msdn® training

Module 3: Using Microsoft ADO.NET to Access Data

Contents

Microsoft

Overview

- Overview of ADO.NET
- Connecting to a Data Source
- Accessing Data with DataSets
- Using Stored Procedures
- Accessing Data with DataReaders
- Binding to XML Data

One of the reasons for the widespread use of Microsoft® Active Server Pages (ASP) is that it facilitates access to data stores. ASP.NET expands on this capability with the introduction of ADO.NET, which offers a rich suite of data handling and data binding functions for manipulating all types of data.

After completing this module, you will be able to:

- Describe the Microsoft ADO.NET object model.
- Connect to a data source by using ADO.NET.
- Retrieve data from a database by using **DataReaders** and **DataSets**.
- Display the data from a database on the client by using list-bound controls.
- Customize the look of **Repeater** controls with templates.
- Use stored procedures to return **Recordsets**.
- Read data from an Extensible Markup Language (XML) file into **DataSets**.

Note There were many changes between the Beta 1 and Beta 2 versions of ASP.NET with respect to data binding. For a detailed list of changes, see Appendix A, in Course 2063B, *Introduction to Microsoft® ASP.NET*.

◆ Overview of ADO.NET

- The ADO.NET Object Model
- Animation: Using ADO.NET to Access Data
- RecordSets vs. DataSets
- Using Namespaces

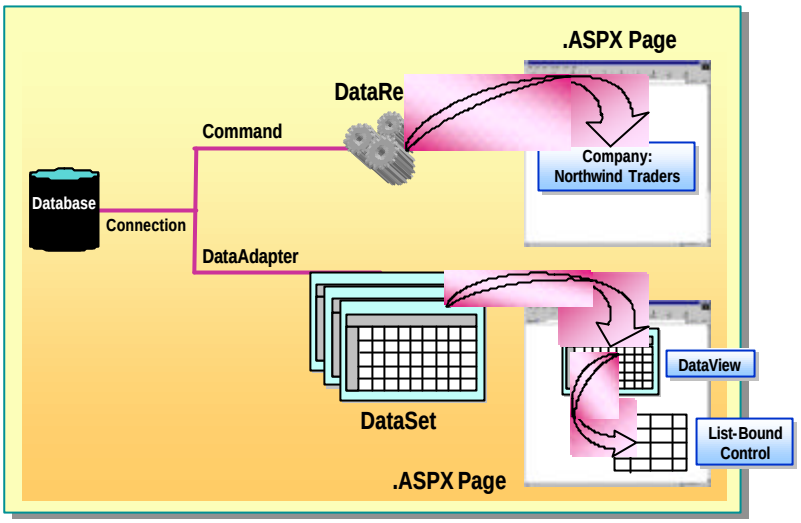
ADO.NET is not a revision of Microsoft ActiveX® Data Objects (ADO), but a new way to manipulate data that is based on disconnected data and XML. Although ADO is an important data access tool within ASP, it does not provide all of the necessary features for developing robust and scalable Web applications. In spite of ADO's rich object model and relative ease of use, it is connected by default, relies on an OLE DB provider to access data, and it is entirely Component Object Model (COM)-based.

ADO.NET has been designed to work with disconnected datasets. Disconnected datasets reduce network traffic.

ADO.NET uses XML as the universal transmission format. This guarantees interoperability as long as the receiving component runs on a platform where an XML parser is available. When the transmission occurs through XML, it is no longer necessary that the receiver be a COM object. The receiving component has no architectural restrictions whatsoever. Any software component can share ADO.NET data, as long as it uses the same XML schema for the format of the transmitted data.

In this section, you will learn about ADO.NET. You will learn about the new and modified objects in ADO.NET. You will also learn about some of the new namespaces that are included in ASP.NET.

The ADO.NET Object Model



ADO.NET evolved from the ADO data access model. By using ADO.NET, you can develop applications that are robust and scalable, and that can use XML.

ADO.NET has some of the same objects as ADO (like the **Connection** and **Command** objects), and introduces new objects, such as the **Dataset**, **DataReader**, and **DataAdapter**.

Connection Objects

Connection objects are used to talk to databases. They have properties, such as **DataSource**, **UserID**, and **Password**, which are needed to access a particular **DataSource**. Commands travel over connections, and result sets are returned in the form of streams that can be read by **DataReaders** or pushed into **DataSet** objects.

There are two kinds of connection objects in ADO.NET: **SqlConnection** and **OleDbConnection**.

Command Objects

Command objects contain the information that is submitted to a database. A command can be a stored procedure call, an update statement, or a statement that returns results. You can also use input and output parameters and return values. In ADO.NET, you can use two kinds of command objects: **SqlCommand** and **OleDbCommand**.

DataReader Objects

A **DataReader** is a read-only/forward-only view on the data. It provides a simple and lightweight way of traversing through recordsets. For example, if you wanted to simply show the results of a search list in a Web page, using a **DataReader** is an ideal way to accomplish this.

A **DataReader** is returned after executing a command. It works similarly to a recordset in ADO, allowing you to simply loop through the records.

ADO.NET includes two types of **DataReader** objects: the **SqlDataReader** for Microsoft SQL Server™ version 7.0 (or later) data, and the **OleDbDataReader** for ADO data. The **DataReader** object is database-specific. The behavior for the **SqlDataReader** may differ from the behavior for the **OleDbDataReader** and additional **DataReader** objects that are introduced in the future.

You use the **OleDbCommand** and **SqlCommand** objects and the **ExecuteReader** method to transfer data into a **DataReader**.

DataSet Objects

The **DataSet** is designed to handle the actual data from a data store. The **DataSet** provides a rich object model to work with when passing data between various components of an enterprise solution. The **DataSet** object is generic. The behavior of a **DataSet** is completely consistent regardless of the underlying database, SQL or OLE DB.

The **DataSet** object represents a cache of data, with database-like behavior. It contains tables, columns, relationships, constraints, and data. Data coming from a database, an XML file, code, or user input can be entered into **DataSet** objects and converted into files, forms, or databases. As changes are made to the **DataSet**, they are tracked in a way similar to the way changes are tracked in a word processing document.

The **DataSet** object has a collection of **DataTable** objects. A **DataTable** represents one table of in-memory data. It contains a collection of columns that represents the table's schema. A **DataTable** also contains a collection of rows, representing the data contained in the table.

You use the **OleDbDataAdapter** and **SqlDataAdapter** objects and the **Fill** method to get data into a **DataSet**.

DataView Objects

A **DataView** provides a custom view of a data table. You can think of the **DataView** as the equivalent of the ADO disconnected recordset, because it contains a single view on top of the table data. You can use a **DataView** to specify criteria for filtering and sorting a **DataSet**.

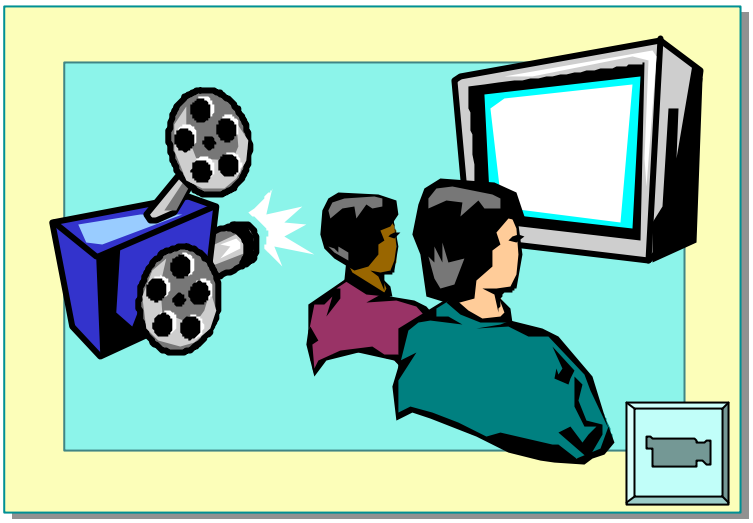
Note A **DataSet** is not a recordset. A **DataView** is more analogous to a recordset.

DataAdapter Object

While the **DataSet** object provides a tool for in-memory data storage, you need another tool to create and initialize the various tables. This tool is the **DataAdapter** object. It represents a centralized console that hides the details of working with connections and commands. The **DataAdapter** object allows for the retrieval and saving of data between a **DataSet** object and the source data store. It is responsible for pulling out data from the physical store and pushing it into data tables and relations. The **DataAdapter** object is also responsible for transmitting any update, insertion, or deletion to the physical database. You can use four command objects to make any updates: **UpdateCommand**, **InsertCommand**, **DeleteCommand**, and **SelectCommand**.

The **DataAdapter** object exists in two forms: **SqlDataAdapter** objects and **OleDbDataAdapter** objects. The data source is SQL Server for **SqlDataAdapter** objects and any other OLE DB provider for **OleDbDataAdapter** objects.

Animation: Using ADO.NET to Access Data



In this animation, you will learn how to access data by using ADO.NET and how you can display that data in an ASP.NET page. To view the animation, open the file **2063B_03A001.swf** from the folder Media.

Recordsets vs. DataSets

Feature	Recordset	DataSet
Number of tables	One table	Multiple tables
Relationships	Based on join	Includes relationships
Moving through data	Move row-by-row	Navigate via relationships
Data connections	Connected or disconnected	Disconnected
Transmitting data	COM marshalling	Transmit XML file

In ADO, the in-memory representation of database data is the recordset. In ADO.NET, it is the **DataSet**. The **DataSet** contains a collection of tables and knowledge of relationships between those tables. Each table contains a collection of columns. These objects represent the schema of the **DataSet**. Each table can then have multiple rows, representing the data held by the **DataSet**. These rows track their original state along with their current state, so that the **DataSet** tracks what kinds of changes have occurred. Additionally, the **DataSet** provides persistence and de-persistence through XML.

There are important differences between recordsets and **DataSets**, which are highlighted in the following table and detailed in the text that follows.

Feature	Recordset	DataSet
Number of tables	One table	Multiple tables
Relationships	Based on join	Includes relationships
Moving through data	Move row-by-row	Navigate via relationships
Data connections	Connected or disconnected	Disconnected
Transmitting data	COM marshalling	Transmit XML file

Number of Tables

An ADO recordset looks like a single table. If a recordset is to contain data from multiple database tables, it must use a **JOIN** query, which assembles the data from the various database tables into a single result table.

In contrast, an ADO.NET **DataSet** is a collection of one or more tables. The tables within a data set are called data tables; specifically, they are **DataTable** objects.

Relationships

Typically, a **DataSet** also contains relationships. A relationship within a **DataSet** is analogous to a foreign-key relationship in a database. In ADO.NET, a **DataRelation** represents the relationship.

Moving Through Data

In ADO.NET, the methods you use to read or modify data differ from the programming methods you use in ADO in the following ways:

- In ADO, you scan sequentially through the rows of the recordset.
- In ADO.NET, you employ a navigation paradigm, moving from a row of one data table to the corresponding row or rows of another data table by following the relationship.

