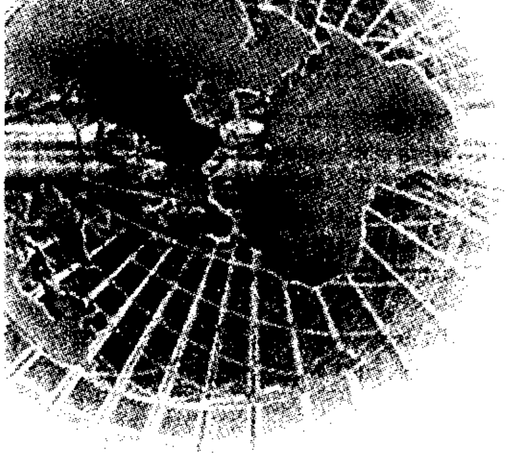


21 Working with Database Controls

What are databases? Why do we need them? How does a .NET application work with databases? These are some of the questions whose answers we need to know before we learn about working with databases in ASP.NET.

A database is a collection of records or information that is stored in the form of tables in a systematic way a computer program can access the information easily whenever required. Structured Query Language (SQL) is used for retrieving, storing, deleting, and updating the records stored in a database. An application may need to use the databases for performing various functions, such as:

- Displaying data in a tabular format by retrieving the records from the database
- Displaying the data after processing the record retrieved from the database, such as calculating the service duration of each employee and then displaying the records
- Processing the retrieved data and updating them in the database
- Deleting the records from the database depending on the choice or condition specified by the user of the application

In this chapter, you learn about the use of various data sources and data bound controls supported by ASP.NET 3.5. You can bind the data bound controls to the data source controls to display the data. Following is a list of data bound controls:

- The `GridView` Control
- The `DataList` Control
- The `DetailsView` Control
- The `FormView` Control
- The `ListView` Control
- The `Repeater` Control
- The `DataPager` Control

The data source controls allow you to work with different types of data sources, such as SQL server or an XML file. Following is a list of data source controls:

- The `SqlDataSource` Control
- The `AccessDataSource` Control
- The `LinqDataSource` Control
- The `ObjectDataSource` Control
- The `XmlDataSource` Control
- The `SiteMapDataSource` Control

Later on, in this chapter, you learn to modify and edit the data using the data bound and data source controls. In Visual Studio 2008, the data bound and the data source controls are placed under the Data tab in the toolbox.

Now, let's start the discussion with the `GridView` Control in detail.

The **GridView** Control

The `GridView` control is a data bound control that displays the values of a data source in the form of a table. In this table, each column represents a field and each row represents a record. The `GridView` control exists within the `System.Web.UI.Controls` namespace. The inheritance hierarchy of the `GridView` control is as follows:

```
System.Object
    System.Web.UI.Control
        System.Web.UI.WebControls.WebControl
            System.Web.UI.WebControls.BaseDataboundControl
                System.Web.UI.WebControls.DataboundControl
                    System.Web.UI.WebControls.CompositeDataboundControl
                        System.Web.UI.WebControls.GridView
```

When you drag and drop the `GridView` control on the design view, the following syntax is added to the source view of the page:

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

The GridView data control has a built-in capability of sorting, paging, updating, and deleting data. You can also set the column fields through the AutoGenerate property to indicate whether bound fields are automatically created for each field in the data source. Table 21.1 lists the different column field types of the GridView class:

Table 21.1: Column Field Types of the GridView Class

Name	Description
BoundField	Shows the value of a field in a data source. It is the default column type of the GridView control.
ButtonField	Shows a command button for each item in the GridView control. It allows to create a column of custom button controls, such as the Add or the Remove button.
CheckBoxField	Shows a CheckBox control for each item in the GridView control. This column field type is commonly used to display fields with a Boolean value.
CommandField	Shows predefined command buttons to perform operations, such as select, edit, or delete.
HyperLinkField	Shows the value of a field in a data source as a hyperlink. This column field type allows you to bind a second field to the hyperlink's URL.
ImageField	Shows an image for each item in the GridView control.
TemplateField	Shows user-defined content for each item in the GridView control according to a specified template. This column field type enables us to create a custom column field.

You can also customize the appearance of the GridView control by setting the style properties. The different style properties are listed in Table 21.2:

Table 21.2: Style Properties of the GridView Class

Style Property	Description
EditRowStyle	Performs the style settings for the row that is edited in the GridView control.
EmptyDataRowStyle	Performs the style settings for the empty data row displayed in the GridView control when the data source does not contain any records.
FooterStyle	Performs the style settings for the footer row of the GridView control.
HeaderStyle	Performs the style settings for the header row of the GridView control.
PagerStyle	Performs the style settings for the pager row of the GridView control.
SelectedRowStyle	Performs the style settings for the selected row in the GridView control.

Noteworthy properties of the GridView class are listed in Table 21.3:

Table 21.3: Noteworthy Properties of the GridView Class

Property	Description
AllowPaging	Obtains or sets a value indicating whether the paging feature is enabled.
AllowSorting	Obtains or sets a value indicating whether the sorting feature is enabled.
AlternatingRowStyle	Obtains a reference to the TableItemStyle object that enables us to set the appearance of alternating data rows in a GridView control.
AutoGenerateColumns	Obtains or sets a value indicating whether bound fields are automatically created for each field in the data source.
AutoGenerateDeleteButton	Obtains or sets a value indicating whether a CommandField field column with a Delete button for each data row is automatically added to a GridView control.

Table 21.3: Noteworthy Properties of the GridView Class

Property	Description
`AutoGenerateEditButton`	Obtains or sets a value indicating whether a `CommandField` field column with an Edit button for each data row is automatically added to a `GridView` control.
`AutoGenerateSelectButton`	Obtains or sets a value indicating whether a `CommandField` field column with a Select button for each data row is automatically added to a `GridView` control.
`BackImageUrl`	Obtains or sets the `URL` to an image to display in the background of a `GridView` control.
`BottomPagerRow`	Obtains a `GridViewRow` object that represents the bottom pager row in a `GridView` control.
`Caption`	Obtains or sets the text to render in an HTML caption element in a `GridView` control. This property is provided to make the control more accessible to users of assistive technology devices.
`CaptionAlign`	Obtains or setsthe horizontal or vertical position of the HTML caption element in a `GridView` control. This property is provided to make the control more accessible to users of assistive technology devices.
`CellPadding`	Obtains or sets the amount of space between the contents of a cell and the cell's border.
`CellSpacing`	Obtains or sets the amount of space between cells.
`Columns`	Obtains a collection of `DataControlField` objects that represent the column fields in a `GridView` control.
`DataKeyNames`	Obtains or sets an array that contains the names of the primary key fields for the items displayed in a `GridView` control.
`DataKeys`	Obtains a collection of `DataKey` objects that represent the data key value of each row in a `GridView` control.
`EditIndex`	Obtains or sets the index of the row to edit.
`EditRowStyle`	Obtains a reference to the `TableItemStyle` object that enables us to set the appearance of the row selected for editing in a `GridView` control.
`EmptyDataRowStyle`	Obtains a reference to the `TableItemStyle` object that enables us to set the appearance of the empty data row rendered when a `GridView` control is bound to a data source that does not contain any records.
`EmptyDataTemplate`	Obtains or sets the user-defined content for the empty data row rendered when a `GridView` control is bound to a data source that does not contain any records.
`EmptyDataText`	Obtains or sets the text to display in the empty data row rendered when a `GridView` control is bound to a data source that does not contain any records.
`EnableSortingAndPagingCallbacks`	Obtains or sets a value indicating whether client-side callbacks are used for sorting and paging operations.
`FooterRow`	Obtains a `GridViewRow` object that represents the footer row in a `GridView` control.
`FooterStyle`	Obtains a reference to the `TableItemStyle` object that enables us to set the appearance of the footer row in a `GridView` control.
`GridLines`	Obtains or sets the gridline style for a `GridView` control.
`HeaderRow`	Obtains a `GridViewRow` object that represents the header row in a GridView control.
`HeaderStyle`	Obtains a reference to the `TableItemStyle` object that enables us to set the appearance of the header row in a `GridView` control.

Table 21.3: Noteworthy Properties of the GridView Class

Property	Description
`HorizontalAlign`	Obtains or sets the horizontal alignment of a `GridView` control on the page.
`PageCount`	Obtains the number of pages required to display the records of the data source in a `GridView` control.
`PageIndex`	Obtains or sets the index of the currently displayed page.
`PagerSettings`	Obtains a reference to the `PagerSettings` object that enables us to set the properties of the pager buttons in a `GridView` control.
`PagerStyle`	Obtains a reference to the `TableItemStyle` object that enables us to set the appearance of the pager row in a `GridView` control.
`PagerTemplate`	Obtains or sets the custom content for the pager row in a `GridView` control.
`PageSize`	Obtains or sets the number of records to display on a page in a `GridView` control.
`RowHeaderColumn`	Obtains or sets the name of the column to use as the column header for the `GridView` control. This property is provided to make the control more accessible to users of assistive technology devices.
`Rows`	Obtains a collection of `GridViewRow` objects that represent the data rows in a GridView control.
`RowStyle`	Obtains a reference to the `TableItemStyle` object that enables us to set the appearance of the data rows in a GridView control.
`SelectedDataKey`	Obtains the `DataKey` object that contains the data key value for the selected row in a GridView control.
`SelectedIndex`	Obtains or sets the index of the selected row in a `GridView` control.
`SelectedRow`	Obtains a reference to a `GridViewRow` object that represents the selected row in the control.
`SelectedRowStyle`	Obtains a reference to the `TableItemStyle` object that enables us to set the appearance of the selected row in a `GridView` control.
`SelectedValue`	Obtains the data key value of the selected row in a `GridView` control.
`ShowFooter`	Obtains or sets a value showing whether the footer row is displayed in a `GridView` control.
`ShowHeader`	Obtains or sets a value showing whether the header row is displayed in a `GridView` control.
`SortDirection`	Obtains the sort direction of the column being sorted.
`SortExpression`	Obtains the sort expression associated with the column or columns being sorted.
`TopPagerRow`	Obtains a `GridViewRow` object that represents the top pager row in a `GridView` control.
`UseAccessibleHeader`	Obtains or sets a value indicating whether a `GridView` control renders its header in an accessible format. This property makes the control more accessible to users of assistive technology devices.

Noteworthy methods of the `GridView` class are listed in Table 21.4:

Table 21.4: Noteworthy Methods of the GridView Class

Methods	Description
`DataBind`	Binds the data source to the `GridView` control

Table 21.4: Noteworthy Methods of the GridView Class

Methods	Description
`DeleteRow`	Deletes the record at the specified index from the data source
`IsBindableType`	Determines whether the specified data type can be bound to a column in a `GridView` control
`Sort`	Sorts the `GridView` control based on the specified sort expression and direction
`UpdateRow`	Updates the record at the specified row index using the field values of the row

Noteworthy events of the `GridView` class are listed in Table 21.5:

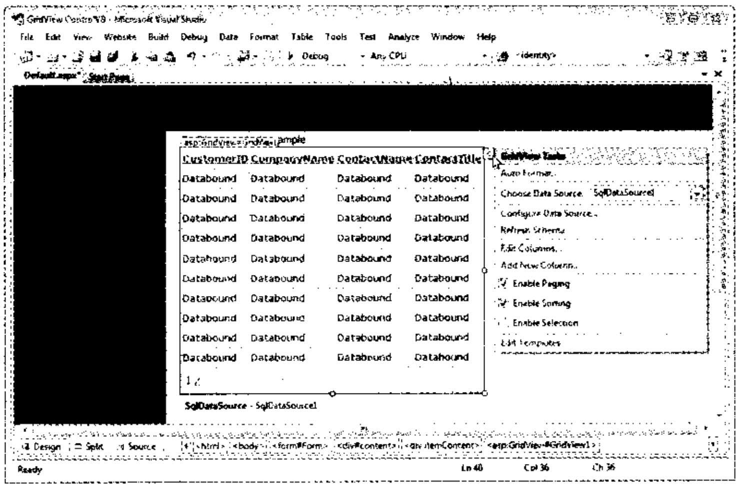
Table 21.5: Noteworthy Events of the GridView Class

Event	Description
`PageIndexChanged`	Invoked when one of the pager buttons is clicked after the `GridView` control handles the paging operation
`PageIndexChanging`	Invoked when one of the pager buttons is clicked, but before the `GridView` control handles the paging operation
`RowCancelingEdit`	Invoked when the Cancel button of a row in edit mode is clicked, but before the row exits the edit mode
`RowCommand`	Invoked when a button is clicked in a `GridView` control
`RowCreated`	Invoked when a row is created in a `GridView` control
`RowDataBound`	Invoked when a data row is bound to data in a `GridView` control
`RowDeleted`	Invoked when a row's Delete button is clicked, but after the `GridView` control deletes the row
`RowDeleting`	Invoked when a row's Delete button is clicked, but before the `GridView` control deletes the row
`RowEditing`	Invoked when a row's Edit button is clicked, but before the `GridView` control enters edit mode
`RowUpdated`	Invoked when a row's Update button is clicked, but after the `GridView` control updates the row
`RowUpdating`	Invoked when a row's Update button is clicked, but before the `GridView` control updates the row
`SelectedIndexChanged`	Invoked when a row's Select button is clicked, but after the `GridView` control handles the select operation
`SelectedIndexChanging`	Invoked when a row's Select button is clicked, but before the `GridView` control handles the select operation.
`Sorted`	Invoked when the hyperlink to sort a column is clicked, but after the `GridView` control handles the sort operation
`Sorting`	Invoked when the hyperlink to sort a column is clicked, but before the `GridView` control handles the sort operation

Using the **GridView** Control

Now, let's enable the paging, sorting, and selection features of a `GridView` control by performing the following steps:

1. Create a website and save it as `GridViewControlVB` Control. You can find the code of `GridViewControlVB` application in the Code\ASP.NET\Chapter 21\`GridViewControlVB` folder on the CD.
2. Select the Enable Paging and Enable Sorting check boxes in the Smart Tag, as shown in Figure 21.1:



[✓ Enable Paging]
[✓ Enable Sorting]
[ ] Enable Selection

Figure 21.1: Enabling Paging, Sorting, and Selection on the GridView Control

NOTE

In this application, we have created a connection string with the Customers table of the Northwind database by using the SqlDataSource control.

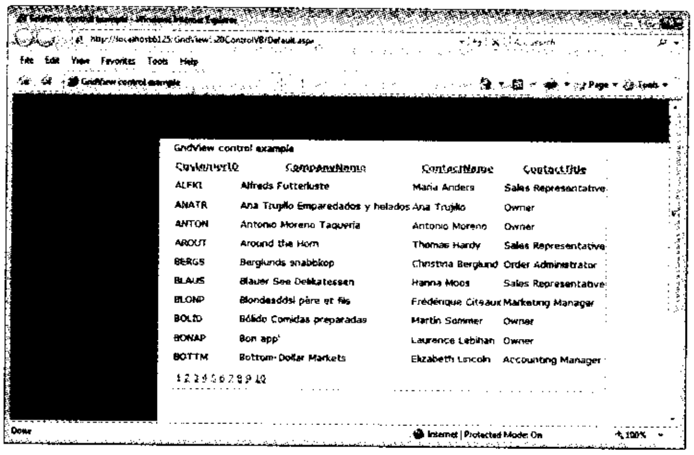
3. The code for the `Default.aspx` page of the `GridView` Control application is shown in Listing 21.1:

Listing 21.1: Showing the Code of the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
   Inherits="_Default" %>
<!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 id="Head1" runat="server">
   <title>GridView control example </title>
   <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
   <form id="Form" runat="server">
        <div id="header">
        </div>
        <div id="sidebar">
            <div id="nav">
                &nbsp;
            </div>
        </div>
        <div id="content">
            <div class ="itemContent">
            GridView control example
            <br />
            <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
            AllowSorting="True" AutoGenerateColumns="False" DataKeyNames="CustomerID"
            DataSourceID="SqlDataSource1">
            <Columns>
```

```
                <asp:BoundField DataField="CustomerID" HeaderText="CustomerID"
                ReadOnly="True"
                SortExpression="CustomerID" />
                <asp:BoundField DataField="CompanyName" HeaderText="CompanyName"
                SortExpression="CompanyName" />
                <asp:BoundField DataField="ContactName" HeaderText="ContactName"
                SortExpression="ContactName" />
                <asp:BoundField DataField="ContactTitle"
                HeaderText="ContactTitle"
                SortExpression="ContactTitle" />
            </Columns>
            </asp:GridView>
            <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
            SelectCommand="SELECT [CustomerID], [CompanyName], [ContactName],
            [ContactTitle] FROM [Customers]">
            </asp:SqlDataSource>
            <div id="footer">
                <p class="left">
                    All content copyright &copy; Kogent Solutions Inc.</p>
            </div>
            </div>
        </div>
    </form>
</body>
</html>
```

4. Run the application by pressing the F5 key. The output of the GridViewControlVB application is shown in Figure 21.2:


GridView control example

CustomerID	CompanyName	ContactName	ContactTitle
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner
AROUT	Around the Horn	Thomas Hardy	Sales Representative
BERGS	Berglunds snabbkop	Christina Berglund	Order Administrator
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager
BOLID	Bólido Comidas preparadas	Martin Sommer	Owner
BONAP	Bon app'	Laurence Lebihan	Owner
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager

1 2 3 4 5 6 7 8 9 10

File Edit View Favorites Tools Help
Page
Tools
Internet | Protected Mode: On
100%

Figure 21.2: Output of the GridViewControlVB Application

Now, let's learn about the DataList control.

The **DataList** Control

The DataList control is a data bound control that displays data by using templates. These templates define controls and HTML elements that should be displayed for an item. The DataList control exists within the System.Web.UI.WebControl namespace. The inheritance hierarchy of the DataList control is as follows:

```
System.Object
  System.Web.UI.Control
      System.Web.UI.WebControls.WebControl
```

```
        System.Web.UI.WebControls.BaseDataList
            System.Web.UI.WebControls.DataList
```

You can change the content of the `DataList` control by using the templates described in Table 21.6:

Table 21.6: Templates of the DataList Class

[illegible]	[illegible]
`AlternatingItemTemplate`	Provides the content and layout for alternating items in the `DataList control`, if defined, otherwise, the `ItemTemplate` control is used.
`EditItemTemplate`	Provides the content and layout for the item currently edited in the `DataList control`, if defined, otherwise the `ItemTemplate` control is used.
`FooterTemplate`	Provides the content and layout for the footer section of the `DataList control`, if defined, otherwise, the footer section is not displayed.
`HeaderTemplate`	Provides the content and layout for the header section of the `DataList control`, if defined, otherwise the header section is not displayed.
`ItemTemplate`	Provides the content and layout for items in the `DataList control`. It is an obligatory template.
`SelectedItemTemplate`	Provides the content and layout for the currently selected item in the `DataList control`, if defined, otherwise the `ItemTemplate` control is used.
`SeparatorTemplate`	Provides the content and layout for the separator between items in the `DataList control`, if defined, otherwise, the separator is not displayed.

We can also configure the `DataList` control to enable users to edit or delete a record in the table and it can render its items horizontally or vertically.

Noteworthy properties of the `DataList` class are listed in Table 21.7:

Table 21.7: Noteworthy Properties of the DataList Class

[illegible]	[illegible]
`AlternatingItemStyle`	Obtains the style properties for alternating items in the `DataList` control
`AlternatingItemTemplate`	Obtains or sets the template for alternating items in the `DataList`
`EditItemIndex`	Obtains or sets the index number of the selected item in the `DataList` control to edit
`EditItemStyle`	Obtains the style properties for the item selected for editing in the `DataList` control
`EditItemTemplate`	Obtains or sets the template for the item selected for editing in the `DataList` control
`ExtractTemplateRows`	Obtains or sets a value that indicates whether the rows of a `Table` control, defined in each template of a `DataList` control, are extracted and displayed
`FooterStyle`	Obtains the style properties for the footer section of the `DataList` control
`FooterTemplate`	Obtains or sets the template for the footer section of the `DataList` control
`GridLines`	Obtains or sets the grid line style for the `DataList` control when the `RepeatLayout` property is set to `RepeatLayout.Table`
`HeaderStyle`	Obtains the style properties for the heading section of the `DataList` control
`HeaderTemplate`	Obtains or sets the template for the heading section of the `DataList` control
`Items`	Obtains a collection of `DataListItem` objects representing the individual items within the control
`ItemStyle`	Obtains the style properties, such as BackColor for the items in the `DataList` control
`ItemTemplate`	Obtains or sets the template for the items in the `DataList` control

Table 21.7: Noteworthy Properties of the DataList Class

[illegible]	[illegible]
`RepeatColumns`	Obtains or sets the number of columns to display in the `DataList` control
`RepeatDirection`	Obtains or sets whether the `DataList` control displays vertically or horizontally
`RepeatLayout`	Obtains or sets whether the control is displayed in a table or flow layout
`SelectedIndex`	Obtains or sets the index of the selected item in the `DataList` control
`SelectedItem`	Obtains the selected item in the `DataList` control
`SelectedItemStyle`	Obtains the style properties for the selected item in the `DataList` control
`SelectedItemTemplate`	Obtains or sets the template for the selected item in the `DataList` control
`SelectedValue`	Obtains the value of the key field for the selected data list item
`SeparatorStyle`	Obtains the style properties of the separator between each item in the `DataList` control
`SeparatorTemplate`	Obtains or sets the template for the separator between the items of the `DataList` control
`ShowFooter`	Obtains or sets a value indicating whether the footer section is displayed in the `DataList` control
`ShowHeader`	Obtains or sets a value indicating whether the header section is displayed in the `DataList` control

Noteworthy events of `DataList` class are listed in Table 21.8:

Table 21.8: Noteworthy Events of the DataList Class

[illegible]	[illegible]
`CancelCommand`	Invoked when the `Cancel` button is clicked for an item in the `DataList` control
`DeleteCommand`	Invoked when the `Delete` button is clicked for an item in the `DataList` control
`EditCommand`	Invoked when the `Edit` button is clicked for an item in the `DataList` control
`ItemCommand`	Invoked when any button is clicked in the `DataList` control
`ItemCreated`	Invoked on the server when an item in the `DataList` control is created
`ItemDatabound`	Invoked when an item is data bound to the `DataList` control
`UpdateCommand`	Invoked when the `Update` button is clicked for an item in the `DataList` control

Using the **DataList** Control

Now, let's use a `DataList` control to display a single data row from the database by performing the following steps:

1. Create an application and save it as `DataListControlVB`. You can find the code of `DataListControlVB` application in the Code\ASP.NET\Chapter 21\`DataListControlVB` folder on the CD.

NOTE

In this application, we have created a connection string with the Customers table of the Northwind database by using the SqlDataSource control.

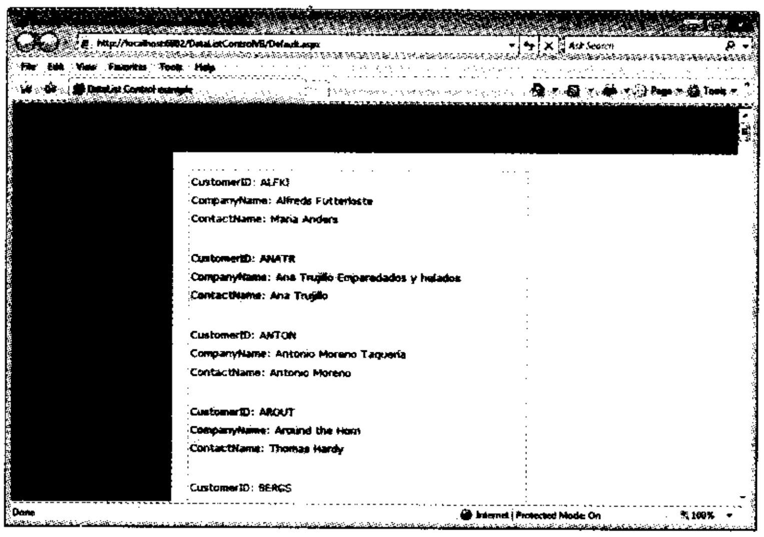
2. Now, replace the code for the `Default.aspx` page with the code shown in Listing 21.2, to add a `DataList` control, three `Label` controls, and a `SQLDataSource` control required for the application:

Listing 21.2: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
 Inherits="_Default" %>
<!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 id="Head1" runat="server">
  <title>DataList Control example </title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="Form1" runat="server">
      <div id="header">

      </div>
      <div id="sidebar">
          <div id="nav">
              &nbsp;
          </div>
      </div>
      <div id="content">
          <div class ="itemContent">
          <br />
          <asp:DataList ID="DataList1" runat="server"
          BackColor="LightGoldenrodYellow" BorderColor="Tan"
          BorderWidth="1px" CellPadding="2" DataKeyField="CustomerID"
          DataSourceID="SqlDataSource1"
          ForeColor="Black" Width="428px" Height="307px">
          <FooterStyle BackColor="Tan" />
          <SelectedItemStyle BackColor="DarkSlateBlue" ForeColor="GhostWhite" />
          <ItemTemplate>
          CustomerID:
          <asp:Label ID="CustomerIDLabel" runat="server" Text='<%#
          Eval("CustomerID") %>'></asp:Label><br />
          CompanyName:
          <asp:Label ID="CompanyNameLabel" runat="server"
          Text='<%# Eval("CompanyName") %>'></asp:Label><br />
          ContactName:
          <asp:Label ID="ContactNameLabel" runat="server"
          Text='<%# Eval("ContactName") %>'></asp:Label><br />
          <br />
          </ItemTemplate>
          <AlternatingItemStyle BackColor="PaleGoldenrod" />
          <HeaderStyle BackColor="Tan" Font-Bold="True" />
          </asp:DataList>
          <asp:SqlDataSource ID="SqlDataSource1" runat="server"
          ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
          SelectCommand="SELECT [CustomerID], [CompanyName], [ContactName] FROM
          [Customers]">
          </asp:SqlDataSource>
          <br />
          <div id="footer">
              <p class="left">
                  All content copyright &copy; Kogent Solutions Inc.</p>
          </div>
          </div>
      </div>
```

