

## UNIT-V

### SYLLABUS

Data Binding in ASP.Net: Data source Controls – Configuring data source control caching – storing connection information-Using Bound list controls with Data Source Controls – Other Databound Controls-Data Management with ADO.Net

### **Data Binding in ASP.NET 2.0**

#### **Data Source Controls:**

ASP.NET provides another set of controls for data bindings. Using the “*DataSource*” property of a data-bind control you can bind the data object to the control directly. Another way is to use *DataSource* controls. Any control that implements the “*IDataSource*” interface becomes a data source control and you can bind the data-source control to a data-bind control using the “*DataSourceID*” property.

*DataSource* controls are very easy to use and you can create a page with data without writing any code. But what you can does not always means what you should do. In professional applications, the *DataSource* controls are not generally used. Data access code should be separated from UI and be optimized for your business needs.

*DataSource* controls can still be very useful when you create a small personal application or the test pages to look at the result without relying on data access code.

#### **1. DataSource Controls**

*DataSource* controls are server controls that allow you to access the data source directly from a web page. They are configurable with parameters.

A parameter values can be obtained from:

- a cookie
- the session
- a form field
- a query string

The *DataSource* controls can be bound to data-bind controls very easily. You just need to add a the *DataSource* control to a page, set the properties of it, and set the “*DataSourceControlID*” property of a data-bind control. That’s all. The magic will happen.

The *DataSource* controls perform data binding after the “*PreRender*” event, which is the last page creation stage before HTML output is rendered. When you perform “*Update*” or “*Insert*” operations using the controls, the events are handled just before the “*PreRender*” event. This life cycle is very important. When you perform “*update*” or “*insert*”, you cannot catch the change in the “*Page\_Load*”.

ASP.NET provides a couple of built-in controls for you.

- *SqlDataSource*: ADO.NET provider (SQL Server, Oracle, OLE DB, ODBC, and a new access file (.accdb) )
- *AccessDataSource*: Access database file (.mdb)
- *XmlDataSource*: XML files
- *SiteMapDataSource*: .sitemap
- *LinqDataSource*: LINQ to SQL
- *ObjectDataSource*: a custom data access class
- *EntityDataSource*: Entity Framework

Some controls can cache data: *ObjectDataSource*, *SqlDataSource*, and *AccessDataSource* controls.

## 2. *IDataSource* interface

Any controls that implements “*System.Web.UI.IDataSource*” interface can be a *DataSource* control.

The interface has 2 methods to be implemented:

- *DataSourceView* *GetView*(string viewName)
- *ICollection* *GetViewNames*()

## 3. *SqlDataSource*

The “*SqlDataSource*” control represents the data access to an ADO.NET provider.

- *Attributes* -

- *ID*: used when referring to the data source during data binding
- *ProviderName*: *System.Data.SqlClient*, *System.Data.OracleClient*, *System.Data.OleDb*, or *System.Data.Odbc*

- *ConnectionString*: connection string or a page script to read the connection string ( <%\$ ConnectionStrings:NorthwindConnectionString %> )
- *SelectCommand*, *UpdateCommand*, *InsertCommand*, *DeleteCommand*:
- *SelectCommandType*, *UpdateCommandType*, *InsertCommandType*, *DeleteCommandType*: *SqlDataSourceCommandType.Text* / *SqlDataSourceCommandType.StoredProcedure*
- *DataSourceMode*: *SqlDataSourceMode.DataSet* / *SqlDataSourceMode.DataReader*

- Filtering -

- The “*DataSourceMode*” attribute must be set to “*DataSet*”
- Use “*FilterExpression*” and “*FilterParameters*” attributes

- Caching -

- The “*DataSourceMode*” attribute must be set to “*DataSet*”
- Use “*EnableCaching*”, “*CacheExpirationPolicy*”, and “*CacheDuration*” attributes
- Even without writing a single line of code, you can create a web page with 2 data-bind controls (parent-child).

- Exception Handlings -

- Handle *Selected*, *Updated*, *Inserted* and *Deleted* events
- Check the *Exception* property of the *EventArgs* object
- 

#### 4. AccessDataSource

The “*AccessDataSource*” control works with Access database (.mdb). The “*DataFile*” attribute is used instead of the “*ConnectionString*” attribute.

You need to use *SqlDataSource* to connect to the newer version of Access files (.accdb).

Provider=Microsoft.ACE.OLEDB.12.0;

Data Source=C:\myFolder\myAccess2007file.accdb;

Persist Security Info=False;

#### 5. XmlDataSource

The “*XmlDataSource*” control is for XML files, which are typically stored in *App\_Data* folder.

- *Attributes* -

- *DataFile*: the path of the xml file
- *TransformFile*: the path of the xsl (XML Stylesheet Language) file
- *XPath*: filtering using XPath expressions

```
1 <asp:XmlDataSource ID = "XmlDataSource1"
  runat="server"
2   DataFile="~/App_Data/product.xml"
3   TransformFile="~/App_Data/producttransform.xsl"
4   XPath="/Products/Product[Category='Food']" >
5 </asp:XmlDataSource>
```

## 6. SiteMapDataSource

The “*SiteMapDataSource*” control automatically picks up the “*Web.sitemap*” file in the root directory of the web application.

- *Attributes* -

- *StartingNodeUrl*:
- *ShowStartingNode*:
- *StartingFromCurrentNode*

## 7. LinqDataSource

The “*LinqDataSource*” control is used with “*LINQ to SQL*”.

- *Attributes* -

- *ContextTypeName*: the name of the class that represents the database context
- *EnableDelete, EnableInsert, EnableUpdate*: Boolean
- *TableName*

- *Sorting* -

- *OrderBy*

```
1 <asp:LinqDataSource ID = "LinqDataSource1" runat="server"
2   ContextTypeName="NorthwindDataContext"
3   EnableDelete="True" EnableInsert="True" EnableUpdate="True"
4   TableName="Suppliers" OrderBy="CompanyName" Where="Country==@Country" >
5   <WhereParameters>
6     <asp:QueryStringParameter DefaultValue="us" Name="Country"
7       QueryStringField="country" Type="String" />
8   </WhereParameters>
9 </ asp: LinqDataSource >
```

### 8. EntityDataSource

The “*EntityDataSource*” control is used with “*Entity Framework*” which becomes replacing LINQ to SQL.

### 9. ObjectDataSource

The “*ObjectDataSource*” control is used with a custom data access object.