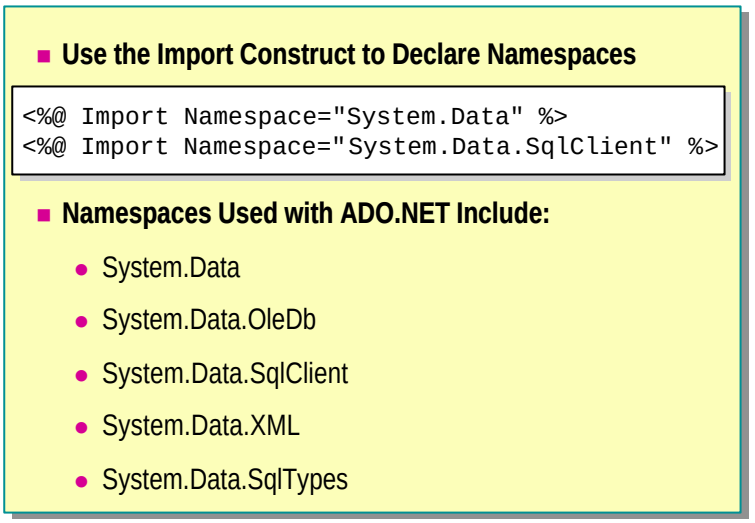
Data Connections

In ADO.NET, the **DataSet** provides disconnected access to database data. In ADO, the recordset can provide disconnected access, but is typically used to provide connected access.

Transmitting Data

To transmit an ADO disconnected recordset from one component to another, you use COM marshalling. To transmit an ADO.NET data set, you simply transmit an XML file.

Using Namespaces



The Microsoft .NET Framework is an object-oriented system. When using specific parts of the framework, you need to include references to the appropriate namespace in your ASP.NET page.

When using ADO.NET from either Microsoft Visual Basic® 7.0 or Microsoft VisualC#™, you must reference the **System.Data** namespace, plus either the **System.Data.OleDb** or **System.Data.SqlClient** namespace, depending on the data source you choose to use. **System.Data** provides the code facilities, while **System.Data.OleDb** and **System.Data.SqlClient** are the namespaces for the two managed providers.

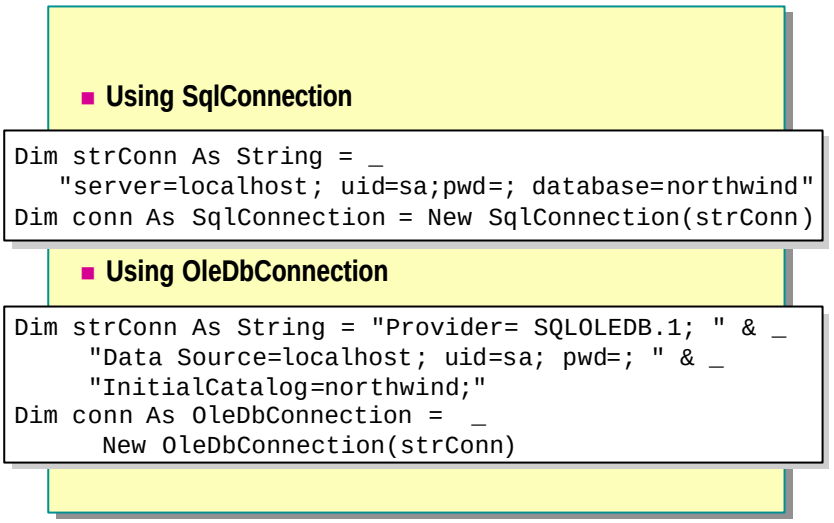
In the C# programming language, you use the keyword **Using** to import a namespace. In Visual Basic, you declare namespaces at the top of an ASP.NET page, using the **Import** construct.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
```

The following table summarizes the list of available namespaces with ADO.NET.

Namespace	Contains
System.Data	Base objects and types for ADO.NET
System.Data.OleDb	Managed OLE DB data store objects
System.Data.SqlClient	SQL Server specific implementations of ADO.NET objects
System.Data.XML	XML objects
System.Data.SqlTypes	SQL data types

Connecting to a Data Source



When you connect to a database through an ASP.NET page, there are two routes that you can take: use OLE DB or use the native SQL provider. The native SQL provider is faster, but you must be using Microsoft SQL Server as your database. If you are using Microsoft Access, Microsoft Excel, a comma-delimited file, or some other data source, you must use the OLE DB provider. You can use the OLE DB provider with a Microsoft SQL Server database; however, it is not as fast as using the native SQL provider.

The **Connection** object defines how to connect to a specific data store. The .NET framework provides two **Connection** objects: **SqlConnection** and **OleDbConnection**. The **SqlConnection** object defines how to connect to SQL Server databases and the **OleDbConnection** object allows you to establish a connection to a database through an OLE DB provider.

Using SqlConnection

The following code illustrates how to create and open a connection to a Microsoft SQL Server database by using the **SqlConnection** object.

```
Dim strConn As String = _
  "server=localhost;uid=sa;pwd=;database=northwind"
Dim conn As SqlConnection = New SqlConnection(strConn)
```

Using OleDbConnection

For the OLE DB Managed Provider, the connection string format is quite similar to the connection string format used in OLE DB.

The following code illustrates how to create and open a connection to a Microsoft SQL Server database by using **OleDbConnection**.

```
Dim strConn As String = "Provider=SQLOLEDB.1; " & _
  "Data Source=localhost; uid=sa; pwd=; " & _
  "Initial Catalog=northwind;"
Dim conn As OleDbConnection = New OleDbConnection(strConn)
```

We will be using **SqlConnection** objects for the examples in this module. Implementation is slightly different for using **OleDbConnection** objects. For more information about using **OleDbConnection** objects, search for **OleDbConnection** in the Microsoft .NET Framework SDK documentation.

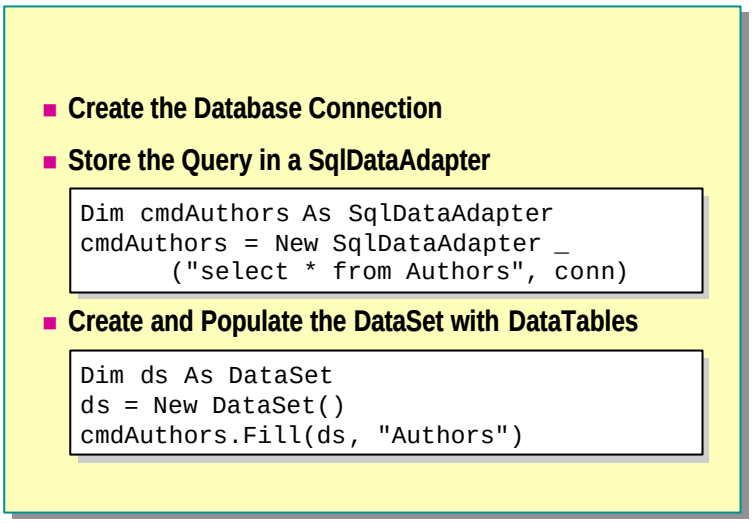
Accessing Data with DataSets

- Using DataSets to Read Data
- Storing Multiple Tables in a DataSet
- Using DataViews
- Displaying Data in the DataGrid Control
- Demonstration: Displaying Data in a DataGrid
- Using Templates
- Using the Repeater Control
- Demonstration: Displaying Data in a Repeater Control

ADO.NET provides two ways to access data, **DataSets** and **DataReaders**.

In this section, you will learn how to access data by using **DataSets**. You will also learn about **DataViews** and displaying data in list-bound controls. At the end of the section, you will learn how to customize a list-bound control by using templates.

Using DataSets to Read Data



After you establish a connection to a database, you can access its data.

ADO.NET provides multiple ways to access data.

Using DataSets

The **DataSet** object is the centerpiece of ADO.NET. It represents a complete set of data, including multiple, related tables, and constraints.

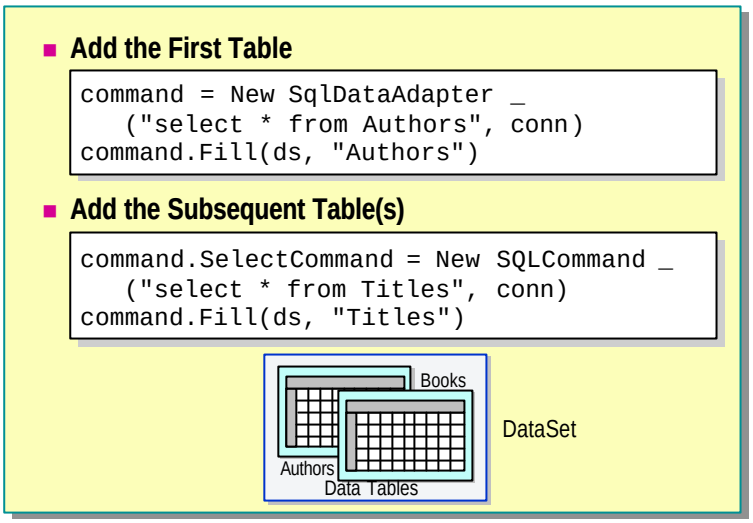
Although a **DataSet** stores data, you need **DataAdapter** objects to create and initialize the various tables. You also need the **Fill** method to populate a **DataSet** with the results from a query.

The **Fill** method takes two parameters: a **DataSet** instance and a string. The **DataSet** instance represents the **DataSet** to be filled, and the string identifies the **DataTable** that will be created inside the **DataSet**. A **DataSet** can contain many **DataTables**. You use the string supplied to the **Fill** method to reference the **DataTable** after it is created.

The following code example illustrates how to create a **SqlDataAdapter** object that contains the query statement. The **Fill** method then populates the **DataSet** with the results from the query.

```
'Create a connection
Dim conn As SqlConnection = New SqlConnection _
  ("server=localhost;uid=sa;pwd=;database=pubs")
'Create the DataAdapter
Dim cmdAuthors As SqlDataAdapter = New SqlDataAdapter _
  ("select * from Authors", conn)
'Create and populate the DataSet
Dim ds As DataSet
ds= New DataSet()
cmdAuthors.Fill(ds, "Authors")
```

Storing Multiple Tables in a DataSet



A **DataSet** can contain multiple tables. You can retrieve multiple tables from a database and store them in a **DataSet**.

Note You can store tables from different databases in the same **DataSet**.

↙ To retrieve and store multiple tables in a DataSet

1. Create and populate the first **DataSet**.

```
ds = New DataSet()
SQL = "Select * from Authors"
command = New SqlDataAdapter (SQL, conn)
command.Fill(ds, "Authors")
```

2. Reset the **SelectCommand**, **InsertCommand**, or **DeleteCommand** property of the **DataAdapter** object to a new **Command** object with a new SQL statement.