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

3. Run the application by pressing the F5 key. The output of the DataListControlVB application is shown in Figure 21.3:


http://localhost6802/DataListControlVB/Default.aspx
Ask Search
File Edit View Favorites Tools Help
DataList Control example
Page Tools
Done
Internet | Protected Mode On
100%

```
CustomerID: ALFKI
CompanyName: Alfreds Futterkiste
ContactName: Maria Anders

CustomerID: ANATR
CompanyName: Ana Trujillo Emparedados y helados
ContactName: Ana Trujillo

CustomerID: ANTON
CompanyName: Antonio Moreno Taquería
ContactName: Antonio Moreno

CustomerID: AROUT
CompanyName: Around the Horn
ContactName: Thomas Hardy

CustomerID: BERGS
```

Figure 21.3: Output of the DataListControlVB Application

NOTE

We have changed the format of the DataList control to Sand & Sky, by selecting the Auto Format option in the Smart Tag

Now, let's learn about the DetailsView control.

The **DetailsView** Control

The DetailsView control is a data bound control that is used to display a single record from the associated data source in a table format, where each row of the table represents a field of the record. The DetailsView control uses the DataSourceID property to support two-way binding; consequently, it also supports insert, update, and delete operations.

The DetailsView control exists within the System.Web.UI.WebControls namespace. The inheritance hierarchy of the DetailsView control is:

```
System.Object
  System.Web.UI.Control
      System.Web.UI.WebControls. WebControl
          System.Web.UI.WebControls. BaseDataboundControl
              System.Web.UI.WebControls. DataboundControl
                  System.Web.UI.WebControls. CompositeDataboundControl
                      System.Web.UI.WebControls.DetailsView
```

Data rows of a DetailsView control are created by declaring a field control. Table 21.9 shows the different row fields:

NOTE

Row fields, Row field types, and field types are the same in the DetailsView control.

Table 21.9: Row Field Types of the DetailsView Class

[illegible]	[illegible]
`BoundField`	Displays the value of a field in a data source as text.
`ButtonField`	Displays a command button in the `DetailsView` control. This allows you to display a row with a custom button control, such as an Add or a Remove button.
`CheckBoxField`	Displays a check box in the `DetailsView` control. This row field type is commonly used to display fields with a boolean value.
`CommandField`	Displays built-in command buttons to perform edit, insert, or delete operations in the `DetailsView` control.
`HyperLinkField`	Displays the value of a field in a data source as a hyperlink. This row field type allows you to bind a second field to the hyperlink's URL.
`ImageField`	Displays an image in the `DetailsView` control.
`TemplateField`	Displays user-defined content for a row in the `DetailsView` control according to a specified template. This row field type allows you to create a custom row field.

Noteworthy properties of the `DetailsView` class are listed in Table 21.10:

Table 21.10: Noteworthy Properties of the DetailsView Class

[illegible]	[illegible]
`AllowPaging`	Gets or sets a value indicating whether the paging feature is enabled.
`AlternatingRowStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the alternating data rows in a `DetailsView` control.
`AutoGenerateDeleteButton`	Obtains or sets a value indicating whether the built-in controls, such as the `GridView` control, to delete the current record is displayed in a `DetailsView` control.
`AutoGenerateEditButton`	Obtains or sets a value indicating whether the built-in controls to edit the current record are displayed in a `DetailsView` control.
`AutoGenerateInsertButton`	Obtains or sets a value indicating whether the built-in controls to insert a new record are displayed in a `DetailsView` control.
`AutoGenerateRows`	Obtains or sets a value indicating whether the row fields for each field in the data source are automatically generated and displayed in a `DetailsView` control.
`BackImageUrl`	Obtains or sets the URL to an image to display in the background of a `DetailsView` control.
`BottomPagerRow`	Obtains a `DetailsViewRow` object that represents the bottom pager row in a `DetailsView` control.
`Caption`	Obtains or sets the text to render in an HTML caption element in a `DetailsView` control. This property is provided to make the control more accessible to users of assistive technology devices.
`CaptionAlign`	Obtains or sets the horizontal or vertical position of the HTML caption element in a `DetailsView` control. This property is provided to make the control more accessible to users of assistive technology devices.
`CellPadding`	Obtains or sets the amount of space between the contents of a cell and the cell's border.
`CellSpacing`	Gets or sets the amount of space between cells.
`CommandRowStyle`	Gets a reference to the `TableItemStyle` object that allows you to set the appearance of a command row in a `DetailsView` control.

Table 21.10: Noteworthy Properties of the DetailsView Class

[illegible]	[illegible]
`CurrentMode`	Obtains the current data-entry mode of the `DetailsView` control.
`DataItem`	Obtains the data item bound to the `DetailsView` control.
`DataItemCount`	Obtains the number of items in the underlying data source.
`DataItemIndex`	Obtains the index of the item displayed in a `DetailsView` control from the underlying data source.
`DataKey`	Obtains a `DataKey` object that represents the primary key of the displayed record.
`DataKeyNames`	Obtains or sets an array that contains the names of the key fields for the data source.
`DefaultMode`	Obtains or sets the default data-entry mode of the `DetailsView` control.
`EditRowStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the data rows when a `DetailsView` control is in edit mode.
`EmptyDataRowStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the empty data row displayed when the data source bound to a `DetailsView` control does not contain any records. '
`EmptyDataTemplate`	Obtains or sets the user-defined content for the empty data row rendered when a `DetailsView` control is bound to a data source that does not contain any records.
`EmptyDataText`	Obtains or sets the text to display in the empty data row rendered when a `DetailsView` control is bound to a data source that does not contain any records.
`EnablePagingCallbacks`	Obtains or sets a value indicating whether client-side callback functions are used for paging operations in the `DetailsView` control.
`FieldHeaderStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the header column in a `DetailsView` control.
`Fields`	Obtains a collection of `DataControlField` objects that represent the explicitly declared row fields in a `DetailsView` control.
`FooterRow`	Obtains a `DetailsViewRow` object that represents the footer row in a `DetailsView` control.
`FooterStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the footer row in a `DetailsView` control.
`FooterTemplate`	Obtains or sets the user-defined content for the footer row in a `DetailsView` control.
`FooterText`	Obtains or sets the text to display in the footer row of a `DetailsView` control.
`GridLines`	Obtains or sets the gridline style for a `DetailsView` control.
`HeaderRow`	Obtains a `DetailsViewRow` object that represents the header row in a `DetailsView` control.
`HeaderStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the header row in a `DetailsView` control.
`HeaderTemplate`	Obtains or sets the user-defined content for the header row in a `DetailsView` control.
`HeaderText`	Obtains or sets the text to display in the header row of a `DetailsView` control.
`HorizontalAlign`	Obtains or sets the horizontal alignment of a `DetailsView` control on the page.

Table 21.10: Noteworthy Properties of the DetailsView Class

[illegible]	[illegible]
`InsertRowStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the data rows in a `DetailsView` control when the `DetailsView` control is in insert mode.
`PageCount`	Obtains the number of the records in the data source.
`PageIndex`	Obtains or sets the index of the displayed record.
`PagerSettings`	Obtains a reference to the `PagerSettings` object that allows you to set the properties of the pager buttons in a `DetailsView` control.
`PagerStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the pager row in a `DetailsView` control.
`PagerTemplate`	Obtains or sets the custom content for the pager row in a `DetailsView` control.
`Rows`	Obtains a collection of `DetailsViewRow` objects that represent the data rows in a `DetailsView` control.
`RowStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the data rows in a `DetailsView` control.
`SelectedValue`	Obtains the data key value of the current record in a `DetailsView` control.

NOTE

The AutoGenerateRows property of the DetailsView control is used to generate the row fields in the datasource. The default value for the AutoGenerateRows property is true, which auitomatically generates the row fieds. To explicitly declare the row fields, you need to set the AutoGenerateRows property to false.

Noteworthy methods of the `DetailsView` class are listed in Table 21.11:

Table 21.11: Noteworthy Methods of the DetailsView Class

[illegible]	[illegible]
`ChangeMode`	Use to switche the `DetailsView` control to the specified mode
`DataBind`	Use to bind data from the data source to the control
`DeleteItem`	Use to delete the current record from the data source
`InsertItem`	Use to insers the current record in the data source
`IsBindableType`	Use to determine whether the specified data type can be bound to a field in the `DetailsView` control
`UpdateItem`	Use to update the current record in the data source

Noteworthy events of the `DetailsView` class are listed in Table 21.12:

Table 21.12: Noteworthy Events of the DetailsView Class

[illegible]	[illegible]
`ItemCommand`	Invoked when a button within a `DetailsView` control is clicked
`ItemCreated`	Invoked when a record is created in a `DetailsView` control
`ItemDeleted`	Invoked when a Delete button within a `DetailsView` control is clicked, but after the delete operation

Table 21.12: Noteworthy Events of the DetailsView Class

`ItemDeleting`	Invoked when a Delete button within a `DetailsView` control is clicked, but before the delete operation
`ItemInserted`	Invoked when an Insert button within a `DetailsView` control is clicked, but after the insert operation
`ItemInserting`	Invoked when an Insert button within a `DetailsView` control is clicked, but before the insert operation
`ItemUpdated`	Invoked when an Update button within a `DetailsView` control is clicked, but after the update operation
`ItemUpdating`	Invoked when an Update button within a `DetailsView` control is clicked, but before the update operation
`ModeChanged`	Invoked when a `DetailsView` control attempts to change between edit, insert, and read-only mode, but after the `CurrentMode` property is updated
`ModeChanging`	Invoked when a `DetailsView` control attempts to change between edit, insert, and read-only mode, but before the `CurrentMode` property is updated
`PageIndexChanged`	Invoked when the value of the `PageIndex` property changes after a paging operation
`PageIndexChanging`	Invoked when the value of the `PageIndex` property changes before a paging operation

Using the **DetailsView** Control

Now, let's use a `DetailsView` control to display data from a database by performing the following steps:

1. Create an application and save it as `DetailsViewControlVB`. You can find the code of `DetailsViewControlVB` application in the Code\ASP.NET\Chapter 21\`DetailsViewControlVB` folder on the CD.

NOTE

In this application, we have created a connection string with the Customers table of the Northwind database by using the SqlDataSource control,

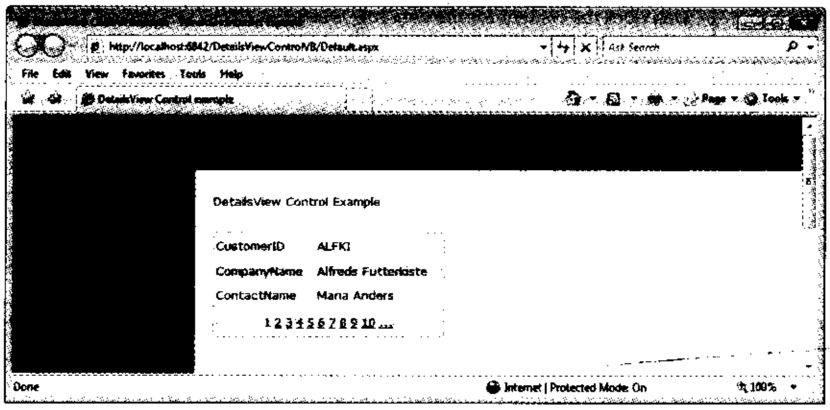
2. Now, replace the code for the `Default.aspx` page with the code shown in Listing 21.3, to add the controls required for the application:

Listing 21.3: Showing the Code for the `Deafault.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!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 id="Head1" runat="server">
  <title>DetailsView Control example </title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="Form" runat="server">
      <div id="header">

      </div>
      <div id="sidebar">
          <div id="nav">
              &nbsp;
          </div>
      </div>
```

```
        <div id="content">
            <div class ="itemContent">
            <br />
            <asp:Label ID="Label1" runat="server" Text="DetailsView Control
            Example"></asp:Label><br />
            <br />
            <asp:DetailsView ID="DetailsView1" runat="server" AllowPaging="True"
            AutoGenerateRows="False"
            BackColor="LightGoldenrodYellow" BorderColor="Tan" BorderWidth="1px"
            CellPadding="2"
            DataKeyNames="CustomerID" DataSourceID="SqlDataSource1" ForeColor="Black"
            GridLines="None" Height="50px" Width="256px">
            <FooterStyle BackColor="Tan" />
            <EditRowStyle BackColor="DarkSlateBlue" ForeColor="GhostWhite" />
            <PagerStyle BackColor="PaleGoldenrod" ForeColor="DarkSlateBlue"
            HorizontalAlign="Center" />
            <Fields>
            <asp:BoundField DataField="CustomerID" HeaderText="CustomerID"
            ReadOnly="True" SortExpression="CustomerID" />
            <asp:BoundField DataField="CompanyName" HeaderText="CompanyName"
            SortExpression="CompanyName" />
            <asp:BoundField DataField="ContactName" HeaderText="ContactName"
            SortExpression="ContactName" />
            </Fields>
            <HeaderStyle BackColor="Tan" Font-Bold="True" />
            <AlternatingRowStyle BackColor="PaleGoldenrod" />
            </asp:DetailsView>
            <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
            SelectCommand="SELECT [CustomerID], [CompanyName], [ContactName] FROM
            [Customers]">
            </asp:SqlDataSource>
            <br />
            <div id="footer">
                <p class="left">
                All content copyright &copy; Kogent Solutions Inc.</p>
            </div>
            </div>
        </div>
    </form>
</body>
</html>
```

3. Run the application by pressing the F5 key. The output of the DetailsViewControlVB application is shown in Figure 21.4:


http://localhost:6842/DetailsViewControlVB/Default.aspx

DetailsView Control Example

CustomerID	ALFKI
CompanyName	Alfreds Futterkiste
ContactName	Maria Anders
1 2 3 4 5 6 7 8 9 10 ...	

Done

Figure 21.4: Output of the DetailsViewControlVB Application

NOTE

We have changed the format of the DetailsView control to Sand & Sky, by selecting the Auto Format option in the Smart Tag, and also enable the paging option.

Now, let's explore the FormView control.

The **FormView** Control

The FormView control displays a single record from the associated data source. Each row of the table displays each field of the record. For the FormView control to display content, you need to create templates for the different parts of the control. You must create a template for the mode in which the control is configured, even though most of the templates are optional. For example, a FormView control that supports inserting records must have an insert item template defined. The FormView control exists within the System.Web.UI.WebControl namespace. The inheritance hierarchy of the FormView class is as follows:

```
System.Object
   System.Web.UI.Control
      System.Web.UI.WebControls. ebControl
         System.Web.UI.WebControls.BaseDataboundControl
            System.Web.UI.WebControls.DataboundControl
               System.Web.UI.WebControls.CompositeDataboundControl
                  System.Web.UI.WebControls.FormView
```

To display the content through the FormView control, you need to create templates for different parts of the control. The mode of the control and template should be same. For example, a FormView control that supports editing records must have an EditItem template. Table 21.13 lists the templates that are supported by the FormView control:

Table 21.13: Templates for the FormView Control

Template Type	Description
EditItemTemplate	Describes the content for the data row when the FormView control is in edit mode. This template usually contains input controls and command buttons with which the user can edit an existing record.
EmptyDataTemplate	Describes the content for the empty data row displayed when the FormView control is bound to a data source that does not contain any records. This template usually contains content to alert the user that the data source does not contain any records.
FooterTemplate	Describes the content for the footer row. This template usually contains any additional content you would like to display in the footer row.
HeaderTemplate	Describes the content for the header row. This template usually contains any additional content you would like to display in the header row.
ItemTemplate	Describes the content for the data row when the FormView control is in read-only mode. This template usually contains content to display the values of an existing record.
InsertItemTemplate	Describes the content for the data row when the FormView control is in insert mode. This template usually contains input controls and command buttons with which the user can add a new record.
PagerTemplate	Describes the content for the pager row displayed when the paging feature is enabled, that is, when the AllowPaging property is set to true. This template usually contains controls with the help of which the user can navigate to another record.

NOTE

We have to manually include command buttons to update, delete, or insert operations, because the FormView control does not provide a way to automatically generate command buttons.

Noteworthy properties of the `FormView` class are listed in Table 21.14:

Table 21.14: Noteworthy Properties of the FormView Class

Property	Description
`AllowPaging`	Obtains or sets a value indicating whether the paging feature is enabled.
`BackImageUrl`	Obtains or sets the URL to an image to display in the background of a `FormView` control.
`BottomPagerRow`	Obtains the `FormViewRow` object that represents the pager row displayed at the bottom of the `FormView` control.
`Caption`	Obtains or sets the text to render in an HTML caption element in a `FormView` control. This property is provided to make the control more accessible to users of assistive technology devices.
`CaptionAlign`	Obtains or sets the horizontal or vertical position of the HTML caption element in a `FormView` control.
`CellPadding`	Obtains or sets the amount of space between the contents of a cell and the cell's border.
`CellSpacing`	Obtains or sets the amount of space between cells.
`CurrentMode`	Obtains the current data-entry mode of the `FormView` control.
`DataItem`	Obtains the data item bound to the `FormView` control.
`DataItemCount`	Obtains the number of data items in the data source.
`DataItemIndex`	Obtains the index of the data item bound to the `FormView` control from the data source.
`DataKey`	Obtains a `DataKey` object that represents the primary key of the displayed record.
`DataKeyNames`	Obtains or sets an array that contains the names of the key fields for the data source.
`DefaultMode`	Obtains or sets the data-entry mode to which the `FormView` control returns after an update, insert, or cancel operation.
`EditItemTemplate`	Obtains or sets the custom content for an item in edit mode.
`EditRowStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the data row when a `FormView` control is in edit mode.
`EmptyDataRowStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the empty data row displayed when the data source bound to a `FormView` control does not contain any records.
`EmptyDataTemplate`	Obtains or sets the user-defined content for the empty data row rendered when a `FormView` control is bound to a data source that does not contain any records.
`EmptyDataText`	Obtains or sets the text to display in the empty data row rendered when a `FormView` control is bound to a data source that does not contain any records.
`EnableModelValidation`	Obtains or sets a value that indicates whether a validator control will handle exceptions that occur during insert or update operations.
`FooterRow`	Obtains the `FormViewRow` object that represents the footer row in a `FormView` control.
`FooterStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the footer row in a `FormView` control.
`FooterTemplate`	Obtains or sets the user-defined content for the footer row in a `FormView` control.
`FooterText`	Obtains or sets the text to display in the footer row of a `FormView` control.
`GridLines`	Obtains or sets the gridline style for a `FormView` control.
`HeaderRow`	Obtains the `FormViewRow` object that represents the header row in a `FormView` control.

Table 21.14: Noteworthy Properties of the FormView Class

[illegible]	[illegible]
`HeaderStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the header row in a `FormView` control.
`HeaderTemplate`	Obtains or sets the user-defined content for the header row in a `FormView` control.
`HeaderText`	Obtains or sets the text to display in the header row of a `FormView` control.
`HorizontalAlign`	Obtains or sets the horizontal alignment of a `FormView` control on the page.
`InsertItemTemplate`	Obtains or sets the custom content for an item in insert mode.
`InsertRowStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the data row in a `FormView` control when the control is in insert mode.
`ItemTemplate`	Obtains or sets the custom content for the data row in a `FormView` control when the control is in read-only mode.
`PageCount`	Obtains the total number of pages required to display every record in the data source.
`PageIndex`	Obtains or sets the index of the displayed page.
`PagerSettings`	Obtains a reference to the `PagerSettings` object that allows you to set the properties of the pager buttons in a `FormView` control.
`PagerStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the pager row in a `FormView` control.
`PagerTemplate`	Obtains or sets the custom content for the pager row in a `FormView` control.
`Row`	Obtains the `FormViewRow` object that represents the data row in a `FormView` control.
`RowStyle`	Obtains a reference to the `TableItemStyle` object that allows you to set the appearance of the data row in a `FormView` control when the control is in read-only mode.
`SelectedValue`	Obtains the data key value of the current record in a `FormView` control.
`TopPagerRow`	Obtains the `FormViewRow` object that represents the pager row displayed at the top of a `FormView` control.

Noteworthy Methods of the `FormView` class are listed in Table 21.15:

Table 21.15: Noteworthy Methods of the FormView Class

[illegible]	[illegible]
`ChangeMode`	Use to switch the `DetailsView` control to the specified mode
`DataBind`	Use to bind data from the data source to the control
`DeleteItem`	Use to delete the current record from the data source
`InsertItem`	Use to insert the current record in the data source
`IsBindableType`	Use to determine whether the specified data type can be bound to a field in the `DetailsView` control
`UpdateItem`	Use to update the current record in the data source

Noteworthy events of the `FormView` class are listed in Table 21.16:

Table 21.16: Noteworthy Events of the FormView Class

[illegible]	[illegible]
`ItemCommand`	Invokes when a button within a `FormView` control is clicked.

Table 21.16: Noteworthy Events of the FormView Class

[illegible]	[illegible]
`ItemCreated`	Invokes after all the rows are created in a `FormView` control.
`ItemDeleted`	Invokes when a Delete button within a `FormView` control is clicked, but after the delete operation.
`ItemDeleting`	Invokes when a Delete button within a `FormView` control is clicked, but before the delete operation.
`ItemInserted`	Invokes when an Insert button within a `FormView` control is clicked, but after the insert operation.
`ItemInserting`	Invokes when an Insert button within a `FormView` control is clicked, but before the insert operation.
`ItemUpdated`	Invokes when an Update button within a `FormView` control is clicked, but after the update operation.
`ItemUpdating`	Invokes when an Update button within a `FormView` control is clicked, but before the update operation.
`ModeChanged`	Invokes when the `FormView` control switches between edit, insert, and read-only mode, but after the mode has changed.
`ModeChanging`	Invokes when the `FormView` control switches between edit, insert, and read-only mode, but before the mode changes.
`PageIndexChanged`	Invokes when the value of the `PageIndex` property changes after a paging operation.
`PageIndexChanging`	Invokes when the value of the `PageIndex` property changes before a paging operation.

Using the **FormView** Control

Now, let's see how a `FormView` control is used to display data from a database, by performing the following steps:

1. Create a Web application and save it as `FormViewControlVB`. You can find the code of `FormViewControlVB` application in the Code\ASP.NET\Chapter 21\`FormViewControlVB` folder on the CD.

NOTE

In this application, we have created a connection string with the Customers table of the Northwind database by using the SqlDataSource control.

2. Replace the code for the `Default.aspx` page with the code shown in Listing 21.4, to add the controls required for the application:

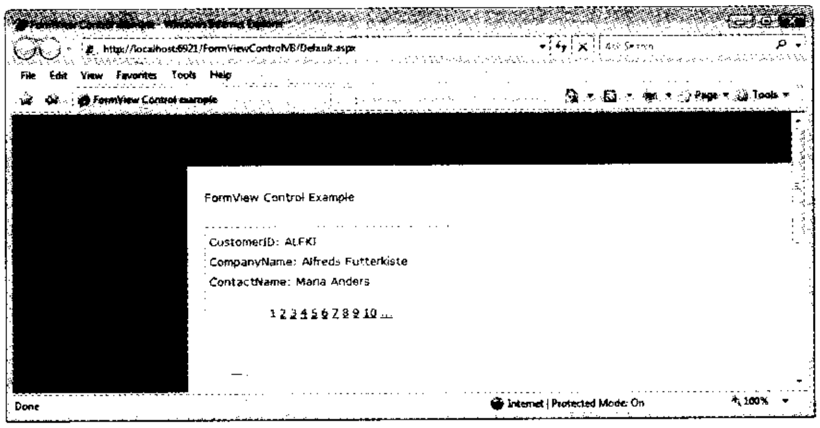
Listing 21.4: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>

<!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 id="Head1" runat="server">
  <title>FormView Control example</title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="Form1" runat="server">
```

```
        <div id="header">

        </div>
        <div id="sidebar">
            <div id="nav">
                &nbsp;
            </div>
        </div>
        <div id="content">
            <div class ="itemContent">
            <br />
            <asp:Label ID="Label1" runat="server" Text="FormView Control
            Example"></asp:Label><br />
            <br />
            <asp:FormView ID="FormView1" runat="server" AllowPaging="True"
            DataKeyNames="CustomerID"
            DataSourceID="SqlDataSource1" Width="292px" BackColor="White"
            BorderColor="#CC9966" BorderStyle="None" BorderWidth="1px"
            CellPadding="4"
            GridLines="Both">
            <FooterStyle BackColor="#FFFFCC" ForeColor="#330099" />
            <RowStyle BackColor="White" ForeColor="#330099" />
            <EditItemTemplate>
            CustomerID:
            <asp:Label ID="CustomerIDLabel1" runat="server"
            Text='<%# Eval("CustomerID") %>'></asp:Label><br />
            CompanyName:
            <asp:TextBox ID="CompanyNameTextBox" runat="server"
            Text='<%# Bind("CompanyName") %>'></asp:TextBox><br />
            ContactName:
            <asp:TextBox ID="ContactNameTextBox" runat="server"
            Text='<%# Bind("ContactName") %>'></asp:TextBox><br />
            <asp:LinkButton ID="UpdateButton" runat="server" CausesValidation="True"
            CommandName="Update"
            Text="Update"></asp:LinkButton>
            &nbsp;<asp:LinkButton ID="UpdateCancelButton" runat="server"
            CausesValidation="False" CommandName="Cancel"
            Text="Cancel"></asp:LinkButton>
            </EditItemTemplate>
            <InsertItemTemplate>
            CustomerID:
            <asp:TextBox ID="CustomerIDTextBox" runat="server"
            Text='<%# Bind("CustomerID") %>'></asp:TextBox><br />
            CompanyName:
            <asp:TextBox ID="CompanyNameTextBox" runat="server"
            Text='<%# Bind("CompanyName") %>'></asp:TextBox><br />
            ContactName:
            <asp:TextBox ID="ContactNameTextBox" runat="server"
            Text='<%# Bind("ContactName") %>'></asp:TextBox><br />
            <asp:LinkButton ID="InsertButton" runat="server" CausesValidation="True"
            CommandName="Insert"
            Text="Insert"></asp:LinkButton>
            &nbsp;<asp:LinkButton ID="InsertCancelButton" runat="server"
            CausesValidation="False" CommandName="Cancel"
            Text="Cancel"></asp:LinkButton>
            </InsertItemTemplate>
            <ItemTemplate>
            CustomerID:
            <asp:Label ID="CustomerIDLabel" runat="server" Text='<%#
            Eval("CustomerID") %>'></asp:Label><br />
```

```
                CompanyName:
                <asp:Label ID="CompanyNameLabel" runat="server"
                Text='<%# Bind("CompanyName") %>'></asp:Label><br />
                ContactName:
                <asp:Label ID="ContactNameLabel" runat="server"
                Text='<%# Bind("ContactName") %>'></asp:Label><br />
                </ItemTemplate>
                <PagerStyle BackColor="#FFFFCC" ForeColor="#330099"
                HorizontalAlign="Center" />
                <HeaderStyle BackColor="#990000" Font-Bold="True" ForeColor="#FFFFCC" />
                <EditRowStyle BackColor="#FFCC66" Font-Bold="True" ForeColor="#663399" />
                </asp:FormView>
                <asp:SqlDataSource ID="SqlDataSource1" runat="server"
                ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
                SelectCommand="SELECT [CustomerID], [CompanyName], [ContactName] FROM
                [Customers]">
                </asp:SqlDataSource>
                <br />
                <div id="footer">
                    <p class="left">
                    All content copyright &copy; Kogent Solutions Inc.</p>
                </div>
                </div>
            </div>
        </form>
    </body>
    </html>
```

3. Run the application by pressing the F5 key. The output of the FormViewControlVB application is shown in Figure 21.5:


http://localhost:6921/FormViewControlVB/Default.aspx
File Edit View Favorites Tools Help
FormView Control example
Page
Tools
Done
Internet | Protected Mode: On
100%

FormView Control Example

CustomerID: ALFKI
CompanyName: Alfreds Futterkiste
ContactName: Maria Anders

1 2 3 4 5 6 7 8 9 10 ...

Figure 21.5: Output of the FormViewControlVB Application

Now, let's learn about the ListView control.

The **ListView** Control

The ListView control is a data bound control used to display data from the associated data source. The ListView control enables a developer to display data in any format using templates and styles. This control provides excellent customization and extensibility features. It provides the developer with control on the rendered HTML output and provides support for adding new row, sorting, and the rest. When you drag the ListView control to the ASP.NET page, the following code appears in the HTML code of the page:

```
<asp:ListView ID="ListView1" runat="server"> </asp:ListView>
```

The inheritance hierarchy of the `ListView` control is as follows:

```
System.Object
   System.Web.UI.Control
      System.Web.UI.WebControls.WebControl
         System.Web.UI.WebControls.BaseDataboundControl
           System.Web.UI.WebControls.DataboundControl
              System.Web.UI.WebControls.ListView
```

Noteworthy properties of the `ListView` class are listed in Table 21.17:

Table 21.17: Noteworthy Properties of the ListView Class

[illegible]	[illegible]
`AccessKey`	Overrides the `WebContro.AccessKey` property. Setting this property is not supported by the `ListView` control.
`AlternatingItemTemplate`	Obtains or sets the custom content for the alternating data item in a `ListView` control.
`Controls`	Obtains a `ControlCollection` object that represents the child controls of the ListView control.
`ConvertEmptyStringToNull`	Obtains or sets a value that indicates whether empty string values (" ") are automatically converted to null values when the data field is updated in the data source.
`CssClass`	Overrides the `WebControl.CssClass` property. Setting this property is not supported by the `ListView` control.
`DataKeyNames`	Obtains or sets an array that contains the names of the primary key fields for the items displayed in a `ListView` control.
`DataKeys`	Obtains a collection of `DataKey` objects that represent the data-key value for each item in a `ListView` control.
`EditIndex`	Obtains or sets the index of the item being edited.
`EditItem`	Obtains the item that is in edit mode in a `ListView` control.
`EditItemTemplate`	Obtains or sets the custom content for the item in edit mode.
`EmptyDataTemplate`	Obtains or sets the user-defined content for the empty template that is rendered when a `ListView` control is bound to a data source that does not contain any records.
`EmptyItemTemplate`	Obtains or sets the user-defined content for the empty item that is rendered in a `ListView` control when there are no more data items to display in the last row of the current data page.
`Font`	Overrides the `WebControl.Font` property. This property is not supported by the `ListView` control.
`ForeColor`	Overrides the `WebControl.ForeColor` property. Setting this property is not supported by the `ListView` control.
`GroupItemCount`	Obtains or sets the number of items to display per group in a `ListView` control.
`GroupPlaceholderID`	Obtains or sets the ID for the group placeholder in a `ListView` control.
`GroupSeparatorTemplate`	Obtains or sets the user-defined content for the separator between groups in a `ListView` control.
`GroupTemplate`	Obtains or sets the user-defined content for the group container in a `ListView` control.
`Height`	Overrides the `WebControl.Height` property. Setting this property is not supported by the `ListView` control.
`InsertItem`	Obtains the insert item of a `ListView` control.

Table 21.17: Noteworthy Properties of the ListView Class

Property	Description
`InsertItemPosition`	Obtains or sets the location of the `InsertItemTemplate` template when it is rendered as part of the `ListView` control.
`InsertItemTemplate`	Obtains or sets the custom content for an insert item in the `ListView` control.
`ItemPlaceholderID`	Obtains or sets the ID for the item placeholder in a `ListView` control.
`Items`	Obtains a collection of `ListViewDataItem` objects that represent the data items of the current page of data in a `ListView` control.
`ItemSeparatorTemplate`	Obtains or sets the custom content for the separator between the items in a `ListView` control.
`ItemTemplate`	Obtains or setsr the custom content for the data item in a `ListView` control.
`LayoutTemplate`	Obtains or sets the custom content for the root container in a `ListView` control.
`SelectedDataKey`	Obtains the data-key value for the selected item in a `ListView` control.
`SelectedIndex`	Obtains or sets the index of the selected item in a `ListView` control.
`SelectedItemTemplate`	Obtains or sets the custom content for the selected item in a `ListView` control.
`SelectedValue`	Obtains the data-key value of the selected item in a `ListView` control.
`SortDirection`	Obtains the sort direction of the field or fields that are being sorted.
`SortExpression`	Obtains the sort expression that is associated with the field or fields that are being sorted.

Noteworthy methods of the `ListView` class are listed in Table 21.18:

Table 21.18: Noteworthy Methods of the ListView Class

Method	Description
`DeleteItem`	Deletes the record at the specified index from the data source
`ExtractItemValues`	Retrieves the values of each field that is declared in the specified item, and stores them in the specified `IOrderedDictionary` object
`InsertNewItem`	Inserts the current record in the data source
`Sort`	Sorts the `ListView` control, depending on the specified sort expression and direction
`UpdateItem`	Updates the record at the specified index in the data source

Noteworthy events of the `ListView` class are listed in Table 21.19:

Table 21.19: Noteworthy Events of the ListView Class

Event	Description
`ItemCanceling`	Invoked when a cancel operation is requested, but before the `ListView` control cancels the insert or edit operation
`ItemCommand`	Invoked when a button in a `ListView` control is clicked
`ItemCreated`	Invoked when an item is created in a `ListView` control
`ItemDatabound`	Invoked when a data item is bound to the data in a `ListView` control
`ItemDeleted`	Invoked when a delete operation is requested, after the `ListView` control deletes the item

Table 21.19: Noteworthy Events of the ListView Class

[illegible]	Description
`ItemDeleting`	Invoked when a delete operation is requested, before the `ListView` control deletes the item
`ItemEditing`	Invoked when an edit operation is requested, but before the `ListView` item is put in edit mode
`ItemInserted`	Invoked when an insert operation is requested, after the `ListView` control has inserted the item in the data source
`ItemInserting`	Invoked when an insert operation is requested, before the `ListView` control performs the insert operation
`ItemUpdated`	Invoked when an update operation is requested, after the `ListView` control updates the item
`ItemUpdating`	Invoked when an update operation is requested, before the `ListView` control updates the item
`LayoutCreated`	Invoked when the `LayoutTemplate` template is created in a `ListView` control.
`PagePropertiesChanged`	Invoked when the page properties change, after the `ListView` control sets the new values
`PagePropertiesChanging`	Invoked when the page properties change, but before the `ListView` control sets the new values
`SelectedIndexChanged`	Invoked when an item's Select button is clicked, after the `ListView` control handles the select operation
`SelectedIndexChanging`	Invoked when an item's Select button is clicked, but before the ListView control handles the select operation
`Sorted`	Invoked when a sort operation is requested, after the `ListView` control handles the sort operation
`Sorting`	Invoked when a sort operation is requested, but before the `ListView` control handles the sort operation

Table 21.20 lists the templates supported by the `ListView` class:

Table 21.20: Types of Templates of the ListView Class

Template Type	Description
`LayoutTemplate`	Defines a container object, such as a table row (tr), div, or span element. These objects will contain the content defined in the `ItemTemplate` or `GroupTemplate` template. It might also contain a `DataPager` object. It is used to define a Custom UI for the root container of the `Listview` control.
`ItemTemplate`	Describes the data bound content to display for individual items.
`ItemSeparatorTemplate`	Describes the content to render between individual items.
`GroupTemplate`	Describes a container object, such as a table cell (td), div, or span element, that contains the content defined in the `ItemTemplate` and `EmptyItemTemplate` templates. The number of items that are displayed in a group is specified by the `GroupItemCount` property. It is used to create a tiled layout in the `ListView` control.
`GroupSeparatorTemplate`	Describes the content to render between groups of items.
`EmptyItemTemplate`	Describes the content to render for an empty item when a `GroupTemplate` template is used. For example, if the `GroupItemCount` property is set to 5, and the total

Table 21.20: Types of Templates of the ListView Class

Template Type	Description
	number of items returned from the data source is 8, the last group of data displayed by the `ListView` control will contain three items as specified by the `ItemTemplate` template, and two items as specified by the `EmptyItemTemplate` template.
`EmptyDataTemplate`	Describes the content to render if the data source returns no data.
`SelectedItemTemplate`	Describes the content to render for the selected data item to differentiate the selected item from the other items.
`AlternatingItemTemplate`	Describes the content to render for alternating items to make it easier to distinguish between consecutive items.
`EditItemTemplate`	Describes the content to render when an item is being edited. The `EditItemTemplate` template is rendered in place of the `ItemTemplate` template for the data item that is edited.
`InsertItemTemplate`	Describes the content to render inserting an item. The `InsertItemTemplate` template is rendered in place of an `ItemTemplate` template at either the start or the end of the items that are displayed by the `ListView` control. You can specify where the `InsertItemTemplate` template is rendered using the `InsertItemPosition` property of the `ListView` control.

Now, let's explore the `Repeater` control.

The **Repeater** Control

The `Repeater` control is a data bound control that is used to display repeated list of items from the associated data source. It is a single Webcontrol that allows splitting markup tags across the templates. The `Repeater` control does not provide support to in-built layout or styles. Therefore, while working with the `Repeater` control, you have to explicitly declare all layouts, formatting, and style. In other words, the `Repeater` control does not support built-in selection or editing features.

NOTE

The basic difference between the Repeater control and DataList control is that the latter explicitly places items in an HTML table but the former does not explicitly places items in an HTML table.

The `Repeater` control exists within the `System.Web.UI.WebControls` namespace. The inheritance hierarchy of the `Repeater` control is as follows:

```
System.Object
  System.Web.UI.Control
    System.Web.UI.WebControls.Repeater
```

Noteworthy properties of the `Repeater` class are listed in Table 21.21:

Table 21.21: Noteworthy Properties of the Repeater Class

Property	Description
`AlternatingItemTemplate`	Obtains or sets the object implementing the `System.Web.UI.ITemplate` namespace that defines how alternating items in the control are displayed.
`AppRelativeTemplateSourceDirectory`	Gets or sets the application-relative virtual directory of the Page or UserControl object that contains this control. (Inherited from Control.)
`BindingContainer`	Infrastructure. Gets the control that contains this control's data binding. (Inherited from Control.)
`ChildControlsCreated`	Gets a value that indicates whether the server control's child controls have been created. (Inherited from Control.)

Table 21.21: Noteworthy Properties of the Repeater Class

`ClientID`	Gets the server control identifier generated by ASP.NET. (Inherited from Control.)
`ClientIDSeparator`	Gets a character value representing the separator character used in the ClientID property. (Inherited from Control.)
`Context`	Gets the HttpContext object associated with the server control for the current Web request. (Inherited from Control.)
`Controls`	Obtains a `System.Web.UI.ControlCollection` namespace that contains the child controls of the `Repeater` control
`DataMember`	Obtains or sets the specific table in the `DataSource` to bind to the `Repeater` control
`DataSource`	Obtains or sets the data source that provides data for populating the list
`DataSourceID`	Obtains or sets the `ID` property of the data source control that the `Repeater` control should use to retrieve its data source
`DesignMode`	Gets a value indicating whether a control is being used on a design surface. (Inherited from Control.)
`EnableTheming`	Obtains or sets a value indicating whether the themes are applied to this control.
`EnableViewState`	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client. (Inherited from Control.)
`Events`	Gets a list of event handler delegates for the control. This property is read-only. (Inherited from Control.)
`FooterTemplate`	Obtains or sets the `System.Web.UI.ITemplate` namespace that defines how the footer section of the Repeater control will be displayed
`HasChildViewState`	Gets a value indicating whether the current server control's child controls have any saved view-state settings. (Inherited from Control.)
`HeaderTemplate`	Obtains or sets the `System.Web.UI.ITemplate` that defines how the header section of the `Repeater` control will be displayed
`ID`	Gets or sets the programmatic identifier assigned to the server control. (Inherited from Control.)
`IdSeparator`	Infrastructure. Gets the character used to separate control identifiers. (Inherited from Control.)
`Initialized`	Infrastructure. Returns a value indicating whether the control has been initialized.
`IsBoundUsingDataSourceID`	Infrastructure. Gets a value indicating whether the DataSourceID property is set.
`IsChildControlStateClea red`	Gets a value indicating whether controls contained within this control have control state. (Inherited from Control.)
`IsTrackingViewState`	Gets a value that indicates whether the server control is saving changes to its view state. (Inherited from Control.)
`IsViewStateEnabled`	Gets a value indicating whether view state is enabled for this control. (Inherited from Control.)
`Items`	Obtains a collection of the `RepeaterItem` objects in the `Repeater` control
`ItemTemplate`	Obtains or sets the `System.Web.UI.ITemplate` namespace that defines how items in the `Repeater` control will be displayed

Table 21.21: Noteworthy Properties of the Repeater Class

[illegible]	[illegible]
`LoadViewStateByID`	Gets a value indicating whether the control participates in loading its view state by ID instead of index. (Inherited from Control.)
`Page`	Gets a reference to the Page instance that contains the server control. (Inherited from Control.)
`Parent`	Gets a reference to the server control's parent control in the page control hierarchy. (Inherited from Control.)
`RequiresDataBinding`	Gets or sets a value indicating whether the Repeater control needs to bind to its specified data source.
`SelectArguments`	Infrastructure. Gets a DataSourceSelectArguments object that the Repeater control uses when retrieving data from a data source control.
`SeparatorTemplate`	Obtains or sets the `System.Web.UI.ITemplate` interface that defines how the separator between the items is displayed

Noteworthy events of the `Repeater` class are listed in Table 21.22:

Table 21.22: Noteworthy Events of the Repeater Class

[illegible]	[illegible]
`ItemCommand`	Invoked when a button is clicked in the `Repeater` control
`ItemCreated`	Invoked when an item is created in the `Repeater` control
`ItemDatabound`	Invoked after an item in the `Repeater` control is data bound but before it is rendered on the page

Using the **Repeater** Control

Now, let's use a `Repeater` control to display data from a database by performing the following steps:

1. Create a Web application and save it as `RepeaterVB`. You can find the code of `RepeaterVB` application in the Code\ASP.NET\Chapter 21\`RepeaterVB` folder on the CD.

NOTE

In this application, we have created a connection string with the Customers table of the Northwind database using the SqlDataSource.

