataSet** back into the data source.

Creating the DataAdapter

There are two ways to create a **DataAdapter** object:

- Use an existing, open **Connection** object.
- Open the **Connection** as needed

Using an Existing Connection Object

Create a **Command** object within a **Connection** object, and assign the **SelectCommand** property of the previously instantiated **DataAdapter** object to that command. This technique is useful if you need to create a **Connection** object specifically for the **DataAdapter** object to use.

The following example shows how to use **Connection** and **Command** objects to instantiate a **DataAdapter**:

```
Dim conSQL As SqlClient.SqlConnection
conSQL = New SqlClient.SqlConnection( )
conSQL.ConnectionString = "Integrated Security=True;" & _
"Data Source=LocalHost;Initial Catalog=Pubs;"
conSQL.Open( )

Dim comSQL As SqlClient.SqlCommand
comSQL = New SqlClient.SqlCommand( )
comSQL.Connection = conSQL
comSQL.CommandText = "Select * from authors"

Dim adaptSQL As SqlClient.SqlDataAdapter
adaptSQL = New SqlClient.SqlDataAdapter( )
adaptSQL.SelectCommand = comSQL
```

Using a Closed Connection

Instantiate the **DataAdapter** object, passing a query string and a **Connection** object. The **DataAdapter** will check whether the **Connection** is open, and, if it is not open, it will open it for you and close it when your method call is complete. This method is useful if you have already set the properties of a **Connection** object in your application and only need the connection to be opened to populate the data tables.

The following example shows how to instantiate a **DataAdapter** object:

```
Private conSQL as SqlClient.SqlConnection

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MyBase.Load
  conSQL = New SqlClient.SqlConnection( )
  conSQL.ConnectionString = "Integrated " & _
      "Security=True;Data Source" & _
      "=LocalHost;Initial Catalog=Pubs;"
  End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button1.Click
  Dim adaptSQL As New SqlClient.SqlDataAdapter( _
  "Select * from authors", conSQL)
End Sub
```

Filling the DataTable

After the **DataAdapter** object is created, you use the **Fill** method, passing a **DataSet** and, optionally, the required **DataTable** name as parameters. You can then work with the data in your application, and, if required, you can use the **Update** method of the **DataAdapter** to synchronize those changes back to the data source.

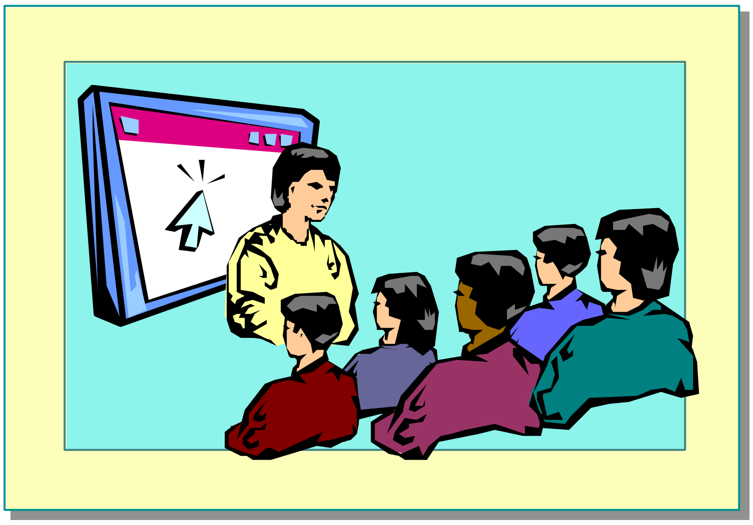
You can use a single **DataAdapter** to fill and update multiple **DataSets**. A single **DataAdapter** is linked to a particular **DataSet** only when a method is actually being called. The following example shows how to use a **DataAdapter** to fill a **DataSet**:

```
Dim conSQL As SqlClient.SqlConnection
conSQL = New SqlClient.SqlConnection( )
conSQL.ConnectionString = "Integrated Security=True;" & _
"Data Source=LocalHost;Initial Catalog=Pubs;"
conSQL.Open( )

Dim adaptSQL As New SqlClient.SqlDataAdapter("Select * from
authors", conSQL)

Dim datPubs As DataSet = New DataSet( )
adaptSQL.Fill(datPubs, "NewTable")

'Manipulate the data locally using the DataSet

adaptSQL.Update (datPubs, "NewTable")
```

Demonstration: Retrieving Data Using ADO .NET

Topic Objective
To demonstrate how to retrieve data from a SQL Server database by using ADO .NET.

Lead-in
Having reviewed the code to retrieve data by using ADO .NET, you will now see the code working.



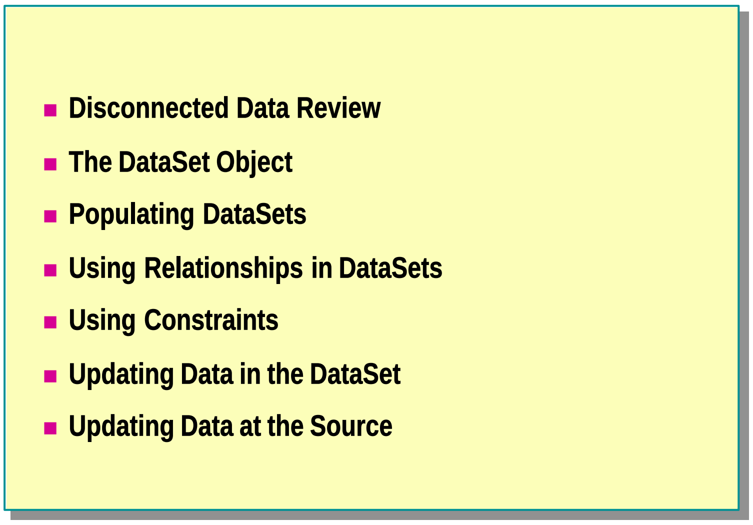
Delivery Tip
The step-by-step instructions for this demonstration are in the instructor notes for this module.

In this demonstration, you will learn how to retrieve data from a SQL Server database by using the **SQLDataReader** object in a Visual Basic .NET–based application.

◆ The DataSet Object

Topic Objective
To provide an overview of the topics covered in this lesson.

Lead-in
You have just seen how to populate a **DataSet** by using a **DataAdapter**. Now you to use this object



- Disconnected Data Review
- The DataSet Object
- Populating DataSets
- Using Relationships in DataSets
- Using Constraints
- Updating Data in the DataSet
- Updating Data at the Source

DataSets are the primary object that you will work with when accessing disconnected sets of data. They are similar in concept to groups of ADO disconnec ted recordsets, but in ADO .NET, there are many enhancements, including the ability to relate tables together.

In this lesson, you will learn how to create DataSets and populate tables within them. You will also learn how to edit these tables and propagate those changes to the data source.

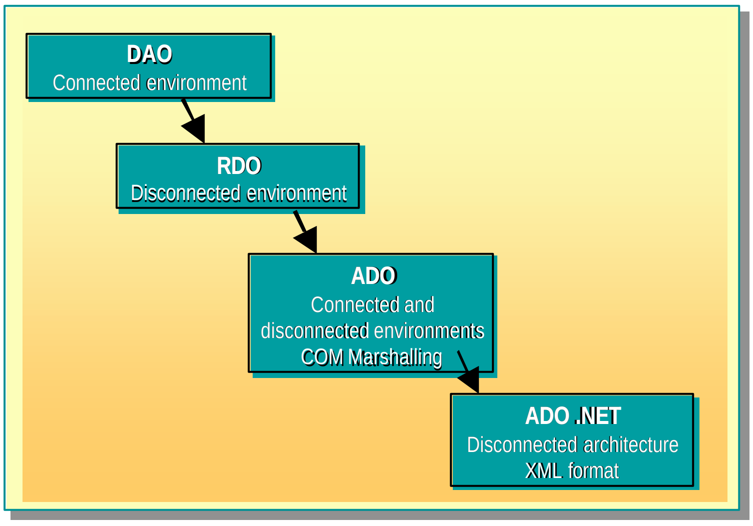
After completing this lesson, you will be able to:

- Create DataSets and populate tables within them.
- Edit tables within DataSets.
- Propagate changes to the data source.

Disconnected Data Review

Topic Objective
To review the concepts of disconnected data, to ensure understanding for the rest of this lesson.

Lead-in
Disconnected data is an important part of contemporary application development.



Delivery Tip
The slide associated with this topic is an animated slide.
It begins by showing only the DAO information. The other boxes are revealed on subsequent mouse clicks.

Each new data access technology has improved on the concept of disconnected data, but ADO .NET is the first one to provide a truly enterprise-wide solution.

Problems with Two-Tier Applications

In traditional two-tier applications, a data source connection was often made at the start of the application and held open until the application ended. This can cause many problems, including:

- Poor performance

  Database connections use valuable system resources, such as memory and CPU utilization. The database server performance will be affected if a large number of connections are needlessly held open.

- Limited scalability

  Applications that consume a large number of database connections are not scalable because most data sources can only support a limited number of connections.

Disconnected Data in RDO and ADO

To overcome these problems, Remote Data Objects (RDO) and ADO introduced the concept of disconnected data. This was implemented so that you could retrieve a set of records, disconnect from the data source, and work with the data locally. You could then reconnect and submit your changes to the database. The **Recordsets** were marshaled between the tiers as COM objects, requiring that both the server and client computer could handle COM components.

Disconnected Data in ADO .NET

Delivery Tip
Students may be concerned that using a text-based format for transmission may compromise security. Remind them that you can use Internet standard security measures such as Secure Sockets Layer (SSL), TLS, and Internet Protocol Security (IPSec) to ensure the privacy of data.

ADO .NET is designed for use in the Internet world, whereas COM may not be supported by all tiers, and may not be transmitted through firewalls. The disconnected architecture has been updated from the previous two-tier, RDO, and ADO architectures.

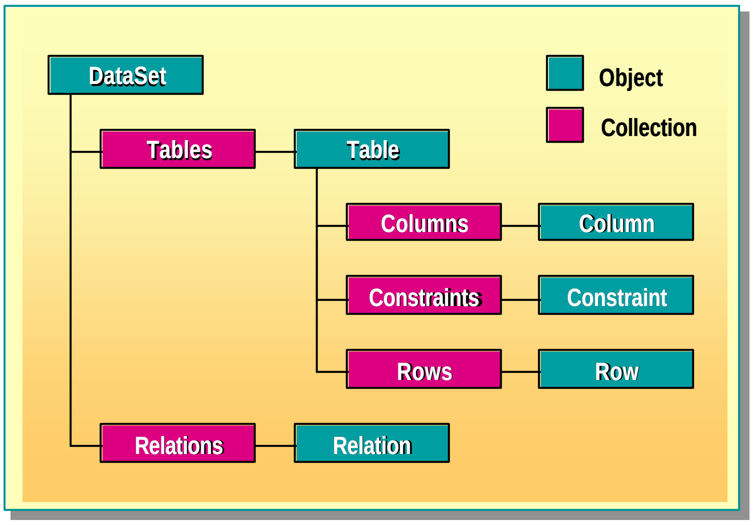
ADO .NET uses XML as its transmission format. This is a text-based format, alleviating the problems associated with the transmission of COM objects and ensuring true cross-platform interoperability.

ADO .NET provides you with a new object for the caching of data on the client computer. This object is known as a **DataSet**. This object is automatically disconnected from the data source but maintains the ability to later update the source based on changes made at the client.

The DataSet Object

Topic Objective
To introduce the **DataSet** object.

Lead-in
The **DataSet** object is a disconnected cache of data.



The **DataSet** object is a disconnected, memory resident cache of data. It is structured in a similar manner to a database in that it contains **DataTable**, **DataRelation**, and **Constraint** objects.