- *Attributes* -

- *TypeName*: the type name of the object for DB access
- *DataObjectName*: the type name of the object for Data Model
- *SelectMethod*: works with *IEnumerable*, *IListSource*, *IDataSource*, *IHierarchicalDataSource* (return classes as a *DataTable*, *DataSet*, or some form of a collection)
- *InsertMethod*, *UpdateMethod*, and *DeleteMethod*:

- *Filtering* -

- Use *FilterExpression* and *FilterParameters* attributes

- *Sorting*-

- Use *SortParameterName* attribute

- Paging -

- *EnablePaging:*
- *StartRowIndexParameterName:*
- *MaximumRowsParameterName:*
- *SelectCountMethod:*

- Caching -

- *EnableCaching:*
- *CacheDuration:* in seconds

When you create a “Data Object” class, you need to decorate a class properly.

- Set the “*DataObject*” attribute to the class
- Add “*DataObjectMethod*” attribute to methods – Pass a “*DataObjectMethodType*” enum (*Select, Insert, Update, and Delete*)

```
[System.ComponentModel.DataObject()]
1
2 public class MyData
3 {
4     [System.ComponentModel.DataObjectMethod(Syste.ComponentModel.DataObjectMethod.S
    elect)]
5     public static DataTable GetAllData()
6     {
7     }
8 }
```

## Data Binding in ASP.NET

The following controls are list controls which support data binding:

- `asp:RadioButtonList`
- `asp:CheckBoxList`
- `asp:DropDownList`
- `asp:Listbox`

The selectable items in each of the above controls are usually defined by one or more `asp:ListItem` controls, like this:

```
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="countrylist" runat="server">
<asp:ListItem value="N" text="Norway" />
<asp:ListItem value="S" text="Sweden" />
<asp:ListItem value="F" text="France" />
<asp:ListItem value="I" text="Italy" />
</asp:RadioButtonList>
</form>
</body>
</html>
```

- However, with data binding we may use a separate source, like a database, an XML file, or a script to fill the list with selectable items.
- By using an imported source, the data is separated from the HTML, and any changes to the items are made in the separate data source.

### Create an ArrayList

- The `ArrayList` object is a collection of items containing a single data value.
- Items are added to the `ArrayList` with the `Add()` method.

The following code creates a new `ArrayList` object named `mycountries` and four items are added:

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
    dim mycountries=New ArrayList
```

```
mycountries.Add("Norway")
mycountries.Add("Sweden")
mycountries.Add("France")
mycountries.Add("Italy")
end if
end sub
</script>
```

- By default, an ArrayList object contains 16 entries.

An ArrayList can be sized to its final size with the TrimToSize() method:

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New ArrayList
mycountries.Add("Norway")
mycountries.Add("Sweden")
mycountries.Add("France")
mycountries.Add("Italy")
mycountries.TrimToSize()
end if
end sub
</script>
```

An ArrayList can also be sorted alphabetically or numerically with the Sort() method:

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New ArrayList
mycountries.Add("Norway")
mycountries.Add("Sweden")
mycountries.Add("France")
mycountries.Add("Italy")
```



```
mycountries.TrimToSize()  
mycountries.Sort()  
end if  
end sub  
</script>
```

To sort in reverse order, apply the Reverse() method after the Sort() method:

```
<script runat="server">  
Sub Page_Load  
if Not Page.IsPostBack then  
    dim mycountries=New ArrayList  
    mycountries.Add("Norway")  
    mycountries.Add("Sweden")  
    mycountries.Add("France")  
    mycountries.Add("Italy")  
    mycountries.TrimToSize()  
    mycountries.Sort()  
    mycountries.Reverse()  
end if  
end sub  
</script>
```

### **Data Binding to an ArrayList**

An ArrayList object may automatically generate the text and values to the following controls:

- asp:RadioButtonList
- asp:CheckBoxList
- asp:DropDownList
- asp:Listbox

To bind data to a RadioButtonList control, first create a RadioButtonList control (without any asp:ListItem elements) in an .aspx page:

```
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server" />
</form>
</body>
</html>
```

Then add the script that builds the list and binds the values in the list to the RadioButtonList control:

### **Example**

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
    dim mycountries=New ArrayList
    mycountries.Add("Norway")
    mycountries.Add("Sweden")
    mycountries.Add("France")
    mycountries.Add("Italy")
    mycountries.TrimToSize()
    mycountries.Sort()
    rb.DataSource=mycountries
    rb.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server" />
</form>
</body>
</html>
```

- The DataSource property of the RadioButtonList control is set to the ArrayList and it defines the data source of the RadioButtonList control.
- The DataBind() method of the RadioButtonList control binds the data source with the RadioButtonList control.

**Note:** The data values are used as both the Text and Value properties for the control. To add Values that are different from the Text, use either the Hashtable object or the SortedList object.

### Create a Hashtable

- The Hashtable object contains items in key/value pairs.
- The keys are used as indexes, and very quick searches can be made for values by searching through their keys.
- Items are added to the Hashtable with the Add() method.

The following code creates a Hashtable named mycountries and four elements are added:

```
<script runat="server">  
Sub Page_Load  
if Not Page.IsPostBack then  
    dim mycountries=New Hashtable  
    mycountries.Add("N","Norway")  
    mycountries.Add("S","Sweden")  
    mycountries.Add("F","France")  
    mycountries.Add("I","Italy")  
end if  
end sub  
</script>
```

### Data Binding

A Hashtable object may automatically generate the text and values to the following controls:

- asp:RadioButtonList
- asp:CheckBoxList
- asp:DropDownList
- asp:Listbox

To bind data to a RadioButtonList control, first create a RadioButtonList control (without any asp:ListItem elements) in an .aspx page:

```
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>
</body>
</html>
```

Then add the script that builds the list:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
    dim mycountries=New Hashtable
    mycountries.Add("N","Norway")
    mycountries.Add("S","Sweden")
    mycountries.Add("F","France")
    mycountries.Add("I","Italy")
    rb.DataSource=mycountries
    rb.DataValueField="Key"
    rb.DataTextField="Value"
    rb.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>
</body>
</html>
```

Then we add a sub routine to be executed when the user clicks on an item in the RadioButtonList control.

When a radio button is clicked, a text will appear in a label:

### **Example**

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
    dim mycountries=New Hashtable
    mycountries.Add("N","Norway")
    mycountries.Add("S","Sweden")
    mycountries.Add("F","France")
    mycountries.Add("I","Italy")
    rb.DataSource=mycountries
    rb.DataValueField="Key"
    rb.DataTextField="Value"
    rb.DataBind()
end if
end sub
sub displayMessage(s as Object,e As EventArgs)
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text
end sub
</script>

<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
<p><asp:label id="lbl1" runat="server" /></p>
</form>
</body>
</html>
```

## **The SortedList Object**

- The SortedList object contains items in key/value pairs. A SortedList object automatically sort the items in alphabetic or numeric order.
- Items are added to the SortedList with the Add() method. A SortedList can be sized to its final size with the TrimToSize() method.