2. Replace the code for the `Default.aspx` page with the code shown in Listing 21.5, to add the required controls for the application:

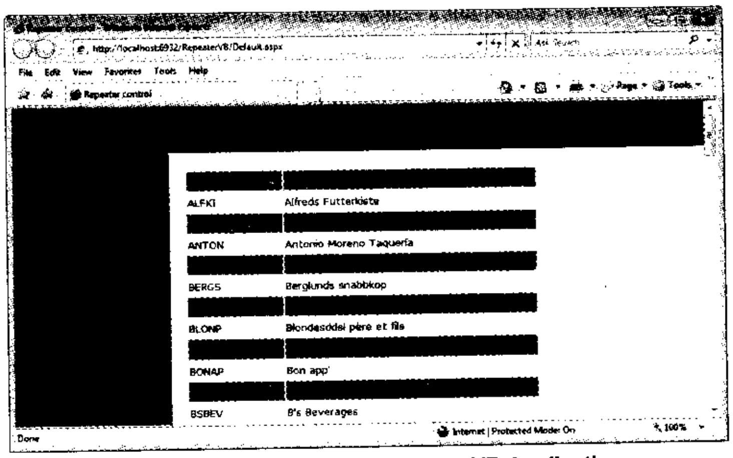
Listing 21.5: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!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 id="Head1" runat="server">
  <title>Repeater control </title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="Form" runat="server">
      <div id="header">
      </div>
```

```
<div id="sidebar">
    <div id="nav">
        &nbsp;
    </div>
</div>
<div id="content">
    <div class ="itemContent">
    <br />
    <table width="70%">
    <asp:Repeater ID="Repeater1" runat="server"
    DataSourceID="SqlDataSource1">
    <HeaderTemplate>
    <tr style="background-color: Gray">
    <th>
        CustomerID</th>
    <th>
        Name of Company </th>
    </tr>
    </HeaderTemplate>
    <ItemTemplate>
    <tr>
        <td>
        <%# DataBinder.Eval(Container, "DataItem.CustomerID") %>
        </td>
        <td>
        <%# DataBinder.Eval(Container,"DataItem.CompanyName") %>
        </td>
    </tr>
    </ItemTemplate>
    <AlternatingItemTemplate>
    <tr>
        <td style="background-color:Gray">
        <%# DataBinder.Eval(Container, "DataItem.CustomerID")%>
        </td>
        <td style="background-color:Gray">
        <%# DataBinder.Eval(Container, "DataItem.CompanyName")%>
        </td>
    </tr>
    </AlternatingItemTemplate>
    <FooterTemplate>
    <tr style="background-color: Gray">
    <th>
        Customer ID</th>
    <th>
        Company Name </th>
    </tr>
    </FooterTemplate>
    </asp:Repeater>
    <asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
    SelectCommand="SELECT [CustomerID], [CompanyName] FROM [Customers]">
    </asp:SqlDataSource>
    </table>
    <br />
    <div id="footer">
    <p class="left">
        All content copyright &copy; Kogent Solutions Inc.</p>
    </div>
    </div>
```

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

3. Run the application by pressing the F5 key. The output of the RepeaterVB application is shown in Figure 21.6:



```
http://localhost:6932/RepeaterVB/Default.aspx

ALFKI    Alfreds Futterkiste
ANTON    Antonio Moreno Taquería
BERGS    Berglunds snabbkop
BLONP    Blondesddsl père et fils
BONAP    Bon app'
BSBEV    B's Beverages
```

Figure 21.6: Output of the RepeaterVB Application

Now, let's learn about the DataPager control.

The **DataPager** Control

The DataPager control is used to provide paging functionality to the data bound controls, such as the ListView control. When you drag a DataPager control to the page, the following code appears in the HTML code of the page:

```
<asp:DataPager ID="DataPager1" runat="server"> </asp:DataPager>
```

The inheritance hierarchy of the DataPager control is as follows:

```
System.Object
   System.Web.UI.Control
      System.Web.UI.WebControls.DataPager
```

Noteworthy properties of the DataPager class are listed in Table 21.23:

Table 21.23: Noteworthy Properties of the DataPager Class

Property	Description
Attributes	Gets a collection of custom attribute name/value pairs for the DataPager control.
BindingContainer	Infrastructure. Gets the control that contains this control's data binding. (Inherited from Control.)
ChildControlsCreated	Gets a value that indicates whether the server control's child controls have been created. (Inherited from Control.)
ClientID	Gets the server control identifier generated by ASP.NET. (Inherited from Control.)
ClientIDSeparator	Gets a character value representing the separator character used in the ClientID property. (Inherited from Control.)
Context	Gets the HttpContext object associated with the server control for the current Web request. (Inherited from Control.)
Controls	Obtains a ControlCollection object that represents the child controls for the DataPager control in the UI hierarchy.

Table 21.23: Noteworthy Properties of the DataPager Class

[illegible]	[illegible]
`DesignMode`	Gets a value indicating whether a control is being used on a design surface. (Inherited from Control.)
`EnableTheming`	Gets or sets a value indicating whether themes apply to this control. (Inherited from Control.)
`EnableViewState`	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client. (Inherited from Control.)
`Events`	Gets a list of event handler delegates for the control. This property is read-only. (Inherited from Control.)
`Fields`	Obtains a collection of `DataPagerField` objects that represent the pager fields that are specified in a `DataPager` control.
`HasChildViewState`	Gets a value indicating whether the current server control's child controls have any saved view-state settings. (Inherited from Control.)
`ID`	Gets or sets the programmatic identifier assigned to the server control. (Inherited from Control.)
`IdSeparator`	Infrastructure. Gets the character used to separate control identifiers. (Inherited from Control.)
`IsChildControlStateCleared`	Gets a value indicating whether controls contained within this control have control state. (Inherited from Control.)
`IsTrackingViewState`	Gets a value that indicates whether the server control is saving changes to its view state. (Inherited from Control.)
`IsViewStateEnabled`	Gets a value indicating whether view state is enabled for this control. (Inherited from Control.)
`LoadViewStateByID`	Gets a value indicating whether the control participates in loading its view state by ID instead of index. (Inherited from Control.)
`MaximumRows`	Obtains the maximum number of records that are displayed for each page of data.
`Page`	Gets a reference to the Page instance that contains the server control. (Inherited from Control.)
`PagedControlID`	Obtains or sets the ID of the control that contains the data that will be paged by the `DataPager` control.
`PageSize`	Obtains or sets the number of records that are displayed for each page of data.
`Parent`	Gets a reference to the server control's parent control in the page control hierarchy. (Inherited from Control.)
`QueryStringField`	Obtains or sets the name of the query string field.
`Site`	Gets information about the container that hosts the current control when rendered on a design surface. (Inherited from Control.)
`SkinID`	Gets or sets the skin to apply to the control. (Inherited from Control.)
`StartRowIndex`	Obtains the index of the first record that is displayed on a page of data.
`TagKey`	Gets the HTML element that is used to render the DataPager control.
`TemplateControl`	Gets or sets a reference to the template that contains this control. (Inherited from Control.)

Table 21.23: Noteworthy Properties of the DataPager Class

[illegible]	[illegible]
`TemplateSourceDirectory`	Gets the virtual directory of the Page or UserControl that contains the current server control. (Inherited from Control.)
`TotalRowCount`	Obtains the total number of records that are retrieved by the underlying data source object. This data source object is referenced by the associated data bound control.

Noteworthy methods of the `DataPager` class are listed in Table 21.24:

Table 21.24: Noteworthy Methods of the DataPager Class

[illegible]	[illegible]
`DataBind`	Use to bind a `DataPager` control and all its child controls to a data source.
`RenderBeginTag`	Use to render the HTML opening tag of the `DataPager` control to the specified writer.
`SetPageProperties`	Use to set the page-related properties in the `DataPager` control.

Using the **ListView** and **DataPager** Controls

Let's learn how to use the `ListView` and `DataPager` controls to display a set of records, by performing the following steps:

1. Create an application and save it as `ListViewControlVB`. You can find the code of `ListViewControlVB` application in the Code\ASP.NET\Chapter 21\`ListViewControlVB` folder on the CD.
2. Replace the code for the `Default.aspx` page with the code shown in Listing 21.6, to add the controls required for the application:

Listing 21.6: Showing the Code for the `Default.aspx` page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!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 id="Head1" runat="server">
  <title>ListView and DataPager Control </title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="Form" runat="server">
      <div id="header">

      </div>
      <div id="sidebar">
          <div id="nav">
              &nbsp;
          </div>
      </div>
      <div id="content">
          <div class ="itemContent">
          ListView and DataPager Control
          <br />
          <br />
          <asp:ListView ID="ListView1" runat="server" DataKeyNames="CustomerID"
          DataSourceID="SqlDataSource1" GroupItemCount="3">
          <EmptyItemTemplate>
          <td id="Td1" runat="server" />
          </EmptyItemTemplate>
```

```
<ItemTemplate>
<td id="Td2" runat="server" style="background-color:#DCDCDC;color:
#000000;">
CustomerID:
<asp:Label ID="CustomerIDLabel" runat="server"
Text='<%# Eval("CustomerID") %>' />
<br />
CompanyName:
<asp:Label ID="CompanyNameLabel" runat="server"
Text='<%# Eval("CompanyName") %>' />
<br />
ContactName:
<asp:Label ID="ContactNameLabel" runat="server"
Text='<%# Eval("ContactName") %>' />
<br />
</td>
</ItemTemplate>
<AlternatingItemTemplate>
<td id="Td3" runat="server" style="background-color:#FFF8DC;">
CustomerID:
<asp:Label ID="CustomerIDLabel" runat="server"
Text='<%# Eval("CustomerID") %>' />
<br />
CompanyName:
<asp:Label ID="CompanyNameLabel" runat="server"
Text='<%# Eval("CompanyName") %>' />
<br />
ContactName:
<asp:Label ID="ContactNameLabel" runat="server"
Text='<%# Eval("ContactName") %>' />
<br />
</td>
</AlternatingItemTemplate>
<EmptyDataTemplate>
<table id="Table1" runat="server"
style="background-color: #FFFFFF;border-collapse: collapse;border-color:
#999999;border-style:none;border-width:1px;">
<tr>
    <td>
    No data was returned.</td>
</tr>
</table>
</EmptyDataTemplate>
<InsertItemTemplate>
<td id="Td4" runat="server" style="">
CustomerID:
<asp:TextBox ID="CustomerIDTextBox" runat="server"
Text='<%# Bind("CustomerID") %>' />
<br />
CompanyName:
<asp:TextBox ID="CompanyNameTextBox" runat="server"
Text='<%# Bind("CompanyName") %>' />
<br />
ContactName:
<asp:TextBox ID="ContactNameTextBox" runat="server"
Text='<%# Bind("ContactName") %>' />
<br />
<asp:Button ID="InsertButton" runat="server" CommandName="Insert"
Text="Insert" />
```

```
<br />
<asp:Button ID="CancelButton" runat="server" CommandName="Cancel"
Text="Clear" />
<br />
</td>
</InsertItemTemplate>
<LayoutTemplate>
<table id="Table2" runat="server">
<tr id="Tr1" runat="server">
    <td id="Td5" runat="server">
    <table ID="groupPlaceholderContainer" runat="server" border="1"
    style="background-color: #FFFFFF;border-collapse:
    collapse;border-color: #999999;border-style:none;border-
    width:1px;font-family: Verdana, Arial, Helvetica, sans-serif;">
    <tr ID="groupPlaceholder" runat="server">
    </tr>
    </table>
    </td>
</tr>
<tr id="Tr2" runat="server">
    <td id="Td6" runat="server"
    style="text-align: center;background-color: #CCCCCC;font-family:
    Verdana, Arial, Helvetica, sans-serif;color: #000000;">
    <asp:DataPager ID="DataPager1" runat="server" PageSize="12">
    <Fields>
    <asp:NextPreviousPagerField ButtonType="Button"
    ShowFirstPageButton="True"
    ShowLastPageButton="True" />
    </Fields>
    </asp:DataPager>
    </td>
</tr>
</table>
</LayoutTemplate>
<EditItemTemplate>
<td id="Td7" runat="server" style="background-color:#008A8C;color:
#FFFFFF;">
CustomerID:
<asp:Label ID="CustomerIDLabel1" runat="server"
Text='<%# Eval("CustomerID") %>' />
<br />
CompanyName:
<asp:TextBox ID="CompanyNameTextBox" runat="server"
Text='<%# Bind("CompanyName") %>' />
<br />
ContactName:
<asp:TextBox ID="ContactNameTextBox" runat="server"
Text='<%# Bind("ContactName") %>' />
<br />
<asp:Button ID="UpdateButton" runat="server" CommandName="Update"
Text="Update" />
<br />
<asp:Button ID="CancelButton" runat="server" CommandName="Cancel"
Text="Cancel" />
<br />
</td>
</EditItemTemplate>
<GroupTemplate>
<tr ID="itemPlaceholderContainer" runat="server">
    <td ID="itemPlaceholder" runat="server">
```

```
                    </td>
                </tr>
                </GroupTemplate>
                <SelectedItemTemplate>
                <td id="Td8" runat="server"
                style="background-color:#008A8C;font-weight: bold;color: #FFFFFF;">
                CustomerID:
                <asp:Label ID="CustomerIDLabel" runat="server"
                Text='<%# Eval("CustomerID") %>' />
                <br />
                CompanyName:
                <asp:Label ID="CompanyNameLabel" runat="server"
                Text='<%# Eval("CompanyName") %>' />
                <br />
                ContactName:
                <asp:Label ID="ContactNameLabel" runat="server"
                Text='<%# Eval("ContactName") %>' />
                <br />
                </td>
                </SelectedItemTemplate>
                </asp:ListView>
                <asp:SqlDataSource ID="SqlDataSource1" runat="server"
                ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
                SelectCommand="SELECT [CustomerID], [CompanyName], [ContactName] FROM
                [Customers]">
                </asp:SqlDataSource>
                <div id="footer">
                    <p class="left">
                        All content copyright &copy; Kogent Solutions Inc.</p>
                </div>
                </div>
            </div>
        </form>
    </body>
    </html>
```

In the preceding listing, the highlighted code shows the settings for the `DataPager` control, which you are using with the `ListView` control:

3. To configure a `ListView` control, click the Show Smart Tag of the `ListView` control and then select the Configure ListView option from the Smart Tag, as shown in Figure 21.7:

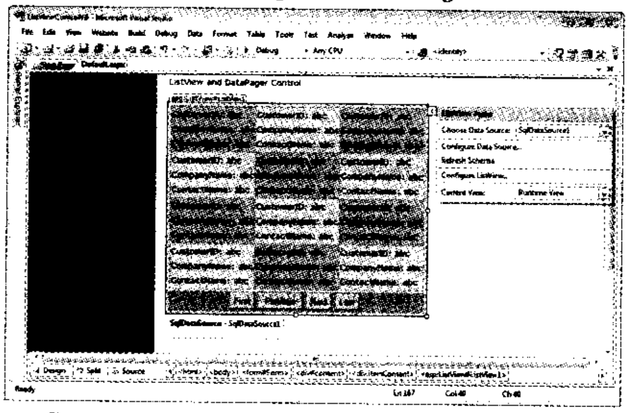
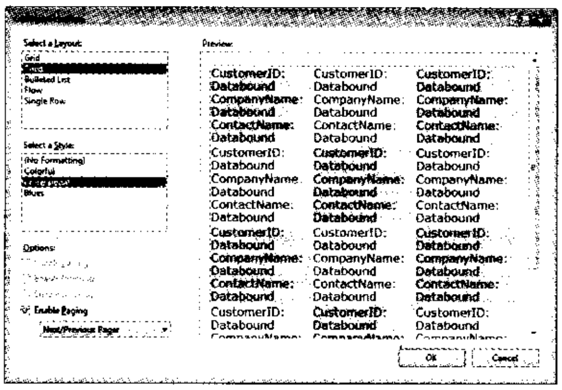

ListView and DataPager Control
ListView Tasks
Choose Data Source: SqlDataSource1
Configure Data Source...
Refresh Schema
Configure ListView...
Current View: Runtime View
SqlDataSource - SqlDataSource1

Figure 21.7: Selecting the Configure ListView Option

The Configure ListView dialog box opens.

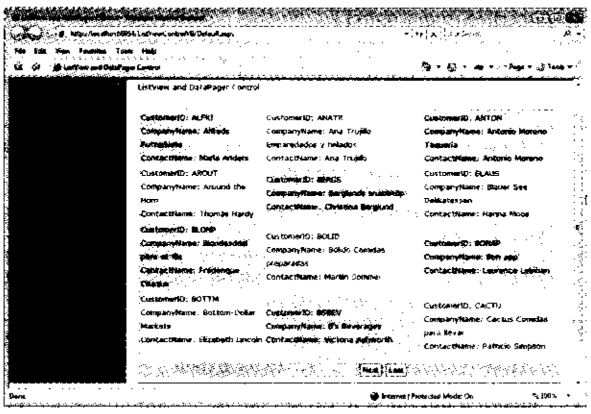
4. You can select different layouts and styles for the `ListView` control in the Configure ListView dialog box, as shown in Figure 21.8:



```
Select a Layout:        Preview:
Grid                    CustomerID:     CustomerID:     CustomerID:
Tiled                   Databound       Databound       Databound
Bulleted List           CompanyName:    CompanyName:    CompanyName:
Flow                    Databound       Databound       Databound
Single Row              ContactName:    ContactName:    ContactName:
                        Databound       Databound       Databound
Select a Style:         CustomerID:     CustomerID:     CustomerID:
(No Formatting)         Databound       Databound       Databound
Colorful                CompanyName:    CompanyName:    CompanyName:
Professional            Databound       Databound       Databound
Blues                   ContactName:    ContactName:    ContactName:
                        Databound       Databound       Databound
Options:                CustomerID:     CustomerID:     CustomerID:
                        Databound       Databound       Databound
                        CompanyName:    CompanyName:    CompanyName:
                        Databound       Databound       Databound
                        ContactName:    ContactName:    ContactName:
                        Databound       Databound       Databound
☑ Enable Paging         CustomerID:     CustomerID:     CustomerID:
  Next/Previous Pager   Databound       Databound       Databound

                                                 OK      Cancel
```

Figure 21.8: Configuring the ListView Control

5. Now, run the application by pressing the `F5` key. The output of the `ListViewControlVB` application is shown in Figure 21.9:



```
ListView and DataPager Control

CustomerID: ALFKI                 CustomerID: ANATR                 CustomerID: ANTON
CompanyName: Alfreds              CompanyName: Ana Trujillo         CompanyName: Antonio Moreno
Futterkiste                       Emparedados y helados             Taquería
ContactName: Maria Anders         ContactName: Ana Trujillo         ContactName: Antonio Moreno
CustomerID: AROUT                 CustomerID: BERGS                 CustomerID: BLAUS
CompanyName: Around the           CompanyName: Berglunds snabbköp   CompanyName: Blauer See
Horn                              ContactName: Christina Berglund   Delikatessen
ContactName: Thomas Hardy                                           ContactName: Hanna Moos
CustomerID: BLONP                 CustomerID: BOLID                 CustomerID: BONAP
CompanyName: Blondesddsl          CompanyName: Bólido Comidas       CompanyName: Bon app'
père et fils                      preparadas                        ContactName: Laurence Lebihan
ContactName: Frédérique           ContactName: Martín Sommer
Citeaux
CustomerID: BOTTM                 CustomerID: BSBEV                 CustomerID: CACTU
CompanyName: Bottom-Dollar        CompanyName: B's Beverages        CompanyName: Cactus Comidas
Markets                           ContactName: Victoria Ashworth    para llevar
ContactName: Elizabeth Lincoln                                      ContactName: Patricio Simpson

                                  Next  Last
```

Figure 21.9: Output of the ListViewControlVB Application

Now, let's learn about the `SqlDataSource` control.