DataSets

A typical use of a **DataSet** is through Web Services. A client application will make a request for data to a Web Service that will populate a **DataSet** (using a **DataAdapter**) and return it to the client. The client can then view and modify the **DataSet** by using properties and methods that are consistent with database operations, and then pass it back to the Web Service. The Web Service will then update the database with the clients' changes. The **DataSet** is transmitted between tiers as XML, which means that it can be also be used by non-ADO .NET clients.

DataTables and DataRelations

The **DataSet** contains the **Tables** and **Relations** collections. Using objects within these two collections, you can build up a group of related tables within your **DataSet**. The **DataTable** object consists of the **Columns** collection and the **Rows** collection. You can use the objects in these collections to manipulate the fields and query their properties. The **Relations** collection contains definitions of all the relationships between the **DataTable** objects in the **DataSet**. You can use these to enforce constraints on your data or to navigate across tables.

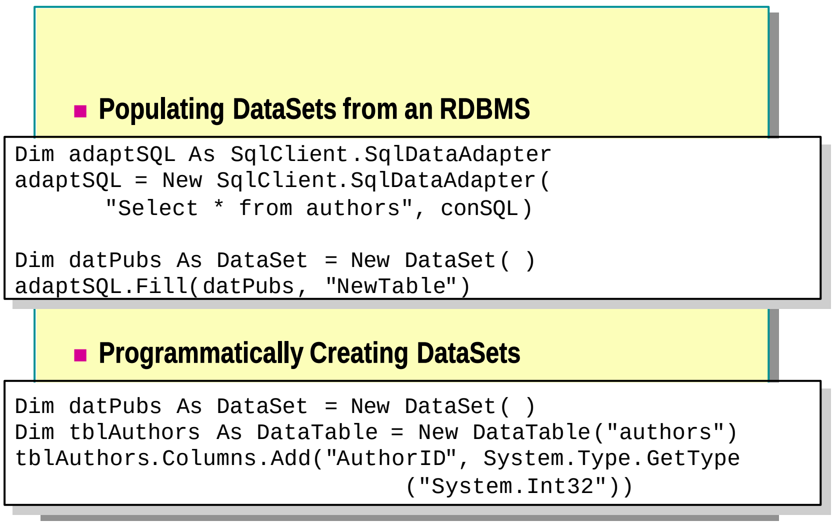
The System.Data Namespace

The **System.Data** namespace contains the **DataSet** and its objects because they are generic ways of handling data. Unlike the Provider objects, there are not different objects for different data sources.

Populating DataSets

Topic Objective
To discuss how to create and populate **DataSets**.

Lead-in
You can populate **DataSets** in three different ways.



- Populating DataSets from an RDBMS

```
Dim adaptSQL As SqlClient.SqlDataAdapter
adaptSQL = New SqlClient.SqlDataAdapter(
       "Select * from authors", conSQL)

Dim datPubs As DataSet = New DataSet( )
adaptSQL.Fill(datPubs, "NewTable")
```

- Programmatically Creating DataSets

```
Dim datPubs As DataSet = New DataSet( )
Dim tblAuthors As DataTable = New DataTable("authors")
tblAuthors.Columns.Add("AuthorID", System.Type.GetType
                              ("System.Int32"))
```

Delivery Tip
Populating **DataSets** from XML data will be covered in the last lesson of this module.

Because a **DataSet** is simply a memory resident representation of data, you do not necessarily need to take it from a traditional data source, such as a relational database management system (RDBMS) or a message store. You can create it at run time to manipulate data created within an application, or you can use it to view XML data.

Populating DataSets from an RDBMS

You use a **DataAdapter** to access data stored in a database, and store the data in **DataTable** objects within a **DataSet** in your application.

The following example shows how to populate a **DataTable** called *NewTable* with data from a SQL Server database:

```
Dim conSQL As SqlClient.SqlConnection
conSQL = New SqlClient.SqlConnection( )
conSQL.ConnectionString = "Integrated Security=True;" & _
  "Data Source=LocalHost;Initial Catalog=Pubs;"
conSQL.Open( )

Dim adaptSQL As SqlClient.SqlDataAdapter
adaptSQL = New SqlClient.SqlDataAdapter("Select * from
authors", conSQL)

Dim datPubs As DataSet = New DataSet( )
adaptSQL.Fill(datPubs, "NewTable")
```

Programmatically Creating DataSets

You sometimes need to work with non-standard data sources. In this situation, you can programmatically create **DataSets**, **DataTables**, **DataRelations**, and **Constraints**, and then populate the tables with your data. This will give you the ability to use standard ADO .NET functions to access your data.

For Your Information
The addition of data and constraints to this table will be covered later in this lesson.

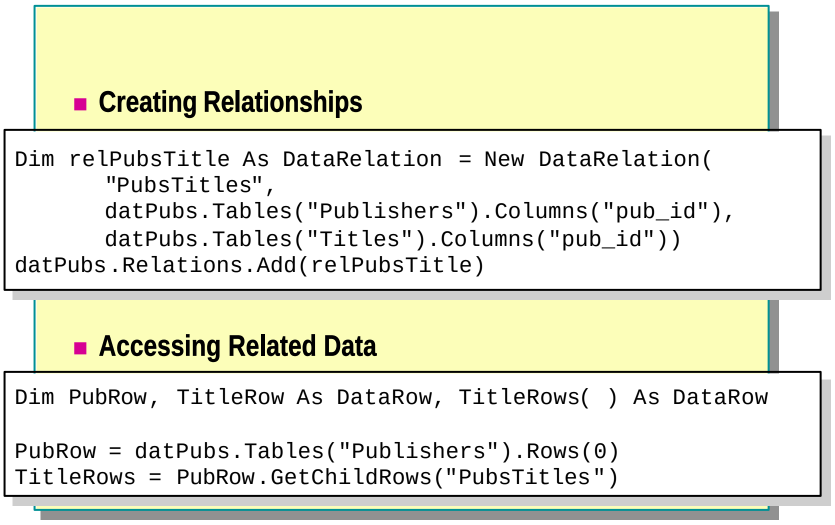
The following example shows how to create a **DataSet** containing a **DataTable** with three **DataColumns**. This could then be extended to add more columns, and then populate them with data.

```
Dim conSQL As SqlClient.SqlConnection
conSQL = New SqlClient.SqlConnection( )
conSQL.ConnectionString = "Integrated Security=True;" & _
"Data Source=LocalHost;Initial Catalog=Pubs;"
conSQL.Open( )

Dim adaptSQL As SqlClient.SqlDataAdapter
adaptSQL = New SqlClient.SqlDataAdapter("Select * from
authors", conSQL)

Dim datPubs As DataSet = New DataSet( )
Dim tblAuthors As DataTable = New DataTable("authors")
tblAuthors.Columns.Add("AuthorID", _
System.Type.GetType("System.Int32"))
tblAuthors.Columns.Add("au_lname", _
System.Type.GetType("System.String"))
tblAuthors.Columns.Add("au_fname", _
System.Type.GetType("System.String"))
```

Using Relationships in DataSets

Topic Objective
To discuss how to use **DataRelations** within a **DataSet** to create and maintain relationships.

Lead-in
Relationships are the basis of an RDBMS.



- Creating Relationships

```
Dim relPubsTitle As DataRelation = New DataRelation(
      "PubsTitles",
      datPubs.Tables("Publishers").Columns("pub_id"),
      datPubs.Tables("Titles").Columns("pub_id"))
datPubs.Relations.Add(relPubsTitle)
```

- Accessing Related Data

```
Dim PubRow, TitleRow As DataRow, TitleRows( ) As DataRow

PubRow = datPubs.Tables("Publishers").Rows(0)
TitleRows = PubRow.GetChildRows("PubsTitles")
```

The basis of most RDBMSs is the ability to relate tables to each other. ADO .NET provides this ability within **DataSets** through the **DataRelation** class.

Each **DataRelation** object contains an array of **DataColumn** objects that define the parent column or columns, or primary key, and the child column or columns, or foreign key, in the relationship. Referential integrity is maintained by the relationship, and you can specify how to deal with related changes.

Creating Relationships

The following example shows how to create a relationship between two **DataTable** objects in a **DataSet**. The same **DataAdapter** is used to populate the **DataTable** objects, and then a **DataRelation** is created between the two.

```
Dim conSQL As SqlClient.SqlConnection
conSQL = New SqlClient.SqlConnection( )
conSQL.ConnectionString = "Integrated Security=True;" & _
"Data Source=LocalHost;Initial Catalog=Pubs;"
conSQL.Open( )

Dim adaptSQL As SqlClient.SqlDataAdapter
Dim datPubs As DataSet = New DataSet( )

adaptSQL = New SqlClient.SqlDataAdapter("Select pub_id," & _
  "pub_name, city, state from publishers", conSQL)
adaptSQL.Fill(datPubs, "Publishers")
adaptSQL = New SqlClient.SqlDataAdapter("Select pub_id," & _
"title, type, price from titles", conSQL)
adaptSQL.Fill(datPubs, "Titles")

Dim relPubsTitle As DataRelation
relPubsTitle = New DataRelation("PubsTitles", _
datPubs.Tables("Publishers").Columns("pub_id"), _
datPubs.Tables("Titles").Columns("pub_id"))
datPubs.Relations.Add(relPubsTitle)
```

Accessing Related Data

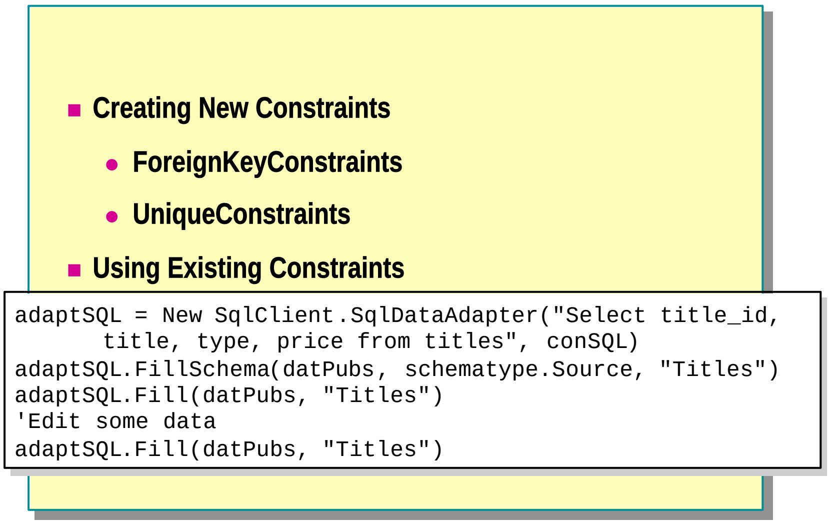
The main use of a **DataRelation** is to allow access to related records in a different table. You can do this by using the **GetChildRows** method of a **DataRow** object that returns an array of **DataRow** objects. The following example shows how to use this method to access the child rows that match the first publisher by using the relationship created in the previous example:

```
Dim PubRow, TitleRow As DataRow
Dim TitleRows( ) As DataRow 'Array of DataRow objects

PubRow = datPubs.Tables("Publishers").Rows(0)
TitleRows = PubRow.GetChildRows("PubsTitles")

Dim i As Integer
For i = 0 To UBound(TitleRows)
  TitleRow = TitleRows(i)
  listBox1.Items.Add(TitleRow("title").ToString)
Next i
```

Using Constraints

Topic Objective
To explain the importance of constraints and how to use them.

Lead-in
Constraints are needed to ensure the validity of data.