```
command.SelectCommand = New SqlCommand _
   ("select * from Titles", conn)
```

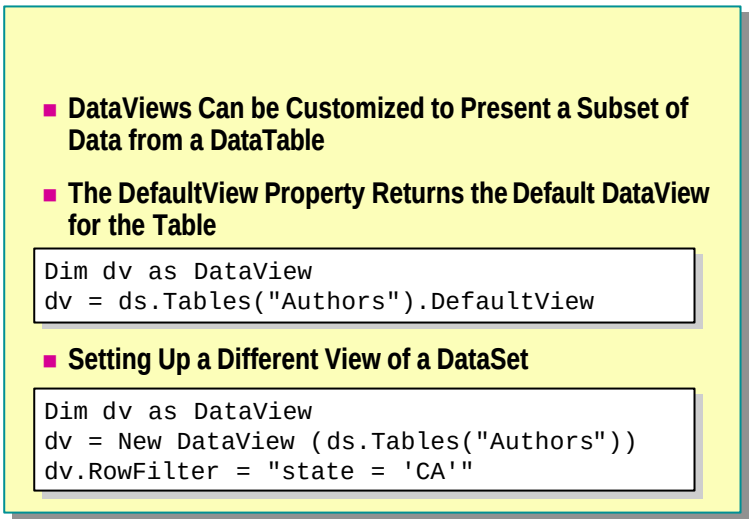
3. Call **Fill** again.

```
command.Fill(ds, "Titles")
```

The following code shows how you can add two tables from two different queries, one for authors and the other for titles, to the same **DataSet**.

```
Dim strConn As String
Dim conn As SqlConnection
Dim SQL As String
Dim command As SqlDataAdapter
Dim ds As DataSet

'create connection to database
strConn = "server=localhost;uid=sa;pwd=;database=pubs"
conn = New SqlConnection(strConn)

'fill DataSet with first set of data
SQL = "Select * from Authors"
command = New SqlDataAdapter (SQL, conn)
ds = New DataSet()
command.Fill(ds, "Authors")

'fill DataSet with second set of data
SQL = "select * from Titles"
command.SelectCommand = New SqlCommand(SQL, conn)
command.Fill(ds, "Titles")
```

Using DataViews



To display the data held in a **DataSet**, you need to use a **DataView**.

DataViews can be customized to present a subset of data from the **DataTable**. This capability allows you to have two controls bound to the same **DataTable**, but showing different versions of the data. For example, one control may be bound to a **DataView** that shows all the rows in the table, and a second may be configured to display only the rows that have been deleted from the **DataTable**.

Each **DataTable** in a **DataSet** has a **DefaultView** property, which returns the default view for the table. You can access the default **DataView** on a **DataSet** as follows:

```
Dim dv As DataView
dv = ds.Tables("Authors").DefaultView
```

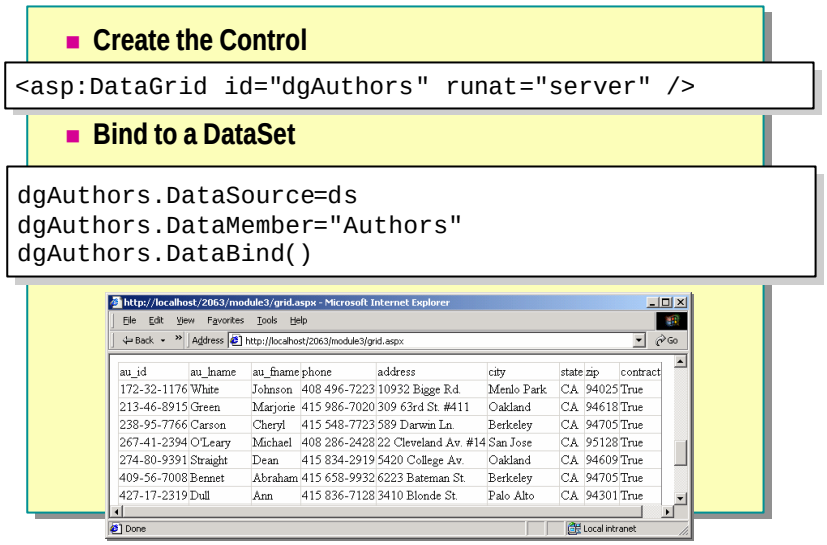
Note The **DataSet** object contains a **Tables** collection. You reference the **DataTable** you are interested in by name.

You can also create a view of a subset of the data in a **DataTable**. For example, you can set the **RowFilter** property on a **DataView** to retrieve only authors from California.

```
Dim dv as DataView
dv = New DataView (ds.Tables("Authors"))
dv.RowFilter = "state = 'CA'"
```

For more information on the properties of the **DataView** object, see the Microsoft .NET Framework SDK documentation.

Displaying Data in the DataGrid Control



Displaying data from a data source is extremely simple and flexible in ASP.NET. ASP.NET includes a set of controls that perform the function of displaying data. The developers only need to bind these controls to a data source.

To display data on the client, you can use any list-bound control, such as **DataGrid**, **DataList**, or **DataRepeater**.

Using the DataGrid Control

The **DataGrid** control is designed to produce Hypertext Markup Language (HTML) output that resembles a spreadsheet.

```
<asp:DataGrid id="dgAuthors" runat="server" />
```

To bind a **DataSet** to a **DataGrid** control, you first need to set the **DataSource** property of the **DataGrid** to a **DataSet**, **DataTable**, or **DataView**—either the **DefaultView** property of a **DataSet** or a custom **DataView** object—and then call the **DataBind** method.

If you set the **DataSource** property of the **DataGrid** directly to a **DataSet**, the **DataTable** with index 0 is used by default. To specify a different **DataTable**, set the **DataMember** property of the **DataGrid** to the name of the **DataTable**.

Example Binding to a DataSet

```
dgAuthors.DataSource = ds
dgAuthors.DataMember = "Authors"
dgAuthors.DataBind()
```

Alternatively, you can use the **Tables** collection of the **DataSet** to assign the **DataTable** directly to the **DataSource** of the **DataGrid**, as in the following example:

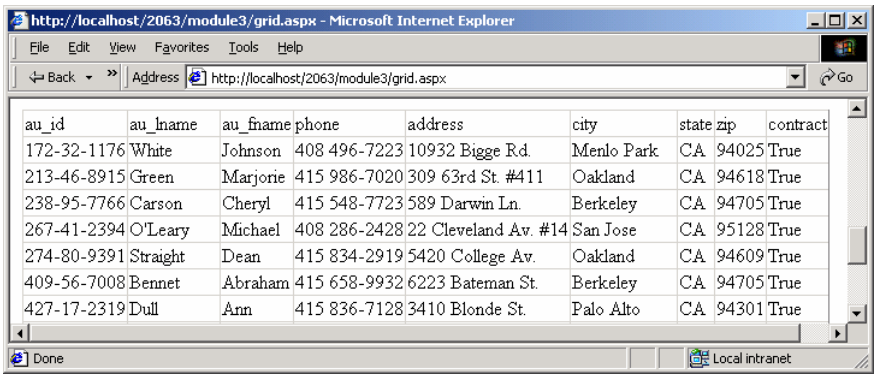
```
dgAuthors.DataSource = ds.Tables("Authors")
dgAuthors.DataBind()
```

If you want to display a different view of data in the **DataGrid** control, create a new **DataView** object from the **DataSet** and bind that to the control. For example, the following code will display only those authors living in California.

Example Using a Custom View

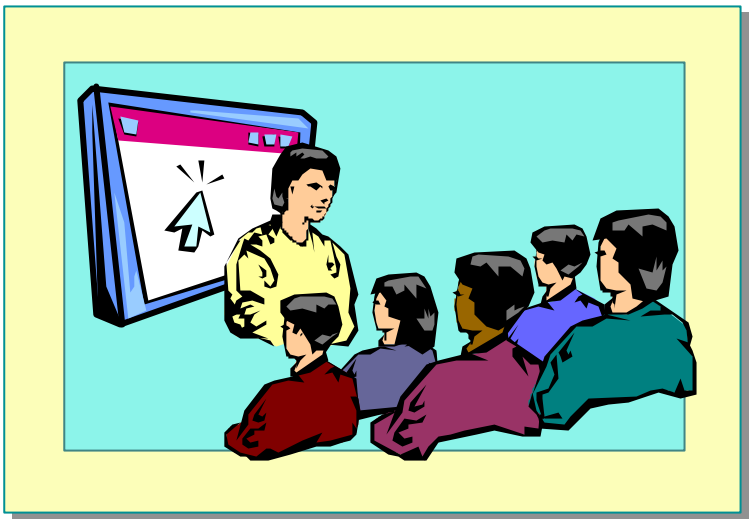
```
Dim dv as DataView
dv = New DataView (ds.Tables("Authors"))
dv.RowFilter = "state = 'CA'"
dgAuthors.DataSource = dv
dgAuthors.DataBind()
```

The following illustration shows the default format of the **DataGrid** control displaying data for authors living in California.



au_id	au_lname	au_fname	phone	address	city	state	zip	contract
172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA	94025	True
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA	94618	True
238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA	94705	True
267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose	CA	95128	True
274-80-9391	Straight	Dean	415 834-2919	5420 College Av.	Oakland	CA	94609	True
409-56-7008	Bennet	Abraham	415 658-9932	6223 Bateman St.	Berkeley	CA	94705	True
427-17-2319	Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA	94301	True

Demonstration: Displaying Data in a DataGrid



In this demonstration, you will see how to read data from a database into a **DataSet** and then display it in a **DataGrid** control.

↙ To run the demonstration

1. Edit the file <*install folder*>\DemoCode\Mod03\grid.aspx.
   a. There are two **DataGrid** controls. The first of these sets the formatting attributes to make the DataGrid control more aesthetically pleasing to the viewer.

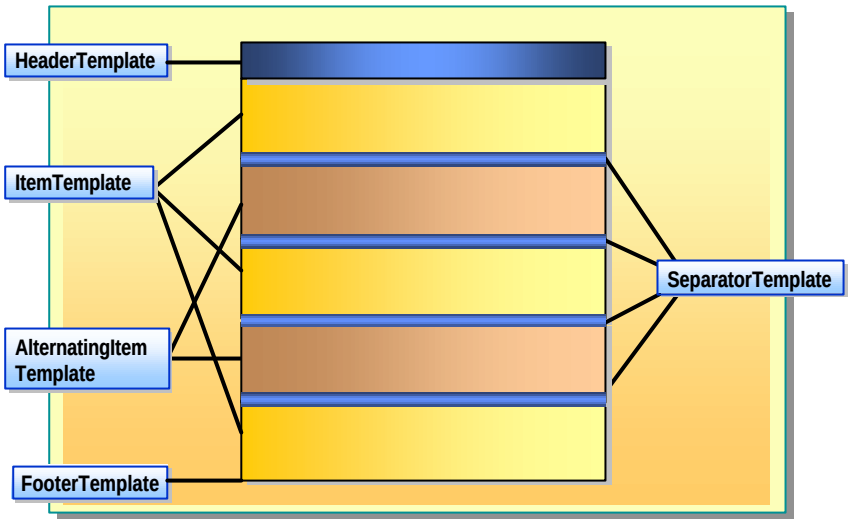
      Select one of the **DataGrid** controls and view the **Properties** window in Microsoft Visual Studio® .NET to learn about the properties of the **DataGrid** control. Look specifically at the **Style** properties.

   b. The **Page_Load** event procedure makes a connection to the database and creates a **DataSet**.

   c. The first **DataGrid** is bound to the **DefaultView** of the **DataSet**.

   d. The second **DataGrid** is bound to a new **DataView** that is a filtered version of the data.
2. View the page in Microsoft Internet Explorer.

Using Templates



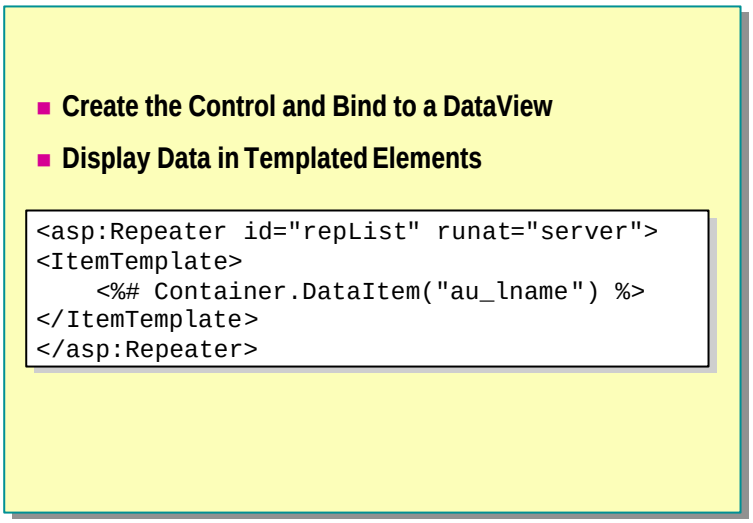
Most ASP.NET controls have a standard appearance, but the user may require something different. ASP.NET allows users to customize the appearance of some controls by using templates. When a control supports a template, you add the ASP.NET template elements. Then, within the template, you insert the elements and controls that you want to be displayed.

List-bound controls support five types of templates. The **Repeater** and **DataList** controls can use these templates directly. The **DataGrid** control can use these templates only if they are bound to a column.

Template	Use
HeaderTemplate	Elements to render once before any data-bound rows have been rendered. A typical use is to begin a container element, such as a table.
ItemTemplate	Elements that are rendered once for each row in the data source. To display data in the **ItemTemplate**, declare one or more Web controls and set their data-binding expressions to evaluate to a field in the **Repeater** control's (that is, in the container control's) **DataSource**: `<asp:Label runat="server" Text="<%#Container.DataItem.LstName %>" />`
AlternatingItemTemplate	Elements that are rendered for every other row in the **Repeater** control.
SeparatorTemplate	Elements to render between each row, typically line breaks (tags), lines (<HR> tags), and so on.
FooterTemplate	Elements to render once when all data-bound rows have been rendered. A typical use is to close an element opened in the **HeaderTemplate** item (with a tag such as </TABLE>).

For more information about using templates with the **DataGrid** control, refer to the “DataGrid Web Control” topic in the Microsoft .NET Framework SDK documentation.

Using the Repeater Control



DataList and **Repeater** controls give a developer greater flexibility over the rendering of list-like data. You might call these controls appearance-less, because they have no standard appearance; that is, they have no default layout or styles. The way data is displayed is completely determined by the HTML in the control's templates, which describe how to present data items.

An advantage of the **DataList** control is that it allows you to display items in multiple columns. As a result, elements can flow horizontally or vertically. For more information about using templates with the **DataList** control, refer to the "DataList Web Control" topic in the Microsoft .NET Framework SDK documentation.

To use the **DataList** or **Repeater** controls, you first bind the controls to a **DataTable** in a **DataSet** as shown in the following example code:

```
replist.DataSource= ds
Replist.DataMember= "Authors"
replist.DataBind()
```

Next, you bind data held in the **Repeater** control to templates that describe how to display the data. For example, to display a list of authors' last names, use the following binding syntax in an **ItemTemplate** template:

```
<asp:Repeater id="repList" runat="server">
  <ItemTemplate>
      <%# Container.DataItem("au_lname") %><BR>
  </ItemTemplate>
</asp:Repeater>
```

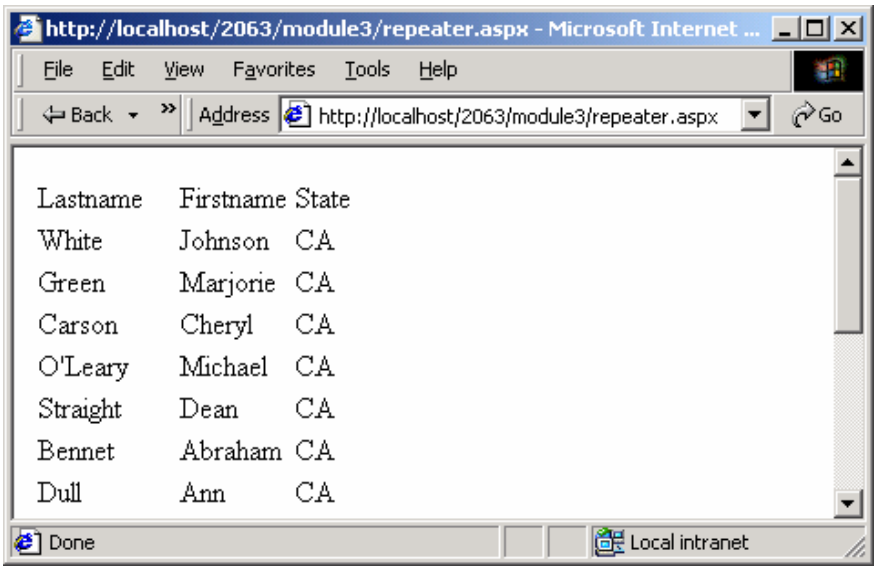
The <% %> is the syntax for inline server-side code, the # is the binding expression, the **Container** object refers to the current record of the **Repeater** control, and the **DataItem** method refers to a field in the record. You are binding to a field in the current record.

Example of a Repeater Control

The following sample code creates a very simple table to display information about the set of authors retrieved from the database.

```
<asp:Repeater id="repList" runat="server">
<HeaderTemplate>
  <table>
      <tr>
          <td>Lastname</td>
          <td>Firstname</td>
          <td>State</td>
      </tr>
</HeaderTemplate>

<ItemTemplate>
  <tr>
      <td><%# Container.DataItem("au_lname") %></td>
      <td><%# Container.DataItem("au_fname") %></td>
      <td><%# Container.DataItem("state") %></td>
  </tr>
</ItemTemplate>

<FooterTemplate>
  </table>
</FooterTemplate>
</asp:Repeater>
```

The following illustration shows the table created by the above sample code:

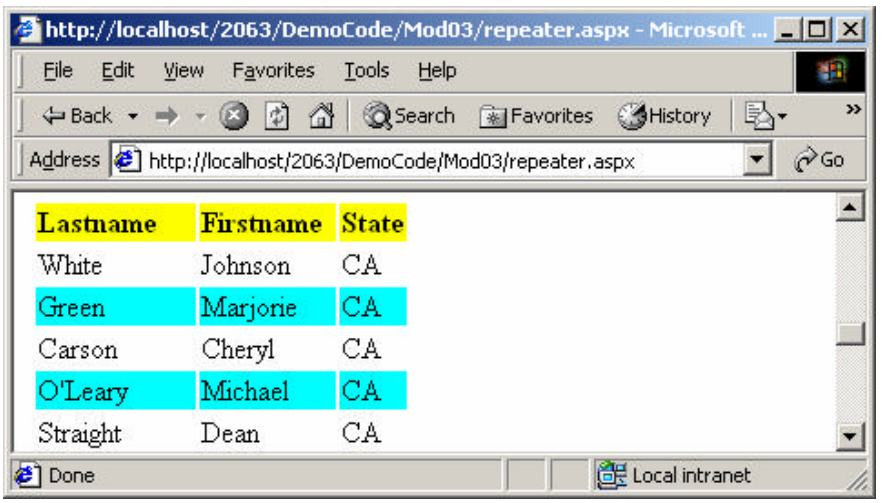


Using the AlternatingItemTemplate with the Repeater Control

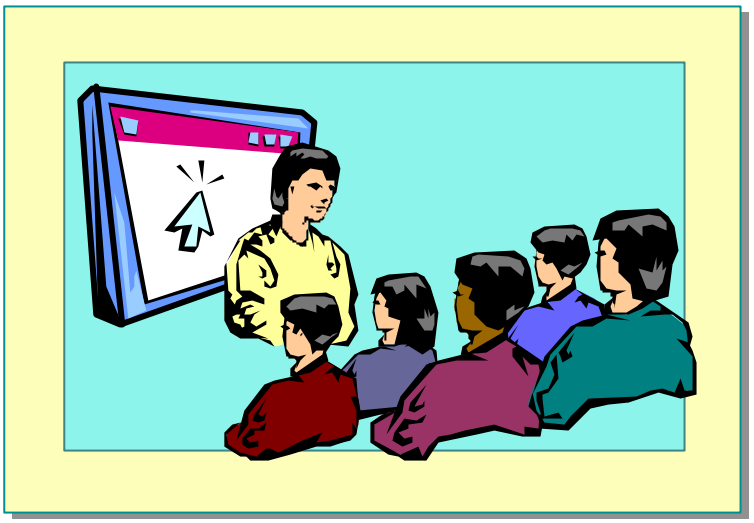
As mentioned previously, **DataList** and **Repeater** controls can use the templates directly. The following is an example of using the **AlternatingItemTemplate** with the **Repeater** control. The use of **AlternatingItemTemplate** highlights every alternate line.