The **SqlDataSource** Control

The `SqlDataSource` control is a data source control that allows a server control, such as the `GridView` control, to access data located in a relational database, such as Microsoft SQL Server and Oracle. This control also generates a connection string to interact with data in an ASP.NET page. These connection strings are generated through the `Configure Data Source` wizard. A connection string contains information about the server, database, and security you are working with in your application, for example:

```
Data Source=.\sqlexpress;Initial Catalog=Northwind;Integrated Security=True
```

The `SqlDataSource` control automatically opens a database connection, which is provided in the connection string, executes the SQL queries, and then closes the connection. When you add a `SqlDataSource` control to an ASP.NET page, the following code appears in the HTML code of a page:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"></asp:SqlDataSource>
```

The inheritance hierarchy of the `SqlDataSource` control is as follows:

```
System.Object
  System.Web.UI.Control
     System.Web.UI.DataSourceControl
        System.Web.UI.WebControls.SqlDataSource
```

Noteworthy properties of the `SqlDataSource` class are listed in Table 21.25:

Table 21.25: Noteworthy Properties of the SqlDataSource Class

Property	Description
`CacheDuration`	Obtains or sets the length of time, in seconds, for which the data source control caches data that is retrieved by the `Select` method.
`CacheExpirationPolicy`	Obtains or sets the cache expiration behavior that, when combined with the duration, describes the behavior of the cache that the data source control uses.
`CacheKeyDependency`	Obtains or sets a user-defined key dependency that is linked to all data cache objects that are created by the data source control. All cache objects explicitly expire when the key expires.
`CancelSelectOnNullParameter`	Obtains or sets a value indicating whether a data retrieval operation is canceled when any parameter that is contained in the `SelectParameters` collection evaluates to null reference
`ChildControlsCreated`	Gets a value that indicates whether the server control's child controls have been created. (Inherited from Control.)
`ClientID`	Gets the server control identifier generated by ASP.NET. (Inherited from DataSourceControl.)
`ClientIDSeparator`	Gets a character value representing the separator character used in the ClientID property. (Inherited from Control.)
`ConflictDetection`	Obtains or sets the value indicating how the `SqlDataSource` control performs, updates and deletes when data in a row in the underlying database changes during the time of the operation.
`ConnectionString`	Obtains or sets the ADO.NET provider-specific connection string that the `SqlDataSource` control uses to connect to an underlying database.
`Context`	Gets the HttpContext object associated with the server control for the current Web request. (Inherited from Control.)
`Controls`	Gets a ControlCollection object that represents the child controls for a specified server control in the UI hierarchy. (Inherited from DataSourceControl.)
`DataSourceMode`	Obtains or sets the data retrieval mode such as DataSet, DataReader that the `SqlDataSource` control uses to fetch data.
`DeleteCommand`	Obtains or sets the SQL string that the `SqlDataSource` control uses to delete the data from the underlying database.
`DeleteCommandType`	Obtains or sets a value indicating whether the text in the `DeleteCommand` property is an SQL statement or is the name of a stored procedure.
`DeleteParameters`	Obtains the parameters collection that contains the parameters that are used by the `DeleteCommand` property from the `SqlDataSourceView` object that is associated with the `SqlDataSource` control.
`DesignMode`	Gets a value indicating whether a control is being used on a design surface. (Inherited from Control.)
`EnableCaching`	Obtains or sets a value indicating whether the `SqlDataSource` control has data caching enabled.
`EnableTheming`	Gets a value indicating whether this control supports themes. (Inherited from DataSourceControl.)
`EnableViewState`	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client. (Inherited from Control.)

Table 21.25: Noteworthy Properties of the SqlDataSource Class

Property	Description
`Events`	Gets a list of event handler delegates for the control. This property is read-only. (Inherited from Control.)
`FilterExpression`	Obtains or sets a filtering expression that is applied when the `Select`Method property of data source is called.
`FilterParameters`	Obtains a collection of parameters that are associated with any parameter placeholders that are in the `FilterExpression` string.
`HasChildViewState`	Gets a value indicating whether the current server control's child controls have any saved view-state settings. (Inherited from Control.)
`ID`	Gets or sets the programmatic identifier assigned to the server control. (Inherited from Control.)
`IdSeparator`	Infrastructure. Gets the character used to separate control identifiers. (Inherited from Control.)
`InsertCommand`	Obtains or sets the SQL string that the `SqlDataSource` control uses to insert data into the underlying database.
`InsertCommandType`	Obtains or sets a value indicating whether the text in the `InsertCommand` property is an SQL statement or the name of a stored procedure.
`InsertParameters`	Obtains the parameters collection that contains the parameters that are used by the `InsertCommand` property from the `SqlDataSourceView` object that is associated with the `SqlDataSource` control.
`IsChildControlStat eCleared`	Gets a value indicating whether controls contained within this control have control state. (Inherited from Control.)
`IsTrackingViewState`	Gets a value that indicates whether the server control is saving changes to its view state. (Inherited from Control.)
`IsViewStateEnabled`	Gets a value indicating whether view state is enabled for this control. (Inherited from Control.)
`LoadViewStateByID`	Gets a value indicating whether the control participates in loading its view state by ID instead of index. (Inherited from Control.)
`OldValuesParameter FormatString`	Obtains or sets a format string to apply to the names of any parameters that are passed to the `Delete` or `Update` method.
`Page`	Gets a reference to the Page instance that contains the server control. (Inherited from Control.)
`Parent`	Gets a reference to the server control's parent control in the page control hierarchy. (Inherited from Control.)
`ProviderName`	Obtains or sets the name of the .NET Framework data provider that the `SqlDataSource` control uses to connect to an underlying data source.
`SelectCommand`	Obtains or sets the SQL string that the `SqlDataSource` control uses to retrieve data from the underlying database.
`SelectCommandType`	Obtains or sets a value indicating whether the text in the `SelectCommand` property is an SQL query or the name of a stored procedure.
`SelectParameters`	Obtains the parameters collection that contains the parameters that are used by the `SelectCommand` property from the `SqlDataSourceView` object that is associated with the `SqlDataSource` control.
`Site`	Gets information about the container that hosts the current control when rendered on a design surface. (Inherited from Control.)

Table 21.25: Noteworthy Properties of the SqlDataSource Class

[illegible]	[illegible]
`SkinID`	Gets the skin to apply to the DataSourceControl control. (Inherited from DataSourceControl.)
`SortParameterName`	Obtains or sets the name of a stored procedure parameter that is used to sort retrieved data when data retrieval is performed using a stored procedure.
`SqlCacheDependency`	Obtains or sets a semicolon-delimited string that indicates which databases and tables to use for the Microsoft SQL Server cache dependency.
`TemplateControl`	Gets or sets a reference to the template that contains this control. (Inherited from Control.)
`TemplateSourceDirectory`	Gets the virtual directory of the Page or UserControl that contains the current server control. (Inherited from Control.)
`UniqueID`	Gets the unique, hierarchically qualified identifier for the server control. (Inherited from Control.)
`UpdateCommand`	Obtains or sets the SQL string that the SqlDataSource control uses to update data in the underlying database.
`UpdateCommandType`	Obtains or sets a value indicating whether the text in the `UpdateCommand` property is an SQL statement or the name of a stored procedure.
`UpdateParameters`	Obtains the parameters' collection that contains the parameters that are used by the `UpdateCommand` property from the `SqlDataSourceView` control. The `SqlDataSourceView` control is associated with the `SqlDataSource` control.

Noteworthy methods of `SqlDataSource` class are listed in the following Table 21.26:

Table 21.26: Noteworthy Methods of the SqlDataSource Class

[illegible]	[illegible]
`Delete`	Deletes the data from the database by using the `DeleteCommand` SQL string and any parameters that are in the `DeleteParameters` collection
`Insert`	Inserts the data in the database by using the `InsertCommand` SQL string and any parameters that are in the `InsertParameters` collection
`Select`	Retrieves the data from the database by using the `SelectCommand` SQL string and any parameters that are in the `SelectParameters` collection
`Update`	Updates the data in the database by using the `UpdateCommand` SQL string and any parameters that are in the `UpdateParameters` collection

Using the **SqlDataSource** Control

You have already learned that you can display or customize data through data bound controls, such as the `GridView` control, using any data source controls, such as the `SqlDataSource` control, and the rest. The selection of a data source control and data bound control depends on the requirement of an application.

Now we will learn all the data source controls using the `GridView` control. Let's understand the SqlDataSource control. Implement the following steps to display data in a `GridView` control by using the `SqlDataSource` control:

1. Create a Web application and save it as `SqlDataSourceVB`. You can find the code of `SqlDataSourceVB` application in the Code\ASP.NET\Chapter 21\`SqlDataSourceVB` folder on the CD.
2. Drag and drop a `GridView` control on the design view of the `Default.aspx` page. Now, click Smart Tag, as shown in Figure 21.10:

NOTE

The arrow symbol on the right of the GridView control in Figure 21.10 represnts the Smart Tag.

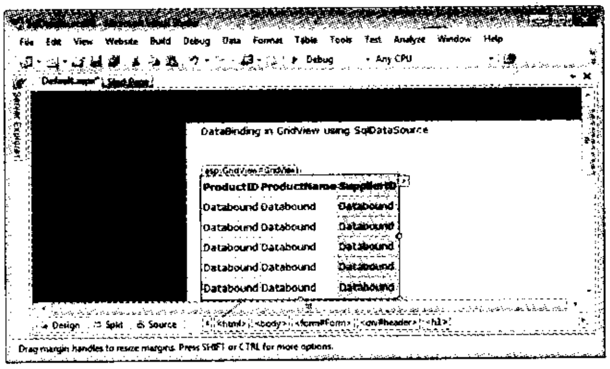

DataBinding in GridView using SqlDataSource
ProductID ProductName SupplierID
Databound Databound Databound
Design Split Source

Figure 21.10: Adding a GridView to the Web Page

On clicking the arrow symbol the Smart Tag appears.

3. Click the New data source option to configure a data source with the `GridView` control, as shown in Figure 21.11:

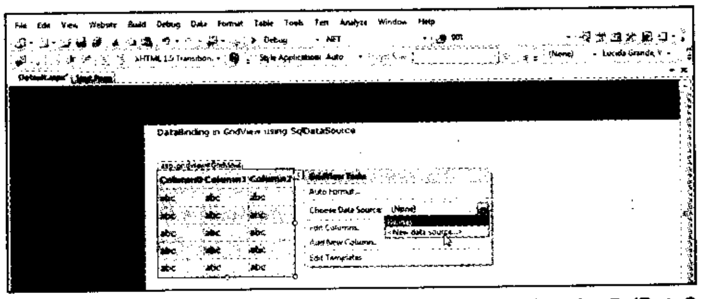


Figure 21.11: Configuring Data Source of the GridView Control Using the SqlDataSource Control

A Data Source Configuration Wizard appears, as shown in Figure 21.12:

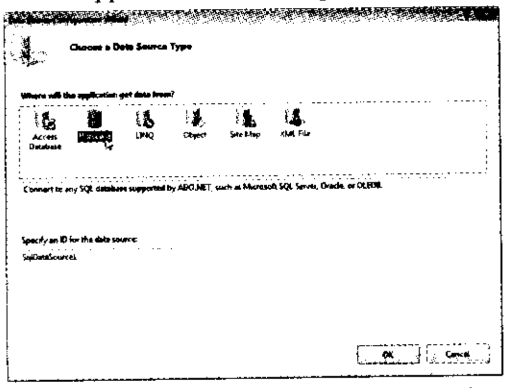

Choose a Data Source Type
Where will the application get data from?
Access Database
Database
LINQ
Object
Site Map
XML File
Connect to any SQL database supported by ADO.NET, such as Microsoft SQL Server, Oracle, or OLEDB.
Specify an ID for the data source:
SqlDataSource1
OK
Cancel

Figure 21.12: The Data Source Configuration Wizard

Select the Database option to configure a data source (Figure 21.12).

4. Select a data connection to create a connection string. Click the New Connection button, as shown in Figure 21.13:

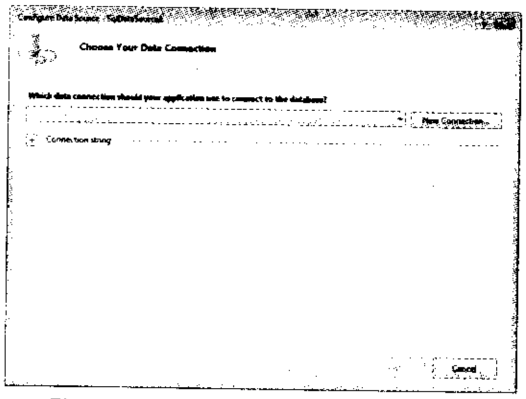
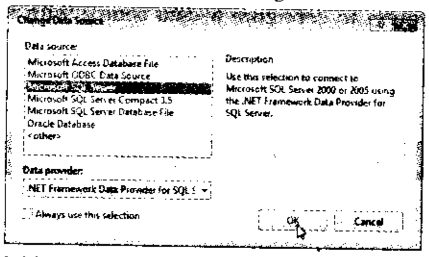

Configure Data Source - SqlDataSource1
Choose Your Data Connection
New Connection...
Connection string
Cancel
Which data connection should your application use to connect to the database?

Figure 21.13: Selecting a Data Connection

The Add Connection dialog box appears (Figure 21.15).

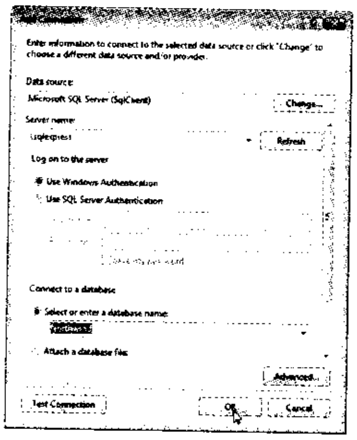
5. You can change the data source by clicking the Change button of the Add Connection dialog box. The Change Data Source dialog box appears, as shown in Figure 21.14:


Change Data Source
Data source:
Microsoft Access Database File
Microsoft ODBC Data Source
Microsoft SQL Server
Microsoft SQL Server Compact 3.5
Microsoft SQL Server Database File
Oracle Database
<other>
Description
Data provider:
Always use this selection
OK
Cancel
Use this selection to connect to Microsoft SQL Server 2000 or 2005 using the .NET Framework Data Provider for SQL Server.

Data provider: .NET Framework Data Provider for SQL S

Figure 21.14: Showing the Change Data Source Dialog Box

6. Select the data source according to your requirement and click the OK button. In our case we have selected the Microsof SQL Service data source (Figure 21.14).
7. Provide a server name and a database name under the Server name and Select or enter a database name fields, respectively in the Add Connection dialog box. In this example, we have selected Northwind as the database, as shown in Figure 21.15:



Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

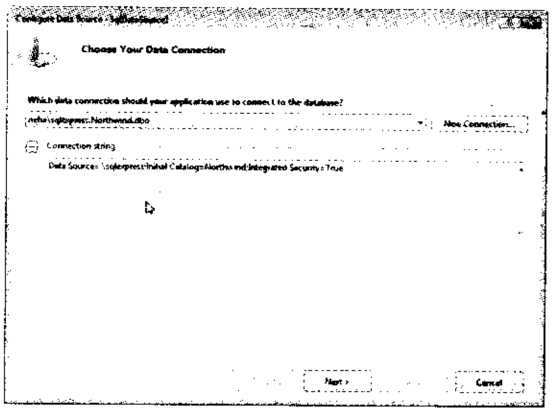
Data source: Microsoft SQL Server (SqlClient)

Server name: \sqlexpress

Select or enter a database name: Northwind

Figure 21.15: Adding a Connection

Now, a connection to the Northwind database is created. The connection string is shown in Figure 21.16:



Choose Your Data Connection

Which data connection should your application use to connect to the database?

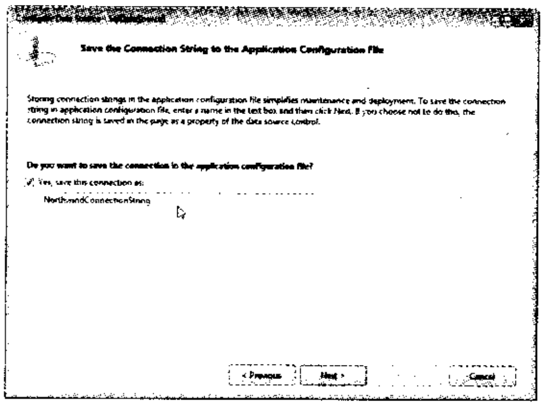
neha\sqlexpress.Northwind.dbo

Connection string

Data Source=.\sqlexpress;Initial Catalog=Northwind;Integrated Security=True

Figure 21.16: Selecting a Data Connection

8. Next, enter the name of the connection as shown in Figure 21.17, to save the connection string to the Application Configuration File:



Save the Connection String to the Application Configuration File

Storing connection strings in the application configuration file simplifies maintenance and deployment. To save the connection string in application configuration file, enter a name in the text box and then click Next. If you choose not to do this, the connection string is saved in the page as a property of the data source control.

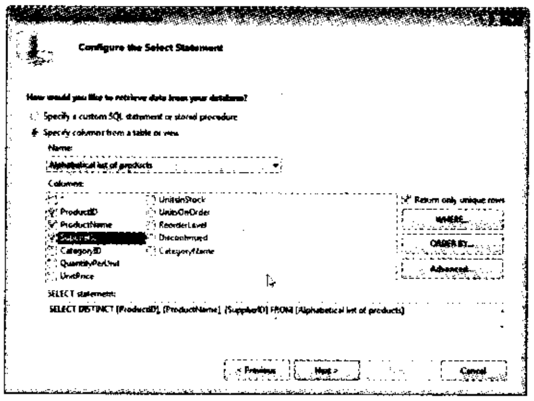
Do you want to save the connection in the application configuration file?

[✓] Yes, save this connection as:

NorthwindConnectionString

Figure 21.17: Saving a Connection String

9. Now, select a table or a view from the drop-down list and select the columns of the table. In this example, you have taken the `Products` table of the `Northwind` database and selected the `ProductID`, `ProductName` and `SupplierID` columns to display these columns in a `GridView` control at runtime, as shown in Figure 21.18:



Configure the Select Statement

How would you like to retrieve data from your database?

Specify a custom SQL statement or stored procedure [✓ Specify columns from a table or view]

Name:

Alphabetical list of products

Columns:

* ProductID ✓, ProductName ✓, SupplierID ✓, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued, CategoryName

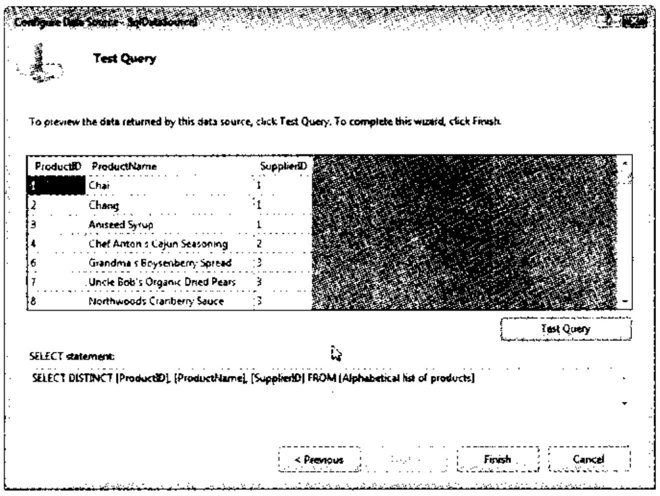
✓ Return only unique rows

SELECT statement:

SELECT DISTINCT [ProductID], [ProductName], [SupplierID] FROM [Alphabetical list of products]

Figure 21.18: Configuring the Select Statement

10. Test the query created through the wizard, as shown in Figure 21.19:



To preview the data returned by this data source, click Test Query. To complete this wizard, click Finish.

ProductID	ProductName	SupplierID
1	Chai	1
2	Chang	1
3	Aniseed Syrup	1
4	Chef Anton's Cajun Seasoning	2
6	Grandma's Boysenberry Spread	3
7	Uncle Bob's Organic Dried Pears	3
8	Northwoods Cranberry Sauce	3

SELECT statement:

SELECT DISTINCT [ProductID], [ProductName], [SupplierID] FROM [Alphabetical list of products]

Figure 21.19: Testing the Query Results

The Configure Data Source wizard enables a developer to test the results of a query to avoid the complexity of coding and time consumption.

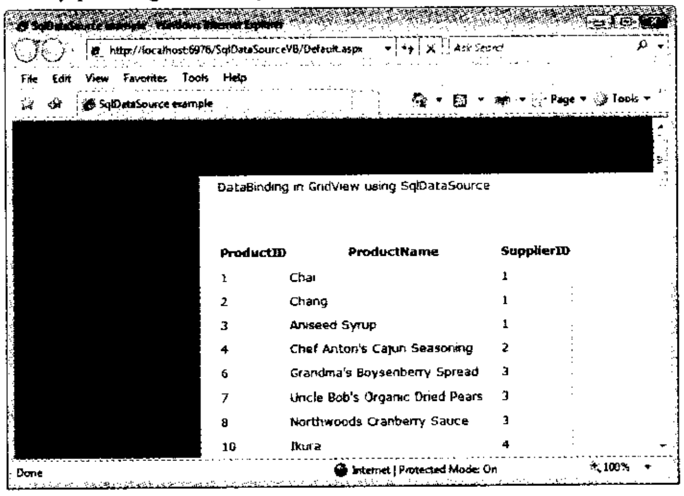
After configuring the data source, the `Default.aspx` page for the `SqlDataSource` application is shown in Listing 21.7:

Listing 21.7: Showing the code of the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!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 id="Head1" runat="server">
  <title>SqlDataSource example </title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="Form" runat="server">
      <div id="header">

      </div>
      <div id="sidebar">
          <div id="nav">
              &nbsp;
          </div>
      </div>
      <div id="content">
          <div class ="itemContent">
          DataBinding in GridView using SqlDataSource<br />
          <br />
          &nbsp;<asp:GridView ID="GridView1" runat="server"
          AutoGenerateColumns="False"
          DataKeyNames="ProductID" DataSourceID="SqlDataSource1">
          <Columns>
          <asp:BoundField DataField="ProductID" HeaderText="ProductID"
          ReadOnly="True"
          SortExpression="ProductID" />
          <asp:BoundField DataField="ProductName" HeaderText="ProductName"
```

```
                SortExpression="ProductName" />
                <asp:BoundField DataField="SupplierID" HeaderText="SupplierID"
                SortExpression="SupplierID" />
                </Columns>
                </asp:GridView>
                <asp:SqlDataSource ID="SqlDataSource1" runat="server"
                ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
                SelectCommand="SELECT DISTINCT [ProductID], [ProductName], [SupplierID]
                FROM [Alphabetical list of products]">
                </asp:SqlDataSource>
                <div id="footer">
                <p class="left">
                    All content copyright &copy; Kogent Solutions Inc.</p>
                </div>
                </div>
            </div>
        </form>
    </body>
    </html>
```

11. Run the application by pressing the F5 key. The output is shown in Figure 21.20:



http://localhost:6976/SqlDataSourceVB/Default.aspx

SqlDataSource example

DataBinding in GridView using SqlDataSource

ProductID	ProductName	SupplierID
1	Chai	1
2	Chang	1
3	Aniseed Syrup	1
4	Chef Anton's Cajun Seasoning	2
6	Grandma's Boysenberry Spread	3
7	Uncle Bob's Organic Dried Pears	3
8	Northwoods Cranberry Sauce	3
10	Ikura	4

Done — Internet | Protected Mode: On — 100%

Figure 21.20: Output of the SqlDataSourceVB Application

The AccessDataSource Control

The `AccessDataSource` control is a data source control that allows a Webserver control to access a Microsoft Access database. This control does not support connection strings, but the `DataFile` property of this control allows you to specify the Access file (`.mdb` file) that you want to use to access the data. When you add an `AccessDataSource` control to your ASP.NET page, the following line of code appears in the HTML page:

```
<asp:AccessDataSource ID="AccessDataSource1" runat="server">    </asp:AccessDataSource>
```

The `AccessDataSource` control does not allow you to provide username and password to access the data. If you need to use a password-protected Access database, select the `SqlDataSource` control to access the database. The inheritance hierarchy of the `AccessDataSource` control is as follows:

```
System.Object
  System.Web.UI.Control
     System.Web.UI.DataSourceControl
       System.Web.UI.WebControls.SqlDataSource
          System.Web.UI.WebControls.AccessDataSource
```

Noteworthy properties of the `AccessDataSource` class are listed in Table 21.27:

Table 21.27: Noteworthy Properties of the AccessDataSource Class

Property	Description
`ConnectionString`	Obtains the connection string that is used to connect to the Microsoft Access database
`Context`	Gets the HttpContext object associated with the server control for the current Web request. (Inherited from Control.)
`Controls`	Gets a ControlCollection object that represents the child controls for a specified server control in the UI hierarchy. (Inherited from DataSourceControl.)
`DataFile`	Gets or sets the location of the Microsoft Access `.mdb` file
`DataSourceMode`	Gets or sets the data retrieval mode that the SqlDataSource control uses to fetch data. (Inherited from SqlDataSource.)
`DeleteCommand`	Gets or sets the SQL string that the SqlDataSource control uses to delete data from the underlying database. (Inherited from SqlDataSource.)
`DeleteCommandType`	Gets or sets a value indicating whether the text in the DeleteCommand property is an SQL statement or the name of a stored procedure. (Inherited from SqlDataSource.)
`DeleteParameters`	Gets the parameters collection that contains the parameters that are used by the DeleteCommand property from the SqlDataSourceView object that is associated with the SqlDataSource control. (Inherited from SqlDataSource.)
`DesignMode`	Gets a value indicating whether a control is being used on a design surface. (Inherited from Control.)
`EnableCaching`	Gets or sets a value indicating whether the SqlDataSource control has data caching enabled. (Inherited from SqlDataSource.)
`EnableTheming`	Gets a value indicating whether this control supports themes. (Inherited from DataSourceControl.)
`EnableViewState`	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client. (Inherited from Control.)
`Events`	Gets a list of event handler delegates for the control. This property is read-only. (Inherited from Control.)
`FilterExpression`	Gets or sets a filtering expression that is applied when the Select method is called. (Inherited from SqlDataSource.)
`FilterParameters`	Gets a collection of parameters that are associated with any parameter placeholders that are in the FilterExpression string. (Inherited from SqlDataSource.)
`HasChildViewState`	Gets a value indicating whether the current server control's child controls have any saved view-state settings. (Inherited from Control.)
`ID`	Gets or sets the programmatic identifier assigned to the server control. (Inherited from Control.)
`IdSeparator`	Infrastructure. Gets the character used to separate control identifiers. (Inherited from Control.)
`InsertCommand`	Gets or sets the SQL string that the SqlDataSource control uses to insert data into the underlying database. (Inherited from SqlDataSource.)
`InsertCommandType`	Gets or sets a value indicating whether the text in the InsertCommand property is an SQL statement or the name of a stored procedure. (Inherited from SqlDataSource.)