The following code creates a SortedList named mycountries and four elements are added:

```
<script runat="server">  
sub Page_Load  
if Not Page.IsPostBack then  
    dim mycountries=New SortedList  
    mycountries.Add("N","Norway")  
    mycountries.Add("S","Sweden")  
    mycountries.Add("F","France")  
    mycountries.Add("I","Italy")  
end if  
end sub  
</script>
```

## **Data Binding**

A SortedList object may automatically generate the text and values to the following controls:

- asp:RadioButtonList
- asp:CheckBoxList
- asp:DropDownList
- asp:Listbox

To bind data to a RadioButtonList control, first create a RadioButtonList control (without any asp:ListItem elements) in an .aspx page:

```
<html>  
<body>  
<form runat="server">  
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
```

```
</form>
</body>
</html>
```

Then add the script that builds the list:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
    dim mycountries=New SortedList
    mycountries.Add("N","Norway")
    mycountries.Add("S","Sweden")
    mycountries.Add("F","France")
    mycountries.Add("I","Italy")
    rb.DataSource=mycountries
    rb.DataValueField="Key"
    rb.DataTextField="Value"
    rb.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>
</body>
</html>
```

- Then we add a sub routine to be executed when the user clicks on an item in the RadioButtonList control.

When a radio button is clicked, a text will appear in a label:

### **Example**

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
    dim mycountries=New SortedList
    mycountries.Add("N","Norway")
    mycountries.Add("S","Sweden")
    mycountries.Add("F","France")
    mycountries.Add("I","Italy")
    rb.DataSource=mycountries
    rb.DataValueField="Key"
    rb.DataTextField="Value"
    rb.DataBind()
end if
end sub
sub displayMessage(s as Object,e As EventArgs)
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
<p><asp:label id="lbl1" runat="server" /></p>
</form>
</body>
</html>
```

### **Bind a DataSet to a List Control**

First, import the "System.Data" namespace. We need this namespace to work with DataSet objects.

Include the following directive at the top of an .aspx page:



---

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

Next, create a DataSet for the XML file and load the XML file into the DataSet when the page is first loaded:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
    dim mycountries=New DataSet
    mycountries.ReadXml(MapPath("countries.xml"))
end if
end sub
```

To bind the DataSet to a RadioButtonList control, first create a RadioButtonList control (without any asp:ListItem elements) in an .aspx page:

```
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>
</body>
</html>
```

Then add the script that builds the XML DataSet:

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

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
    dim mycountries=New DataSet
    mycountries.ReadXml(MapPath("countries.xml"))
    rb.DataSource=mycountries
    rb.DataValueField="value"
    rb.DataTextField="text"
```

```
rb.DataBind()  
end if  
end sub  
</script>  
<html>  
<body>  
<form runat="server">  
<asp:RadioButtonList id="rb" runat="server"  
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />  
</form>  
</body>  
</html>
```

Then we add a sub routine to be executed when the user clicks on an item in the RadioButtonList control. When a radio button is clicked, a text will appear in a label:

### **Example**

```
<%@ Import Namespace="System.Data" %>  
  
<script runat="server">  
sub Page_Load  
if Not Page.IsPostBack then  
dim mycountries=New DataSet  
mycountries.ReadXml(MapPath("countries.xml"))  
rb.DataSource=mycountries  
rb.DataValueField="value"  
rb.DataTextField="text"  
rb.DataBind()  
end if  
end sub  
sub displayMessage(s as Object,e As EventArgs)  
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text  
end sub  
</script>
```

```
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
<p><asp:label id="lbl1" runat="server" /></p>
</form>
</body>
</html>
```

### **Bind a DataSet to a Repeater Control**

- The Repeater control is used to display a repeated list of items that are bound to the control.
- The Repeater control may be bound to a database table, an XML file, or another list of items.
- Here we will show how to bind an XML file to a Repeater control.

We will use the following XML file in our examples ("cdcatalog.xml"):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
<cd>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>
</cd>
<cd>
<title>Hide your heart</title>
<artist>Bonnie Tyler</artist>
<country>UK</country>
<company>CBS Records</company>
<price>9.90</price>
```

```
<year>1988</year>
</cd>
<cd>
  <title>Greatest Hits</title>
  <artist>Dolly Parton</artist>
  <country>USA</country>
  <company>RCA</company>
  <price>9.90</price>
  <year>1982</year>
</cd>
<cd>
  <title>Still got the blues</title>
  <artist>Gary Moore</artist>
  <country>UK</country>
  <company>Virgin records</company>
  <price>10.20</price>
  <year>1990</year>
</cd>
<cd>
  <title>Eros</title>
  <artist>Eros Ramazzotti</artist>
  <country>EU</country>
  <company>BMG</company>
  <price>9.90</price>
  <year>1997</year>
</cd>
</catalog>
```

Take a look at the XML file: [cdcatalog.xml](#)

First, import the "System.Data" namespace. We need this namespace to work with DataSet objects. Include the following directive at the top of an .aspx page:

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

Next, create a DataSet for the XML file and load the XML file into the DataSet when the page is first loaded:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
    dim mycdcatalog=New DataSet
    mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
end if
end sub
```

Then we create a Repeater control in an .aspx page.

The contents of the <HeaderTemplate> element are rendered first and only once within the output, then the contents of the <ItemTemplate> element are repeated for each "record" in the DataSet, and last, the contents of the <FooterTemplate> element are rendered once within the output:

```
<html>
<body>
<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">
<HeaderTemplate>
...
</HeaderTemplate>

<ItemTemplate>
...
</ItemTemplate>

<FooterTemplate>
...
</FooterTemplate>
</asp:Repeater>
</form>
```

</body>  
</html>

- Then we add the script that creates the DataSet and binds the mycdcatalog DataSet to the Repeater control.

We also fill the Repeater control with HTML tags and bind the data items to the cells in the<ItemTemplate> section with the <%#Container.DataItem("fieldname")%> method:

### Example

```
<%@ Import Namespace="System.Data" %>
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
    dim mycdcatalog=New DataSet
    mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
    cdcatalog.DataSource=mycdcatalog
    cdcatalog.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">
<HeaderTemplate>
<table border="1" width="100%">
<tr>
<th>Title</th>
<th>Artist</th>
<th>Country</th>
<th>Company</th>
<th>Price</th>
<th>Year</th>
```

```
</tr>
</HeaderTemplate>
<ItemTemplate>
<tr>
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</ItemTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:Repeater>
</form>
</body>
</html>
```

Using the <AlternatingItemTemplate>

You can add an <AlternatingItemTemplate> element after the <ItemTemplate> element to describe the appearance of alternating rows of output.

