CHAPTER 10

Data Binding in ASP.NET 4.5

The chapters you have explored so far in this book have been focused on data access. You learned how easily you can create a data access layer using Entity Framework. You also learned about the different data access paradigms and how data access technologies are becoming standardized by means of protocols like OData for universal distribution.

In this chapter, we will shift gears and focus on data presentation. Although it is not unfamiliar territory, there has been continuous improvement in this area since ASP.NET 2.0. While ASP.NET 2.0 brought with it the addition of powerful data presentation capabilities in the form of Data Source Controls and an array of Web controls, ASP.NET 4.5 has taken it further by introducing features like Strongly Typed Data Controls and the all-new Model Binding capability—straight out of the ASP.NET MVC Framework—that you will learn about shortly.

A significant and visible improvement has also occurred in the area of ASP.NET MVC and, with increasing popularity, ASP.NET MVC 4 is aiming to take its fair share of the new Web application development market that supports modern Web platforms like HTML 5. ASP.NET MVC also introduced a new view engine called Razor. Although not strictly tied to ASP.NET MVC, Razor brings to the table a host of data presentation capabilities. Coupled with Ajax, you will learn how easy it is to fetch data asynchronously and bind it to HTML controls for display.

In this chapter, we will cover the following:

- How to perform CRUD operations using LinqDataSource and EntityDataSource Controls.
- How the new data presentation controls work with the data source controls.
- What the new data-binding features in ASP.NET 4.5 are.
- How to do data binding in ASP.NET MVC 4.

Furthermore, in this chapter, you will apply your knowledge of data access and learn how to create useful visualizations using presentation tier components.

Data Source Controls

Data source controls are not new. They have provided easy access to data from a variety of sources, including databases like Microsoft SQL Server and CLR objects, since ASP.NET 2.0. They can be used either declaratively or in code, providing lots of flexibility and ease when performing CRUD operations in conjunction with presentation Web controls like GridView. There has been continuous improvement in ASP.NET Web controls—stack and newer controls are added with each new version of .NET. You explored the SqlDataSource control in ASP.NET 2.0 that allowed you to perform data access and manipulation operations on a Microsoft SQL Server database. In this section, you will explore two new data source controls, LinqDataSource and EntityDataSource, which have been introduced to support ORM-style data access using LINQ to SQL and Entity Framework.
Note Data source controls like SqlDataSource have been discussed in detail in the previous version of this book, *Beginning ASP.NET 2.0 Databases: From Novice to Professional*. You are strongly encouraged to pick up a copy of it if you are not familiar with the concept.

### LinqDataSource

The LinqDataSource (`System.Web.UI.WebControls`) control allows you to use a LINQ to SQL data context class as a source of data and then perform CRUD operations. A LinqDataSource control can be associated with data presentation controls like GridView and ListView. The following code demonstrates a simple example of LinqDataSource control used declaratively in an ASPX file:

```xml
<asp:LinqDataSource ID="StoreDataSource" runat="server"
    ContextTypeName="DataSourceControls.StoreDataContext"
    TableName="Contacts"
    Select="new (Name, Email, State, ID)">
</asp:LinqDataSource>
```

The preceding code uses a LINQ to SQL data context class generated from the Store database, and it fetches information from the Contacts table. Associated with a GridView control, it will display the records present in the Contacts table.

```xml
<asp:GridView ID="StoreContactsView" runat="server" DataSourceID="StoreDataSource"
    AutoGenerateColumns="False">
    <Columns>
        <asp:BoundField DataField="Name" HeaderText="Name" ReadOnly="True"
            SortExpression="Name" />
        <asp:BoundField DataField="Email" HeaderText="Email" ReadOnly="True"
            SortExpression="Email" />
        <asp:BoundField DataField="State" HeaderText="State" ReadOnly="True"
            SortExpression="State" />
        <asp:BoundField DataField="ID" HeaderText="ID" ReadOnly="True"
            SortExpression="ID" />
    </Columns>
</asp:GridView>
```

The control opens up lot of possibilities in querying data. For example, you could filter the results using the `Where` clause. To select Contacts in the State of CA, you could provide the following `Where` clause:

```xml
<asp:LinqDataSource ID="StoreDataSource" runat="server"
    ContextTypeName="DataSourceControls.StoreDataContext"
    TableName="Contacts"
    Where='State = "CA"'
    Select="new (Name, Email, State, ID)">
</asp:LinqDataSource>
```

A more useful and practical possibility will be to let users choose the State from a drop-down, and the results will automatically be filtered based on the selected State. To implement this feature, first create a DropDownList control holding the filter condition values (AZ and CA in our case).