20.8.1 Building a Web Form that Displays Data from a Database

You'll now build this GUI and set up the data binding between the `GridView` control and the database. We discuss the code-behind file in Section 20.8.2. To build the guestbook application, perform the following steps:

Step 1: Creating the Web Site

To begin, follow the steps in Section 20.4.1 to create an **Empty Web Site** named `Guestbook` then add a Web Form named `Guestbook.aspx` to the project. Set the document's `Title` property to `"Guestbook"`. To ensure that `Guestobook.aspx` loads when you execute this application, right click it in the **Solution Explorer** and select **Set As Start Page**.

Step 2: Creating the Form for User Input

In **Design** mode, add the text `Please leave a message in our guestbook:`, then use the **Block Format** `ComboBox` in the IDE's toolbar to change the text to **Heading 3** format. Insert a table with four rows and two columns, configured so that the text in each cell aligns with the top of the cell. Place the appropriate text (see Fig. 20.31) in the top three cells in the table's left column. Then place `TextBoxes` named `nameTextBox`, `emailTextBox` and `messageTextBox` in the top three table cells in the right column. Configure the `TextBoxes` as follows:

- Set the `nameTextBox`'s width to `300px`.
- Set the `emailTextBox`'s width to `300px`.
- Set the `messageTextBox`'s width to `300px` and height to `100px`. Also set this control's `TextMode` property to `MultiLine` so the user can type a message containing multiple lines of text.

Finally, add `Buttons` named `submitButton` and `clearButton` to the bottom-right table cell. Set the buttons' `Text` properties to `Submit` and `Clear`, respectively. We discuss the buttons' event handlers when we present the code-behind file. You can create these event handlers now by double clicking each `Button` in **Design** view.

Step 3: Adding a `GridView` Control to the Web Form

Add a `GridView` named `messagesGridView` that will display the guestbook entries. This control appears in the **Data** section of the **Toolbox**. The colors for the `GridView` are specified through the **Auto Format...** link in the **GridView Tasks** smart-tag menu that opens when you place the `GridView` on the page. Clicking this link displays an **AutoFormat** dialog with several choices. In this example, we chose **Professional**. We show how to set the `GridView`'s data source (that is, where it gets the data to display in its rows and columns) shortly.

Step 4: Adding a Database to an ASP.NET Web Application

To use a SQL Server Express database file in an ASP.NET web application, you must first add the file to the project's `App_Data` folder. For security reasons, this folder can be accessed only by the web application on the server—clients cannot access this folder over a network. The web application interacts with the database on behalf of the client.

The **Empty Web Site** template does not create the `App_Data` folder. To create it, right click the project's name in the **Solution Explorer**, then select **Add ASP.NET Folder > App_Data**. Next, add the `Guestbook.mdf` file to the `App_Data` folder. You can do this in one of two ways:

- Drag the file from Windows Explorer and drop it on the `App_Data` folder.
- Right click the `App_Data` folder in the **Solution Explorer** and select **Add Existing Item...** to display the **Add Existing Item** dialog, then navigate to the databases folder with this chapter's examples, select the `Guestbook.mdf` file and click **Add**. [*Note:* Ensure that **Data Files** is selected in the `ComboBox` above or next to the **Add** `Button` in the dialog; otherwise, the database file will not be displayed in the list of files.]

Step 5: Creating the LINQ to SQL Classes
You'll use LINQ to interact with the database. To create the LINQ to SQL classes for the `Guestbook` database:

1. Right click the project in the **Solution Explorer** and select **Add New Item...** to display the **Add New Item** dialog.
2. In the dialog, select **LINQ to SQL Classes**, enter `Guestbook.dbml` as the **Name**, and click **Add**. A dialog appears asking if you would like to put your new LINQ to SQL classes in the `App_Code` folder; click **Yes**. The IDE will create an `App_Code` folder and place the LINQ to SQL classes information in that folder.
3. In the **Database Explorer** window, drag the `Guestbook` database's `Messages` table from the **Database Explorer** onto the **Object Relational Designer**. Finally, save your project by selecting **File > Save All**.

Step 6: Binding the `GridView` to the `Messages` Table of the `Guestbook` Database
You can now configure the `GridView` to display the database's data.