In the following example each other row in the table will be displayed in a light grey color:

### Example

```
<%@ Import Namespace="System.Data" %>
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
```

```
cdcatalog.DataSource=mycdcatalog
cdcatalog.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">
<HeaderTemplate>
<table border="1" width="100%">
<tr>
<th>Title</th>
<th>Artist</th>
<th>Country</th>
<th>Company</th>
<th>Price</th>
<th>Year</th>
</tr>
</HeaderTemplate>
<ItemTemplate>
<tr>
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</ItemTemplate>
<AlternatingItemTemplate>
<tr bgcolor="#e8e8e8">
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
```



```
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</AlternatingItemTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:Repeater>
</form>
</body>
</html>
```

### Using the <SeparatorTemplate>

The <SeparatorTemplate> element can be used to describe a separator between each record.

The following example inserts a horizontal line between each table row:

### Example

```
<%@ Import Namespace="System.Data" %>
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
cdcatalog.DataSource=mycdcatalog
cdcatalog.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
```

```
<asp:Repeater id="cdcatalog" runat="server">
  <HeaderTemplate>
    <table border="0" width="100%">
      <tr>
        <th>Title</th>
        <th>Artist</th>
        <th>Country</th>
        <th>Company</th>
        <th>Price</th>
        <th>Year</th>
      </tr>
    </HeaderTemplate>
    <ItemTemplate>
      <tr>
        <td><%#Container.DataItem("title")%></td>
        <td><%#Container.DataItem("artist")%></td>
        <td><%#Container.DataItem("country")%></td>
        <td><%#Container.DataItem("company")%></td>
        <td><%#Container.DataItem("price")%></td>
        <td><%#Container.DataItem("year")%></td>
      </tr>
    </ItemTemplate>
    <SeparatorTemplate>
      <tr>
        <td colspan="6"><hr /></td>
      </tr>
    </SeparatorTemplate>
    <FooterTemplate>
  </table>
</FooterTemplate>
</asp:Repeater>
</form>
</body>
</html>
```

## Caching Data Using Data Source Controls

- Data source controls provide data services to data-bound controls such as the [GridView](#), [FormView](#), and [DetailsView](#) controls.
- These services include caching data to help improve the performance of applications in which the data does not change frequently.
- To cache data using the [SqlDataSource](#) or [AccessDataSource](#) controls, you must set the [DataSourceMode](#) property of those controls to DataSet.
- The [ObjectDataSource](#) control can cache objects returned by the underlying business object.
- However, you should not cache objects that hold resources or that maintain state that cannot be shared among multiple requests, such as an open DataReader object.
- Caching is not enabled by default for data source controls, but you can enable it by setting the control's EnableCaching property to true.
- Cached data is refreshed based on the number of seconds you specify using the CacheDuration property.
- You can further refine the behavior of a data source control's caching behavior by setting its CacheExpirationPolicy property.
- Setting the property's value to Absolute forces the cache to be refreshed when the CacheDuration value is exceeded.
- Setting the CacheExpirationPolicy property to Sliding refreshes the cache only if the CacheDuration value has been exceeded since the last time the cached item was accessed.

```
<% @ Page language="VB" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
```

```
  <head runat="server">
```

```
    <title>ASP.NET Example</title>
```

```
  </head>
```

```
  <body>
```

```
    <form id="form1" runat="server">
```

```
      <asp:SqlDataSource
```

```
        id="SqlDataSource1"
```

```
        runat="server"
```

```
DataSourceMode="DataSet"
ConnectionString="<%%$ ConnectionStrings:MyNorthwind%>"
EnableCaching="True"
CacheDuration="20"
SelectCommand="SELECT      EmployeeID,FirstName,LastName,Title      FROM
Employees">
</asp:SqlDataSource>

<asp:GridView
id="GridView1"
runat="server"
AutoGenerateColumns="False"
DataSourceID="SqlDataSource1">
<columns>
<asp:BoundField HeaderText="First Name" DataField="FirstName" />
<asp:BoundField HeaderText="Last Name" DataField="LastName" />
<asp:BoundField HeaderText="Title" DataField="Title" />
</columns>
</asp:GridView>

</form>
</body>
</html>
```

### Storing Connection Information

- One of the best aspects of the .NET Framework is ADO.NET and data providers that negate the need for ODBC connections.
- Data providers offer a liaison between your code and the database.
- They also provide better performance, as well as easier setup (no one misses the process of installing and configuring an ODBC driver).
- .NET database providers make it necessary to specify database connection information, but it's the developer's discretion to decide where to store this information.
- Connecting to a database requires knowledge of the database server, as well as the username and password to use when accessing it.
- Also, you can specify parameters such as which database to use based upon your database platform.

The following VB.NET code demonstrates a sample connection string for working with SQL Server:

```
Dim sConn As String
Dim conn As SqlConnection
sConn = "server=(local);InitialCatalog=Northwind;UID=tester;PWD=123456"
conn = New SqlConnection(sConn)
conn.Open()
```

Here's the C# equivalent:

```
string sConn;
SqlConnection conn;
sConn = "server=(local);InitialCatalog=Northwind;UID=tester;PWD=123456";
conn = new SqlConnection(sConn);
conn.Open();
```

- This code works but there are potential problems.
- First, including database connection information in your code is a security risk; a malicious user could use this information to gain access.
- Also, problems may arise if the connection string changes—you must recompile the code.
- For these reasons, developers often choose to store database connection information outside of the code.
- There are many options for accomplishing this task, but two popular choices are using an XML configuration file or utilizing the Windows registry.

### Choosing XML

- Storing database connection properties in an XML file allows you to change the settings without recompiling the code.
- It's easy to edit the XML file with any standard text editor. A good example of this approach is utilizing the XML-based web.config file in an ASP.NET application.
- This file allows you to add custom elements (which are accessible from the code) to the application.
- In addition, this file isn't viewable from a Web browser so its contents remain hidden.

You can add new items using the add element with key and value attributes. The following sample web.config shows a possible way to add the connection:

```
<configuration>
<add key="dbconnection" value=" server=(local);
Initial Catalog=Northwind;UID=tester;PWD=123456"/>
</configuration>
```

- With this data stored in the web.config file, you can use it in your code with the ConfigurationSettings class.

The following VB.NET reads the connection string from the application's web.config file.

It uses the AppSettings property and passes the key to be read to it.

```
Dim sConn As String
Dim conn As SqlConnection
sConn = ConfigurationSettings.AppSettings("dbconnection")
conn = New SqlConnection(sConn)
conn.Open()
```

Here is the code in C#:

```
String sConn;
SqlConnection conn;
sConn = ConfigurationSettings.AppSettings("dbconnection");
conn = new SqlConnection(sConn);
conn.Open();
```

- You may also use this straightforward approach with a simple XML file (for a non-ASP.NET application).
- The problem is that it's still open to security threats. If a malicious user gains access to the server, the web.config file is a simple text file that they can easily read to gain database access.
- To provide tighter access control, many developers utilize the Windows registry to store connection information.
- With the information in the registry, it's easy to read it in the code.