- **Creating New Constraints**
  - **ForeignKeyConstraints**
  - **UniqueConstraints**
- **Using Existing Constraints**

```
adaptSQL = New SqlClient.SqlDataAdapter("Select title_id,
      title, type, price from titles", conSQL)
adaptSQL.FillSchema(datPubs, schematype.Source, "Titles")
adaptSQL.Fill(datPubs, "Titles")
'Edit some data
adaptSQL.Fill(datPubs, "Titles")
```

You can create your own constraints within a **DataSet**, or you can copy the existing constraints from the data source. Each of these options is available to you in ADO .NET.

Creating New Constraints

You can apply two types of constraint classes to **DataColumns**: **ForeignKeyConstraint** and **UniqueConstraint**.

ForeignKeyConstraint

This constraint controls what happens to a child row when a parent row is updated or deleted. You can specify different behaviors for different circumstances. The following table shows the values for the **DeleteRule** and **UpdateRule** properties of the **ForeignKeyConstraint**.

Value	Description
Cascade	Deletes or updates any child records based on the parent record
SetNull	Sets related values to **DBNull**
SetDefault	Sets related values to their defaults
None	Does not affect related rows

The following example shows how to apply a foreign key constraint with specific actions between two tables in an existing **DataSet**. If a row in the parent table is deleted, the child value will be set to **DBNull**. If a row in the parent table is updated, the child values will be also be updated.

```
Dim colParent As DataColumn
Dim colChild As DataColumn
Dim fkcPubsTitles As ForeignKeyConstraint

colParent = datPubs.Tables("publishers").Columns("pub_id")
colChild = datPubs.Tables("titles").Columns("pub_id")
fkcPubsTitles = New _
  ForeignKeyConstraint("PubsTitlesFKConstraint", colParent, _
  colChild)

fkcPubsTitles.DeleteRule = Rule.SetNull
fkcPubsTitles.UpdateRule = Rule.Cascade
        datPubs.Tables("titles").Constraints.Add(fkcPubsTitles)
datPubs.EnforceConstraints = True
```

UniqueConstraint

This constraint can be added to one column or to an array of columns. It ensures that all values in the column or columns are unique. When this constraint is added, ADO .NET verifies that the existing data does not violate the constraint and maintains the setting for all changes to that **DataTable**.

The following example shows how to add a **UniqueConstraint** to a column:

```
Dim ucTitles As UniqueConstraint
ucTitles = New UniqueConstraint("UniqueTitles", _
  datPubs.Tables("titles").Columns("title"))
datPubs.EnforceConstraints = True
```

Using Existing Constraints

If constraints already exist in the RDBMS, you can copy them directly into your **DataSet**. This can save a lot of time that might be spent coding for frequently occurring problems. For example, if you fill a **DataSet**, modify some data, and then use **Fill** again to return to the original data, all the rows will be appended to your existing **DataTable** s, unless you define primary keys. You can avoid this type of problem by copying the table schema.

The following example shows how to use the **FillSchema** method to copy constraint information into a **DataSet**:

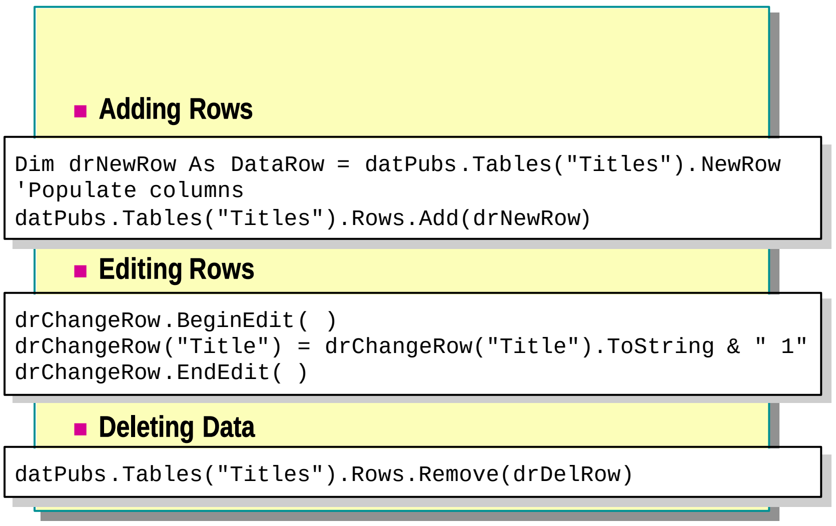
```
adaptSQL = New SqlClient.SqlDataAdapter("Select title_id," & _
  "title, type, price from titles", conSQL)
adaptSQL.FillSchema(datPubs, schematype.Source, "Titles")
adaptSQL.Fill(datPubs, "Titles")
'Edit some data
adaptSQL.Fill(datPubs, "Titles")
```

Note Constraints are automatically added to columns when you create a relationship between them. A **UniqueConstraint** is added to the primary key, and a **ForeignKeyConstraint** is added to the foreign key.

Updating Data in the DataSet

Topic Objective
To explain how to update data in a **DataSet** and pass those changes back to the underlying data source.

Lead-in
Once you have access to data, you are likely to want to be able to change some of it.



- **Adding Rows**

```
Dim drNewRow As DataRow = datPubs.Tables("Titles").NewRow
'Populate columns
datPubs.Tables("Titles").Rows.Add(drNewRow)
```

- **Editing Rows**

```
drChangeRow.BeginEdit( )
drChangeRow("Title") = drChangeRow("Title").ToString & " 1"
drChangeRow.EndEdit( )
```

- **Deleting Data**

```
datPubs.Tables("Titles").Rows.Remove(drDelRow)
```

After you have created a **DataSet** of **DataTables**, you might want to add, update, and delete data. Any changes you make to the data are stored in memory and later used to apply the changes to the data source.

Adding Rows

Use the following steps to add new rows to a table:

1. Instantiate a **DataRow** object by using the **NewRow** method of the **DataTable**.
2. Populate the columns with data.
3. Call the **Add** method of the **DataRows** collection, passing the **DataRow** object.

The following example shows how to add rows to a **DataSet**:

```
Dim drNewRow As DataRow = datPubs.Tables("Titles").NewRow
drNewRow("title") = "New Book"
drNewRow("type") = "business"
datPubs.Tables("Titles").Rows.Add(drNewRow)
```

Editing Rows

Use the following steps to edit existing rows:

1. Call the **BeginEdit** method of the row.
2. Change the data in the columns.
3. Call **EndEdit** or **CancelEdit** to accept or reject the changes.

The following example shows how to edit data in an existing column:

```
Dim drChangeRow As DataRow = datPubs.Tables("Titles").Rows(0)
drChangeRow.BeginEdit( )
drChangeRow("Title") = drChangeRow("Title").ToString & " 1"
drChangeRow.EndEdit( )
```

Deleting Data

Use either of the following methods to delete a row:

- **Remove** method

  Call the **Remove** method of the **DataRows** collection. This permanently removes the row from the **DataSet**.

- **Delete** method

  Call the **Delete** method of the **DataRow** object. This only marks the row for deletion in the **DataSet**, and calling **RejectChanges** will undo the deletion.

The following example shows how to delete an existing row from a **DataSet**:

```
Dim drDelRow As DataRow = datPubs.Tables("Titles").Rows(0)
datPubs.Tables("Titles").Rows.Remove(drDelRow)
```

Confirming the Changes

To update the **DataSet**, you use the appropriate methods to edit the table, and then call **AcceptChanges** or **RejectChanges** for the individual rows or for the entire table.

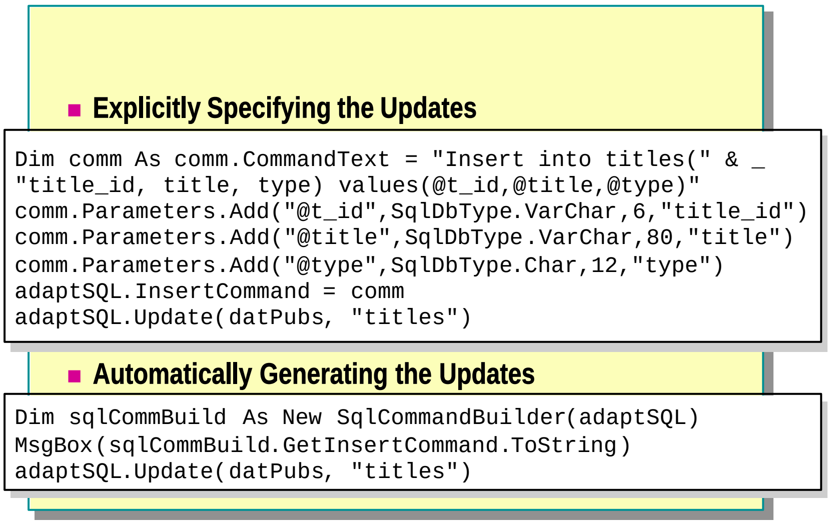
You can discover whether any changes have been made to a row since **AcceptChanges** was last called by querying its **RowState** property. The following table describes the valid settings for this property.

Value	Description
Unchanged	No changes have been made.
Added	The row has been added to the table.
Modified	Something in the row has been changed.
Deleted	The row has been deleted by the **Delete** method.
Detached	The row has been deleted, or the row has been created, but the **Add**method has not been called.

Updating Data at the Source

Topic Objective
To discuss how to update changes from the **DataSet** to the source.

Lead-in
After you have updated your **DataSet**, you will want to replicate those changes to the data source.



- **Explicitly Specifying the Updates**

```
Dim comm As comm.CommandText = "Insert into titles(" & _
"title_id, title, type) values(@t_id,@title,@type)"
comm.Parameters.Add("@t_id",SqlDbType.VarChar,6,"title_id")
comm.Parameters.Add("@title",SqlDbType.VarChar,80,"title")
comm.Parameters.Add("@type",SqlDbType.Char,12,"type")
adaptSQL.InsertCommand = comm
adaptSQL.Update(datPubs, "titles")
```

- **Automatically Generating the Updates**

```
Dim sqlCommBuild As New SqlCommandBuilder(adaptSQL)
MsgBox(sqlCommBuild.GetInsertCommand.ToString)
adaptSQL.Update(datPubs, "titles")
```

After you have updated the tables in your **DataSet**, you will want to replicate those changes to the underlying data source. To do this, you use the **Update** method of the **DataAdapter** object, which is the link between **DataSet** and data source.

The **Update** method, like the **Fill** method, takes two parameters: the **DataSet** in which the changes have been made and the name of the **DataTable** in which the changes are. It determines the changes to the data and executes the appropriate SQL command (Insert, Update or Delete) against the source data.

Explicitly Specifying the Updates

You use the **InsertCommand**, **UpdateCommand**, and **DeleteCommand** properties of the **DataAdapter** to identify the changes occurring in your DataSet. You specify each of these as an existing command object for an Insert, Update, or Delete SQL statement. For any variable columns in the statements, you use **SqlParameter** objects to identify the column, data type, size, and data to be inserted.

The following example shows how to use the **InsertCommand** property to add a row to the Titles table in the Pubs database:

```
Dim conSQL As SqlClient.SqlConnection
Dim adaptSQL As SqlClient.SqlDataAdapter
Dim datPubs As DataSet = New DataSet( )

conSQL = New SqlClient.SqlConnection( )
conSQL.ConnectionString = "Integrated Security=True;" & _
  "Data Source=LocalHost;Initial Catalog=Pubs;"
conSQL.Open( )

adaptSQL = New SqlClient.SqlDataAdapter("Select pub_id," & _
  "title_id, title, type, price from titles", conSQL)
adaptSQL.Fill(datPubs, "Titles")

Dim drNewRow As DataRow = datPubs.Tables("Titles").NewRow
drNewRow("title_id") = "hg3454"
drNewRow("title") = "New Book"
drNewRow("type") = "business"
datPubs.Tables("Titles").Rows.Add(drNewRow)

Dim comm As SqlClient.SqlCommand
comm = New SqlClient.SqlCommand( )
comm.Connection = conSQL
comm.CommandText = "Insert into titles(title_id, title, " & _
"type) values (@title_id,@title,@type)"

comm.Parameters.Add("@title_id", SqlDbType.VarChar, 6, _
"title_id")
comm.Parameters.Add("@title", SqlDbType.VarChar, 80, "title")
comm.Parameters.Add("@type", SqlDbType.Char, 12, "type")

adaptSQL.InsertCommand = comm
adaptSQL.Update(datPubs, "titles")
```

Automatically Generating the Updates

If your DataTable is generated from only one table in the data source, you can use the **CommandBuilder** object to automatically create the **InsertCommand**, **UpdateCommand**, and **DeleteCommand** properties.