1. In the **GridView Tasks** smart-tag menu, select **<New data source...>** from the **Choose Data Source** `ComboBox` to display the **Data Source Configuration Wizard** dialog.
2. In this example, we use a **`LinqDataSource`** control that allows the application to interact with the `Guestbook.mdf` database through LINQ. Select **LINQ**, then set the `ID` of the data source to `messagesLinqDataSource` and click **OK** to begin the **Configure Data Source** wizard.
3. In the **Choose a Context Object** screen, ensure that `GuestbookDataContext` is selected in the `ComboBox`, then click **Next >**.
4. The **Configure Data Selection** screen (Fig. 20.33) allows you to specify which data the `LinqDataSource` should retrieve from the data context. Your choices on this page design a `Select` LINQ query. The **Table** drop-down list identifies a table in the data context. The `Guestbook` data context contains one table named `Messages`, which is selected by default. *If you haven't saved your project* since creating your LINQ to SQL classes (*Step 5*), the list of tables *will not appear*. In the **Select** pane, ensure that the checkbox marked with an asterisk (*) is selected to indicate that you want to retrieve all the columns in the `Messages` table.
5. Click the **Advanced...** button, then select the **Enable the LinqDataSource to perform automatic inserts** `CheckBox` and click **OK**. This configures the `LinqDataSource` control to automatically insert new data into the database when new data is inserted in the data context. We discuss inserting new guestbook entries based on users' form submissions shortly.
6. Click **Finish** to complete the wizard.

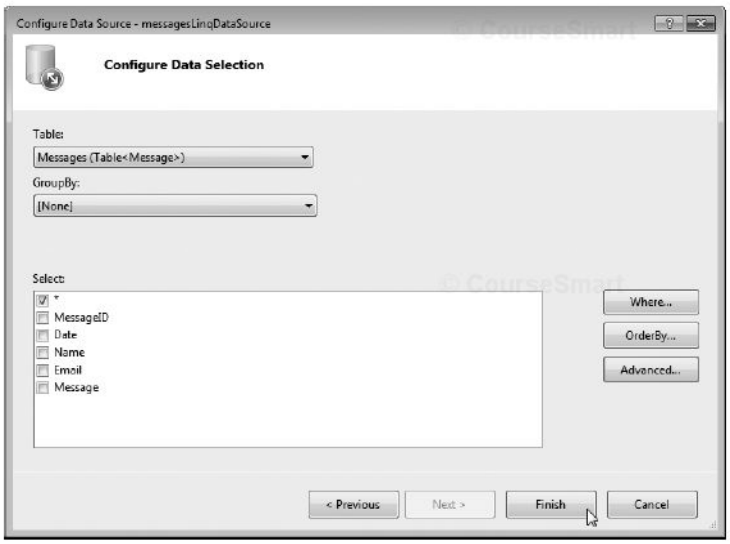

Fig. 20.33 | Configuring the query used by the `LinqDataSource` to retrieve data.

A control named `messagesLinqDataSource` now appears on the Web Form directly below the `GridView` (Fig. 20.34). It's represented in **Design** mode as a gray box containing its type and name. It will *not* appear on the web page—the gray box simply provides a way to manipulate the control visually through **Design** mode—similar to how the objects in the component tray are used in **Design** mode for a Windows Forms application.

The `GridView` now has column headers that correspond to the columns in the `Messages` table. The rows each contain either a number (which signifies an autoincremented column) or **abc** (which indicates string data). The actual data from the `Guestbook.mdf` database file will appear in these rows when you view the ASPX file in a web browser.

Step 7: Modifying the Columns of the Data Source Displayed in the `GridView`

It's not necessary for site visitors to see the `MessageID` column when viewing past guestbook entries—this column is merely a unique primary key required by the `Messages` table within the database. So, let's modify the `GridView` to prevent this column from displaying on the Web Form. We'll also modify the column **Message1** to read **Message**.

1. In the **GridView Tasks** smart tag menu, click **Edit Columns** to display the **Fields** dialog (Fig. 20.35).
2. Select **MessageID** in the **Selected fields** pane, then click the `Button`. This removes the `MessageID` column from the `GridView`.
3. Next select **Message1** in the **Selected fields** pane and change its `HeaderText` property to `Message`. The IDE renamed this field to prevent a naming conflict in the LINQ to SQL classes.
4. Click **OK** to return to the main IDE window, then set the `Width` property of the `GridView` to `650px`.

The `GridView` should now appear as shown in Fig. 20.31.