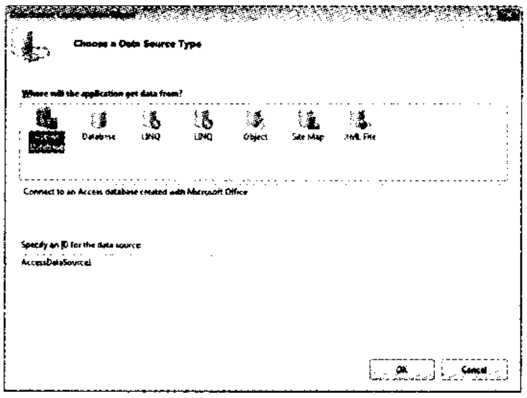
Table 21.27: Noteworthy Properties of the AccessDataSource Class

Property	Description
`InsertParameters`	Gets the parameters collection that contains the parameters that are used by the InsertCommand property from the SqlDataSourceView object that is associated with the SqlDataSource control. (Inherited from SqlDataSource.)
`IsChildControlStateCleared`	Gets a value indicating whether controls contained within this control have control state. (Inherited from Control.)
`IsTrackingViewState`	Gets a value that indicates whether the server control is saving changes to its view state. (Inherited from Control.)
`IsViewStateEnabled`	Gets a value indicating whether view state is enabled for this control. (Inherited from Control.)
`LoadViewStateByID`	Gets a value indicating whether the control participates in loading its view state by ID instead of index. (Inherited from Control.)
`OldValuesParameterFormatString`	Gets or sets a format string to apply to the names of any parameters that are passed to the Delete or Update method. (Inherited from SqlDataSource.)
`Page`	Gets a reference to the Page instance that contains the server control. (Inherited from Control.)
`Parent`	Gets a reference to the server control's parent control in the page control hierarchy. (Inherited from Control.)
`ProviderName`	Obtains the name of the .NET data provider that the `AccessDataSource` control uses to connect to a Microsoft Access database
`SelectCommand`	Gets or sets the SQL string that the SqlDataSource control uses to retrieve data from the underlying database. (Inherited from SqlDataSource.)
`SelectCommandType`	Gets or sets a value indicating whether the text in the SelectCommand property is an SQL query or the name of a stored procedure. (Inherited from SqlDataSource.)
`SelectParameters`	Gets the parameters collection that contains the parameters that are used by the SelectCommand property from the SqlDataSourceView object that is associated with the SqlDataSource control. (Inherited from SqlDataSource.)
`Site`	Gets information about the container that hosts the current control when rendered on a design surface. (Inherited from Control.)
`SkinID`	Gets the skin to apply to the DataSourceControl control. (Inherited from DataSourceControl.)
`SortParameterName`	Gets or sets the name of a stored procedure parameter that is used to sort retrieved data when data retrieval is performed using a stored procedure. (Inherited from SqlDataSource.)
`SqlCacheDependency`	Gets or sets a semicolon-delimited string that indicates which databases and tables to use for the Microsoft SQL Server cache dependency

Using the **AccessDataSource** Control

Let's understand how to use the `AccessDataSource` control in a website by performing the following steps:

1. Create a Web application and save it as `AccessDataSourceVB`. You can find the code of `AccessDataSourceVB` application in the Code\ASP.NET\Chapter 21\`AccessDataSourceVB` folder on the CD.
2. Drag and drop the `GridView` control on the design view. Select a new data source and then select Access Database to configure the data in `AccessDataSource`, as shown in Figure 21.21:


Choose a Data Source Type

Where will the application get data from?

Connect to an Access database created with Microsoft Office

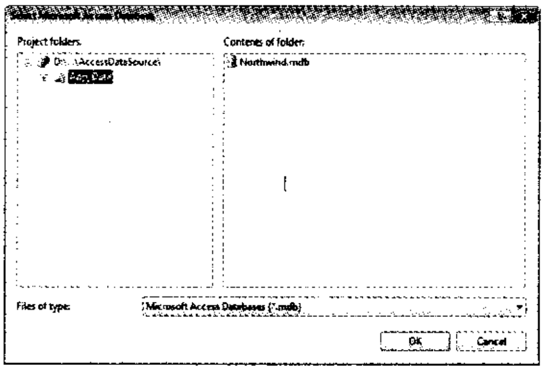
Specify an ID for the data source:
AccessDataSource1

Figure 21.21: Selecting an Access Database as the Data Source

NOTE

You need to explicitly add the Northwind.mdb file in your application.

3. Browse to the `Access Database` file, and then select the `Northwind.mdb` file that is placed in the `App_Data` folder of the `AccessDataSource` control application, as shown in Figure 21.22:

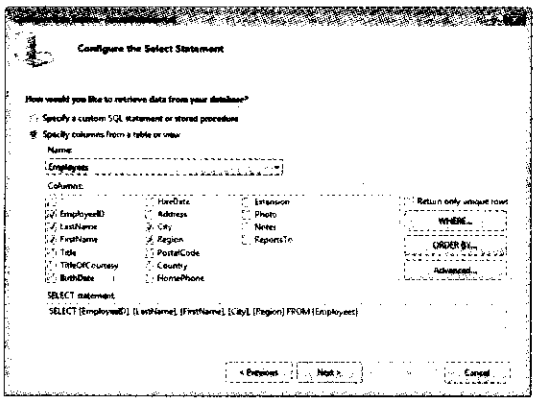

Project folders: D:\...\AccessDataSource\ App_Data

Contents of folder: Northwind.mdb

Files of type: Microsoft Access Databases (*.mdb)

Figure 21.22: Selecting an .mdb File

4. Create a `SELECT` statement, as shown in Figure 21.23 (we have used `EmployeeID`, `LastName`, `FirstName`, `City`, and `Region` in our application):


Configure the Select Statement

How would you like to retrieve data from your database?

Specify a custom SQL statement or stored procedure [✓ Specify columns from a table or view]

Name: Employees

Columns:

*	HireDate	Extension
✓ EmployeeID	Address	Photo
✓ LastName	✓ City	Notes
✓ FirstName	✓ Region	ReportsTo
Title	PostalCode	
TitleOfCourtesy	Country	
BirthDate	HomePhone	

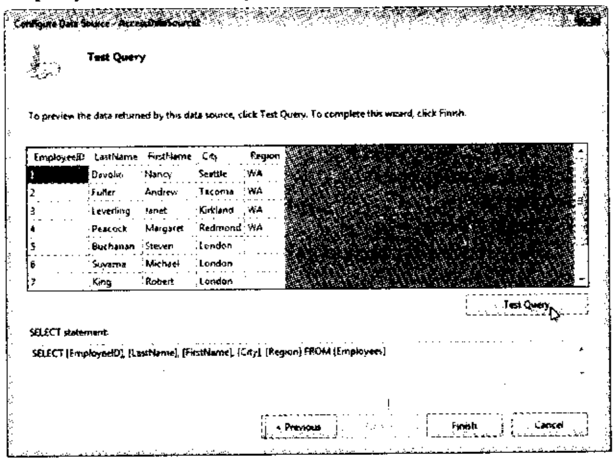
Return only unique rows

SELECT statement:

```
SELECT [EmployeeID], [LastName], [FirstName], [City], [Region] FROM [Employees]
```

Figure 21.23: Creating a SELECT Statement

5. Test the results of the query, as shown in Figure 21.24:



To preview the data returned by this data source, click Test Query. To complete this wizard, click Finish.

EmployeeID	LastName	FirstName	City	Region
1	Davolio	Nancy	Seattle	WA
2	Fuller	Andrew	Tacoma	WA
3	Leverling	Janet	Kirkland	WA
4	Peacock	Margaret	Redmond	WA
5	Buchanan	Steven	London	
6	Suyama	Michael	London	
7	King	Robert	London	

SELECT statement:

SELECT [EmployeeID], [LastName], [FirstName], [City], [Region] FROM [Employees]

Figure 21.24: Testing the Query

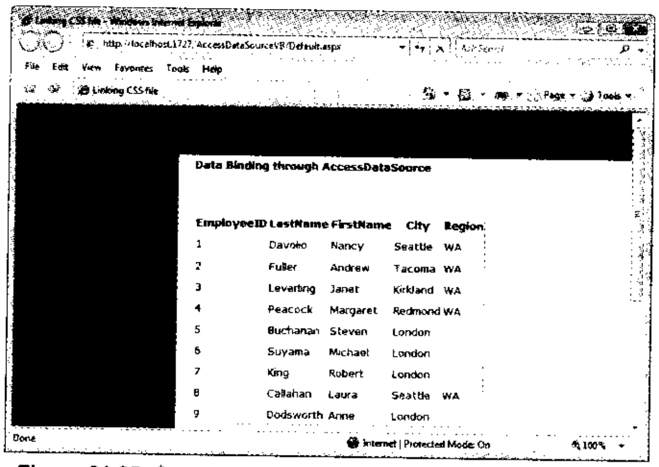
After configuring the data source through the `AccessDataSource` control, the code for the `Default.aspx` page is shown in Listing 21.8:

Listing 21.8: Showing the Code of the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!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 id="Head1" runat="server">
  <title> AccessDataSource </title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="Form" runat="server">
      <div id="header">

      </div>
      <div id="sidebar">
          <div id="nav">
              &nbsp;
          </div>
      </div>
      <div id="content">
          <div class ="itemContent">
          <b>Data Binding through AccessDataSource</b><br />
          <br />
          <br />
          <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
          DataKeyNames="EmployeeID" DataSourceID="AccessDataSource1">
          <Columns>
          <asp:BoundField DataField="EmployeeID" HeaderText="EmployeeID"
          InsertVisible="False" ReadOnly="True" SortExpression="EmployeeID" />
          <asp:BoundField DataField="LastName" HeaderText="LastName"
          SortExpression="LastName" />
          <asp:BoundField DataField="FirstName" HeaderText="FirstName"
          SortExpression="FirstName" />
          <asp:BoundField DataField="City" HeaderText="City" SortExpression="City" />
```

```
                <asp:BoundField DataField="Region" HeaderText="Region"
                SortExpression="Region" />
                </Columns>
                </asp:GridView>
                <asp:AccessDataSource ID="AccessDataSource1" runat="server"
                DataFile="~/App_Data/Northwind.mdb"
                SelectCommand="SELECT [EmployeeID], [LastName], [FirstName], [City],
                [Region] FROM [Employees]">
                </asp:AccessDataSource>
                <div id="footer">
                    <p class="left">
                        All content copyright &copy; Kogent Solutions Inc.</p>
                </div>
                </div>
            </div>
        </form>
    </body>
    </html>
```

6. Run the application by pressing the F5 key. The output of the `AccessDataSourceVB` application is shown in Figure 21.25:


Linking CSS file - Windows Internet Explorer
http://localhost:1727/AccessDataSourceVB/Default.aspx
File Edit View Favorites Tools Help
Linking CSS file
Page Tools
Done
Internet | Protected Mode: On
100%

Data Binding through AccessDataSource

EmployeeID	LastName	FirstName	City	Region
1	Davolio	Nancy	Seattle	WA
2	Fuller	Andrew	Tacoma	WA
3	Leverling	Janet	Kirkland	WA
4	Peacock	Margaret	Redmond	WA
5	Buchanan	Steven	London	
6	Suyama	Michael	London	
7	King	Robert	London	
8	Callahan	Laura	Seattle	WA
9	Dodsworth	Anne	London	

Figure 21.25: Output of the AccessDataSourceVB Application

Now, let's explore the `LinqDataSource` control.

The **LinqDataSource** Control

The `LinqDataSource` control is a data source control that allows a Web server control to access data located in the relational databases or in the memory data collection, such as an array and context objects. This control enables Web developers to use LINQ, which simplifies the interaction between object-oriented programming and relational data by applying the principles of object-oriented programming to relational data. The `LinqDataSource` control provides with the capability to connect to data from either a database or an in-memory collection, such as an array. This control enables a user to handle operations, such as insert and delete without using SQL commands to perform these tasks.

The inheritance hierarchy of the `LinqDataSource` control is as follows:

```
System.Object
  System.Web.UI.Control
     System.Web.UI.DataSourceControl
        System.Web.UI.WebControls.LinqDataSource
```

Noteworthy properties of the `LinqDataSource` class are listed in Table 21.28:

Table 21.28: Noteworthy Properties of the LinqDataSource Class

Property	Description
`AutoGenerateOrderByClause`	Gets or sets a value that indicates whether the `LinqDataSource` control dynamically creates an Order By clause based on values in the `OrderByParameters` collection
`AutoGenerateWhereClause`	Obtains or sets a value that indicates whether the `LinqDataSource` control dynamically creates a Where clause based on values defined in the `WhereParameters` collection
`AutoPage`	Obtains or sets a value that indicates whether the `LinqDataSource` control supports navigation through sections of the data at run time
`AutoSort`	Obtains or sets a value that indicates whether the `LinqDataSource` control supports sorting of the data at run time
`BindingContainer`	Infrastructure. Gets the control that contains this control's data binding. (Inherited from Control.)
`ChildControlsCreated`	Gets a value that indicates whether the server control's child controls have been created. (Inherited from Control.)
`ClientID`	Gets the server control identifier generated by ASP.NET. (Inherited from DataSourceControl.)
`ClientIDSeparator`	Gets a character value representing the separator character used in the ClientID property (Inherited from Control.)
`Context`	Gets the HttpContext object associated with the server control for the current Web request. (Inherited from Control.)
`ContextTypeName`	Obtains or sets the name of the type of data that contains the property whose value has the data that you want to retrieve
`Controls`	Gets a ControlCollection object that represents the child controls for a specified server control in the UI hierarchy. (Inherited from DataSourceControl.)
`DeleteParameters`	Obtains the collection of parameters that are used during a delete operation
`DesignMode`	Gets a value indicating whether a control is being used on a design surface. (Inherited from Control.)
`EnableDelete`	Obtains or sets a value that indicates whether data records can be deleted through the `LinqDataSource` control
`EnableInsert`	Obtains or sets a value that indicates whether data records can be inserted through the `LinqDataSource` control
`EnableTheming`	Gets a value indicating whether this control supports themes. (Inherited from DataSourceControl.)
`EnableUpdate`	Obtains or sets a value that indicates whether data records can be updated through the `LinqDataSource` control
`EnableViewState`	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client. (Inherited from Control.)
`Events`	Gets a list of event handler delegates for the control. This property is read-only. (Inherited from Control.)
`GroupBy`	Obtains or sets a value that specifies which properties are used for grouping the retrieved data
`GroupByParameters`	Obtains the collection of parameters that are used to create the Group By clause

Table 21.28: Noteworthy Properties of the LinqDataSource Class

Property	Description
`HasChildViewState`	Gets a value indicating whether the current server control's child controls have any saved view-state settings. (Inherited from Control.)
`ID`	Gets or sets the programmatic identifier assigned to the server control. (Inherited from Control.)
`IdSeparator`	Infrastructure. Gets the character used to separate control identifiers. (Inherited from Control.)
`InsertParameters`	Obtains the collection of parameters that are used during an insert operation
`IsChildControlStateCleared`	Gets a value indicating whether controls contained within this control have control state. (Inherited from Control.)
`IsTrackingViewState`	Gets a value that indicates whether the server control is saving changes to its view state. (Inherited from Control.)
`IsViewStateEnabled`	Gets a value indicating whether view state is enabled for this control. (Inherited from Control.)
`LoadViewStateByID`	Gets a value indicating whether the control participates in loading its view state by ID instead of index. (Inherited from Control.)
`OrderBy`	Obtains or sets a value that specifies which fields are used for ordering the retrieved data
`OrderByParameters`	Obtains the collection of parameters that are used to create the Order By clause.
`Page`	Gets a reference to the Page instance that contains the server control. (Inherited from Control.)
`Parent`	Gets a reference to the server control's parent control in the page control hierarchy. (Inherited from Control.)
`Select`	Obtains or sets the properties and calculated values that are included in the retrieved data
`SelectParameters`	Obtains the collection of parameters that are used during a data-retrieval operation
`Site`	Gets information about the container that hosts the current control when rendered on a design surface. (Inherited from Control.)
`SkinID`	Gets the skin to apply to the DataSourceControl control. (Inherited from DataSourceControl.)
`StoreOriginalValuesInViewState`	Obtains or sets a value that indicates whether the data from the data source should be stored in view state to make sure that the data has not been changed by another process before it is updated or deleted
`TableName`	Obtains or sets the name of the property or field in the data context object that represents a data collection
`TemplateControl`	Gets or sets a reference to the template that contains this control. (Inherited from Control.)
`TemplateSourceDirectory`	Gets the virtual directory of the Page or UserControl that contains the current server control. (Inherited from Control.)
`UniqueID`	Gets the unique, hierarchically qualified identifier for the server control. (Inherited from Control.)
`UpdateParameters`	Obtains the collection of parameters that are used during an update operation
`ViewState`	Gets a dictionary of state information that allows you to save and restore the view state of a server control across multiple requests for the same page. (Inherited from Control.)

Table 21.28: Noteworthy Properties of the LinqDataSource Class

Property	Description
`ViewStateIgnoresCase`	Gets a value that indicates whether the StateBag object is case-insensitive. (Inherited from Control.)
`Visible`	Gets or sets a value indicating whether the control is visually displayed. (Inherited from DataSourceControl.)
`Where`	Obtains or sets a value that specifies what conditions must be true for a record to be included in the retrieved data
`WhereParameters`	Obtains the collection of parameters that is used to create the Where clause

Noteworthy methods of the `LinqDataSource` control are listed in Table 21.29:

Table 21.29: Noteworthy Methods of the LinqDataSource Class

Methods	Description
`Delete`	Deletes the data from the database
`Insert`	Inserts the data in the database
`Update`	Updates the data in the database

Noteworthy events of the `LinqDataSource` class are listed in Table 21.30:

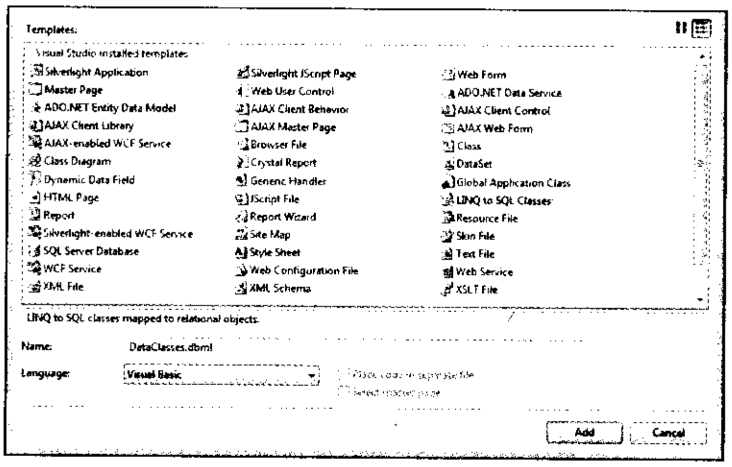
Table 21.30: Noteworthy Events of the LinqDataSource Class

Event	Description
`ContextCreated`	Invoked after an instance of the context type object is created
`ContextCreating`	Invoked before an instance of the context type object is created
`ContextDisposing`	Invoked before disposing the context type object
`DataBinding`	Occurs when the server control binds to a data source. (Inherited from Control.)
`Deleted`	Invoked when a delete operation has finished
`Deleting`	Invoked before a delete operation starts
`Disposed`	Occurs when a server control is released from memory, which is the last stage of the server control lifecycle when an ASP.NET page is requested. (Inherited from Control.)
`Init`	Occurs when the server control is initialized, which is the first step in its lifecycle. (Inherited from Control.)
`Inserted`	Invoked when an insert operation has finished
`Inserting`	Invoked before an insert operation starts
`Load`	Occurs when the server control is loaded into the Page object. (Inherited from Control.)
`PreRender`	Occurs after the Control object is loaded but prior to rendering. (Inherited from Control.)
`Selected`	Invoked when a data retrieval operation has finished
`Selecting`	Invoked before a data-retrieval operation starts
`Unload`	Occurs when the server control is unloaded from memory. (Inherited from Control.)
`Updated`	Invoked when an update operation has finished
`Updating`	Invoked before an update operation starts

Using the **LinqDataSource** Control

Now, let's use the `LinqDataSource` control with a `data bound` control, such as a `GridView` control, by performing the following steps:

1. Create a Web application and save it as `LinqDataSourceControlVB`. You can find the code of `LinqDataSourceControlVB` application in the Code\ASP.NET\Chapter 21\`LinqDataSourceControlVB` folder on the CD.
2. Right-click the application name in the Solution Explorer and select the Add New Item option from the context menu. The Add New Item dialog box appears.
3. Add a `LINQ to SQL Classes` file to the project and rename the default name of the `.dbml` file to `Northwind.dbml`, as shown in Figure 21.26:



LINQ to SQL classes mapped to relational objects.

Name: DataClasses.dbml

Language: Visual Basic

Figure 21.26: Adding a .dbml File to the Project

4. Open Server Explorer and create a connection with the `Northwind` database. Drag and drop the Products table to the design view of the `Northwind.dbml` file. The output of `Northwind.dbml` is shown in Figure 21.27:

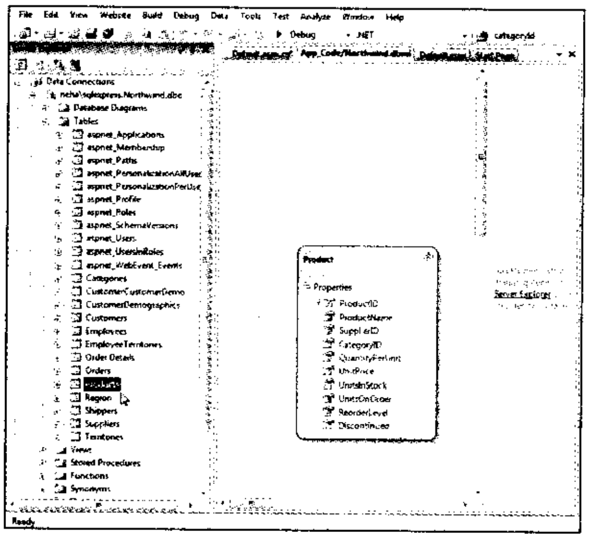
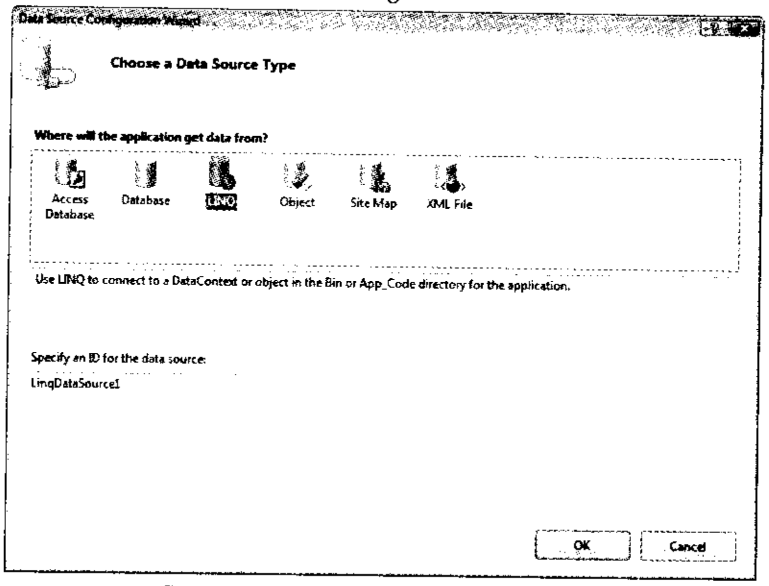


Figure 21.27: Products Table on Design Surface of Northwind.dbml

5. Drag and drop the `GridView` control on the design view of the Web page, and select the LINQ option from the Configure Data Source wizard, as shown in Figure 21.28:

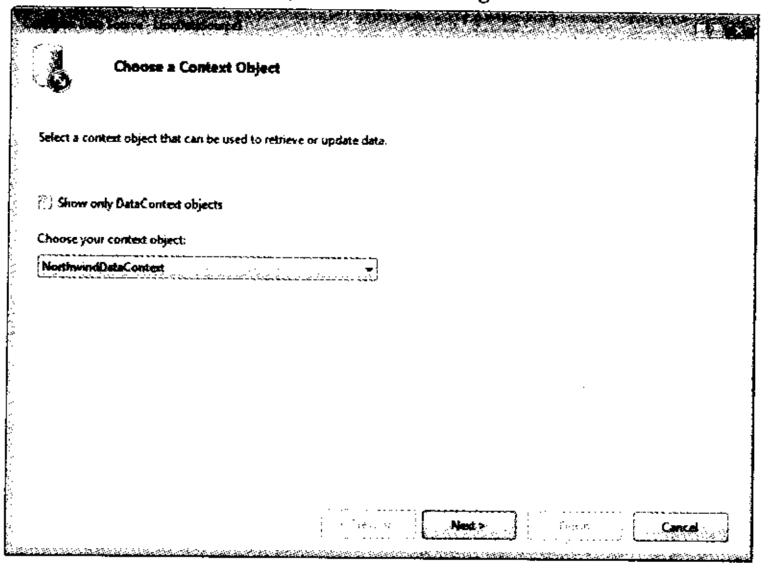

Data Source Configuration Wizard
Choose a Data Source Type
Where will the application get data from?
Access Database
Database
LINQ
Object
Site Map
XML File
OK
Cancel

Use LINQ to connect to a DataContext or object in the Bin or App_Code directory for the application.

Specify an ID for the data source:
LinqDataSource1

Figure 21.28: Selecting a Data Source Type

6. Select the `NorthwindDataContext` object, as shown in Figure 21.29:



Select a context object that can be used to retrieve or update data.

Show only DataContext objects

Choose your context object:
NorthwindDataContext

Figure 21.29: Selecting a DataContext Object

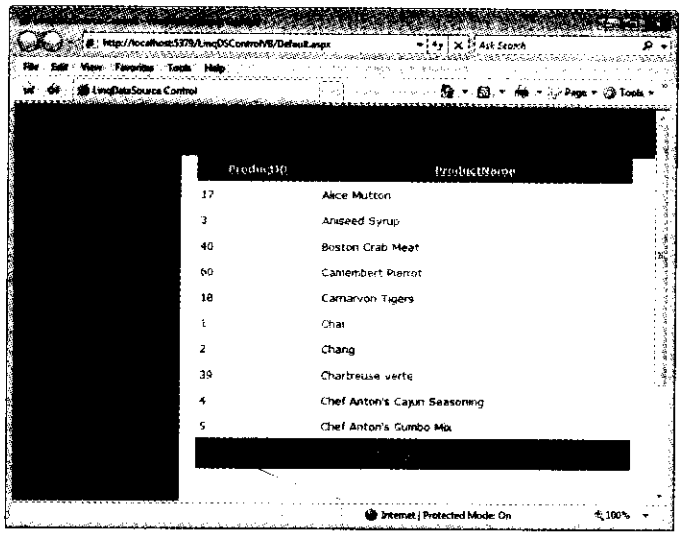
7. Create a query based on the data contained in the data context object. After configuring the data source through the `LinqDataSource` control, the code for the `Default.aspx` page is shown in Listing 21.9:

Listing 21.9: Showing the Code of the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
   Inherits="_Default" %>
<!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 id="Head1" runat="server">
  <title>LinqDataSource Control </title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />

</head>

<body>
  <form id="Form" runat="server">
       <div id="header">

       </div>
       <div id="sidebar">
            <div id="nav">
                 &nbsp;
            </div>
       </div>
       <div id="content">
            <div class ="itemContent">
            <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
            AllowSorting="True" AutoGenerateColumns="False" CellPadding="4"
            DataSourceID="LinqDataSource1" ForeColor="#333333" GridLines="None"
            Height="205px" Width="526px">
            <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />
            <Columns>
            <asp:BoundField DataField="ProductID" HeaderText="ProductID"
            ReadOnly="True"
            SortExpression="ProductID" />
            <asp:BoundField DataField="ProductName" HeaderText="ProductName"
            ReadOnly="True" SortExpression="ProductName" />
            </Columns>
            <FooterStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
            <PagerStyle BackColor="#284775" ForeColor="White"
              HorizontalAlign="Center" />
            <SelectedRowStyle BackColor="#E2DED6" Font-Bold="True"
              ForeColor="#333333" />
            <HeaderStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
            <EditRowStyle BackColor="#999999" />
            <AlternatingRowStyle BackColor="White" ForeColor="#284775" />
            </asp:GridView>
            <asp:LinqDataSource ID="LinqDataSource1" runat="server"
            ContextTypeName="NorthwindDataContext" Select="new (ProductID,
              ProductName)"
            TableName="Products">
            </asp:LinqDataSource>
            <div id="footer">
            <p class="left">
                 All content copyright &copy; Kogent Solutions Inc.</p>
            </div>
            </div>
       </div>
   </form>

 </body>

 </html>
```