```
<AlternatingItemTemplate>
  <tr>
      <td style="background-color: green">
          <%# Container.DataItem("au_lname") %></td>
      <td style="background-color: green">
          <%# Container.DataItem("au_fname") %></td>
      <td style="background-color: green">
          <%# Container.DataItem("state") %></td>
  </tr>
</AlternatingItemTemplate>
```

The following illustration shows the table with the **AlternatingItemTemplate**:



Demonstration: Displaying Data in a Repeater Control



In this demonstration, you will see how to display data in a **Repeater** control.

↙ To run the demonstration

1. Edit the file <*install folder*>\DemoCode\Mod03\repeater.aspx.
   a. The **Page_Load** event procedure makes a connection to the database, creates a **DataSet**, and binds it to the two **Repeater** controls.
   b. The first **Repeater** control, **repList**, simply outputs the data from the **DataSet** in a list.
   c. The second **Repeater** control, **repTable**, uses **HeaderTemplate**, **ItemTemplate**, **AlternatingItemTemplate**, and **FooterTemplate** templates to display data in a table.
2. View the page in Microsoft Internet Explorer.

Using Stored Procedures

- Calling Stored Procedures
- Passing Parameters
- Calling Action Stored Procedures
- Demonstration: Calling Stored Procedures

In the past, data processing has been primarily connection-based. Now, in an effort to make multi-tiered applications more efficient, data processing is turning to a message-based approach that revolves around chunks of information. In ADO.NET, this is accomplished by the **DataAdapter** object, which provides a bridge to retrieve and save data between a **DataSet** object and its source data store. The **DataAdapter** object accomplishes this by invoking the appropriate SQL commands against the data store.

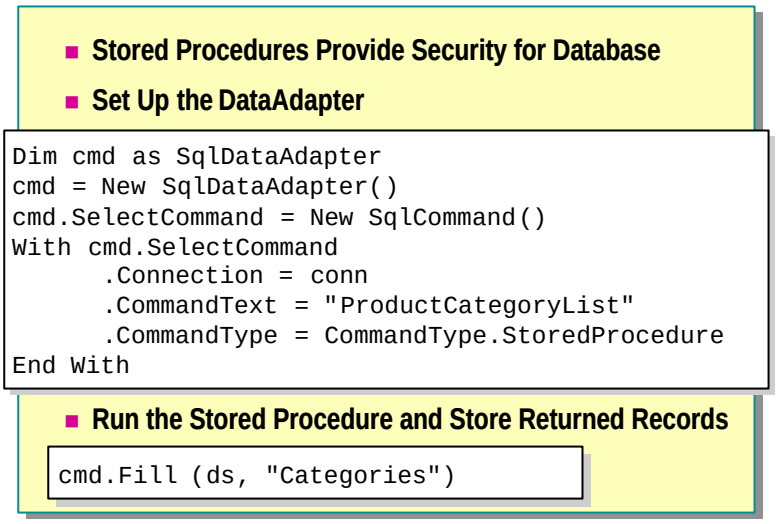
Both the **SqlDataAdapter** and the **OleDbDataAdapter** classes feature four command objects, **InsertCommand**, **UpdateCommand**, **DeleteCommand**, and **SelectCommand**. These objects provide the **create**, **update**, **delete**, and **read** functions for a specific **DataTable** in the **DataSet**.

These command objects are used when you want to perform a number of updates at the same time. Instead of having one stored procedure to do it all, you can put SQL Statements in each object and call the **Update** method.

For more information about these command objects, see the .NET Framework SDK documentation.

However, the easier and more commonly used method of updating data in a database is to use stored procedures. You can use stored procedures to read and modify data from a database. You can call stored procedures both from **DataAdapter** and **Command** objects.

Calling Stored Procedures



A stored procedure is a sequence of Transact-SQL (T-SQL) statements stored on the database server. Stored procedures provide a level of security to a database. The database designer can create stored procedures to retrieve and modify data, and not allow developers access to the actual tables of the database. In this way, the database designer can then change the structure of the database without breaking applications that use it.

Stored procedures are able to return a set of records. This will not always be the case. Stored procedures can encapsulate repetitive tasks and execute them efficiently. When you call a stored procedure that returns a set of records, use a **DataAdapter** and the **Fill** method. When you call a stored procedure that performs some function on the database but does not return a set of records, use a **Command** object and the **ExecuteNonQuery** method.

Calling a Select Stored Procedure

Using stored procedures in ADO.NET is similar to ADO. You create a command object and point it to the database connection. Next, you set the **CommandText** property to the name of the stored procedure and the **CommandType** property to **CommandType.StoredProcedure**.

The following is the **ProductCategoryList** stored procedure. It returns a list of Categories.

```
Procedure ProductCategoryList
As
  SELECT CategoryID,CategoryName
  FROM Categories
```

The following example code uses a **Connection** object and a **DataAdapter** object to call the **ProductCategoryList** stored procedure:

```
Dim cmd as SqlDataAdapter
cmd = New SqlDataAdapter()
cmd.SelectCommand = New SqlCommand()
With cmd.SelectCommand
  .Connection = conn
  .CommandText = "ProductCategoryList"
  .CommandType = CommandType.StoredProcedure
End With
```

Note You can directly set the connection and command text when creating the **SqlDataAdapter** object:

```
cmd = New SqlDataAdapter("ProductCategoryList", conn)
```

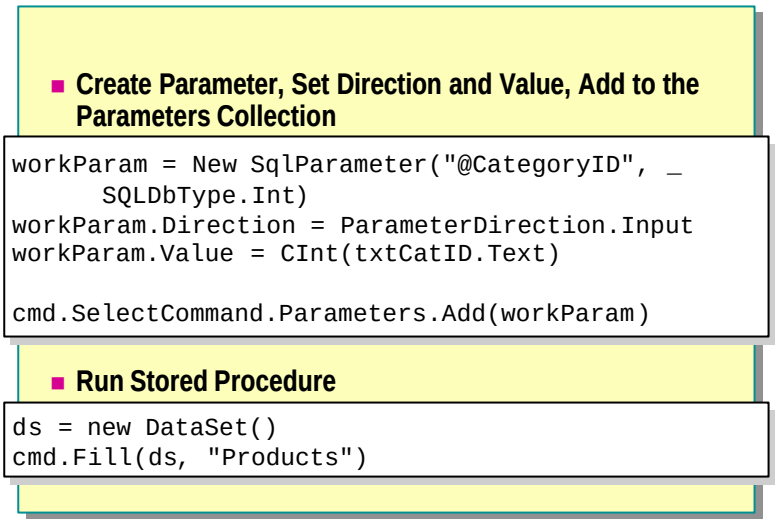
You then simply need to set the **CommandType** property before you call the **Fill** method.

To execute the stored procedure, call the **Fill** method of the **SqlDataAdapter** object. This fills a **DataTable** object with the returned records of the stored procedure.

```
cmd.Fill(ds, "Categories")
```

After you have filled a **DataTable** with the results of a **Select** stored procedure, you can bind it to a list-bound control to display the data.

Passing Parameters



When using Microsoft SQL Server, or other procedure-based databases, parameters can be used to pass and retrieve information from the database. Using parameters in ADO.NET works just as it does in ADO. You can pass the string in the command or use the parameters collection.

When using parameters, the names of the parameters added to the parameters collection of the command must match the names of the parameter markers in the stored procedure.

Parameters

Your application can pass specific data to a stored procedure by using parameters. The following table describes the types of parameters available to you.

Direction	Use
Input	Used by your application to send specific data values to a stored procedure.
Output	Used by a stored procedure to send specific values back to the calling application.
InputOutput	Used by a stored procedure to both retrieve information sent by your application and to send specific values back to the application.
ReturnValue	Used by a stored procedure to send a return value back to the calling application.

Creating a Parameter

To create a parameter for the **SqlDataAdapter** object, create a new **SqlParameter** object with the name and data type of the parameter. Next, set the **Direction** property of the new parameter to indicate how the parameter is used by the stored procedure. If the stored procedure returns a return value, create a parameter named **returnValue**. If the parameter is an input parameter, set the **Value** property to specify the data that should be sent to the server.

For example, the **ProductsByCategory** stored procedure takes one input parameter:

```
Procedure ProductsByCategory (
      @CategoryID int )
As
  SELECT ProductID, ModelName, UnitCost, ProductImage,
      Chairman
  FROM Products
  WHERE CategoryID=@CategoryID
```

To call the **ProductsByCategory** stored procedure, create an input parameter named @CategoryID and set its value to the value of a text box.

```
Dim cmd as SqlDataAdapter
cmd = New SqlDataAdapter ()
cmd.SelectCommand = New SqlCommand()
With cmd.SelectCommand
  .Connection = conn
  .CommandText = "ProductCategoryList"
  .CommandType = CommandType.StoredProcedure
End With
Dim workParam as SqlParameter
workParam = New SqlParameter("@CategoryID", SqlDbType.Int)
workParam.Direction = ParameterDirection.Input
workParam.Value = CInt(txtCatID.Text)
```

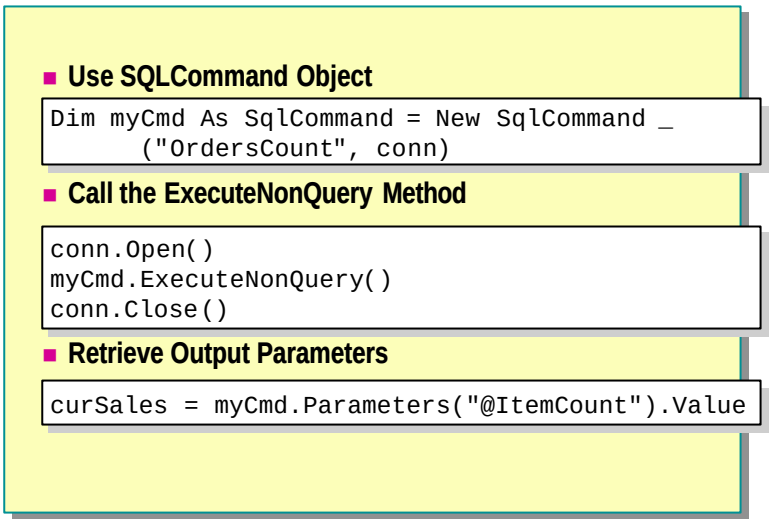
After you have created the parameter, use the **Add** method of the **Parameters** collection of the **SelectCommand** object. The **Add** method takes a **SqlParameter** as an argument. If a stored procedure has more than one parameter, it does not matter what order you add them in because you create them by name.

```
cmd.SelectCommand.Parameters.Add(workParam)
```

Use the **Fill** method to run the stored procedure and retrieve the records.

```
ds = new DataSet()
cmd.Fill(ds, "Products")
```

Calling Action Stored Procedures



When calling an update, insert, or delete stored procedure, you use the **SqlCommand** directly and call the **ExecuteNonQuery** method to run it.

The **OrdersCount** stored procedure takes a customer's ID and returns the number of outstanding orders that customer has:

```
Procedure OrdersCount (
      @CustomerID int,
      @ItemCount int OUTPUT )
As
  SELECT @ItemCount=COUNT(OrderID)
  FROM Orders
  WHERE CustomerID=@CustomerID
```

Because this stored procedure does not return a set of records, you do not need to use a **DataAdapter** object. Instead, you can use a **Command** object directly, and call the **ExecuteNonQuery** method to run the stored procedure.

To call this stored procedure, create an input parameter named @CustomerID, an output parameter named @ItemCount, add them to the **Parameters** collection of a **Command** object, and then call **ExecuteNonQuery** to run the stored procedure:

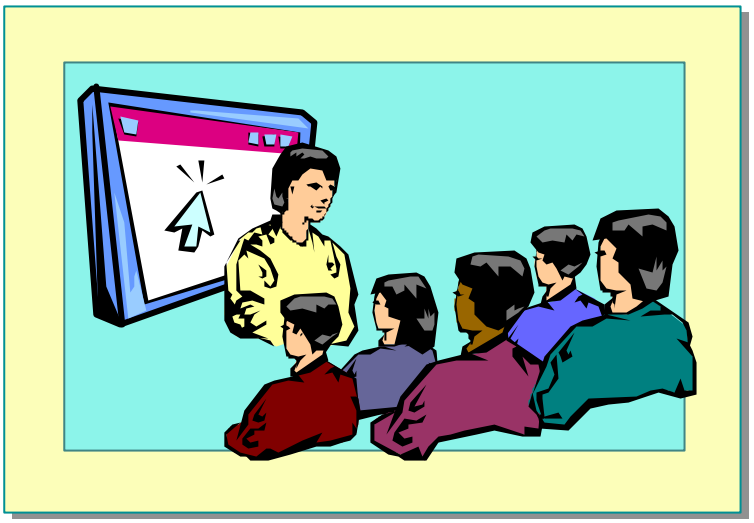
```
Dim myCmd As SqlCommand = New SqlCommand("OrdersCount", conn)
myCmd.CommandType = CommandType.StoredProcedure