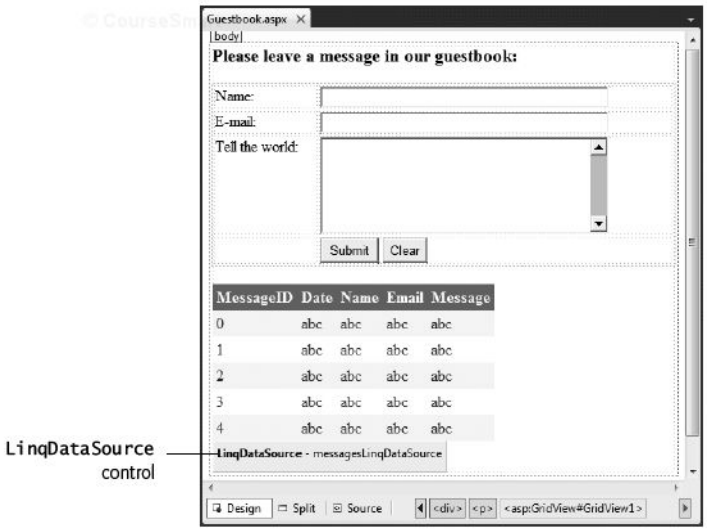

Fig. 20.34 | **Design** mode displaying `LinqDataSource` control for a `GridView`.

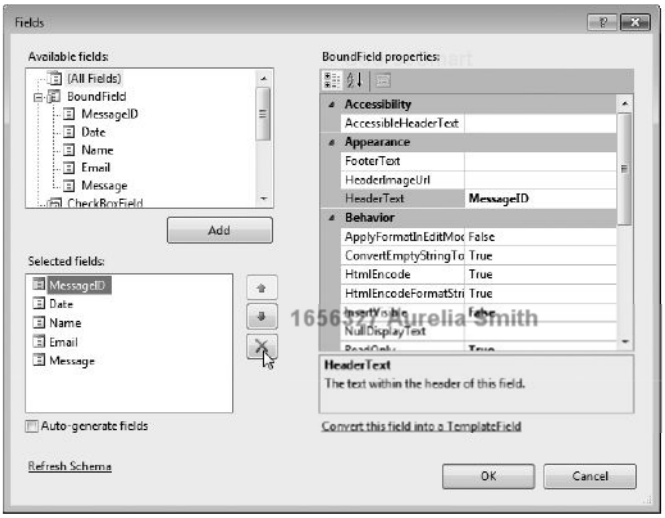

Fig. 20.35 | Removing the `MessageID` column from the `GridView`.

20.8.2 Modifying the Code-Behind File for the Guestbook Application

After building the Web Form and configuring the data controls used in this example, double click the **Submit** and **Clear** buttons in **Design** view to create their corresponding `Click` event handlers in the code-behind file (Fig. 20.36). The IDE generates empty event handlers, so we must add the appropriate code to make these buttons work properly. The

event handler for `clearButton` (lines 37–42) clears each `TextBox` by setting its `Text` property to an empty string. This resets the form for a new guestbook submission.

```
1  // Fig. 20.36: Guestbook.aspx.cs
2  // Code-behind file that defines event handlers for the guestbook.
3  using System;
4  using System.Collections.Specialized; // for class ListDictionary
5
6  public partial class Guestbook : System.Web.UI.Page
7  {
8     // Submit Button adds a new guestbook entry to the database,
9     // clears the form and displays the updated list of guestbook entries
10    protected void submitButton_Click( object sender, EventArgs e )
11    {
12       // create dictionary of parameters for inserting
13       ListDictionary insertParameters = new ListDictionary();
14
15       // add current date and the user's name, e-mail address
16       // and message to dictionary of insert parameters
17       insertParameters.Add( "Date", DateTime.Now.ToShortDateString() );
18       insertParameters.Add( "Name", nameTextBox.Text );
19       insertParameters.Add( "Email", emailTextBox.Text );
20       insertParameters.Add( "Message", messageTextBox.Text );
21
22       // execute an INSERT LINQ statement to add a new entry to the
23       // Messages table in the Guestbook data context that contains the
24       // current date and the user's name, e-mail address and message
25       messagesLinqDataSource.Insert( insertParameters );
26
27       // clear the TextBoxes
28       nameTextBox.Text = String.Empty;
29       emailTextBox.Text = String.Empty;
30       messageTextBox.Text = String.Empty;
31
32       // update the GridView with the new database table contents
33       messagesGridView.DataBind();
34    } // submitButton_Click
35
36    // Clear Button clears the Web Form's TextBoxes
37    protected void clearButton_Click( object sender, EventArgs e )
38    {
39       nameTextBox.Text = String.Empty;
40       emailTextBox.Text = String.Empty;
41       messageTextBox.Text = String.Empty;
42    } // clearButton_Click
43 } // end class Guestbook
```

Fig. 20.36 | Code-behind file for the guestbook application.