### Choosing the registry

- The Microsoft.Win32 namespace provides everything necessary to interact with a Windows registry.
- You can use its CreateSubKey method to create registry entries.
- The following C# snippet will do the trick.
- It uses the Registry class to access a machine's Windows registry, and it uses the RegistryKey class to work with individual elements within the registry.

```
Using Microsoft.Win32;
class Class1 {
static void Main(string[] args) {
Registry.LocalMachine.CreateSubKey("SOFTWARE\\MyApp\\DBConn");
RegistryKey regKey;
regKey = Registry.OpenSubKey("SOFTWARE\\MyApplication\\DBConn");
if (regKey != null) {
regKey.SetValue("Server", "(local)");
regKey.SetValue("Database", "Northwind");
regKey.SetValue("UserID", "tester");
regKey.SetValue("Password", "123456");
}}}

```

This is the VB.NET equivalent:

```
Imports Microsoft.Win32;
Module Module1
Sub Main()
Registry.LocalMachine.CreateSubKey("SOFTWARE\\MyApp\\DBConn")
Dim regKey As RegistryKey
If Not (regKey Is Nothing) Then
regKey.SetValue("Server", "(local)")
regKey.SetValue("Database", "Northwind")
regKey.SetValue("UserID", "tester")
regKey.SetValue("Password", "123456")
End If

```

End Sub

End Module

- This approach is inherently more secure since a person will need full server access to work with its registry.
- With the necessary values stored in the registry, you may utilize them to connect to the database.
- You can perform this with the GetValue method of the RegistryKey class.

```
using Microsoft.Win32;
using System.Data.SqlClient;
class Class1 {
    static void Main(string[] args) {
        RegistryKey regKey;
        string server, db, uid, pwd
        string sConn
        SqlConnection conn;
        regKey = Registry.LocalMachine.OpenSubKey("SOFTWARE\\MyApp\\DBConn");
        if (regKey != null) {
            server = regKey.GetValue("Server").ToString();
            db = regKey.GetValue("Database").ToString();
            uid = regKey.GetValue("UserID").ToString();
            pwd = regKey.GetValue("Password").ToString();
            sConn = "server=" + server + ";Initial Catalog=" + db + ";UID=" + uid + ";PWD=" + pwd;
            conn = new SqlConnection(sConn);
            conn.Open();
        }
    }
}
```

This is the VB.NET equivalent:

```
Imports Microsoft.Win32;
Imports System.Data.SqlClient
Module Module1
    Sub Main()
        Dim regKey As RegistryKey
        Dim server, db, uid, pwd As String
```



```
Dim sConn As String
regKey = Registry.LocalMachine.OpenSubKey("SOFTWARE\\MyApp\\DBConn")
If Not (regKey Is Nothing) Then
server = regKey.GetValue("Server").ToString()
db = regKey.GetValue("Database").ToString()
uid = regKey.GetValue("UserID").ToString()
pwd = regKey.GetValue("Password").ToString()
sConn = "server=" + server + ";Initial Catalog=" + db + ";UID=" + uid + ";PWD=" + pwd
conn = New SqlConnection(sConn)
conn.Open()
End If
End Sub
End Module
```

Peppering the registry calls throughout an application is tedious--especially if you need to change anything. Many developers place these calls in the Global.asax (for ASP.NET applications) or create a special class for it.

**Note:** Editing the registry is risky, so make sure you have a verified backup before making any changes.

- Using Boundlist controls with data source controls:
- A data source control interacts with the data-bound controls and hides the complex data binding processes.
- These are the tools that provide data to the data bound controls and support execution of operations like insertions, deletions, sorting and updates.

Each data source control wraps a particular data provider-relational databases, XML documents or custom classes and helps in:

- Managing connection.
- Selection of data
- Managing presentation aspects like paging, caching etc.
- Manipulation of data

There are many data source controls available in ASP.Net for accessing data from SQL Server, from ODBC or OLE DB servers, from XML files and from business objects.

Based on type of data, these controls could be divided into two categories: hierarchical data source controls and table-based data source controls.

The data source controls used for hierarchical data are:

- **XMLDataSource**-allows binding to XML files and strings with or without schema information
- **SiteMapDataSource**-allows binding to a provider that supplies site map information

**The data source controls used for tabular data are:**

Data source controls	Description
SqlDataSource	represents a connection to an ADO.Net data provider that returns SQL data, including data sources accessible via OLEDB and QDBC
ObjectDataSource	allows binding to a custom .Net business object that returns data
LinkDataSource	allows binding to the results of a Link-to-SQL query (supported by ASP.Net 3.5 only)
AccessDataSource	represents connection to a Microsoft Access database

### The Data Source Views

- Data source views are objects of the DataSourceView class and represent a customized view of data for different data operations like sorting, filtering etc.
- The DataSourceView class serves as the base class for all data source view classes, which define the capabilities of data source controls.

Following table provides the properties of the DataSourceView class:

Properties	Description
------------	-------------

CanDelete	Indicates whether deletion is allowed on the underlying data source.
CanInsert	Indicates whether insertion is allowed on the underlying data source.
CanPage	Indicates whether paging is allowed on the underlying data source.
CanRetrieveTotalRowCount	Indicates whether total row count information is available.
CanSort	Indicates whether the data could be sorted.
CanUpdate	Indicates whether updates are allowed on the underlying data source.
Events	Gets a list of event-handler delegates for the data source view.
Name	Name of the view.

**Following table provides the methods of the DataSourceView class:**

Methods	Description
CanExecute	Determines whether the specified command can be executed.
ExecuteCommand	Executes the specific command.
ExecuteDelete	Performs a delete operation on the list of data that the DataSourceView object represents.
ExecuteInsert	Performs an insert operation on the list of data that the DataSourceView object represents.

ExecuteSelect	Gets a list of data from the underlying data storage.
ExecuteUpdate	Performs an update operation on the list of data that the DataSourceView object represents.
Delete	Performs a delete operation on the data associated with the view.
Insert	Performs an insert operation on the data associated with the view.
Select	Returns the queried data.
Update	Performs an update operation on the data associated with the view.
OnDataSourceViewChanged	Raises the DataSourceViewChanged event.
RaiseUnsupportedCapabilitiesError	Called by the RaiseUnsupportedCapabilitiesError method to compare the capabilities requested for an ExecuteSelect operation against those that the view supports.