The following example shows how to use the **CommandBuilder** object to achieve the same results as the previous example:

```
Dim conSQL As SqlClient.SqlConnection
Dim adaptSQL As SqlClient.SqlDataAdapter
Dim datPubs As DataSet = New DataSet( )

conSQL = New SqlClient.SqlConnection( )
conSQL.ConnectionString = "Integrated Security=True;" & _
  "Data Source=LocalHost;Initial Catalog=Pubs;"
conSQL.Open( )

adaptSQL = New SqlClient.SqlDataAdapter("Select pub_id," & _
  "title_id, title, type, price from titles", conSQL)
adaptSQL.Fill(datPubs, "Titles")

Dim drNewRow As DataRow = datPubs.Tables("Titles").NewRow
drNewRow("title_id") = "hg8765"
drNewRow("title") = "New Book"
drNewRow("type") = "business"
datPubs.Tables("Titles").Rows.Add(drNewRow)

Dim sqlCommBuild As New SqlCommandBuilder(adaptSQL)
adaptSQL.Update(datPubs, "titles")
```

Note Even though the automatically generated commands can simplify your coding, you will improve performance by using the **InsertCommand**, **UpdateCommand**, and **DeleteCommand** properties.

Practice: Using DataSets

Topic Objective
To practice using **DataSets**.

Lead-in
In this practice, you will update existing data in a SQL Server database using automatically generated commands.

In this practice, you will update existing data in a SQL Server database by using automatically generated commands.

⇙ To review the application

1. Open Visual Studio .NET.
2. On the **File** menu, point to **Open**, and click **Project**. Set the location to *install folder*\Practices\Mod08\DataSets\Starter, click **DataSets.sln**, and then click **Open**.
3. Run the application, and click **Populate DataSet** and **Refresh List from Data Source**. Note that the list on the left displays data from a **DataSet**, and the list on the right uses a **DataAdapter** object. Quit the application.
4. Review the code in Form1.vb to see how the **DataSet** was created.

⇙ To update the local DataSet

1. In **btnEditName_Click** procedure, locate the comment "add code to update the dataset here."
2. Add the following lines below the comment to update the local dataset:

```
objDataTable = dsCustomers.Tables("customers")
objDataRow = objDataTable.Rows(iCust)
objDataRow("lastname") = strNewName
```

To automatically generate a command

1. In **btnUpdate_Click** procedure, locate the comment “add code to update the data source here.”
2. To generate the update command automatically, add the following lines below the comment:

```
Dim sqlCommBuild As New _
SqlClient.SqlCommandBuilder(adaptCustomers)
adaptCustomers.Update(dsCustomers, "customers")
```

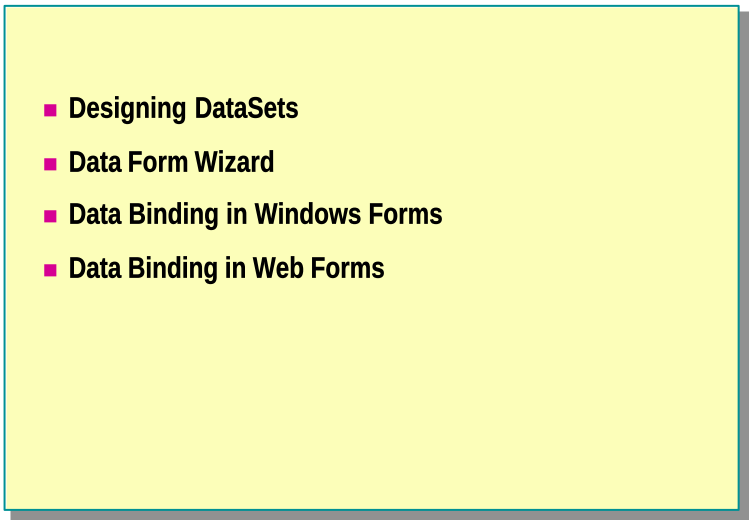
To test your code

1. Run the application, and click **Populate DataSet** and **Refresh List from Data Source**.
2. Click the first name in the **Local DataSet** box. Click **Edit Name**, enter your last name, and then click **OK**. Click **Refresh List from DataSet**, and note that the change has been made to the local dataset. Click **Refresh List from Data Source**, and note that the underlying data source still contains the original data.
3. Click **Update Changes**. Click **Refresh List from Data Source**, and verify that the changes have now been replicated to the underlying data source.

◆ Data Designers and Data Binding

Topic Objective
To provide an overview of the topics covered in this lesson.

Lead-in
Data binding is an important feature of Visual Basic.



- **Designing DataSets**
- **Data Form Wizard**
- **Data Binding in Windows Forms**
- **Data Binding in Web Forms**

Data binding has been an important part of Visual Basic data development for a long time. The tools included in Visual Basic .NET have been enhanced to allow easier creation of data-bound forms and to take advantage of the new features in ADO .NET.

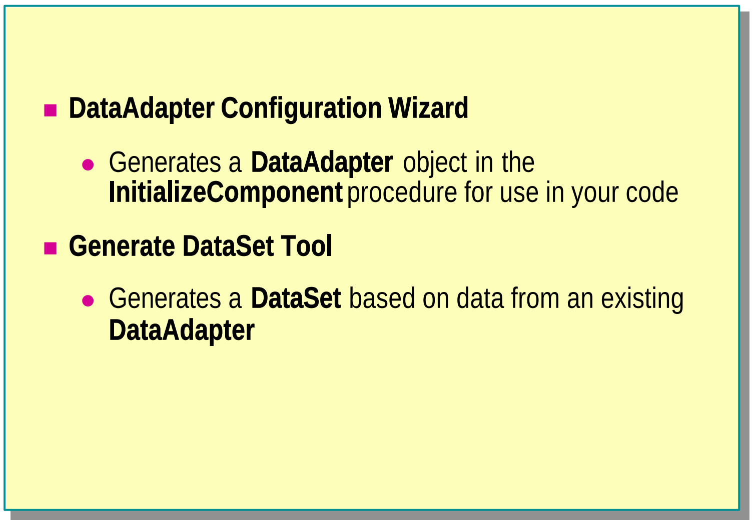
After completing this lesson, you will be able to:

- Describe the data designers available in Visual Basic .NET.
- Create data-bound Windows Forms and Web Forms.

Designing DataSets

Topic Objective
To discuss how to create connections, commands, and **DataSet**s by using the Visual Basic designers.

Lead-in
Visual Basic .NET includes a number of designers to simplify the process of **DataSet** creation.



- **DataAdapter Configuration Wizard**
  - Generates a **DataAdapter** object in the **InitializeComponent** procedure for use in your code
- **Generate DataSet Tool**
  - Generates a **DataSet** based on data from an existing **DataAdapter**

Visual Basic .NET includes a number of designers to simplify the process of DataSet creation. These include the Connection Wizard, DataAdapter Configuration Wizard, and the Generate DataSet Tool.

Delivery Tip
Remind students that these wizards just automate the manual process that they have learned about already.

DataAdapter Configuration Wizard

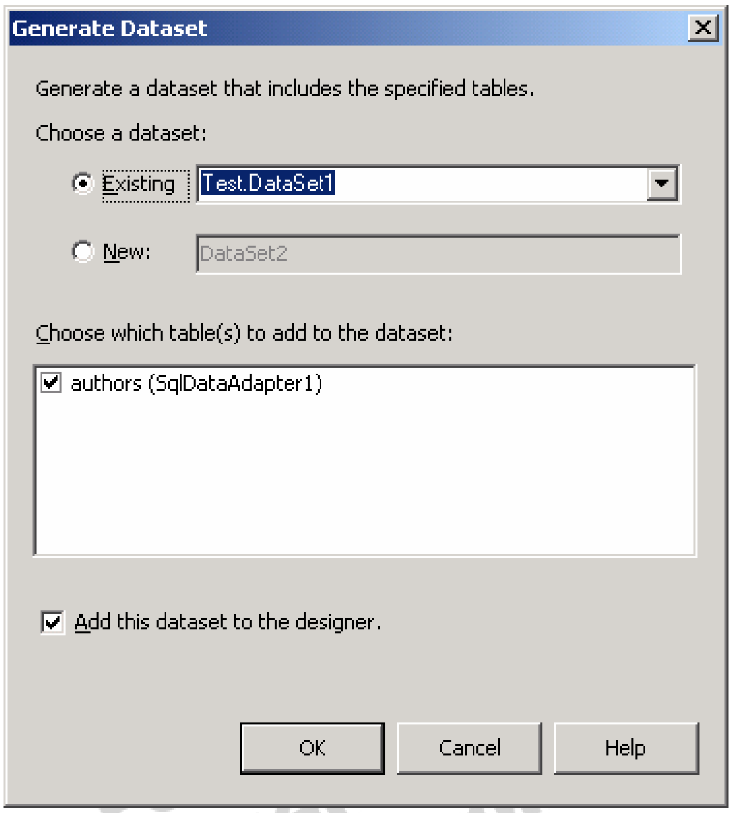
This wizard leads you through the steps needed to create a **DataAdapter** object within an existing connection. You can initiate the wizard by adding a **SqlDataAdapter** or **OleDbDataAdapter** from the Toolbox to a form at design time. It requires the following information to generate a **DataAdapter** object for use in your form:

- Connection name
- Query type
  - SQL statement
  - New stored procedure
  - Existing stored procedure
- Details of the chosen query

Once you have created the **DataAdapter**, you can view the code created by the wizard in the **InitializeComponent** procedure.

Generate DataSet Tool

This tool allows you to generate a **DataSet** automatically from a **DataAdapter**.



Generate Dataset

Generate a dataset that includes the specified tables.

Choose a dataset:

[✓ Existing] Test.DataSet1

New: DataSet2

Choose which table(s) to add to the dataset:

✓ authors (SqlDataAdapter1)

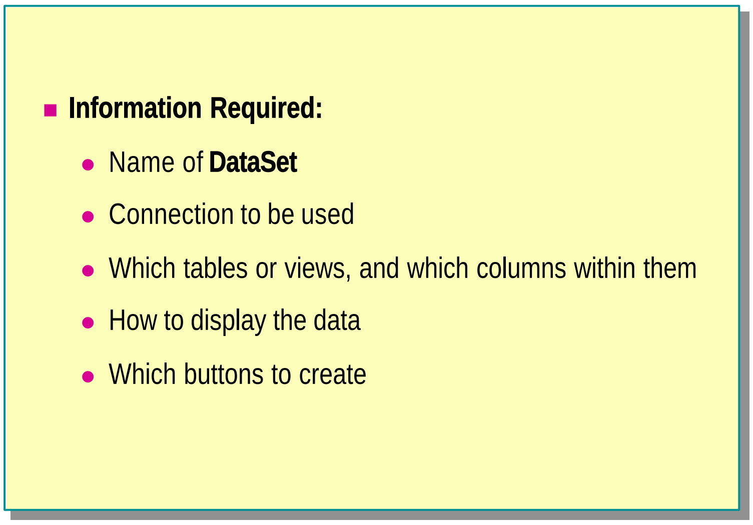
✓ Add this dataset to the designer.