Lines 10–34 contain `submitButton`'s event-handling code, which adds the user's information to the `Guestbook` database's `Messages` table. To use the values of the `TextBoxes` on the Web Form as the parameter values inserted into the database, we must create a **`ListDictionary`** of insert parameters that are key/value pairs.

Line 13 creates a `ListDictionary` object—a set of key/value pairs that is implemented as a linked list and is intended for dictionaries that store 10 or fewer keys. Lines 17–20 use the `ListDictionary`'s `Add` method to store key/value pairs that represent each of the four insert parameters—the current date and the user's name, e-mail address, and message. The keys must match the names of the columns of the `Messages` table in the `.dbml` file. Invoking the `LinqDataSource` method `Insert` (line 25) inserts the data in the data context, adding a row to the `Messages` table and automatically updating the database. We pass the `ListDictionary` object as an argument to the `Insert` method to specify the insert parameters. After the data is inserted into the database, lines 28–30 clear the `TextBoxes`, and line 33 invokes `messagesGridView`'s **`DataBind` method** to refresh the data that the `GridView` displays. This causes `messagesLinqDataSource` (the `GridView`'s source) to execute its `Select` command to obtain the `Messages` table's newly updated data.

20.9 Case Study Introduction: ASP.NET AJAX

In Chapter 21, you learn the difference between a traditional web application and an **Ajax (Asynchronous JavaScript and XML) web application**. You also learn how to use ASP.NET AJAX to quickly and easily improve the user experience for your web applications, giving them responsiveness comparable to that of desktop applications. To demonstrate ASP.NET AJAX capabilities, you enhance the validation example by displaying the submitted form information without reloading the entire page. The only modifications to this web application appear in the `Validation.aspx` file. You use Ajax-enabled controls to add this feature.

20.10 Case Study Introduction: Password-Protected Books Database Application

In Chapter 21, we include a web application case study in which a user logs into a password-protected website to view a list of publications by a selected author. The application consists of several pages and provides website registration and login capabilities. You'll learn about ASP.NET master pages, which allow you to specify a common look-and-feel for all the pages in your app. We also introduce the **Web Site Administration Tool** and use it to configure the portions of the application that can be accessed only by users who are logged into the website.

Summary

Section 20.1 Introduction

- ASP.NET technology is Microsoft's technology for web-application development.
- Web Form files have the file-name extension `.aspx` and contain the web page's GUI. A Web Form file represents the web page that is sent to the client browser.
- The file that contains the programming logic of a Web Form is called the code-behind file.

Section 20.2 Web Basics

- URIs (Uniform Resource Identifiers) identify resources on the Internet. URIs that start with `http://` are called URLs (Uniform Resource Locators).
- A URL contains information that directs a browser to the resource that the user wishes to access. Computers that run web server software make such resources available.
- In a URL, the hostname is the name of the server on which the resource resides. This computer usually is referred to as the host, because it houses and maintains resources.
- A hostname is translated into a unique IP address that identifies the server. This translation is performed by a domain-name system (DNS) server.
- The remainder of a URL specifies the location and name of a requested resource. For security reasons, the location is normally a virtual directory. The server translates the virtual directory into a real location on the server.
- When given a URL, a web browser uses HTTP to retrieve the web page found at that address.

Section 20.3 Multitier Application Architecture

- Multitier applications divide functionality into separate tiers—logical groupings of functionality—that commonly reside on separate computers for security and scalability.
- The information tier (also called the bottom tier) maintains data pertaining to the application. This tier typically stores data in a relational database management system.
- The middle tier implements business logic, controller logic and presentation logic to control interactions between the application's clients and the application's data. The middle tier acts as an intermediary between data in the information tier and the application's clients.
- Business logic in the middle tier enforces business rules and ensures that data is reliable before the server application updates the database or presents the data to users.
- The client tier, or top tier, is the application's user interface, which gathers input and displays output. Users interact directly with the application through the user interface (typically viewed in a web browser), keyboard and mouse. In response to user actions, the client tier interacts with the middle tier to make requests and to retrieve data from the information tier. The client tier then displays to the user the data retrieved from the middle tier.