### The SqlDataSource Control

- The SqlDataSource control represents a connection to a relational database such as SQL Server or Oracle database, or data accessible through OLEDB or Open Database Connectivity (ODBC).
- Connection to data is made through two important properties ConnectionString and ProviderName.

The following code snippet provides the basic syntax for the control:

```
<asp:SqlDataSource runat="server" ID="MySqlSource"
ProviderName='<%= $ ConnectionStrings:LocalNWind.ProviderName %>'
```

```
ConnectionString='<%%$ ConnectionStrings:LocalNWind %>'
```

```
SelectionCommand= "SELECT * FROM EMPLOYEES" />
```

```
<asp:GridView ID="GridView1" runat="server"  
    DataSourceID="MySqlSource">
```

- Configuring various data operations on the underlying data depends upon the various properties (property groups) of the data source control.

The following table provides the related sets of properties of the SqlDataSource control, which provides the programming interface of the control:

Property Group	Description
DeleteCommand, DeleteParameters, DeleteCommandType	Gets or sets the SQL statement, parameters and type for deleting rows in the underlying data.
FilterExpression, FilterParameters	Gets or sets the data filtering string and parameters.
InsertCommand, InsertParameters, InsertCommandType	Gets or sets the SQL statement, parameters and type for inserting rows in the underlying database.
SelectCommand, SelectParameters, SelectCommandType	Gets or sets the SQL statement, parameters and type for retrieving rows from the underlying database.
SortParameterName	Gets or sets the name of an input parameter that the command's stored procedure will use to sort data

UpdateCommand,  
UpdateParameters,  
UpdateCommandType

Gets or sets the SQL statement, parameters and type for updating rows in the underlying data store.

The following code snippet shows a data source control enabled for data manipulation:

```
<asp:SqlDataSource runat="server" ID= "MySqlSource"
ProviderName='<%= $ ConnectionStrings:LocalNWind.ProviderName %>'
ConnectionString=' <%= $ ConnectionStrings:LocalNWind %>'
SelectCommand= "SELECT * FROM EMPLOYEES"
UpdateCommand= "UPDATE EMPLOYEES SET LASTNAME=@lname"
DeleteCommand= "DELETE FROM EMPLOYEES WHERE EMPLOYEEID=@eid"
FilterExpression= "EMPLOYEEID > 10">
.....
.....
</asp:SqlDataSource>
```

### The ObjectDataSource Control:

- The ObjectDataSource Control enables user-defined classes to associate the output of their methods to data bound controls.
- The programming interface of this class is almost same as the SqlDataSource control.

Following are two important aspects of binding business objects:

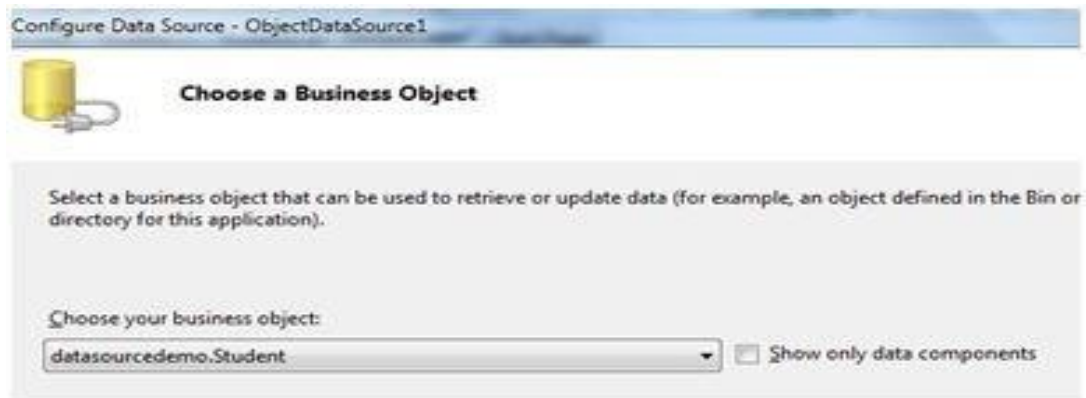
- The bindable class should have a default constructor, be stateless, and have methods that can be mapped to select, update, insert and delete semantics.
- The object must update one item at a time, batch operations are not supported.
- Let us go directly to an example to work with this control. The student class is our class to be used with an object data source.
- This class has three properties: a student id, name and city.
- It has a default constructor and a GetStudents method to be used for retrieving data.

The student class:

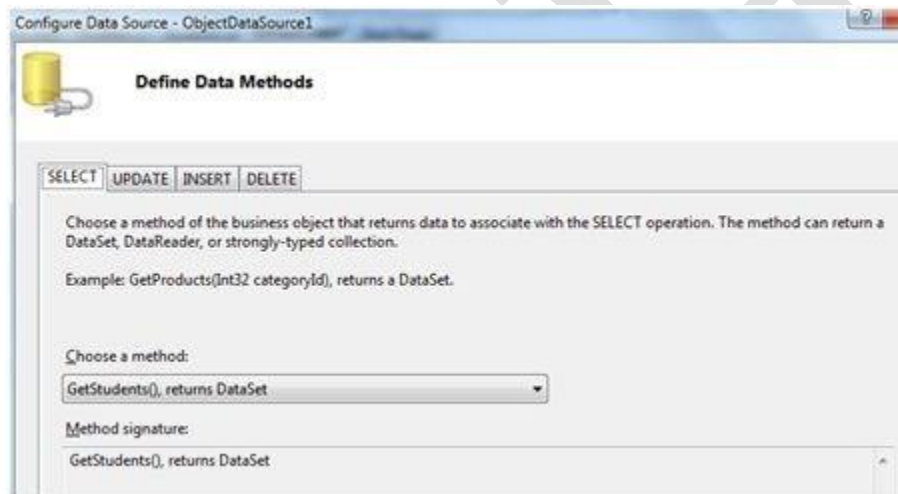
```
public class Student
{
    public int StudentID { get; set; }
    public string Name { get; set; }
    public string City { get; set; }
    public Student()
    { }
    public DataSet GetStudents()
    {
        DataSet ds = new DataSet();
        DataTable dt = new DataTable("Students");
        dt.Columns.Add("StudentID", typeof(System.Int32));
        dt.Columns.Add("StudentName", typeof(System.String));
        dt.Columns.Add("StudentCity", typeof(System.String));
        dt.Rows.Add(new object[] { 1, "M. H. Kabir", "Calcutta" });
        dt.Rows.Add(new object[] { 2, "Ayan J. Sarkar", "Calcutta" });
        ds.Tables.Add(dt);
        return ds;
    }
}
```

Take the following steps to bind the object with an object data source and retrieve data:

- Create a new web site. Add a class (Students.cs) to it by right clicking the project from the Solution Explorer, adding a class template and placing the above code in it.
- Build the solution so that the application can use the reference to the class.
- Place a object data source control in the web form.
- Configure the data source by selecting the object.