'add an input parameter
workParam = New SqlParameter("@CustomerID", SqlDbType.Int)
workParam.Direction = ParameterDirection.Input
workParam.Value = CInt(txtCustID.Text)
myCmd.Parameters.Add (workParam)

'add an output parameter
workParam = New SqlParameter("@ItemCount", SqlDbType.Int)
workParam.Direction = ParameterDirection.Output
myCmd.Parameters.Add (workParam)

'execute the stored procedure
conn.Open()
myCmd.ExecuteNonQuery()
conn.Close()
```

Retrieving an Output Parameter

If you need to retrieve a value from a stored procedure that returns a value or sets an output parameter, use the **Parameters** collection to find the value of the output parameter. You can reference the value of the output parameter by name or index. The following example code retrieves the value of the @ItemCount output parameter by name:

```
curSales = myCmd.Parameters("@ItemCount").Value
```

Demonstration: Calling Stored Procedures



In this demonstration, you will see how to call stored procedures from **DataSets**, both with and without parameters.

↙ To run the demonstration

1. Edit the file *<install folder>*/DemoCode/2063/mod03/storedprocedure.aspx

   There are three sub-procedures called from the **Page_Load** event procedure that call stored procedures in different ways.

   a. The **displayCategories** procedure calls a stored procedure that has no parameters.

   b. The **displayProducts** procedure calls a stored procedure that has one input parameter.

   c. The **displayOrderCount** procedure calls a stored procedure that has input and output parameters.

2. View the page in Internet Explorer.

   The first **DataGrid** is filled from the **displayCategories** procedure.

3. Enter a category number in the text box and click **Get Products**.

   The second **DataGrid** is filled from the **displayProducts** procedure.

4. Enter a customer number in the textbox (such as 31 or 33) and click **Get Order Count**.

   The element is filled from the **displayOrderCount** procedure.

5. View the source code of the page.

   The Hidden field in the form contains the **DataSet**.

6. In Visual Studio .NET, you can view the objects in the Conf SQL Server database.

   a. Open the **Server Explorer** window.

   b. Expand **Servers**, then <*Computer Name*>, then **SQL Servers**, then <*Computer Name*>, then **Conf**.

      You can view the tables and other objects in the SQL Server database.

Lab 3: Using ADO.NET to Access Data

Objectives

After completing this lab, you will be able to:

- Connect to a database.
- Create and fill a **DataSet** with data from a database.
- Display data using the **Repeater** control.

Prerequisite

Before working on this lab, you must know how to use Web controls in an ASP.NET page.

Lab Setup

There are starter and solution files associated with this lab. The starter files are in the folder <*install folder*>\Labs\Lab03\Starter and the solution files for this lab are in the folder <*install folder*>\Labs\Lab03\Solution.

Scenario

There are five tracks in the ASP.NET conference. To view these tracks, a user goes to the ProductsList.aspx page. In Exercise 1, you will connect to the database, create a **DataSet**, fill it, and display it in a **Repeater** control. In Exercise 2, you will display the alternating line differently. In Exercise 3 (optional), you will use a **DataList** control to display a menu of items available in the ASP.NET conference.

Estimated time to complete this lab: 60 minutes

Exercise 1
Accessing a SQL Database by Using ADO.NET

In this exercise, you connect to the Conf SQL Server database and call a stored procedure. You then display the data in a **Repeater** control.

↙ To connect to the database

1. Open the file ProductsList.aspx in the folder InetPub\wwwRoot\ASPNET.
2. Locate the comment:

```
' TO DO: read data from database
```

3. Create a new connection to the Conf database.
   a. Declare a string variable named **strConn**.
   b. Set the variable to: **"server=localhost;uid=sa;pwd=1Aspnet;database=Conf"**
   c. Declare a **SqlConnection** variable named **conn**.
   d. Create a new **SqlConnection** object from the strConn variable and save it in the conn variable.

   Your code should look like the following:

```
Dim strConn As String
Dim conn As SqlConnection

strConn = _
   "server=localhost;uid=sa;pwd=1Aspnet;database=Conf"
conn = New SqlConnection(strConn)
```

↙ To call the ProductsByCategory stored procedure

1. Create a new **SqlDataAdapter** variable to call the ProductsByCategory stored procedure.

   Your code should look like the following:

```
Dim cmdProducts As SqlDataAdapter
cmdProducts = _
  New SqlDataAdapter("ProductsByCategory", conn)
cmdProducts.SelectCommand.CommandType = _
  CommandType.StoredProcedure
```

2. Create an input parameter for the stored procedure with the following properties.

Property	Value
Name	@CategoryID
Type	SqlDbType.Int
Value	categoryID

3. Add the parameter to the **Parameters** collection of the **SQLDataAdapter** variable.

   Your code should look like the following:

```
Dim paramCategoryID As SqlParameter
paramCategoryID = _
  new SqlParameter("@CategoryID", SqlDbType.Int, 4)
paramCategoryID.Value = categoryID
cmdProducts.SelectCommand.Parameters.Add(paramCategoryID)
```

↙ To create and fill the DataSet

1. Create a new **DataSet** object.
2. Fill the **DataSet** by calling the **Fill** method of the **SqlDataAdapter** variable. Save the data in a table named Products.

   Your code should look like the following:

```
Dim dsProducts As New DataSet
cmdProducts.Fill(dsProducts, "Products")
```

↙ To display the data in a Repeater control

- At the end of the **Page_Load** event procedure, display the default view of the Products table in the **DataSet** by binding it to the repList **Repeater** control.

  Your code should look like the following:

```
repList.DataSource = _
   dsProducts.Tables("Products").DefaultView
repList.DataBind()
```

↙ To save and test your work

1. Save your changes to ProductsList.aspx.
2. Using Internet Explorer, view the ProductsList.aspx page by viewing http://localhost/ASPNET/ProductsList.aspx?CategoryID=15

Note The ProductsList.aspx page can display any of the categories of information in the Conf database. CategoryID 15 will display the conference tracks.

You should see the tracks of the conference, as shown in the following illustration.

chairman	subject	price	action
Campbell, David	Creating ASP.NET Web Applications	**Price:** $350.00	**Add To Cart**
Hoffman, Barbara	Creating Web Services	**Price:** $250.00	**Add To Cart**
Wood, John	Displaying Dynamic Data with ADO.NET	**Price:** $300.00	**Add To Cart**
Chai, Sean	Introduction to ASP.NET	**Price:** $100.00	**Add To Cart**
Fortune, John	Using Web Controls	**Price:** $200.00	**Add To Cart**

3. Change the CategoryID to 14 and reload the page.

   You should see the Pre-Conference Tutorials of the conference, as shown in the following illustration.

chairman	subject	price	action
Keil, Kendall	The ABC's of ASP	**Price:** $100.00	**Add To Cart**
Sacksteder, Lane	The Basics of C#	**Price:** $125.00	**Add To Cart**
Henningsen, Jay	The Basics of Visual Basic	**Price:** $120.00	**Add To Cart**
Conroy, Stephanie	Using IIS with Windows 2000 Server	**Price:** $150.00	**Add To Cart**
Dresen, Kate	What's new in Visual Studio.NET	**Price:** $110.00	**Add To Cart**

Exercise 2
Displaying Data Using ASP.NET Controls

In this exercise, you will modify the look of the data displayed in the **Repeater** control on the ProductsList.aspx page by adding an AlternatingItemTemplate.

↙ To add an AlternatingItemTemplate

1. Open the file ProductsList.aspx in the folder InetPub\wwwRoot\ASPNET.
2. Copy the HTML tags for the **ItemTemplate** template, from <ItemTemplate> to </ItemTemplate>.
3. Paste the copied tags following the **ItemTemplate** template.
4. Change the start and end tags of the new template to **AlternatingItemTemplate**, as shown in the following sample code.

```
<AlternatingItemTemplate>
</AlternatingItemTemplate>
```

5. Set the background color of the alternating rows to **BackgroundColor6**, **BackgroundColor5**, **BackgroundColor6**, **BackgroundColor5**.

   These styles are defined in the conference.css style sheet.

   To change the appearance of the columns, set the **class** attribute of the <td> tags to the name of the style as follows:

```
<td class="BackgroundColor6">
```

   The **AlternatingItemTemplate** template should look like the following:

```
<AlternatingItemTemplate>
   <tr>
       <td class="BackgroundColor6">
           <%# Container.DataItem("Chairman") %>
       </td>

       <td class="BackgroundColor5">
           <a href='ProductDetails.aspx?productID=
           <%# Container.DataItem("ProductID") %>'>
           <%# Container.DataItem("ModelName") %><br>
           </a>
       </td>

       <td class="BackgroundColor6">
           <b>Price: </b>
           <%# System.String.Format("{0:c}", _
           Container.DataItem("UnitCost")) %>
           <br>
       </td>

       <td class="BackgroundColor5">
           <a href='AddToCart.aspx?productID=
           <%# Container.DataItem("ProductID") %>'>
           <b>Add To Cart</b>
           </a>
       </td>
   </tr>
</AlternatingItemTemplate>
```

↙ To save and test your work

1. Save your changes to ProductsList.aspx.
2. Using Internet Explorer, view the ProductsList.aspx page by viewing http://localhost/ASPNET/ProductsList.aspx?CategoryID=14

   Your page should now look like the following illustration.

chairman	subject	price	action
Keil, Kendall	The ABC's of ASP	**Price:** $100.00	**Add To Cart**
Sacksteder, Lane	The Basics of C#	**Price:** $125.00	**Add To Cart**
Henningsen, Jay	The Basics of Visual Basic	**Price:** $120.00	**Add To Cart**
Conroy, Stephanie	Using IIS with Windows 2000 Server	**Price:** $150.00	**Add To Cart**
Dresen, Kate	What's new in Visual Studio.NET	**Price:** $110.00	**Add To Cart**

3. Click on a **subject** link.
4. Click on an **Add to Cart** link.

Exercise 3 (Optional)
Creating a Menu Page

In this exercise, you will create a menu for the Web site. The menu.aspx page reads categories of data from the database and displays them in a **DataList** Web control.

↙ To create a new page

1. Create a new ASP.NET page named Menu.aspx in the ASPNET Web site.

> **Note** There is a Menu.aspx file in the <*install folder*>\Labs\Lab03\Starter folder that you can add to your project and edit.

2. Create a **DataList** control named lstMenu.

```
<asp:DataList id="lstMenu" width="145" runat="server">

</asp:DataList>
```

3. Between the <asp:DataList> and </asp:DataList> tags, create an **ItemTemplate** template that displays data in an **asp:HyperLink** control.