Again, once created, you can use this **DataSet** in the usual way in your code.

Data Form Wizard

Topic Objective
To explain how to use the Data Form Wizard to generate a data-bound form.

Lead-in
You can use the Data Form Wizard to automatically bind data to controls on a form.



- **Information Required:**
  - Name of **DataSet**
  - Connection to be used
  - Which tables or views, and which columns within them
  - How to display the data
  - Which buttons to create

You can use the Data Form Wizard to automatically bind data to controls on a form. You can specif y to use an existing DataSet in the project, which will then use your pre-written methods for data access, or to create a new DataSet based on information supplied to the wizard.

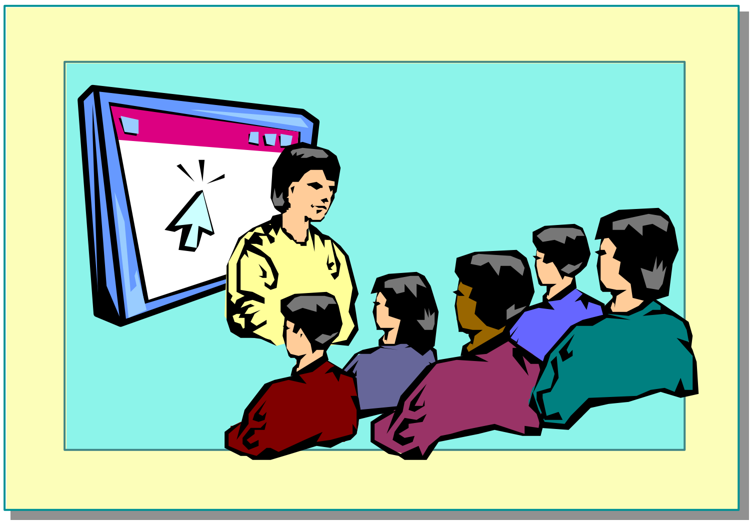
If you wish to create a new DataSet, the Data Form Wizard will require the following information:

- Name of the DataSet to create
- Connection to be used (you can create a new connection at this point)
- Which tables or views to use (if more than one, you can also identify the relationship between them)
- Which columns to include on the form
- How to display the data (data grid or bound controls)
- Whether to include navigation and editing buttons

Demonstration: Using the Data Form Wizard

Topic Objective
To demonstrate how to use the Data Form Wizard.

Lead-in
In this demonstration, you will learn how to use the Data Form Wizard to create a data-bound form from a new **DataSet**.



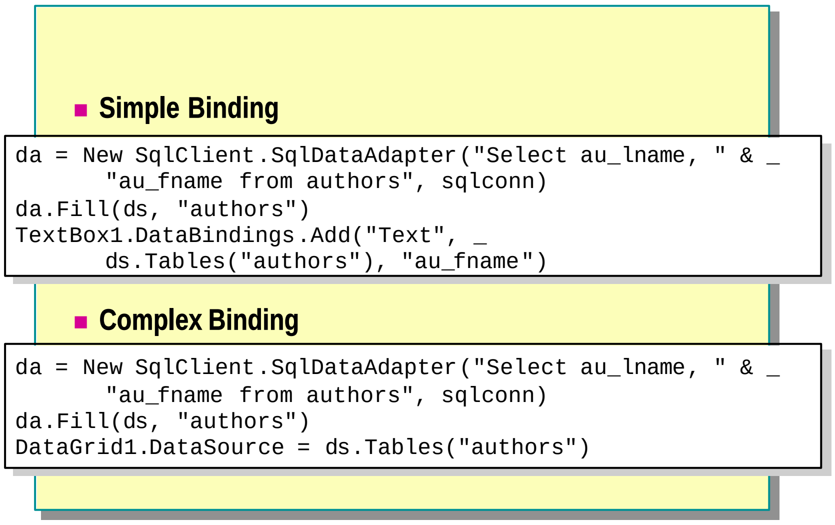
Delivery Tip
The step-by-step instructions for this demonstration are in the instructor notes for this module.

In this demonstration, you will learn how to use the Data Form Wizard to create a data-bound form from a new **DataSet**.

Data Binding in Windows Forms

Topic Objective
To discuss data binding in Windows Forms.

Lead-in
Windows Forms data binding allows for more control over the appearance of your application than applications created with the Data Form Wizard.



- **Simple Binding**

```
da = New SqlClient.SqlDataAdapter("Select au_lname, " & _
       "au_fname from authors", sqlconn)
da.Fill(ds, "authors")
TextBox1.DataBindings.Add("Text", _
       ds.Tables("authors"), "au_fname")
```

- **Complex Binding**

```
da = New SqlClient.SqlDataAdapter("Select au_lname, " & _
       "au_fname from authors", sqlconn)
da.Fill(ds, "authors")
DataGrid1.DataSource = ds.Tables("authors")
```

If you want more control over the appearance of your forms, you can use data binding to design them yourself.

There are two general types of binding that can be used: simple binding and complex binding. Both can be performed at design time by using the Properties window or at run time by using code.

Simple Binding

You use simple binding to link a control to a single field in a **DataSet**. For example, you would use simple binding for a **TextBox** control. Using the **DataBindings** property of a data-aware control, you can specify which **DataSet** and which field to bind to which property.

The following example shows how to bind data to a **TextBox** control:

```
Dim sqlconn As SqlClient.SqlConnection
Dim da As SqlClient.SqlDataAdapter
Dim ds As New DataSet( )
sqlconn = New SqlClient.SqlConnection( )
sqlconn.ConnectionString = "Integrated Security=True;" & _
  "Data Source=LocalHost;Initial Catalog=Pubs;"
sqlconn.Open( )
da = New SqlClient.SqlDataAdapter("Select au_lname, " & _
  "au_fname from authors", sqlconn)
da.Fill(ds, "authors")

TextBox1.DataBindings.Add("Text", _
  ds.Tables("authors"), "au_fname")
TextBox2.DataBindings.Add("Text", _
  ds.Tables("authors"), "au_lname")
```

Complex Binding

You use complex binding to link a control to multiple fields in a **DataSet.** For example, you would use complex binding for a **DataGrid** control. These controls have a **DataSource** property that allows you to specify the table to be used.

The following example shows how to use the **DataSource** property of a **DataGrid** control:

```
Dim sqlconn As SqlClient.SqlConnection
Dim da As SqlClient.SqlDataAdapter
Dim ds As New DataSet( )
sqlconn = New SqlClient.SqlConnection( )
sqlconn.ConnectionString = "Integrated Security=True;" & _
  "Data Source=LocalHost;Initial Catalog=Pubs;"
sqlconn.Open( )
da = New SqlClient.SqlDataAdapter("Select au_lname, " & _
  "au_fname from authors", sqlconn)
da.Fill(ds, "authors")
DataGrid1.DataSource = ds.Tables("authors")
```

Delivery Tip
Remind students that this was covered in the previous lesson.

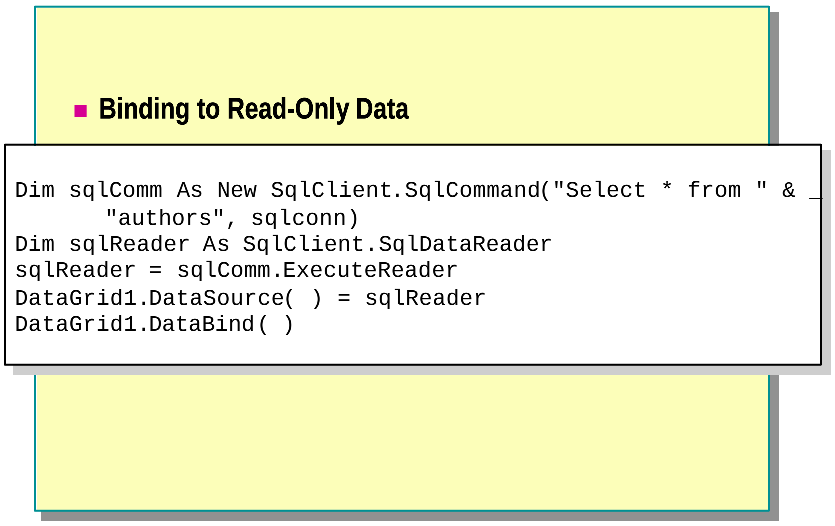
Updating Data

As with manual coding of data access, changing values in data-bound forms only applies to the local **DataSet**. To write these changes to the underlying data source, you must add your own code by using the **Update** method of the **DataAdapter**.

Data Binding in Web Forms

Topic Objective
To discuss how to use data binding in Web forms.

Lead-in
Few Web applications can function without some sort of data display.



- Binding to Read-Only Data

```
Dim sqlComm As New SqlClient.SqlCommand("Select * from " & _
       "authors", sqlconn)
Dim sqlReader As SqlClient.SqlDataReader
sqlReader = sqlComm.ExecuteReader
DataGrid1.DataSource( ) = sqlReader
DataGrid1.DataBind( )
```

Most data displayed on Web Forms will be read only, so you do not need to incorporate the overhead of using a **DataSet** in your Web Form applications; you can use the more efficient **DataReader** object instead. If you want your users to be able to edit data on the Web Form, you must code the edit, update, and cancel events yourself.

Binding to Read-Only Data

You use the **DataBind** method of the **DataGrid** server control to bind data to a grid on a Web Form. The following example shows how to do this using a **DataReader** object:

```
Dim sqlconn As SqlClient.SqlConnection
Dim da As SqlClient.SqlDataAdapter
Dim ds As New DataSet( )
sqlconn = New SqlClient.SqlConnection( )
sqlconn.ConnectionString = "Integrated Security=True;" & _
  "Data Source=LocalHost;Initial Catalog=Pubs;"
sqlconn.Open( )

Dim sqlComm As New SqlClient.SqlCommand("Select * from " & _
  "authors", sqlconn)
Dim sqlReader As SqlClient.SqlDataReader
sqlReader = sqlComm.ExecuteReader
DataGrid1.DataSource( ) = sqlReader
DataGrid1.DataBind( )
```

The **DataGrid** will not be visible until you call the **DataBind** method.

◆ XML Integration

Topic Objective
To provide an overview of the topics covered in this lesson.

Lead-in
XML is tightly integrated into the .NET platform.

- Why Use Schemas?
- Describing XML Structure
- Creating Schemas
- Using XML Data and Schemas in ADO .NET
- DataSets and XmlDataDocuments

Delivery Tip
This lesson assumes that students have reasonable knowledge of XML and an awareness of its associated technologies.
You may find that you need to provide more background information if students are not familiar with these concepts.

Traditionally, XML and ADO data have been two distinct entities, but ADO .NET brings them together and allows you to work with both types in the same way.

XML is tightly integrated into the .NET platform. You have already seen how DataSets are transmitted by using XML format, and now you will learn how DataSets are literally represented as XML and how their structure is defined in an XML Schema Definition (XSD).