Section 20.4.1 Building the `WebTime` Application

- **File System** websites are created and tested on your local computer. Such websites execute in Visual Web Developer's built-in ASP.NET Development Server and can be accessed only by web browsers running on the same computer. You can later "publish" your website to a production web server for access via a local network or the Internet.
- **HTTP** websites are created and tested on an IIS web server and use HTTP to allow you to put your website's files on the server. If you own a website and have your own web server computer, you might use this to build a new website directly on that server computer.
- **FTP** websites use File Transfer Protocol (FTP) to allow you to put your website's files on the server. The server administrator must first create the website on the server for you. FTP is commonly used by so called "hosting providers" to allow website owners to share a server computer that runs many websites.
- A Web Form represents one page in a web application and contains a web application's GUI.
- You can view the Web Form's properties by selecting `DOCUMENT` in the **Properties** window. The `Title` property specifies the title that will be displayed in the web browser's title bar when the page is loaded.

- Controls and other elements are placed sequentially on a Web Form one after another in the order in which you drag-and-drop them onto the Web Form. The cursor indicates the insertion point in the page. This type of layout is known as relative positioning. You can also use absolute positioning in which controls are located exactly where you drop them on the Web Form.
- When a `Label` does not contain text, its name is displayed in square brackets in **Design** view as a placeholder for design and layout purposes. This text is not displayed at execution time.
- Formatting in a web page is performed with Cascading Style Sheets (CSS).
- A Web Form's `Init` event occurs when the page is requested by a web browser. The event handler for this event—named `Page_Init`—initialize the page.

Section 20.4.2 Examining `WebTime.aspx`'s Code-Behind File

- A class declaration can span multiple source-code files—the separate portions of the class declaration in each file are known as partial classes. The `partial` modifier indicates that the class in a particular file is part of a larger class.
- Every Web Form class inherits from class `Page` in namespace `System.Web.UI`. Class `Page` represents the default capabilities of each page in a web application.
- The ASP.NET controls are defined in namespace `System.Web.UI.WebControls`.

Section 20.5 Standard Web Controls: Designing a Form

- An `Image` control's `ImageUrl` property specifies the location of the image to display.
- By default, the contents of a table cell are aligned vertically in the middle of the cell. You can change this with the cell's `valign` property.
- A `TextBox` control allows you to obtain text from the user and display text to the user.
- A `DropDownList` control is similar to the Windows `Forms ComboBox` control, but doesn't allow users to type text. You can add items to the `DropDownList` using the **ListItem Collection Editor**, which you can access by clicking the ellipsis next to the `DropDownList`'s `Items` property in the **Properties** window, or by using the **DropDownList Tasks** menu.
- A `HyperLink` control adds a hyperlink to a Web Form. The `NavigateUrl` property specifies the resource or web page that will be requested when the user clicks the `HyperLink`.
- A `RadioButtonList` control provides a series of radio buttons from which the user can select only one. The **RadioButtonList Tasks** smart-tag menu provides an **Edit Items...** link to open the **ListItem Collection Editor** so that you can create the items in the list.
- A `Button` control triggers an action when clicked.

Section 20.6 Validation Controls

- A validation control determines whether the data in another web control is in the proper format.
- When the page is sent to the client, the validator is converted into `JavaScript` that performs the validation in the client web browser.
- Some client browsers might not support scripting or the user might disable it. For this reason, you should always perform validation on the server.
- A `RequiredFieldValidator` control ensures that its `ControlToValidate` is not empty when the form is submitted. The validator's `ErrorMessage` property specifies what to display on the Web Form if the validation fails. When the validator's `Display` property is set to `Dynamic`, the validator occupies space on the Web Form only when validation fails.
- A `RegularExpressionValidator` uses a regular expression to ensure data entered by the user is in a valid format. Visual Web Developer provides several predefined regular expressions that you can

simply select to validate e-mail addresses, phone numbers and more. A `RegularExpressionValidator`'s `ValidationExpression` property specifies the regular expression to use for validation.

- A Web Form's `Load` event occurs each time the page loads into a web browser. The event handler for this event is `Page_Load`.
- ASP.NET pages are often designed so that the current page reloads when the user submits the form; this enables the program to receive input, process it as necessary and display the results in the same page when it's loaded the second time.
- Submitting a web form is known as a postback. Class `Page`'s `IsPostBack` property returns `true` if the page is being loaded due to a postback.
- Server-side Web Form validation must be implemented programmatically. Class `Page`'s `Validate` method validates the information in the request as specified by the Web Form's validation controls. Class `Page`'s `IsValid` property returns `true` if validation succeeded.