```
<ItemTemplate>
   <asp:HyperLink id="HyperLink1" CssClass="MenuUnselected"
       Text='<%# Container.DataItem("CategoryName") %>'
       NavigateUrl='<%# "productslist.aspx?CategoryID=" & _
              Container.DataItem("CategoryID") %>'
       runat="server" />
</ItemTemplate>
```

The **Text** attribute of the **asp:HyperLink** control displays the **CategoryName** field from the **DataSet** bound to the **DataList**. The **NavigateUrl** attribute of the **asp:HyperLink** control links to the ProductsList.aspx page, passing the ID of the selected category.

↙ To fill the DataList control in the Page_Load event procedure

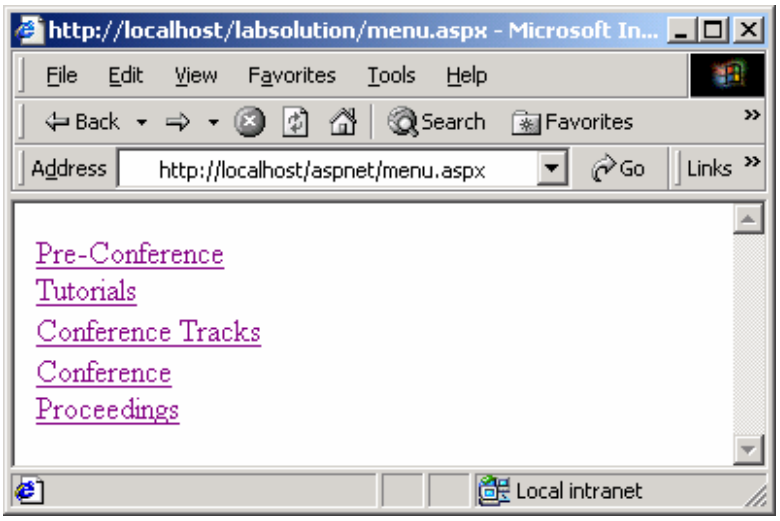
1. Add an **Import** statement to import the **System.Data** namespace.
2. Create a **Page_Load** event procedure in a <script> section.
3. Add the following code to read categories from the Conf database using a method in the Conference component, and bind them to the **lstMenu DataList** control:

```
Dim dvMenuItems As DataView
Dim products As New Conference.ProductsDB
dvMenuItems = products.GetProductCategories(). _
      Tables(0).DefaultView

'Bind the DataView of menu items to the list
lstMenu.DataSource = dvMenuItems
lstMenu.DataBind()
```

↙ To save and test your work

1. Save your changes to Menu.aspx.
2. By using Internet Explorer, go to the Menu.aspx page, http://localhost/ASPNET/menu.aspx

   The page should look like the following illustration.



3. Test each hyperlink on the Menu.aspx page.

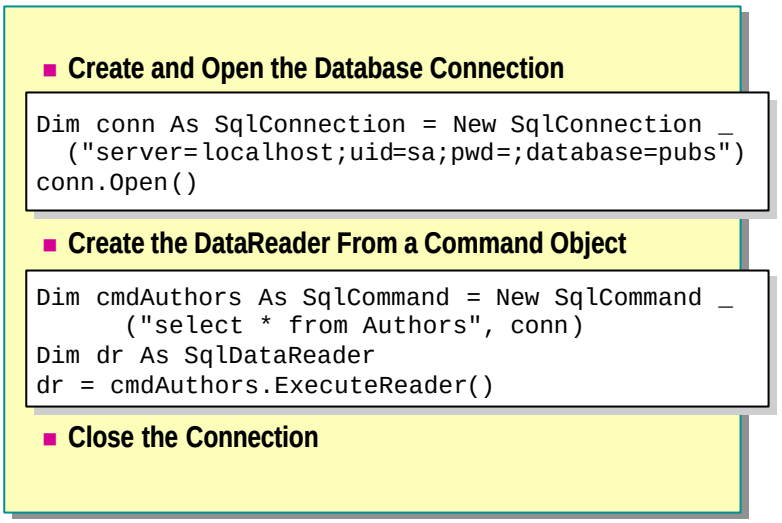
Accessing Data with DataReaders

- Creating a DataReader
- Reading Data from a DataReader
- Demonstration: Accessing Data Using DataReaders
- Using DataSets vs. DataReaders

The benefit of using a **DataSet** is that it gives you a disconnected view of the database. For long running applications, this is often the best approach. For Web applications, developers usually perform short and simple operations with each request, such as displaying data. For such operations, developers do not need to maintain a **DataSet** object. In such cases, you can use a **DataReader**.

In this section, you will learn how to read data from a data source by using **DataReaders**.

Creating a DataReader



When a large amount of data is being retrieved, holding memory open becomes an issue. For example, reading 10,000 rows out of a database causes a **DataTable** to allocate and maintain memory for those 10,000 rows for the lifetime of the table. If 1,000 users do this against the same machine at the same time, memory usage becomes critical.

To address such situations, the **DataReader** is designed to produce a read-only, forward-only stream returned from the database. Only one record at a time is ever in memory. There are two **DataReader** objects, the **SqlDataReader** and the **OleDbDataReader**.

A **DataReader** keeps the connection open until the **DataReader** is closed.

To use a **SqlDataReader**, declare a **SqlCommand** instead of a **SqlDataAdapter**. The **SqlCommand** exposes an **ExecuteReader** method that takes a **SqlDataReader** as a parameter. Note that you must explicitly open and close the **SqlConnection** when you use a **SQLCommand**.

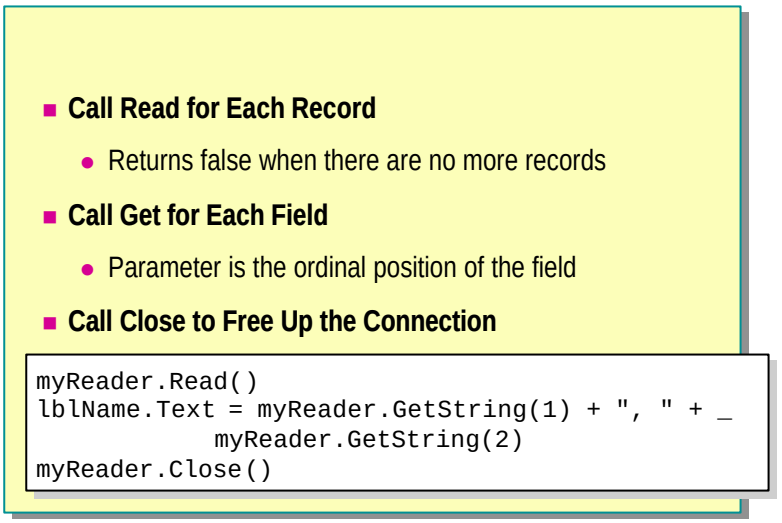
```
Dim conn As SqlConnection = New SqlConnection _
  ("server=localhost;uid=sa;pwd=;database=pubs")
conn.Open()

Dim cmdAuthors As SqlCommand = New SqlCommand _
  ("select * from Authors", conn)
Dim dr As SqlDataReader
dr = cmdAuthors.ExecuteReader()
...
dr.close()
conn.Close()
```

Error Handling with a DataReader

When using connections with the **DataReader** object, you should always use a **Finally** blocking technique to ensure that, if anything fails, connections will be closed.

```
Try
  conn.Open()
  dr = cmdAuthors.ExecuteReader()
  'use the returned data in the datareader
Catch e As Exception
  Console.WriteLine(e.ToString)
Finally
  dr.Close()
  conn.Close()
End Try
```

Reading Data from a DataReader



After you have called the **ExecuteReader** method of the **Command** object, you access a record in the **DataReader** by calling the **Read** method. The default positioning in the **DataReader** is before the first record, therefore you must call **Read** before accessing any data. When no more records are available, the **Read** method returns a null value.

Reading Fields from the Current Record

To get the data from fields in the current record, call an appropriate **Get** method, for example, **GetDateTime**, **GetDouble**, **GetInt32**, or **GetString**. The parameter of the **Get** method is the ordinal value of the field that you want to read.

For example, the following sample code reads the first name and last name fields, both string values, from the first record of the **DataReader**, by using the **GetString()** method:

```
myReader.Read()
lblName.Text = myReader.GetString(1) + ", " + _
      myReader.GetString(2)
```

You can also reference the fields of data in the current record of the data reader by name, and then call an appropriate conversion function, as shown in the following example code:

```
myReader("au_fname").ToString()
```

Looping Through All Records

To loop through all the records in a **DataReader**, you can use a **While** loop, as shown in the following sample code:

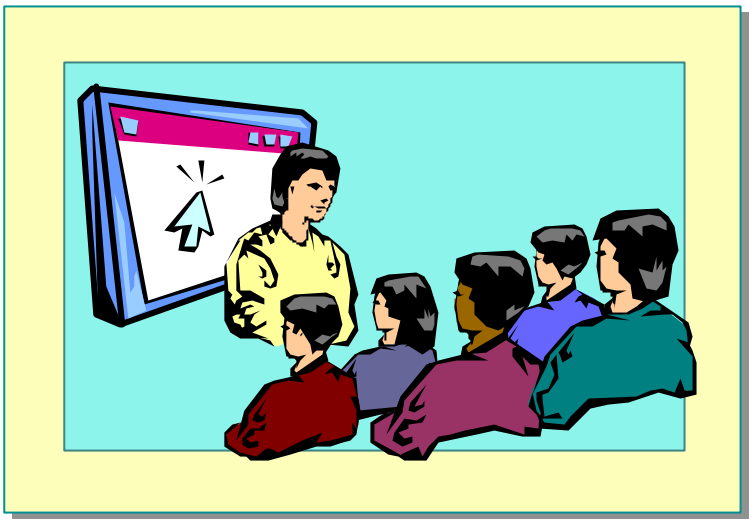
```
While myReader.Read()
  'do something with the data
End While
```

Closing the DataReader

While the **DataReader** is in use, the associated connection is busy serving the **DataReader**. Therefore, you must call **Close** to close the **DataReader** when you are finished using it.