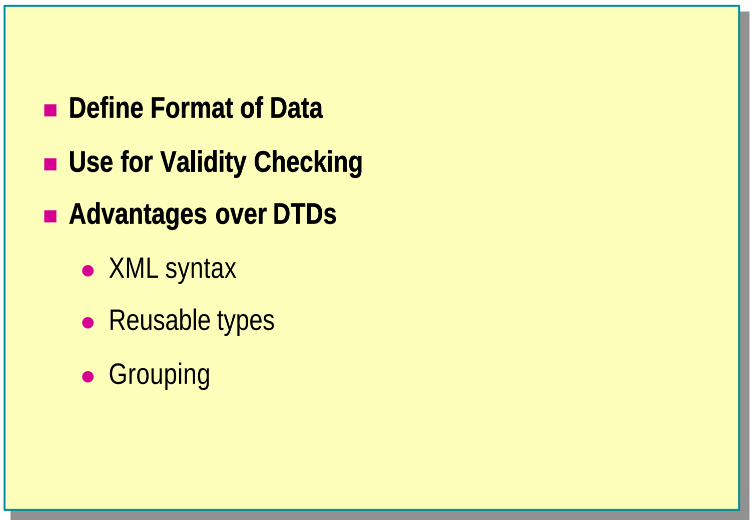
After completing this lesson, you will be able to:

- Describe what an XML schemas is.
- Explain why XML schemas are useful to the Visual Basic .NET developer.
- Create schemas.
- Manipulate XML data within an ADO .NET DataSet by means of an **XMLReader**.

Why Use Schemas?

Topic Objective
To discuss XML schemas and the role they can play in Visual Basic .NET.

Lead-in
XML schemas are an essential part of the developer's tool kit.



- **Define Format of Data**
- **Use for Validity Checking**
- **Advantages over DTDs**
  - XML syntax
  - Reusable types
  - Grouping

When working with traditional database applications, you often need to write validation code to ensure that the data you are inputting matches the database schema. If you do not do this, then you need to write error-handling code for the potential errors that may occur. Either way, there must be some way of checking. The solution to this problem when you are working with XML data is XML schemas.

XML schemas are similar in concept to database schemas. They define the elements and attributes that may appear in your XML documents, and how these elements and attributes relate to each other. Schemas are very important to ensure that the data you are using conforms to your specification. When loading XML data, you can check it against the schema to validate that none of the data entering your system is in an incorrect format. This is becoming more of an issue as business-to-business and business-to-customer commerce becomes more prevalent in the Internet world.

Visual Studio .NET uses XSD to create schemas. This syntax is currently at working-draft status at the World Wide Web Consortium (W3C), but it has many advantages over document type definitions (DTDs).

Delivery Tip
If students are not familiar with DTDs, do not spend too much time justifying the use of a new technology to them.

- XML syntax

  DTDs are written using a DTD syntax, which is not related to any of the other Internet standards currently in use. XSD uses XML syntax, which enables developers to validate data without needing to learn yet another language.
- Reusable types

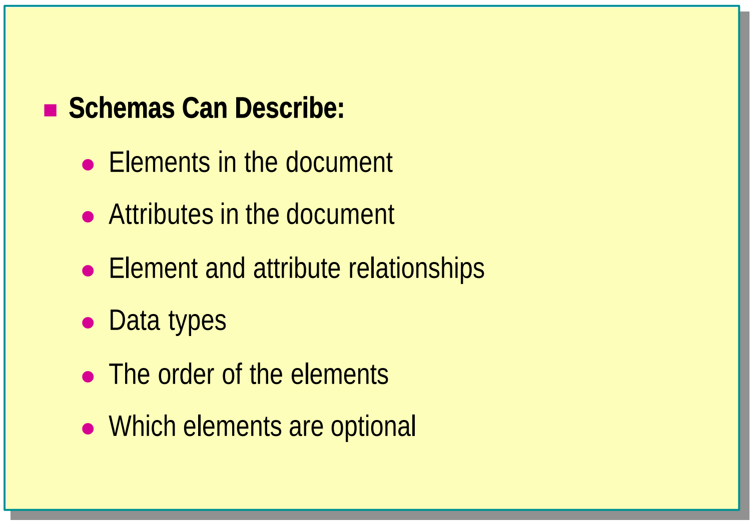
  XSD allows you to define complex data types and reuse those within your schema.
- Grouping

  You can specify that a set of elements always exists as a group and stipulate the order in which they must appear.

Describing XML Structure

Topic Objective
To explain how a schema document describes XML data.

Lead-in
So, what is contained in the schema document?



- **Schemas Can Describe:**
  - Elements in the document
  - Attributes in the document
  - Element and attribute relationships
  - Data types
  - The order of the elements
  - Which elements are optional

Schemas describe the structure of an XML document, and you can use them to validate data within that document. A schema document can describe all or some of the following:

- Elements and attributes contained within the XML document
- Element and attribute relationships
- Data types
- The order of the elements
- Which elements are optional

For example, consider the following XML document:

```
<?xml version="1.0" ?>
<pubs>
  <Publishers>
      <pub_id>0736</pub_id>
      <pub_name>Lucerne Publishing</pub_name>
      <city>Boston</city>
      <state>MA</state>
      <country>USA</country>
  </Publishers>
  <Publishers>
      <pub_id>0877</pub_id>
      <pub_name>Litware, Inc.</pub_name>
      <city>Washington</city>
      <state>DC</state>
      <country>USA</country>
  </Publishers>
</pubs>
```

This document consists of a <pubs> element containing individual <Publishers> elements. Each of these contains a <pub_id>, <pub_name>, <city>, <state>, and <country> element. This defines the structure of the document.

After the XML document is linked to a schema document describing the structure, the schema can be used to verify data being input into the document.

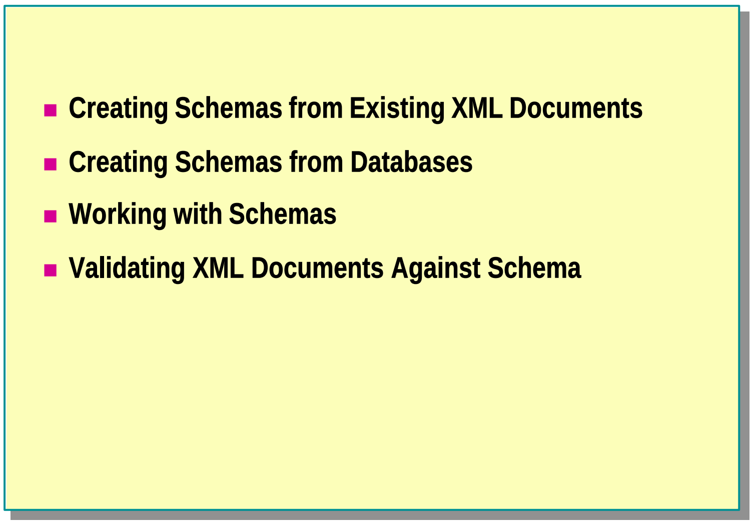
The following example shows the schema generated for this document:

```
<xsd:schema id="pubs"
  targetNamespace="http://tempuri.org/Publishers.xsd"
  xmlns="http://tempuri.org/Publishers.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
  attributeFormDefault="qualified"
  elementFormDefault="qualified">
  <xsd:element name="pubs" msdata:IsDataSet="true"
      msdata:EnforceConstraints="False">
      <xsd:complexType>
          <xsd:choice maxOccurs="unbounded">
              <xsd:element name="Publishers">
                  <xsd:complexType>
                      <xsd:sequence>
                          <xsd:element name="pub_id"
                              type="xsd:string" minOccurs="0" />
                          <xsd:element name="pub_name"
                              type="xsd:string" minOccurs="0" />
                          <xsd:element name="city"
                              type="xsd:string" minOccurs="0" />
                          <xsd:element name="state"
                              type="xsd:string" minOccurs="0" />
                          <xsd:element name="country"
                              type="xsd:string" minOccurs="0" />
                      </xsd:sequence>
                  </xsd:complexType>
              </xsd:element>
          </xsd:choice>
      </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Creating Schemas

Topic Objective
To discuss how to create schemas in Visual Basic .NET.

Lead-in
Schemas can be generated automatically, or they can be explicitly created.



- **Creating Schemas from Existing XML Documents**
- **Creating Schemas from Databases**
- **Working with Schemas**
- **Validating XML Documents Against Schema**

Schemas are automatically generated for you when you work with DataSets. You may find that there are situations when you want to create your own. One example would be when you are exchanging data with a business partner and want to define the structure of that data.

Creating Schemas from Existing XML Documents

You can add an existing XML document to a project and automatically create a schema based on the contents of the document.

When you are working with XML data, an XML menu becomes available. This menu allows you to generate schema and validate data against the schema. When you create schemas from existing documents, all data types are declared as strings. You can alter this manually in the XML Designer.

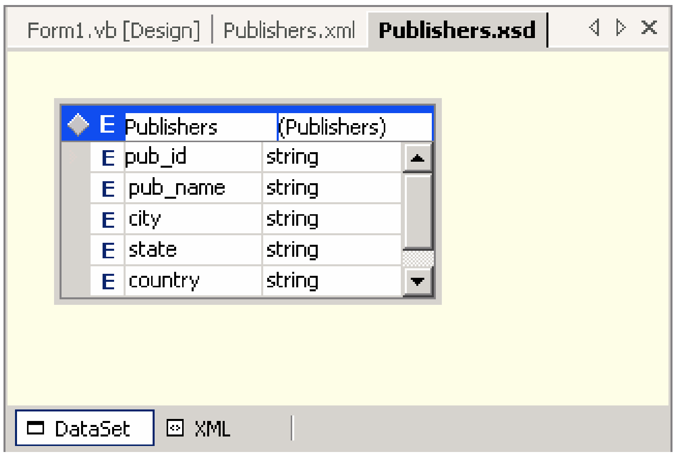
Creating Schemas from Databases

You can also create an XSD schema from the structure of existing data. To do this, you add a new schema item to your project and drag on the chosen tables or views from the hierarchy displayed in the Data Connections of Server Explorer. Again, the data types will need editing.

Working with Schemas

You can view schemas in Visual Studio .NET either as XML or in the Designer Window, which is a user-friendly display of the schema. In this window, you can edit existing schemas (adding, removing, or modifying elements and attributes) and create new schemas.

The following illustration shows the Designer Window:



E	Publishers	(Publishers)
E	pub_id	string
E	pub_name	string
E	city	string
E	state	string
E	country	string

Validating XML Documents Against Schema

You can use the XML Designer to validate data against XML schema. If you have an existing document that you are adding data to, you can use this technique to validate that the new data conforms to the structure described in the schema.

Use the following steps to validate an XML document against an XML schema.

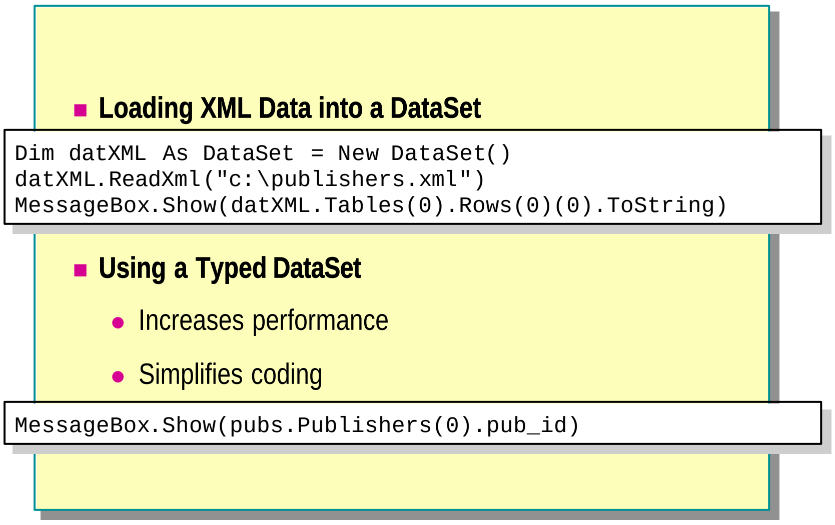
1. Load the XML document into the XML Designer.
2. On the XML menu, click Validate XML Data.

Any validation errors will be noted in the Task List.

Using XML Data and Schemas in ADO .NET

Topic Objective
To discuss how to load XML data into an ADO .NET **DataSet**.

Lead-in
Accessing XML data by means of ADO .NET solves an issue that developers have had in the past.



- **Loading XML Data into a DataSet**