Section 20.7 Session Tracking

- Personalization makes it possible for e-businesses to communicate effectively with their customers and also improves users' ability to locate desired products and services.
- To provide personalized services to consumers, e-businesses must be able to recognize clients when they request information from a site.
- HTTP is a stateless protocol—it does not provide information regarding particular clients.
- Tracking individual clients is known as session tracking.

Section 20.7.1 Cookies

- A cookie is a piece of data stored in a small text file on the user's computer. A cookie maintains information about the client during and between browser sessions.
- The expiration date of a cookie determines how long the cookie remains on the client's computer. If you do not set an expiration date for a cookie, the web browser maintains the cookie for the duration of the browsing session.

Section 20.7.2 Session Tracking with `HttpSessionState`

- Session tracking is implemented with class `HttpSessionState`.

Section 20.7.3 `Options.aspx`*: Selecting a Programming Language*

- Each radio button in a `RadioButtonList` has a `Text` property and a `Value` property. The `Text` property is displayed next to the radio button and the `Value` property represents a value that is sent to the server when the user selects that radio button and submits the form.
- Every Web Form includes a user-specific `HttpSessionState` object, which is accessible through property `Session` of class `Page`.
- `HttpSessionState` property `SessionID` contains a client's unique session ID. The first time a client connects to the web server, a unique session ID is created for that client and a temporary cookie is written to the client so the server can identify the client on subsequent requests. When the client makes additional requests, the client's session ID from that temporary cookie is compared with the session IDs stored in the web server's memory to retrieve the client's `HttpSessionState` object.
- `HttpSessionState` property `Timeout` specifies the maximum amount of time that an `HttpSessionState` object can be inactive before it's discarded. Twenty minutes is the default.
- The `HttpSessionState` object is a dictionary—a data structure that stores key/value pairs. A program uses the key to store and retrieve the associated value in the dictionary.

- The key/value pairs in an `HttpSessionState` object are often referred to as session items. They're placed in an `HttpSessionState` object by calling its `Add` method. Another common syntax for placing a session item in the `HttpSessionState` object is `Session[`*Key*`] = `*Value*.
- If an application adds a session item that has the same name as an item previously stored in the `HttpSessionState` object, the session item is replaced—session items names *must* be unique.

Section 20.7.4 ***`Recommendations.aspx`****: Displaying Recommendations Based on Session Values*

- The `Count` property returns the number of session items stored in an `HttpSessionState` object.
- `HttpSessionState`'s `Keys` property returns a collection containing all the keys in the session.

Section 20.8 Case Study: Database-Driven ASP.NET Guestbook

- A `GridView` data control displays data in tabular format. This control is located in the `Toolbox`'s `Data` section.

Section 20.8.1 Building a Web Form that Displays Data from a Database

- To use a SQL Server Express database file in an ASP.NET web application, you must first add the file to the project's `App_Data` folder. For security reasons, this folder can be accessed only by the web application on the server—clients cannot access this folder over a network. The web application interacts with the database on behalf of the client.
- A `LinqDataSource` control allows a web application to interact with a database through LINQ.

Section 20.8.2 Modifying the Code-Behind File for the Guestbook Application

- To insert data into a database using a `LinqDataSource`, you must create a `ListDictionary` of insert parameters that are formatted as key/value pairs.
- A `ListDictionary`'s `Add` method stores key/value pairs that represent each insert parameter.
- A `GridView`'s `DataBind` method refreshes the data that the `GridView` displays.

Self-Review Exercises

20.1 State whether each of the following is *true* or *false*. If *false*, explain why.

a) Web Form file names end in `.aspx`.
b) `App.config` is a file that stores configuration settings for an ASP.NET web application.
c) A maximum of one validation control can be placed on a Web Form.
d) A `LinqDataSource` control allows a web application to interact with a database.

20.2 Fill in the blanks in each of the following statements:

a) Web applications contain three basic tiers: _____, _____, and _____.
b) The _____ web control is similar to the `ComboBox` Windows control.
c) A control which ensures that the data in another control is in the correct format is called a(n) _____.
d) A(n) _____ occurs when a page requests itself.
e) Every ASP.NET page inherits from class _____.
f) The _____ file contains the functionality for an ASP.NET page.

Answers to Self-Review Exercises

20.1 a) True. b) False. `Web.config` is the file that stores configuration settings for an ASP.NET web application. c) False. An unlimited number of validation controls can be placed on a Web Form. d) True.