```
myReader.Close()
```

Demonstration: Accessing Data Using DataReaders



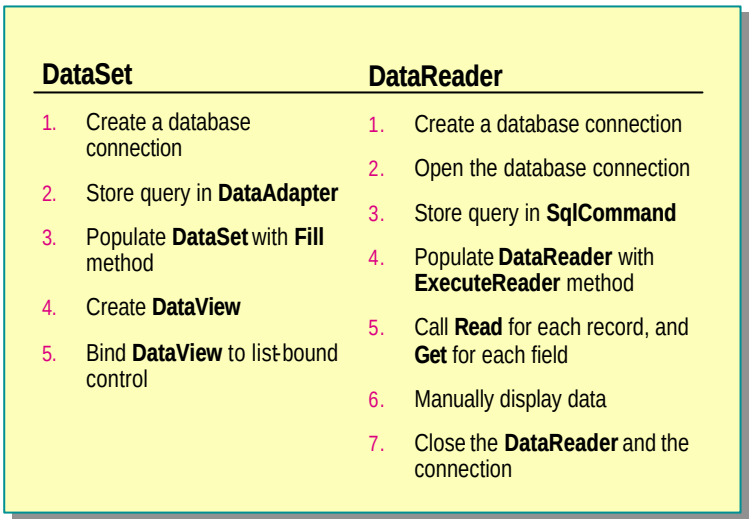
In this demonstration, you will see how to read data from a database by using a **SQLDataReader**.

↙ To run the demonstration

1. Edit the file <*install folder*>\DemoCode\Mod03\Datareader.aspx
   a. The **Page_Load** event procedure sets up the **DataReader** and outputs the first record.
   b. The **Page_Unload** event procedure closes the connection.
2. View the page in Internet Explorer.
3. Edit the file <*install folder*>\Democode\Mod03\datareadersp.aspx

   This page fills a **DataReader** from a stored procedure and then displays the results.
4. View the page in Internet Explorer.

Using DataSets vs. DataReaders



The general procedure for accessing databases from ASP.NET is different depending on whether you will be using a **DataSet** or a **DataReader**:

Using DataSets	Using DataReaders
1. Connect to the database by using **SqlConnection** or **OleDbConnection**.	1. Connect to the database by using **SqlConnection** or **OleDbConnection**.
2. Store the database query in **SqlDataAdapter** or **OleDbDataAdapter** objects.	2. Open the connection with the **Open** method.
3. Populate a **DataSet** from the **DataAdapter** by using **Fill**.	3. Store database query in **SqlCommand** or **OleDbCommand** objects.
4. Set up a new **DataView** for the desired table.	4. Populate a **DataReader** from the **Command** by using **ExecuteReader** method.
5. Bind a server control, such as the **DataGrid**, to the **DataView**.	5. Call **Read** and **Get** methods of the **DataReader** to read data.
	6. Close the **DataReader**.
	7. Close the connection.

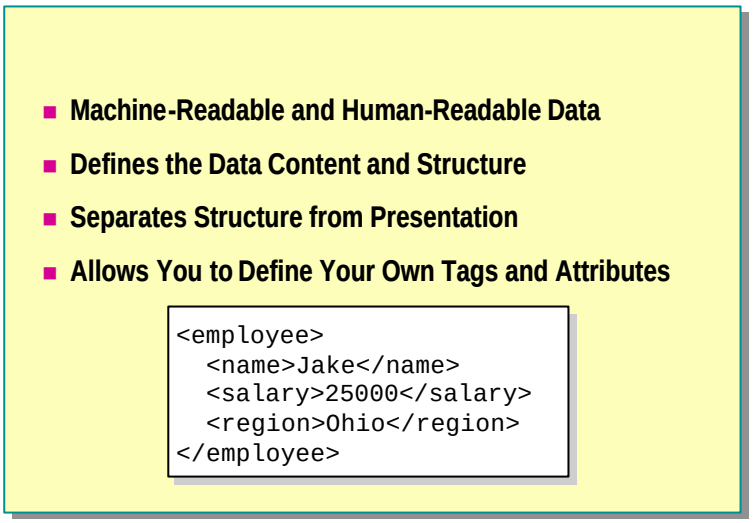
Binding to XML Data

- Overview of XML
- Reading XML Data into a DataSet
- Demonstration: Reading XML Data into a DataSet

HTML is widely used for presenting information on the Web. HTML works well as a presentation language, but it is not suitable for representing data. For example, you can easily format data in an HTML table, but you cannot describe the individual components of the information. To share information between applications, you must have a language that can describe data in a standardized way so that any application, present or future, can understand and use this data correctly. XML is one such standardized language. XML not only helps you structure your data but acts as a common language between different business applications.

In this section, you will get an overview of XML. You will also learn how to read and display XML data by using ADO.NET.

Overview of XML



Businesses today face many problems when it comes to organizing data. They need to meet the following requirements:

- Data needs to be readable by both computers and users.
- Both the content and the structure of the data need to be defined.
- The structure of the data needs to be separate from the presentation of the data.
- The structure needs to be open and extensible.

XML fulfills all these requirements.

XML defines the structure of data in an open and self-describing manner. This allows data to be easily transferred over a network and consistently processed by the receiver. XML describes how data is structured, not how it should be displayed or used. XML documents contain tags that assign meaning to the content of the document. These tags allow programmers to find the data they need in the XML document.

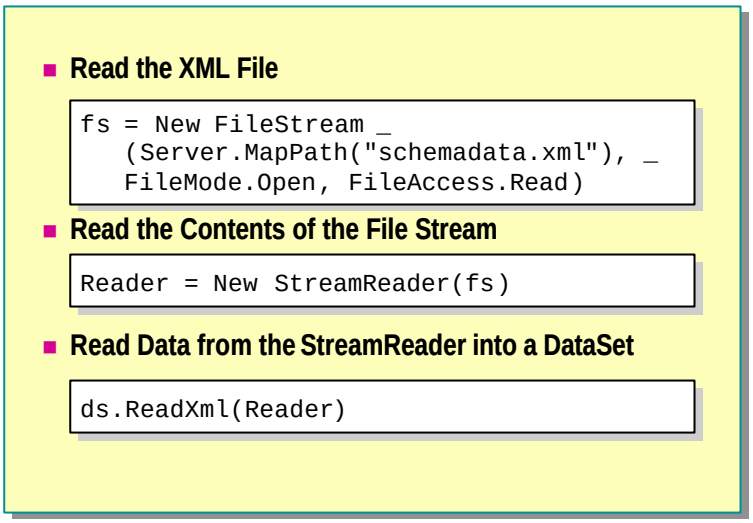
For example, the following XML sample contains information about an employee but does not specify how to display this information:

```
<employee>
  <name>Jake</name>
  <salary>25000</salary>
  <region>Ohio</region>
</employee>
```

XML is considered a markup language because it allows you to define data structure by using markup tags. You can define your own tags that describe the data in whatever way you find useful.

XML data is held in a simple, open format that is easily parsed by other applications. The fact that XML documents contain text rather than binary data is another key advantage. Applications can parse an XML document, looking for specific tags of interest to those applications. Unknown tags and their associated data can be freely ignored.

Reading XML Data into a DataSet



DataSets in ADO.NET are designed to extract data in a way that is independent of its data source. Therefore, reading data from an XML source is similar to reading data from a database. To read XML data, you need to import the **System.IO** namespace in your ASP.NET page.

Note You can't read XML data into a **DataReader**. You can read it only into a **DataSet**.

For XML data, the **DataSet** supports a **ReadXml** method that takes a **FileStream** as its parameter. The **DataSet** expects data to be in the following format:

```
<DocumentElement>
  <TableName>
      <ColumnName1>column value</ColumnName1>
      <ColumnName2>column value</ColumnName2>
      <ColumnName3>column value</ColumnName3>
  </TableName>
  <TableName>
      <ColumnName1>column value</ColumnName1>
      <ColumnName2>column value</ColumnName2>
      <ColumnName3>column value</ColumnName3>
  </TableName>
</DocumentElement>
```

Each **TableName** section corresponds to a single row in the table.

The following example shows how to read the schema and data from an XML file by using the **ReadXml** method, the **FileStream** object, and the **StreamReader** object. Note that after the data is read into the **DataSet**, it is indistinguishable from SQL data—the **DataGrid** binds to it in the same way.

↙ To read XML data and display it in a DataGrid

1. First, open the XML file:

```
Dim fs As FileStream
fs = New FileStream _
   (Server.MapPath("schemadata.xml"), _
   FileMode.Open,FileAccess.Read)
```

2. Next, attach a **StreamReader** to the **FileStream**.

```
Dim Reader As StreamReader
Reader = New StreamReader(fs)
```

3. Finally, read the XML data from the **StreamReader** into the **DataSet**.

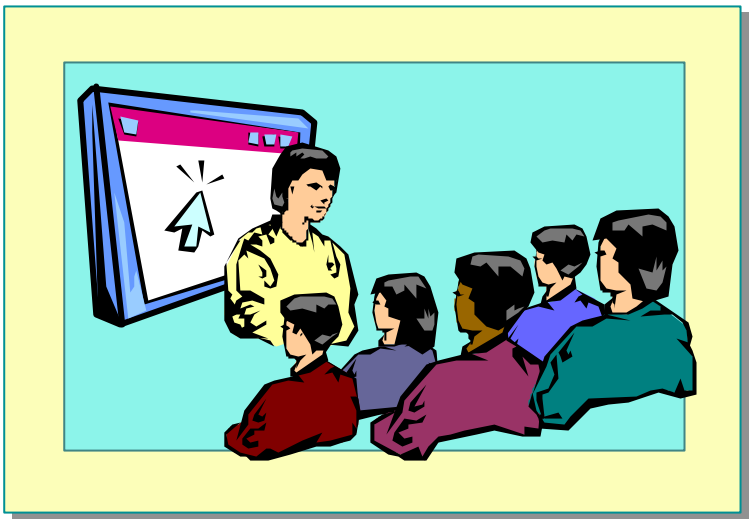
```
Dim ds As New DataSet
ds.ReadXml(Reader)
```

After the data has been read into a **DataSet**, the repeated elements in the XML become the columns in the **DataSet** and can be bound to any control to be displayed on the client.

The full sample code is as follows:

```
Dim ds As New DataSet
Dim fs As FileStream
Dim Reader As StreamReader

'Read the XML data into a DataSet
fs = New FileStream _
  (Server.MapPath("schemadata.xml"), _
  FileMode.Open,FileAccess.Read)
Reader = New StreamReader(fs)
ds.ReadXml(Reader)
fs.Close()

'Bind the DataSet to a DataGrid
Dim Source As DataView
Source = new DataView(ds.Tables(0))
MyDataGrid.DataSource = Source
MyDataGrid.DataBind()
```

Demonstration: Reading XML Data into a DataSet



In this demonstration, you will see how to read information from an XML file and display it in a **DataGrid** control.

↙ To run the demonstration

1. Open the file <*install folder*>\DemoCode\Mod03\Books1.xml.

   This is the data that will be displayed from the ASPX page.

2. Edit the file <*install folder*>\DemoCode\Mod03\xml.aspx.

   a. There is a **DataGrid** control that uses templates to change the look of the table.

   b. The **Page_Load** event procedure reads the data from the Books1.xml file into a **DataSet** and binds it to the **DataGrid**.

3. View the page in Internet Explorer.

4. Edit the file <*install folder*>\Democode\Mod03\aspXMLControl.aspx.

   This page uses the XML Web control to read data from the Books1.xml file, and apply the Books1.xsl style sheet.

5. View the page in Internet Explorer.

Review

- Overview of ADO.NET
- Connecting to a Data Source
- Accessing Data with DataSets
- Using Stored Procedures
- Accessing Data with DataReaders
- Binding to XML Data

---

1. What are some of the new objects in the ADO.NET object model?

2. What is the difference between a **DataSet** and a **DataView**?

3. What is the difference between a **DataSet** and a **DataReader**?

4. What is the purpose of the **DataAdapter** object?

5. Which method is used to populate a **DataSet** with results of a query?