8. Run the application by pressing the F5 key. The output of the `LinqDSControlVB` application is shown in Figure 21.30:



ProductID	ProductName
17	Alice Mutton
3	Aniseed Syrup
40	Boston Crab Meat
60	Camembert Pierrot
18	Carnarvon Tigers
1	Chai
2	Chang
39	Chartreuse verte
4	Chef Anton's Cajun Seasoning
5	Chef Anton's Gumbo Mix

Figure 21.30: Output of the LinqDSControlVB Application

Now, let's learn about the `ObjectDataSource` control.

The **ObjectDataSource** Control

The `ObjectDataSource` control represents the business objects and allows you to use the `ObjectDataSource` control in conjunction with a data-bound control to display, edit, and sort the data on a Web page with little or no code. This control is used to create Web applications that rely on the middle-tier objects to manage data. The inheritance hierarchy of the `ObjectDataSource` control is as follows:

```
System.Object
  System.Web.UI.Control
    System.Web.UI.DataSourceControl
      System.Web.UI.WebControls.ObjectDataSource
```

Noteworthy properties of the `ObjectDataSource` class are listed in Table 21.31:

Table 21.31: Noteworthy Properties of the ObjectDataSource Class

[illegible]	[illegible]
`CacheDuration`	Obtains or sets the length of time, in seconds, for which the data source control caches data that is retrieved by the `SelectMethod` property.
`CacheExpirationPolicy`	Obtains or sets the cache expiration behavior that, when combined with the duration, describes the behavior of the cache that the data source control uses.
`CacheKeyDependency`	Obtains or sets a user-defined key dependency that is linked to all data cache objects that are created by the data source control.
`ChildControlsCreated`	Gets a value that indicates whether the server control's child controls have been created. (Inherited from Control.)
`ClientID`	Gets the server control identifier generated by ASP.NET. (Inherited from DataSourceControl.)
`ClientIDSeparator`	Gets a character value representing the separator character used in the ClientID property. (Inherited from Control.)
`ConflictDetection`	Obtains or sets a value that determines whether or not just the new values are passed to the **Update** method or both the old and new values are passed to the **Update** method.

Table 21.31: Noteworthy Properties of the ObjectDataSource Class

Property	Description
`Context`	Gets the HttpContext object associated with the server control for the current Web request. (Inherited from Control.)
`Controls`	Gets a ControlCollection object that represents the child controls for a specified server control in the UI hierarchy. (Inherited from DataSourceControl.)
`ConvertNullToDBNull`	Obtains or sets a value indicating whether the `Parameter` values that are passed to an update, insert, or delete operation are automatically converted from `nullNothingnullptra null` reference (`Nothing` in Visual Basic) to the `Value` value by the `ObjectDataSource` control.
`DataObjectTypeName`	Obtains or sets the name of a class that the `ObjectDataSource` control uses for a parameter in an update, insert, or delete data operation. It allows you to pass multiple values instead of passing individual values from the databound control.
`DeleteMethod`	Obtains or sets the name of the method or function that the `ObjectDataSource` control invokes to delete the data.
`DeleteParameters`	Obtains the parameters collection that contains the parameters that are used by the `DeleteMethod` method.
`DesignMode`	Gets a value indicating whether a control is being used on a design surface. (Inherited from Control.)
`EnableCaching`	Obtains or sets a value indicating whether the `ObjectDataSource` control has data caching enabled.
`EnablePaging`	Obtains or sets a value that indicates whether the data source control supports paging through the set of data that it retrieves.
`EnableTheming`	Gets a value indicating whether this control supports themes. (Inherited from DataSourceControl.)
`EnableViewState`	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client. (Inherited from Control.)
`Events`	Gets a list of event handler delegates for the control. This property is read-only. (Inherited from Control.)
`FilterExpression`	Obtains or sets a filtering expression that is applied when the method that is specified by the `SelectMethod` property is called.
`FilterParameters`	Obtains a collection of parameters that are associated with any parameter placeholders in the `FilterExpression` string.
`HasChildViewState`	Gets a value indicating whether the current server control's child controls have any saved view-state settings. (Inherited from Control.)
`ID`	Gets or sets the programmatic identifier assigned to the server control. (Inherited from Control.)
`IdSeparator`	Infrastructure. Gets the character used to separate control identifiers. (Inherited from Control.)
`InsertMethod`	Obtains or sets the name of the method or function that the `ObjectDataSource` control invokes to insert data.
`InsertParameters`	Obtains the parameters collection that contains the parameters that are used by the `InsertMethod` property.

Table 21.31: Noteworthy Properties of the ObjectDataSource Class

`IsChildControlStateCleared`	Gets a value indicating whether controls contained within this control have control state. (Inherited from Control.)
`IsTrackingViewState`	Gets a value that indicates whether the server control is saving changes to its view state. (Inherited from Control.)
`IsViewStateEnabled`	Gets a value indicating whether view state is enabled for this control. (Inherited from Control.)
`LoadViewStateByID`	Gets a value indicating whether the control participates in loading its view state by ID instead of index. (Inherited from Control.)
`MaximumRowsParameterName`	Obtains or sets the name of the business object data retrieval method parameter that is used to indicate the number of records to retrieve for data source paging support.
`OldValuesParameterFormatString`	Obtains or sets a format string to apply to the names of the parameters for original values that are passed to the `Delete` or `Update` methods.
`Page`	Gets a reference to the Page instance that contains the server control. (Inherited from Control.)
`Parent`	Gets a reference to the server control's parent control in the page control hierarchy. (Inherited from Control.)
`SelectCountMethod`	Obtains or sets the name of the method or function that the `ObjectDataSource` control invokes to retrieve a row count.
`SelectMethod`	Obtains or sets the name of the method or function that the `ObjectDataSource` control invokes to retrieve data.
`SelectParameters`	Obtains the parameters collection that contains the parameters that are used by the method specified by the `SelectMethod` property.
`Site`	Gets information about the container that hosts the current control when rendered on a design surface. (Inherited from Control.)
`SkinID`	Gets the skin to apply to the DataSourceControl control. (Inherited from DataSourceControl.)
`SortParameterName`	Obtains or sets the name of the business object that the `SelectMethod` parameter uses to specify a sort expression for data source sorting support.
`SqlCacheDependency`	Obtains or sets a semicolon-delimited string that indicates which databases and tables to use for the Microsoft SQL Server cache dependency.
`StartRowIndexParameterName`	Obtains or sets the name of the data retrieval method parameter that is used to indicate the value of the identifier of the first record to retrieve for data source paging support.
`TemplateControl`	Gets or sets a reference to the template that contains this control. (Inherited from Control.)
`TemplateSourceDirectory`	Gets the virtual directory of the Page or UserControl that contains the current server control. (Inherited from Control.)
`TypeName`	Obtains or sets the name of the class that the `ObjectDataSource` object. Represents.
`UniqueID`	Gets the unique, hierarchically qualified identifier for the server control. (Inherited from Control.)
`UpdateMethod`	Obtains or sets the name of the method or function that the `ObjectDataSource` control invokes to update data.

Table 21.31: Noteworthy Properties of the ObjectDataSource Class

[illegible]	[illegible]
`UpdateParameters`	Obtains the parameters collection that contains the parameters that are used by the method that is specified by the `UpdateMethod` property.

Noteworthy methods of the `ObjectDataSource` class are listed in Table 21.32:

Table 21.32: Noteworthy Methods of the LinqObjectDataSource Class

[illegible]	[illegible]
`Delete`	Deletes the data from the database by calling the method that is identified by the `DeleteMethod` property with any parameters that are in the `DeleteParameters` collection
`Insert`	Inserts the data in the database by calling the method that is identified by the `InsertMethod` property and any parameters in the `InsertParameters` collection
`Select`	Retrieves the data from the database by calling the method that is identified by the `SelectMethod` property with the parameters in the `SelectParameters` collection
`Update`	Updates the data in the database by calling the method that is identified by the `UpdateMethod` property and any parameters that are in the `UpdateParameters` collection

Noteworthy events of the `ObjectDataSource` class are listed in Table 21.33:

Table 21.33: Noteworthy Events of the ObjectDataSource Class

[illegible]	[illegible]
`Deleted`	Invoked when a `Delete` operation has been completed
`Deleting`	Invoked before a delete operation starts
`Disposed`	Occurs when a server control is released from memory, which is the last stage of the server control lifecycle when an ASP.NET page is requested. (Inherited from Control.)
`Filtering`	Invoked before a filter operation starts
`Init`	Occurs when the server control is initialized, which is the first step in its lifecycle. (Inherited from Control.)
`Inserted`	Invoked when an `Insert` operation has been completed
`Inserting`	Invoked before an `Insert` operation starts.
`Load`	Occurs when the server control is loaded into the Page object. (Inherited from Control.)
`ObjectCreated`	Invoked after the object that is identified by the `TypeName` property is created
`ObjectCreating`	Invoked before the object that is identified by the `TypeName` property is created
`ObjectDisposing`	Invoked before the object that is identified by the `TypeName` property is discarded
`PreRender`	Occurs after the Control object is loaded but prior to rendering. (Inherited from Control.)
`Selected`	Invoked when a `Select` operation has completed
`Selecting`	Invoked before a `Select` operation starts
`Unload`	Occurs when the server control is unloaded from memory. (Inherited from Control.)
`Updated`	Invoked when an `Update` operation has been completed
`Updating`	Invoked before an `Update` operation starts

Using the **ObjectDataSource** Control

Now, let's understand how to use the `ObjectDataSource` control by performing the following steps:

1. Create a Web application and save it as `ObjectDataSource`.
2. Add a Class File to the project and replace the code of `PubsData.vb` file with the code shown in Listing 21.10:

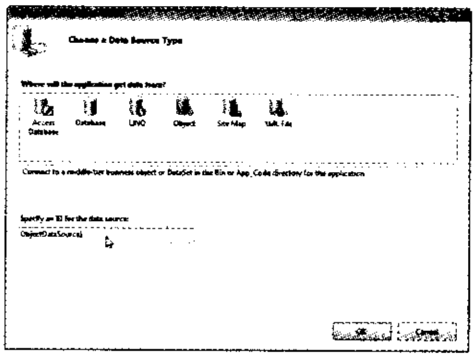
Listing 21.10: Showing the Code for the Class File(`PubsData.vb`)

```
Imports System

Imports System.Data
Imports System.Configuration

Imports System.Web
Imports System.Web.Security
Imports System.Web.UI
Imports System.Web.UI.WebControls

Imports System.Web.UI.WebControls.WebParts
Imports System.Web.UI.HtmlControls
Imports Microsoft.CSharp
Imports System.Data.SqlClient

Public Class PubsData

  Public Sub New()

  End Sub

  Public Shared Function GetInfo() As DataSet
      Dim connectionString As String = "Data Source=.\sqlexpress;Initial
      Catalog=northwind;Integrated Security=True"
      Dim dbConnection As System.Data.IDbConnection = New
      System.Data.SqlClient.SqlConnection(connectionString)
      Dim queryString As String = "SELECT ProductID,ProductName from Products"
      Dim dbCommand As System.Data.IDbCommand = New System.Data.SqlClient.SqlCommand()
      dbCommand.CommandText = queryString
      dbCommand.Connection = dbConnection
              Dim dataAdapter As System.Data.IDbDataAdapter = New
      System.Data.SqlClient.SqlDataAdapter()
      dataAdapter.SelectCommand = dbCommand
      Dim dataSet As New System.Data.DataSet()
      dataAdapter.Fill(dataSet)
      Return dataset
  End Function

End Class
```

NOTE

Change the connection string in the preceding listing as per your system configuration.

3. Drag and drop the `GridView` control on the design view of the Web page to configure `ObjectDataSource` to bind data with the `GridView` control.
4. Select the Object option (to connect to `ObjectDataSource`) from the Data Source Configuration Wizard, as shown in Figure 21.31:



Choose a Data Source Type

Where will the application get data from?

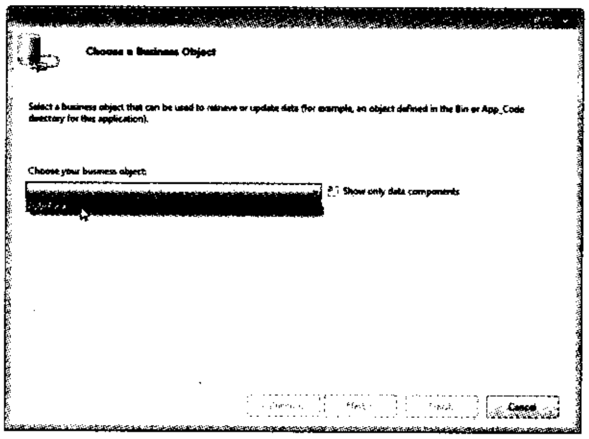
Connect to a middle-tier business object or DataSet in the Bin or App_Code directory for the application

Specify an ID for the data source:

ObjectDataSource1

Figure 21.31: Selecting a Data Source Type

5. Select a business data object that you have created through `PubsData` class file, as shown in Figure 21.32:



Choose a Business Object

Select a business object that can be used to retrieve or update data (for example, an object defined in the Bin or App_Code directory for this application).

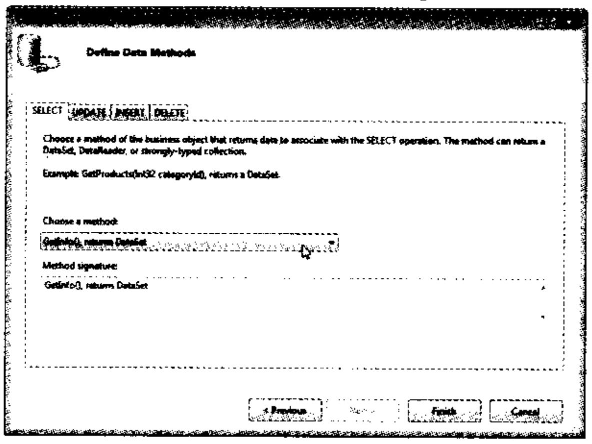
Choose your business object:

[illegible]

Show only data components

Figure 21.32: Selecting a Business Object

6. To retrieve the data from the location of the database given in `PubsData` class file through the Define Data Methods page, select a method from the Choose a method drop-down list, as shown in Figure 21.33:



Define Data Methods

Choose a method of the business object that returns data to associate with the SELECT operation. The method can return a DataSet, DataReader, or strongly-typed collection.

Example: GetProducts(Int32 categoryId), returns a DataSet.