- Select a data method(s) for different operations on data. In this example, there is only one method.



- Place a data bound control like grid view on the page and select the object data source as its underlying data source



**GridView Tasks**

Auto Format...

Choose Data Source: **ObjectDataSource1**

Configure Data Source...

Refresh Schema

Edit Columns...

Add New Column...

☐ Enable Paging

☐ Enable Sorting

☐ Enable Selection

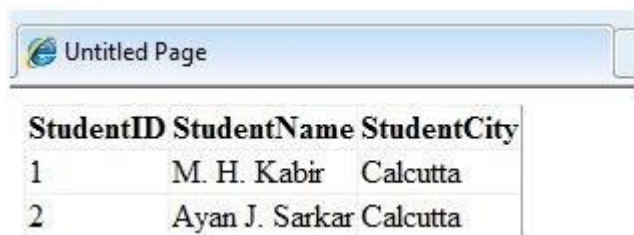
Edit Templates

- At this stage, the design view should look like the following:

**ObjectDataSource - ObjectDataSource1**  
asp:gridview#GridView1

Databound Col0	Databound Col1	Databound Col2
abc	0	abc
abc	1	abc
abc	2	abc
abc	3	abc
abc	4	abc

- Run the project, it retrieves the hard coded tuples from the students class.



Untitled Page

StudentID	StudentName	StudentCity
1	M. H. Kabir	Calcutta
2	Ayan J. Sarkar	Calcutta

### The AccessDataSource Control:

- The AccessDataSource control represents a connection to an Access database.
- It is based on the SqlDataSource control and provides simpler programming interface.

The following code snippet provides the basic syntax for the data source:

```
<asp:AccessDataSource ID="AccessDataSource1"
    runat="server"
    DataFile="~/App_Data/ASPDotNetStepByStep.mdb"
    SelectCommand="SELECT * FROM [DotNetReferences]">
</asp:AccessDataSource>
```

- The AccessDataSource control opens the database in read-only mode.
- However, it can also be used for performing insert, update or delete operations.
- This is done using the ADO.Net commands and parameter collection.
- Updates are problematic for Access databases from within an ASP.Net application because an Access database is a plain file and the default account of the ASP.Net application might not have the permission to write to the database file.

### ASP.NET Data Bound Controls

In this tutorial you will learn about Data Bound Controls - The Hierarchy of Data Bound Controls, Simple Data Bound Controls, Composite DataBound Controls and Hierarchical Data Bound Controls.

### *The Hierarchy of Data Bound Controls*

- Data Bound controls are controls that are bound to data sources.
- Traditionally the DataGrid is the principal data bound control in ASP.NET 1.x. Though DataGrid is still supported, ASP.NET 2.0 introduces three new controls—GridView, FormView and DetailsView.
- Unlike in ASP.NET 1.x, all controls descend from the BaseDataBoundControl class. It has two basic child classes DataBoundControl and HierarchicalDataBoundControl.
- While TreeView and Menu are examples of the latter, AdRotator, ListControls such as BulletedList, CheckedList, DropDownList, ListBox and RadioButtonList, and CompositeDataBoundControls such as DetailsView, FormsView and GridView are examples of the former.
- All Data bound controls can be classified into Simple, composite and hierarchical controls.
- The DataBoundControl Base class defines the common characteristics of non-hierarchical controls which share the same base class.
- It is inherited from the WebControl and has all the visual and style properties of the base class.
- Additionally it has infrastructural properties such as Context, Page and AccessKey.
- DataMember property of the control selects the list of data that the control has to bind to when the DataSource contains more than one list.
- DataSource indicates the source of the data it has to bind to. It is imperative that the DataSource must be an object (unlike ASP.NET 1.x) that implements the IEnumerable or the IListSource interface.
- DataSourceId is the ID of the data source object used to retrieve data.
- The DataBoundControl class has only one method—the DataBind method.
- This method is called when data has to be pumped out of the data source.
- The internal implementation of this method has been tweaked and modified to enhance performance.
- It takes into account the implementation of the IEnumerable based data sources but is more sophisticated in execution as it implements the new interfaces and new data source controls also.

### *Simple Data Bound Controls*

- List based user interface controls have been classified as Simple Data bound controls. ASP.NET 2.0 has two simple data bound controls—the AdRotator and the list control.
- The AdRotator control retrieves information from a XML file which contains the path to the distinctive image, the URL to go to when the control is clicked and the frequency of

the Ad. Arbitrary data sources are supported by the control in ASP.NET 2.0 and it is fully bound to the data source.

- This control has additional capability of creating popup and pop under ads apart from standard banners.
- It supports counters for tracking purposes and updates the ad when the counter is clicked.
- Each ad can be associated with a separate counter.
- The BulletedList control is a new addition to list controls in ASP.NET.
- It can be used to create a list of formatted list items. Individual items can be specified by defining a ListItem for each object.
- The bulleted lists are filled in from the data source and receives the DataTable returned by a query.
- The DataTextField property of the control selects the column to show and the DisplayMode sets the display mode.
- The Hyperlink mode links the page directly to an external URL and click is handled by the browser.
- The LinkButton mode click is handled by the ASP.NET runtime and fires a server side event.
- The OnClick handler will have to be defined in this instance.
- The Index property of the BulletedListEventArgs class contains the base index of the clicked item.
- The BulletStyle property controls the customization of the bullet styles.
- There is no change in the other list controls in ASP.NET 2.0.

### *Composite DataBound Controls*

- Two new base classes have been added to enhance the Composite Data bound controls.
- The new CompositeControl and CompositeDataBoundControl are separate classes with a similar blueprint.
- The CompositeControl class addresses the UI-based needs and CompositeDataBoundControl defines the common foundation for all composite data bound controls.
- The CompositeDataBoundControl is an abstract class that declares and implements the Controls property.
- In ASP.NET 1.x the Controls property stores the references to child controls. The CompositeDataBoundControl additionally exposes the CreateChildControls property.
- It takes a Boolean argument and behaves in accordance with the argument taken to ignore or initialize the view state.

- In ASP.NET 2.0 the developer has to only call the PerformDataBinding method and all boilerplate tasks are performed and the implementation of the CreateChildControls is called to implement the building of the control tree.
- We shall see examples of the GridView, the DetailsView and the FormView controls a little later in this tutorial.