```
Dim datXML As DataSet = New DataSet()
datXML.ReadXml("c:\publishers.xml")
MessageBox.Show(datXML.Tables(0).Rows(0)(0).ToString)
```

- **Using a Typed DataSet**
  - Increases performance
  - Simplifies coding

```
MessageBox.Show(pubs.Publishers(0).pub_id)
```

One of the issues experienced by developers in the past has been trying to manipulate XML data within their applications. In earlier data access technologies, there was no ability to do this. In ADO .NET, you can populate a **DataSet** from an XML document. This allows you to access the data in a simple fashion.

Loading XML Data into a DataSet

You can load an existing XML document into a **DataSet** by using the **ReadXML** method of the **DataSet**. This requires one argument of the fully qualified path and file name of the XML document to be loaded.

The following example shows how to load the data and display the first column in the first row:

```
Dim datXML As DataSet = New DataSet( )
datXML.ReadXml("c:\publishers.xml")
Dim drFirstRow As DataRow = datXML.Tables(0).Rows(0)
MessageBox.Show(drFirstRow(0).ToString)
```

Using Typed DataSets

DataSets can be *typed* or *untyped*. A typed **DataSet** is simply a **DataSet** that uses information in a schema to generate a derived **DataSet** class containing objects and properties based on the structure of the data.

You can create a typed **DataSet** by using the **Generate DataSet** command from the schema view. You can add a DataSet object to a form, and link it to the typed dataset already in the project, and then work with the DataSet in the usual way.

In untyped DataSets, you have been accessing the columns and rows as collections in the hierarchy. In a typed **DataSet**, you can access these directly as objects, as shown in the following example:

```
MessageBox.Show(pubs.Publishers(0).pub_id)
```

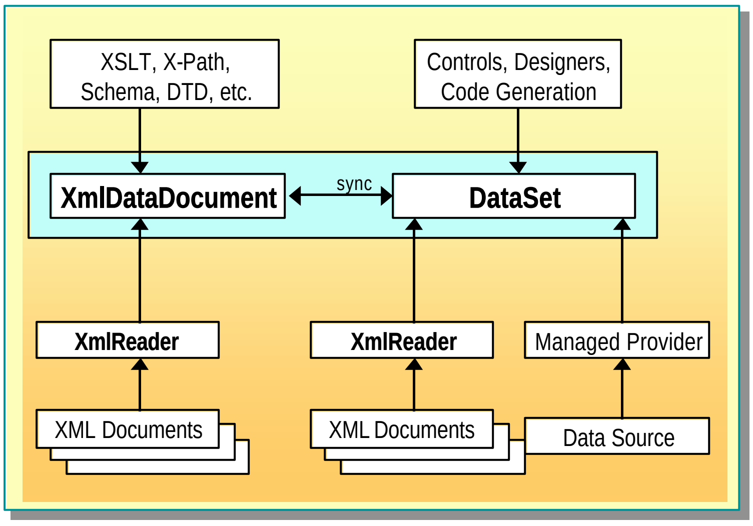
Typed **DataSets** also provide compile-time type checking and better performance than untyped **DataSets**.

Note **DataSets** created using the XML Designer tools are automatically typed.

DataSets and XmlDataDocuments

Topic Objective
To provide an overview of the relationship between **DataSets** and **XmlDataDocuments**.

Lead-in
XML developers have traditionally used the XML DOM to manipulate their XML documents.



Delivery Tip
This topic is not meant to cover how to use **XmlDataDocuments** but to simply provide an overview of the link with **DataSets**.

XML developers have traditionally used the Document Object Model (DOM) to manipulate their XML documents. This is a standard COM interface to XML data, and it allows access to the elements and attributes contained within the document.

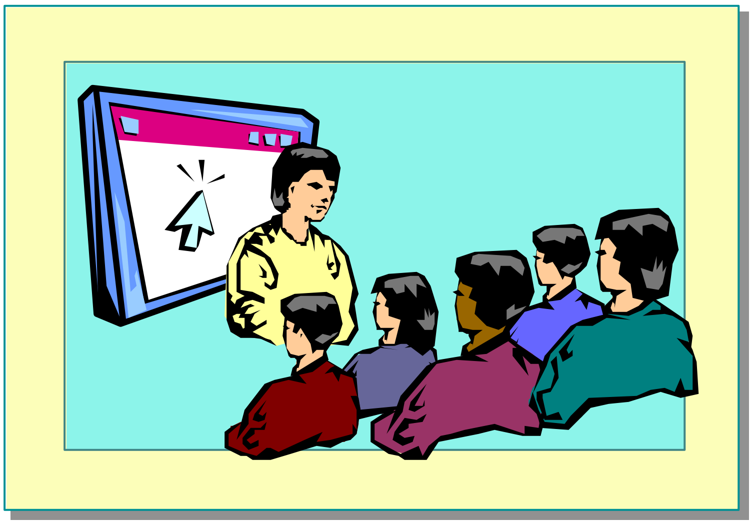
The **XmlDataDocument** in the .NET Framework extends the **DOMDocument** object in the XML DOM to allow .NET developers to use the same functionality. This object is tightly integrated with the ADO .NET **DataSet**, and loading data into either object synchronizes it with the other.

Any manipulation of data by means of the **DataSet** or the **XmlDataDocument** is synchronized in the other. Therefore, if you add a row to the **DataSet**, an element is added to the **XmlDataDocument**, and vice versa.

Demonstration: Using XML Schema

Topic Objective
To demonstrate using XML documents by means of **DataSet**s.

Lead-in
This demonstration will show you how to use an XML schema to validate and XML document.



In this demonstration, you will learn how to create an XML schema from an existing XML document, and how to then use the schema to validate the document.

Lab 8.1: Creating Applications That Use ADO .NET

Topic Objective
To introduce the lab.

Lead-in
In this lab, you will use ADO.NET to access data.

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Retrieve, insert, and update data by using ADO .NET.
- Use data binding on a Web Form.
- Use the XML Designer to create a typed dataset.

Prerequisites

Before working on this lab, you must be familiar with:

- Creating Visual Basic .NET Windows Forms and Web Forms.
- Using ADO .NET data readers.
- Using ADO .NET DataSets.
- Working with XML and XSD.

Scenario

In this lab, you will use ADO .NET to access data for the Cargo application. You will create both Windows Forms and Web Forms that access both customer and invoice information. You will also use the XML Designer to create a typed DataSet.

Starter and Solution Files

There are starter and solution files associated with this lab. The starter files are in the *install folder*\Labs\Lab081\Starter folder, and the solution files are in the *install folder*\Labs\Lab081\Solution folder.

Estimated time to complete this lab: 60 minutes

Exercise 1 Retrieving Data

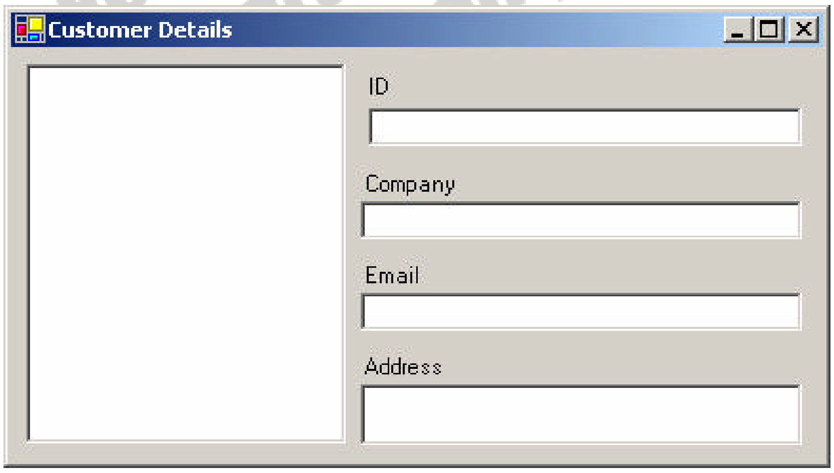
In this exercise, you will create a Windows Forms application to display a list of customers and their details. You will create a **DataSet** object in a class module and use this data set to obtain the customer details for the selected customer.

To create the project

1. Open Visual Studio .NET.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Project Types** box, click **Visual Basic Projects**.
4. In the **Templates** box, click **Windows Application**.
5. Change the name of the project to **CustomerProject**, set the location to *install folder*\Labs\Lab081\Ex01, and then click **OK**.

To create the Windows Form

1. In Solution Explorer, rename the default form to **frmCustomer.vb**.
2. In the Properties window, change the **Text** property of the form to **Customer Details**, and change the **Name** property to **frmCustomer**.
3. Open the Project Property Pages dialog box, and change the Startup object to **frmCustomer**.
4. Add controls to the form, as shown in the following illustration.



5. Set the properties of the controls as shown in the following table.

Control	Property name	Property value
ListBox1	**Name**	**lstCustomers**
Label1	**Name**	**lblID**
	Text	**ID**
TextBox1	**Name**	**txtID**
	Text	**<empty>**
Label2	**Name**	**lblCompany**
	Text	**Company**
TextBox2	**Name**	**txtCompany**
	Text	**<empty>**
Label3	**Name**	**lblEmail**
	Text	**Email**
TextBox3	**Name**	**txtEmail**
	Text	**<empty>**
Label4	**Name**	**lblAddress**
	Text	**Address**
TextBox4	**Name**	**txtAddress**
	Text	**<empty>**
	Multiline	**True**
	Size.Height	**32**

6. Save the project.

To create the Cargo class

1. On the **Project** menu, click **Add Class**.
2. Rename the class **Cargo.vb**, and then click **Open**.
3. At the top of the class, add an **Imports** statement to reference the **System.Data.SqlClient** namespace.
4. In the **Cargo** class, create private class-level variables, using the information in the following table.

Variable name	Type
dstCargo	**DataSet**
conCargo	**SqlConnection**

To create the connection

1. Create a constructor for the class.
2. Instantiate the *conCargo* **Connection** object.
3. Using the information in the following table, set the **ConnectionString** property of *conCargo*.

Name	Value
Data Source	**localhost**
Initial Catalog	**Cargo**
Integrated Security	**True**

To create the GetCustomers method

1. Create a method called **GetCustomers** for the class. The method takes no arguments and has no return value.
2. In this method, declare and instantiate a **SqlDataAdapter** object called *adpCust.* Pass the following information to the constructor.

Argument	Value
selectCommandText	**Select CustomerID, FirstName, LastName, CompanyName, Email, Address from Customers**
selectConnection	**conCargo**

3. Instantiate the **DataSet** object.
4. Use the **Fill** method of *adpCust* to pass the following information.

Argument	Value
dataSet	**dstCargo**
srcTable	**CustTable**

To create the CustList property

1. Define a read-only property called **CustList** that returns a **DataTable**.
2. In the **Get** clause, return the **CustTable** table from *dstCargo*.
3. Save and build the project.

To populate the list box

1. Open the frmCustomers Code Editor.
2. Declare and instantiate a private class-level variable called *objCargo* of type **Cargo**.
3. Create the **frmCustomers_Load** event handler.
4. Call the **GetCustomers** method of the *objCargo* object.
5. Add the following code to loop through the customers and populate the list box:

```
Dim CurRows( ) As DataRow, CurRow As DataRow
CurRows = objCargo.CustList.Select( )
For Each CurRow In CurRows
   lstCustomers.Items.Add(CurRow("FirstName").ToString & _
   " " & CurRow("LastName").ToString)
Next
```

6. Call the SetSelected method of lstCustomer to select the first item in the list.

To populate the text boxes

1. Create the **lstCustomers_SelectedIndexChanged** event handler.
2. Declare an **Integer** variable called *RowNum*, and then store the **SelectedIndex** property of the list box in the variable.
3. Declare a **DataRow** variable called *CurRow*, and then store the row identified by *RowNum* in it by using the following line of code:

```
objCargo.CustList.Rows(rowNum)
```

4. Using the information in the following table, populate the text boxes with data from this row.

Text box	Column name
txtID	CustomerID
txtCompany	CompanyName
txtEmail	Email
txtAddress	Address

5. Save the project.

To test the application

1. On the **Build** menu, click **Build**, and resolve any build errors.
2. On the **Debug** menu, click **Start**. You should see that the list box and text boxes are populated with data.
3. Click on a different customer in the list and verify that the text boxes display the relevant data.
4. Quit the application.
5. Quit Visual Studio .NET.

Exercise 2
Updating Data

In this exercise, you will extend the class created in the previous exercise to allow editing of data. The form in the starter file has been modified to make the text boxes read-only by default and to include a button that you will code to test the functionality.

To prepare the form

1. Open Visual Studio .NET.
2. On the **File** menu, point to **Open**, and then click **Project**.
3. Set the location to *install folder*\Labs\Lab081\Ex02\Starter, click **CustomerProject.sln**, and then click **Open**.
4. In the frmCustomer.vb code window, locate the **btnEdit_Click** procedure and write code to set the **ReadOnly** property of the following textboxes to **False**.
   - txtCompany
   - txtEmail
   - txtAddress

To update the DataSet

1. Open the code window for Cargo.vb.
2. Create a new public method called **EditDetails** that takes the following parameters by value and has no return value.

Parameter name	Type
RowNum	**Integer**
ID	**Integer**
strCompany	**String**
strEmail	**String**
strAddress	**String**

3. Using the information in the following table, declare and initialize a variable.

Name	Type	Initialization Value
editRow	**DataRow**	`dstCargo.Tables(“custtable”).Rows(RowNum)`

4. Call the **BeginEdit** method of the **editRow** object.

5. Update the data in the **DataSet** with the arguments passed to the procedure. Use the information in the following table.

editRow item	Argument
CompanyName	**strCompany**
Email	**strEmail**
Address	**strAddress**

6. Call the **EndEdit** method of the **editRow** object.

To update the underlying data

1. Using the information in the following table, declare and instantiate a variable

Name	Type	Construction Value
commUpdate	**SqlCommand**	`"Update customers set CompanyName= @company, email=@email, address=@address where customerid=@id", conCargo`

2. Use the **Add** method of the command's **Parameters** collection to create parameters for this command. Use the information in the following table.

name	dbType	size	source column
@id	**SqlDbType.Int**	**4**	**CustomerID**
@company	**SqlDbType.VarChar**	**50**	**CompanyName**
@email	**SqlDbType.VarChar**	**255**	**Email**
@address	**SqlDbType.VarChar**	**255**	**Address**

3. In the **GetCustomers** procedure, locate the declaration for *adpCust*.
4. Remove the declaration for *adpCust* (leaving the instantiation code), and then declare *adpCust* as a class-level variable.
5. In the **EditDetails** procedure, set the **UpdateCommand** property of *adpCust* to the **commUpdate** command.
6. Call the **Update** method of *adpCust*, and pass it the existing **DataSet** and **DataTable**.

↙ To call the EditDetails method

1. Open the frmCustomer.vb code window, and create a handler for the **btnSave_Click** event.
2. Add code to call the **EditDetails** method of the **objCargo** object to pass the appropriate parameters. Use the information in the following table.

Parameter	Value
RowNum	**lstCustomers.SelectedIndex**
ID	**txtID.Text**
strCompany	**txtCompany.Text**
strEmail	**txtEmail.Text**
strAddress	**txtAddress.Text**

3. Set the **ReadOnly** property of the following textboxes to **True**.
   - txtCompany
   - txtEmail
   - txtAddress

↙ To test your code

1. Run the application.
2. Click **Edit Details**, replace the company name with **Microsoft**, and then click **Save**.
3. Quit the application.
4. Run the application again, and verify that the changes have been saved.
5. Quit the application.
6. Quit Visual Studio .NET.

Exercise 3
Using Data Binding in Web Forms

In this exercise, you will create a Web Form that displays invoice data in a bound data grid by using a **DataReader** object.

To create the project

1. Open Visual Studio .NET.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Project Types** box, click **Visual Basic Project**.
4. In the **Templates** box, click **ASP.NET Web Application**.
5. Change the name of the project to **Invoices**, set the location to http://localhost/2373/Labs/Lab081/Ex03, and then click **OK**.

To create the Web Form

1. Open the design window for WebForm1.aspx.
2. In the Properties window, click **DOCUMENT**, and change the **title** property to **Cargo Invoices**. This will alter the text in the Internet Explorer title bar.
3. From the Toolbox, add a **DataGrid** control to the form.
4. Save the project.

To create the connection

1. Open the code window for WebForm1.aspx.
2. Add an **Imports** statement to access the **System.Data.SqlClient** namespace.
3. Locate the **Page_Load** procedure, and declare a procedure-level variable, using the information in the following table.

Name	Type
sqlConn	**SqlConnection**

4. Instantiate the connection object, and then set the **ConnectionString** property, using the information in the following table.

Name	Value
Data Source	**localhost**
Initial Catalog	**Cargo**
Integrated Security	**True**

5. Call the **Open** method of the connection object.

✓ To bind the data

1. In the **Page_Load** procedure, declare and instantiate a **SqlCommand** object called **comInvoices**, and pass the parameters listed in the following table.

Parameter	Value
cmdText	**Select * from Invoices**
Connection	**sqlConn**

2. Declare a **SqlDataReader** object called **drInvoices**.
3. Call the **ExecuteReader** method of the command object, and store the results in **drInvoices**.
4. Set the **DataSource** property of the **DataGrid** to **drInvoices**.
5. Call the **DataBind** method of the **DataGrid**.

✓ To test your code

- Run the application and verify that the invoices are displayed in a grid on the Web Form.

Exercise 4
Creating Typed DataSets

In this exercise, you will create an XSD schema from the Invoices table of the Cargo database and use it to create a typed dataset for use in a Windows Form.

To create the project

1. Open Visual Studio .NET.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **Project Types** box, click **Visual Basic Project**.
4. In the **Templates** box, click **Windows Application**.
5. Change the name of the project to **InvoiceApp**, set the location to *install folder*\Labs\Lab081\Ex04, and click **OK**.

To create the connection

1. In Server Explorer, right-click **Data Connections**, and then click **Add Connection**.
2. In the **Data Link Properties** dialog box, enter the following information, and then click **OK**.

Setting	Value
Server name	**localhost**
Security	**Windows NT Integrated Security**
Database name	**Cargo**

To create the schema

1. In the **Project Explorer**, click **InvoiceApp**. On the **Project** menu, click **Add New Item**.
2. In the **Templates** box, click **XML Schema**, rename the item to **InvSchema.xsd**, and then click **Open**.
3. In Server Explorer, expand the connec tion that you just created, expand **Tables**, and then click **Invoices**.
4. Drag **Invoices** onto the **Schema Designer**.
5. Review the schema in both XML view and Schema view.

To create the DataSet

1. In Schema view, on the **Schema** menu, click **Generate DataSet**.
2. In Solution Explorer, click **Show All Files**, and expand **InvSchema.xsd**.
   You will see that an InvSchema.vb file has been created containing code to generate a typed DataSet.
3. Review the code in InvSchema.vb.

To create the form

1. Open the design window for Form1.vb.
2. Add three **Labels**, three **TextBoxes**, and a **Button** to the form. Use the information in the following table to set their properties.

Control	Property	Value
Label1	**Name** **Text**	**lblInvoiceID** **Invoice ID**
Label2	**Name** **Text**	**lblCustomerID** **Customer ID**
Label3	**Name** **Text**	**lblAmount** **Amount**
TextBox1	**Name** **Text**	**txtInvoiceID** **<empty>**
TextBox2	**Name** **Text**	**txtCustomerID** **<empty>**
TextBox3	**Name** **Text**	**txtAmount** **<empty>**
Button	**Name** **Text**	**btnFill** **Fill**

3. From the **Data** tab of the Toolbox, add a DataSet control to the form, using the information in the following table.

Setting	Value
Type of DataSet	**Typed**
Name	**InvoiceApp.InvSchema**

To populate the DataSet

1. Create an event handler for the **btnFill_Click** event.
2. Declare and instantiate a **SQLClient.SqlConnection** object called **sqlConn** using the information in the following table.

Name	Value
Data Source	**localhost**
Initial Catalog	**Cargo**
Integrated Security	**True**

3. Call the **Open** method of **sqlConn**.
4. Declare and instantiate a **SQLClient.SqlDataAdapter** object called **sqlAdapt** using the information in the following table.

Parameter	Value
selectCommandText	**Select * from Invoices**
selectConnection	**sqlConn**

5. Declare and instantiate a variable, using the information in the following table.

Name	Type
invTable	**InvSchema.InvoicesDataTable**

6. Call the **Fill** method of **sqlAdapt**, passing the following parameters.

Parameter	Value
DataSet	**InvSchema1**
srcTable	**invTable.TableName**

To populate the textboxes

1. In **btnFill_Click**, add code to populate the textboxes with the appropriate data. Use the information in the following table.

Control	Text property
txtInvoiceID	**InvSchema1.Invoices(0).InvoiceID**
txtCustomerID	**InvSchema1.Invoices(0).CustomerID**
txtAmount	**InvSchema1.Invoices(0).Amount**

2. Build and save the project.

To test your code

1. Run the application.
2. Click **Fill**, and verify that the textboxes are correctly populated with data.
3. Quit the application, and quit Visual Studio .NET.

Review

Topic Objective
To reinforce module objectives by reviewing key points.

Lead-in
The review questions cover some of the key concepts taught in the module.

- ADO .NET Overview
- .NET Data Providers
- The DataSet Object
- Data Designers and Data Binding
- XML Integration

1. State three benefits that ADO .NET has over earlier data access technologies.

   It is part of the .NET Framework, it is designed for disconnected data, and it is integrated with XML.

2. You have the following code in your application. What would you do to make the code more efficient? Why?

```
Dim sqlConn As New SqlClient.SqlConnection("Integrated
Security=True;Data Source=LocalHost;Initial Catalog=Pubs;")
sqlConn.Open( )
Dim sqlAdapt As New SqlClient.SqlDataAdapter("Select
au_lname from authors", sqlConn)
Dim sqlDataSet As New DataSet( )
sqlAdapt.Fill(sqlDataSet, "Authors")
Dim i As Integer
For i = 0 To sqlDataSet.Tables("Authors").Rows.Count - 1
MessageBox.Show(sqlDataSet.Tables("Authors").Rows(i).Item(0
).ToString)
Next
```

   You should replace the DataSet with a DataReader because a DataReader is more efficient for read-only, forward-only data access. This is because a DataReader only holds one record in memory at a time.

3. If you change the contents of a DataTable in a DataSet, will those changes be reflected in the underlying data source? Why, or why not?

   The changes will only be reflected if you explicitly call the Update method of the DataAdapter. If you do not do this, the changes are made locally in the DataSet, which has no permanent connection to the source.

4. You have the following code in the **Page_Load** event of a Web Form, but the **DataGrid** does not appear. What is wrong, assuming all objects are correctly declared and instantiated?

   ```
   sqlReader = sqlComm.ExecuteReader
   DataGrid1.DataSource( ) = sqlReader
   ```

   You have neglected to call the DataBind method of the DataGrid, as shown in the following line of code:

   ```
   DataGrid1.DataBind( )
   ```

5. Write the code to load an XML document called Books.xml into a DataSet.

   ```
   Dim ds As New DataSet( )
   ds.ReadXml("books.xml")
   ```