Choose a method:

GetInfo(), returns DataSet

Method signature:

GetInfo(), returns DataSet

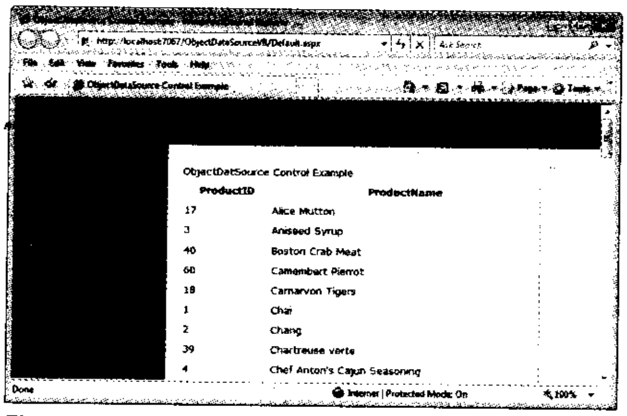
Figure 21.33: Selecting a Method

After configuring the data source through the `ObjectDataSource` control, the code for the `Default.aspx` page is shown in Listing 21.11:

Listing 21.11: Showing the Code of the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>

<!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 id="Head1" runat="server">
  <title>ObjectDataSource Control Example</title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
  <style type="text/css">
  .style1
  {
      width: 32px;
  }
  </style>
</head>

<body>

  <form id="mainForm" runat="server">
  <div>
      <div id="header">

      </div>
      <div id="sidebar" class="style1">
          <div id="nav">
              &nbsp;
          </div>
      </div>
      <div id="content">
          <div class="itemContent">
          <br />
          <asp:Label ID="Label1" runat="server" Text="ObjectDatSource Control
          Example"></asp:Label>
          <br />
          <asp:GridView ID="GridView1" runat="server"
          DataSourceID="ObjectDataSource1"
          Width="470px" Height="235px">
          </asp:GridView>
          <asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
          SelectMethod="GetInfo"
          TypeName="PubsData"></asp:ObjectDataSource>
          <br />
          <br />
          &nbsp;</div>
          <div id="footer">
          <p class="left">
          All content copyright &copy; Kogent Solutions Inc.</p>
          </div>
      </div>
  </div>
  </form>
</body>
</html>
```

7. Run the application by pressing the `F5` key. The output of `ObjectDataSourceVB` application is shown in Figure 21.34:


http://localhost:7067/ObjectDataSourceVB/Default.aspx
Ask Search
File Edit View Favorites Tools Help
ObjectDataSource Control Example
Page Tools
Done
Internet | Protected Mode: On
100%

ObjectDatSource Control Example

ProductID	ProductName
17	Alice Mutton
3	Aniseed Syrup
40	Boston Crab Meat
60	Camembert Pierrot
18	Carnarvon Tigers
1	Chai
2	Chang
39	Chartreuse verte
4	Chef Anton's Cajun Seasoning

Figure 21.34: Output of the ObjectDataSourceVB Application

Now, let's explore the `XmlDataSource` control.

The **XmlDataSource** Control

The XmlDataSource control allows a data bound control to bind the data from a XML document. This control also supports `XPath` expressions that allow to return only certain nodes from the XML document.

The inheritance hierarchy of the `XmlDataSource` control is as follows:

```
System.Object
  System.Web.UI.Control
      System.Web.UI. HierarchicalDataSourceControl
          System.Web.UI.WebControls.XmlDataSource
```

Noteworthy properties of the `XmlDataSource` class are listed in Table 21.34:

Table 21.34: Noteworthy Properties of the XmlDataSource Class

Property	Description
`CacheDuration`	Obtains or sets the length of time, in seconds, for which the data source control caches the data it has retrieved
`CacheExpirationPolicy`	Obtains or sets the cache expiration policy that is combined with the cache duration to describe the caching behavior of the cache that the data source control uses
`CacheKeyDependency`	Obtains or sets a user-defined key dependency that is linked to all data cache objects created by the data source control. All cache objects explicitly expire when the key expires
`ChildControlsCreated`	Gets a value that indicates whether the server control's child controls have been created. (Inherited from Control.)
`ClientID`	Gets the server control identifier generated by ASP.NET. (Inherited from HierarchicalDataSourceControl.)
`ClientIDSeparator`	Gets a character value representing the separator character used in the ClientID property. (Inherited from Control.)
`Context`	Gets the HttpContext object associated with the server control for the current Web request. (Inherited from Control.)
`Controls`	Gets a ControlCollection object that represents the child controls for a specified server control in the UI hierarchy. (Inherited from HierarchicalDataSourceControl.)
`Data`	Obtains or sets a block of XML data that the data source control binds to
`DataFile`	Specifies the file name of an XML file that the data source binds to

Table 21.34: Noteworthy Properties of the XmlDataSource Class

`DesignMode`	Gets a value indicating whether a control is being used on a design surface. (Inherited from Control.)
`EnableCaching`	Obtains or sets a value indicating whether the `XmlDataSource` control has data caching enabled
`EnableTheming`	Gets a value indicating whether this control supports themes. (Inherited from HierarchicalDataSourceControl.)
`EnableViewState`	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client. (Inherited from Control.)
`Events`	Gets a list of event handler delegates for the control. This property is read-only. (Inherited from Control.)
`HasChildViewState`	Gets a value indicating whether the current server control's child controls have any saved view-state settings. (Inherited from Control.)
`ID`	Gets or sets the programmatic identifier assigned to the server control. (Inherited from Control.)
`IdSeparator`	Infrastructure. Gets the character used to separate control identifiers. (Inherited from Control.)
`IsChildControlStateCleared`	Gets a value indicating whether controls contained within this control have control state. (Inherited from Control.)
`IsTrackingViewState`	Gets a value that indicates whether the server control is saving changes to its view state. (Inherited from Control.)
`IsViewStateEnabled`	Gets a value indicating whether view state is enabled for this control. (Inherited from Control.)
`LoadViewStateByID`	Gets a value indicating whether the control participates in loading its view state by ID instead of index. (Inherited from Control.)
`Page`	Gets a reference to the Page instance that contains the server control. (Inherited from Control.)
`Parent`	Gets a reference to the server control's parent control in the page control hierarchy. (Inherited from Control.)
`Site`	Gets information about the container that hosts the current control when rendered on a design surface. (Inherited from Control.)
`SkinID`	Gets or sets the skin to apply to the HierarchicalDataSourceControl control. (Inherited from HierarchicalDataSourceControl.)
`TemplateControl`	Gets or sets a reference to the template that contains this control. (Inherited from Control.)
`TemplateSourceDirectory`	Gets the virtual directory of the Page or UserControl that contains the current server control. (Inherited from Control.)
`Transform`	Obtains or sets a block of Extensible Stylesheet Language (XSL) data that defines an XSL transformation to be performed on the XML data, managed by the `XmlDataSource` control
`TransformArgumentList`	Obtains a list of XSLT arguments that are used with the style sheet defined by the `Transform` or `TransformFile` properties to perform a transformation on the XML data
`TransformFile`	Specifies the file name of an XSL file (`.xsl`) that defines an XSLT transformation to

Table 21.34: Noteworthy Properties of the XmlDataSource Class

[illegible]	[illegible]
	be performed on the XML data, managed by the `XmlDataSource` control
`UniqueID`	Gets the unique, hierarchically qualified identifier for the server control. (Inherited from Control.)
`ViewState`	Gets a dictionary of state information that allows you to save and restore the view state of a server control across multiple requests for the same page. (Inherited from Control.)
`ViewStateIgnoresCase`	Gets a value that indicates whether the StateBag object is case-insensitive. (Inherited from Control.)
`Visible`	Gets or sets a value indicating whether the control is visually displayed. (Inherited from HierarchicalDataSourceControl.)
`XPath`	Specifies an XPath expression to be applied to the XML data contained by the `Data` property or by the XML file indicated by the `DataFile` property

Noteworthy methods of the `XmlDataSource` class are listed in Table 21.35:

Table 21.35: Noteworthy Methods of the XmlDataSource Class

[illegible]	[illegible]
`GetXmlDocument`	Loads the XML data into memory, either directly from the database or from the cache, and returns it in the form of an `XmlDataDocument` object
`Save`	Saves the XML data to disk by using the `XmlDataSource` control

Noteworthy events of the `XmlDataSource` class are listed in Table 21.36:

Table 21.36: Noteworthy Events of the XmlDataSource Class

[illegible]	[illegible]
`Transforming`	Invokes before the style sheet that is defined by the `Transform` property or identified by the `TransformFile` property is applied to XML data

Now, let's learn about the `SiteMapDataSource` control.

Using the **XmlDataSource** Control

Now, let's understand that how to use the `XmlDataSource` control to bind data with the `GridView` control by performing the following steps:

1. Create a Web application and save it as `XmlDataSourceVB`. You can find the code of `XmlDataSourceVB` application in the Code\ASP.NET\Chapter 21\`XmlDataSourceVB` folder on the CD.
2. Add an Xml file to the project. Rename the xml file as `Data.xml` and replace the code of the `Data.xml` file with the following code:

```
<?xml version="1.0" standalone="yes"?>
<Data>
   <authors id="409-56-7008" lname="Bennet" fname="Abraham"/>
   <authors id="648-92-1872" lname="Blotchet-Halls" fname="Reginald"/>
   <authors id="238-95-7766" lname="Carson" fname="Cheryl"/>
</Data>
```

3. Drag and drop the `GridView` control on the design form. Select the new data source option and then select the XML File option from the Data Source Configuration Wizard, as shown in Figure 21.35:

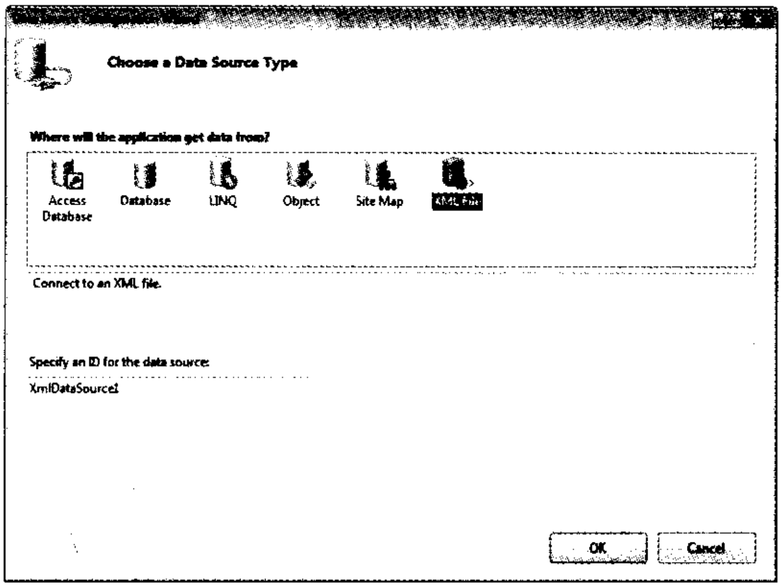

Choose a Data Source Type
Where will the application get data from?
Access Database
Database
LINQ
Object
Site Map
XML File
Connect to an XML file.
Specify an ID for the data source:
XmlDataSource1
OK
Cancel

Figure 21.35: Selecting Xml File Option as a Data Source

4. Now, select the `Data.xml` file, as shown in Figure 21.36:

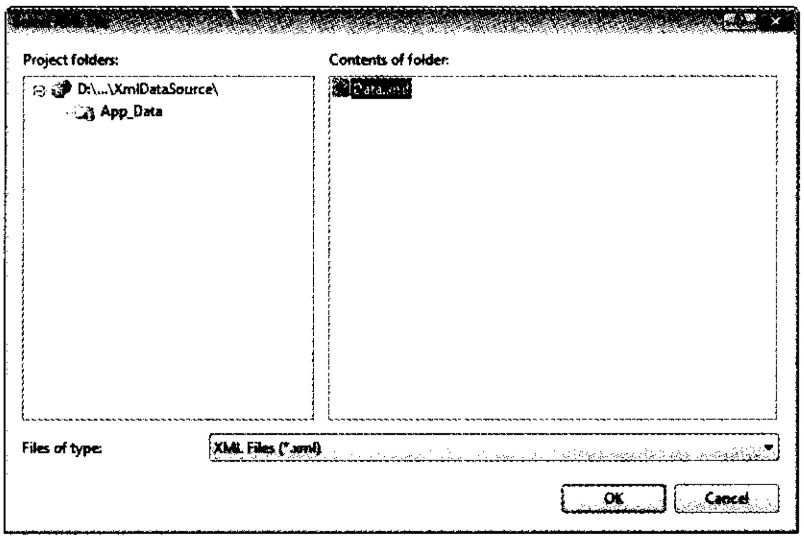


Figure 21.36: Selecting an XML File

`Data.xml` file consists of the data to be displayed in the `GridView` control. Now, you can create the query in the wizard according to the requirement of the application.

NOTE

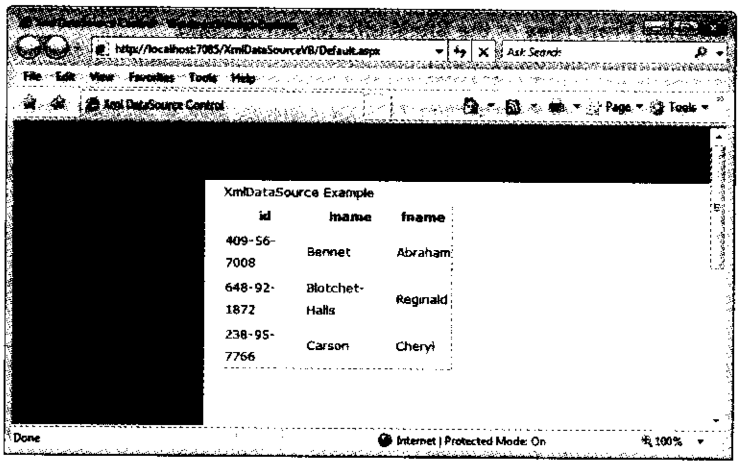
You can also test the query in the Data Source wizard as done in the ObjectDataSource application earlier.

5. After configuring the data source through the `XmlDataSource` control, the code for the `Default.aspx` page is shown in Listing 21.12:

Listing 21.12: Showing the Code of the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
   Inherits="_Default" %>

<!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 id="Head1" runat="server">
  <title>Xml DataSource Control </title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />

</head>

<body>

  <form id="Form" runat="server">
      <div id="header">

      </div>
      <div id="sidebar">
          <div id="nav">
              &nbsp;
          </div>
      </div>

      <div id="content">
          <div class ="itemContent">
          XmlDataSource Example<asp:GridView ID="GridView1"
          runat="server" AutoGenerateColumns="False" DataSourceID="XmlDataSource1"
          Height="136px" Width="241px">
          <Columns>
              <asp:BoundField DataField="id" HeaderText="id"
              SortExpression="id" />
              <asp:BoundField DataField="lname" HeaderText="lname"
              SortExpression="lname" />
              <asp:BoundField DataField="fname" HeaderText="fname"
              SortExpression="fname" />
          </Columns>

          </asp:GridView>
          <asp:XmlDataSource ID="XmlDataSource1" runat="server"
          DataFile="~/App_Data/Data.xml"></asp:XmlDataSource>
          <div id="footer">
              <p class="left">
              All content copyright &copy; Kogent Solutions Inc.</p>
          </div>
          </div>
      </div>
  </form>

</body>

</html>
```

6. Run the application by pressing the `F5` key. The output of the `XmlDataSourceVB` application is shown in Figure 21.37:


http://localhost:7085/XmlDataSourceVB/Default.aspx
Ask Search
File Edit View Favorites Tools Help
Xml DataSource Control
Page Tools
Done
Internet | Protected Mode: On
100%

XmlDataSource Example

id	lname	fname
409-56-7008	Bennet	Abraham
648-92-1872	Blotchet-Halls	Reginald
238-95-7766	Carson	Cheryl

Figure 21.37: Output of the XmlDataSourceVB Application

The **SiteMapDataSource** Control

The `SiteMapDataSource` control allows you to work with the data stored in the `SiteMap` configuration file. This control also provides with the capability to customize site navigation by using site map data with data server controls, irrespective of the fact that controls are not navigation controls. The inheritance hierarchy of the `SiteMapDataSource` control is as follows:

```
System.Object
  System.Web.UI.Control
      System.Web.UI.HierarchicalDataSourceControl
          System.Web.UI.WebControls.SiteMapDataSource
```

Noteworthy properties of the `SiteMapDataSource` class are listed in Table 21.37:

Table 21.37: Noteworthy Properties of the SiteMapDataSource Class

[illegible]	[illegible]
`ContainsListCollection`	Obtains a value indicating whether the data source control contains a collection of data source view objects
`Context`	Gets the HttpContext object associated with the server control for the current Web request. (Inherited from Control.)
`Controls`	Gets a ControlCollection object that represents the child controls for a specified server control in the UI hierarchy. (Inherited from HierarchicalDataSourceControl.)
`DesignMode`	Gets a value indicating whether a control is being used on a design surface. (Inherited from Control.)
`EnableTheming`	Gets a value indicating whether this control supports themes. (Inherited from HierarchicalDataSourceControl.)
`EnableViewState`	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client. (Inherited from Control.)
`Events`	Gets a list of event handler delegates for the control. This property is read-only. (Inherited from Control.)
`HasChildViewState`	Gets a value indicating whether the current server control's child controls have any saved view-state settings. (Inherited from Control.)
`ID`	Gets or sets the programmatic identifier assigned to the server control. (Inherited from Control.)

Table 21.37: Noteworthy Properties of the SiteMapDataSource Class

`IdSeparator`	Infrastructure. Gets the character used to separate control identifiers. (Inherited from Control.)
`IsChildControlStateCleared`	Gets a value indicating whether controls contained within this control have control state. (Inherited from Control.)
`IsTrackingViewState`	Gets a value that indicates whether the server control is saving changes to its view state. (Inherited from Control.)
`IsViewStateEnabled`	Gets a value indicating whether view state is enabled for this control. (Inherited from Control.)
`LoadViewStateByID`	Gets a value indicating whether the control participates in loading its view state by ID instead of index. (Inherited from Control.)
`Page`	Gets a reference to the Page instance that contains the server control. (Inherited from Control.)
`Parent`	Gets a reference to the server control's parent control in the page control hierarchy. (Inherited from Control.)
`Provider`	Gets or sets a `SiteMapProvider` object that is associated with the data source control
`ShowStartingNode`	Obtains or sets a value indicating whether the starting node is retrieved and displayed.
`Site`	Gets information about the container that hosts the current control when rendered on a design surface. (Inherited from Control.)
`SiteMapProvider`	Obtains or sets the name of the site map provider that the data source binds to
`SkinID`	Gets or sets the skin to apply to the HierarchicalDataSourceControl control. (Inherited from HierarchicalDataSourceControl.)
`StartFromCurrentNode`	Obtains or sets a value indicating whether or not the site map node tree is retrieved using the node that represents the current page
`StartingNodeOffset`	Obtains or sets a positive or negative integer offset from the starting node that determines the root hierarchy that is exposed by the data source control
`StartingNodeUrl`	Obtains or sets a node in the site map that the data source then uses as a reference point to retrieve nodes from a hierarchical site map
`Property`	Description
`ContainsListCollection`	Obtains a value indicating whether the data source control contains a collection of data source view objects
`Context`	Gets the HttpContext object associated with the server control for the current Web request. (Inherited from Control.)
`Controls`	Gets a ControlCollection object that represents the child controls for a specified server control in the UI hierarchy. (Inherited from HierarchicalDataSourceControl.)
`DesignMode`	Gets a value indicating whether a control is being used on a design surface. (Inherited from Control.)
`EnableTheming`	Gets a value indicating whether this control supports themes. (Inherited from HierarchicalDataSourceControl.)
`EnableViewState`	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client. (Inherited from Control.)

Noteworthy methods of the `SiteMapDataSource` class are listed in Table 21.38:

Table 21.38: Noteworthy Methods of the SiteMapDataSource Class

Methods	Description
`GetList`	Get a list of data source controls that can be used as sources of lists of data
`GetView`	Get a named view on the site map data of the site map provider according to the starting node and other properties of the data source
`GetViewNames`	Get a collection of named views for the data source control

Summary

In this chapter, you have learned about the use of data bound controls, such as the `GridView`, `DataList`, `DetailsView`, `FormView`, and `Repeater` controls, to display and edit database records. In this chapter, you have also been introduced to two new data bound controls, namely the `ListView` and `DataPager` controls. The chapter has also described data source controls, such as `SqlDataSource`, `AccessDataSource`, `LinqDataSource`, and `ObjectDataSource`, to access data. You have also learned to change and modify various styles and layouts of a data bound control. In addition, you have learned about various data source controls used to configure data connections.

In the next chapter, we explore the concepts of Login Controls.

Quick Revise

Q1. Which controls display only a single record in a table at a time?

Ans: 1. FormView control

2. DetailsView **control**

Q2. What are business objects?

Ans: Business objects are logical independent components, such as classes, modules, Web components, and .NET components that process the data and return the customized or encapsulated data objects.

Q3. Mention data bound controls and their purpose?

Ans: Data bound controls are bind with the data source controls to display the data. Here's the list of data bound controls:

- The `GridView` Control
- The `DataList` Control
- The `DetailsView` Control
- The `FormView` Control
- The `ListView` Control
- The `Repeater` Control
- The `DataPager` Control

Q4. Mention data source controls and their purpose?

Ans: The data source controls allow you to work with different types of data sources, such as SQL server or an XML file. Following is a list of data source controls:

- The `SqlDataSource` Control
- The `AccessDataSource` Control
- The `LinqDataSource` Control
- The `ObjectDataSource` Control
- The `XmlDataSource` Control
- The `SiteMapDataSource` Control

Q5. Describe DataList control and its templates?

Ans: The `DataList` control is a data bound control that displays data by using templates. These templates define controls and HTML elements that should be displayed for an item. The templates of DataList control are as follows:

- **AlternatingItemTemplate** – Provides the content and layout for alternating items in the DataList control, if defined, otherwise, the ItemTemplate control is used.
- **EditItemTemplate** – Provides the content and layout for the item currently edited in the DataList control, if defined, otherwise the ItemTemplate control is used.
- **FooterTemplate** – Provides the content and layout for the footer section of the DataList control, if defined, otherwise, the footer section is not displayed.
- **HeaderTemplate** – Provides the content and layout for the header section of the DataList control, if defined, otherwise the header section is not displayed.
- **ItemTemplate** – Provides the content and layout for items in the DataList control. It is an obligatory template.
- **SelectedItemTemplate** – Provides the content and layout for the currently selected item in the DataList control, if defined, otherwise the ItemTemplate control is used.
- **SeparatorTemplate** – Provides the content and layout for the separator between items in the DataList control, if defined, otherwise, the separator is not displayed.

Q6. Which property is used by DetailsView control to support the two-way binding?

Ans: The `DetailsView` control uses the `DataSourceID` property to support two-way binding.

Q7. FormView control does not provide a way to automatically generate command buttons to perform update, delete, or insert operations. (True/False)

Ans: True

Q8. What is the basic difference between Repeater control and DataList control?

Ans: The Repeater control explicitly places items in an HTML table whereas, the DataList control doesn't.

Q9. Which information is contained in connection string?

Ans: A connection string contains information about the server, database, and security you are working with in your application.

Q10. Describe LinqDataSource Control?

Ans: The LinqDataSource control is a data source control that allows a Web server control to access data located in the relational databases or in the memory data collection, such as an array and context objects. This control enables Web developers to use LINQ, which simplifies the interaction between object-oriented programming and relational data by applying the principles of object-oriented programming to relational data. The LinqDataSource control provides with the capability to connect to data from either a database or an in-memory collection, such as an array. This control enables a user to handle operations, such as insert and delete without using SQL commands to perform these tasks.