### *Hierarchical Data Bound Controls*

- The HierarchicalDataBoundControl class is the base class for hierarchical data bound controls such as TreeView and Menu.
- It is an abstract class but does not have any predefined services.
- It behaves like a logical container for controls that consume the hierarchical data.
- The TreeView control displays a hierarchy of nodes. Each node may contain child nodes.
- Parent nodes can be expanded to display child nodes or collapsed. Checkboxes can be displayed next to nodes or images from an imagelist control can be displayed.
- Nodes can be programmatically selected or cleared.
- The key properties of the TreeView control are Nodes and SelectedNodes.
- The Nodes property defines the top level nodes in the TreeView.
- The SelectedNode property sets the currently selected node.
- The TreeView control binds to any data source object that implements the IHierarchicalDataSource interface.
- It also exposes a DataSource property of the type object which can be assigned to an XmlDataDocument or to plain XML.
- By default the nodes are bound to its own nodes to reflect the name of the node rather than the attribute or the inner text.
- The node to node association can be controlled by binding parameters.
- The nodes can be bound to a data source field by specifying tree node bindings.
- The TreeNodeBinding object defines the relationship between each data item and the node it is binding.
- The Menu control is an end to end site navigation tool. It can be bound to any data source and also supports explicit list of items for simple cases.
- The menu items are stored in a collection and the MenuItem collection property returns all the child items of a given menu.
- A few static and dynamic methods are supported by this class such as selected item, submenus and mouse over items.
- Dynamic menus are implemented using the Dynamic HTML object model.

### **Managing Data with ADO.NET**

### **The Role of DataSets in ADO.NET**

- The important of datasets can never be denied nor it can be questioned.
- DataSet is a mirror image of the table or the query which you run.
- DataSets makes it easier to edit and update the information.
- Another good thing is that datasets are disconnected in nature so, if you make any changes in the dataset it will not reflect in the database unless use special methods to perform the change and confirm it.

### **Using DataSets in ASP.NET**

- Let's see how we can use datasets in Asp.net.
- The most common way of using the DataSet control is with the dataadapter.
- The DataAdapter fills the dataset control with data coming from the database.

Let's see how we can perform this simple action:

```
SqlDataAdapter ad = new SqlDataAdapter("SELECT * FROM Categories",myConnection);
```

```
DataSet ds = new DataSet();
```

```
ad.Fill(ds,"Categories");
```

```
DataGrid1.DataSource = ds;
```

```
DataGrid1.DataBind();
```

#### **Explanation of the code:**

1. First we declare a simple dataadapter instance which fetches all the rows from the Categories table. ( Categories table is present in the Northwind database which is shipped with Sql Server 2000 ).
2. New we created an instance of the DataSet control.
3. Next we fill the dataset with the data coming from the dataadapter control. As you can see that I have written `ad.Fill(ds,"Categories");` you can also write `ad.Fill(ds);` and this will work fine too. In the former case I was telling the dataadapter which table I am using in my query and later the

dataadapter was finding the table itself.

4. Finally I used DataGrid to bind the data on the screen.

### **Iterating through the DataSet:**

Here is a simple code to iterate through the dataset and select individual items instead of selecting all the data in the DataSet.

```
SqlDataAdapter ad = new SqlDataAdapter("SELECT * FROM Categories",myConnection);
```

```
DataSet ds = new DataSet();
```

```
ad.Fill(ds,"Categories");
```

```
DataTable dt = ds.Tables["Categories"];
```

```
Response.Write(dt.Rows[0][1].ToString());
```

1) First few lines are identical which we have done before.

2) In the forth line we declared the DataTable object. DataTable object is used to hold a single table. As I already told you that dataset can hold multiple tables depending upon the query so we can assign a single table from the dataset to the datatable object. I have done that using the line.

```
DataTable dt = ds.Tables["Categories"];
```

3) Next we are printing the first row of the second column. Always remember that the number of the rows and columns in the database does not start with 1 but it starts at 0. So in this line of code:

```
Response.Write(dt.Rows[0][1].ToString());
```

This line of code means that get Row '0' and Column 1 of the from the datatable. And so we get the value and it prints out "Beverages".

- **Saving dataSets in Session State:**



DataSets can also be saved in the Session State so that it can be available in other areas of the application and in new pages.

- The way of storing the DataSet in Session State is very easy.

```
Session["MyDataSet"] = DataSet;
```

- Later if you want to retrieve the DataSet from the Session State you can easily do this by using this line of code:

```
DataSet ds = (DataSet) Session["MyDataSet"]
```

- As you can see that when I am retrieving the values from the Session object I am casting it into the dataset objec.
- This is because this is explicit conversion and requires casting. I am unboxing in this case.

### Using DataTable:

- DataTable is also a collection which you can use like dataset.
- The difference is that DataTable represents only one table.
- Usually its used when you don't have a database and want to save something in the collection.
- // Create a new DataTable.

```
System.Data.DataTable myDataTable = new DataTable("ParentTable");
```

```
// Declare variables for DataColumn and DataRow objects.
```

```
DataColumn myDataColumn;
```

```
DataRow myDataRow;
```

```
// Create new DataColumn, set DataType, ColumnName and add to DataTable.
```

```
myDataColumn = new DataColumn();
```

```
myDataColumn.DataType = System.Type.GetType("System.Int32");
```

```
myDataColumn.ColumnName = "id";
```

```
myDataColumn.ReadOnly = true;
```

```
myDataColumn.Unique = true;
```

```
// Add the Column to the DataColumnCollection.
```

```
myDataTable.Columns.Add(myDataColumn);
```

### DataSet Relations:



- In a relational representation of data, individual tables contain rows that are related to one another using a column or set of columns.
- In the ADO.NET DataSet, the relationship between tables is implemented using a DataRelation.
- When you create a DataRelation, the parent-child relationships of the columns are managed only through the relation.
- The tables and columns are separate entities.
- In the hierarchical representation of data that XML provides, the parent-child relationships are represented by parent elements that contain nested child elements.
- To facilitate the nesting of child objects when a DataSet is synchronized with an XmlDataDocument or written as XML data using WriteXml, the DataRelation exposes a Nested property.
- Setting the Nested property of a DataRelation to true causes the child rows of the relation to be nested within the parent column when written as XML data or synchronized with an XmlDataDocument.

The Nested property of the DataRelation is false, by default. For example, consider the following DataSet:

```
SqlDataAdapter custDA = new SqlDataAdapter("SELECT CustomerID, CompanyName FROM Customers", nwindConn);
SqlDataAdapter orderDA = new SqlDataAdapter("SELECT OrderID, CustomerID, OrderDate FROM Orders", nwindConn);
nwindConn.Open();
DataSet custDS = new DataSet("CustomerOrders");
custDA.Fill(custDS, "Customers");
orderDA.Fill(custDS, "Orders");
nwindConn.Close();
DataRelation custOrderRel = custDS.Relations.Add("CustOrders",
custDS.Tables["Customers"].Columns["CustomerID"],
custDS.Tables["Orders"].Columns["CustomerID"]);
```

KAHE