1

The Online Diary and Organizer

By the end of this chapter you'll have created an online diary, organizer, and contacts manager. So what exactly does the online diary and organizer do? Using a calendar-based interface it allows you to add, delete, and edit a diary entry for any day. It also allows you to create events: for example, to keep a note of your rich Uncle Bob's birthday — wouldn't want to forget that, would you? It's not just limited to birthdays, but any event: meetings, appointments, and so on.

The system has a basic username and password logon system, so that only you and no one else can view your own diary. This is what differentiates it from a blog. This system is a private diary and contacts manager — a place to put all those thoughts and comments you'd rather not have the world see. Unlike a blog, where you want the world to see it!

This whole project demonstrates the power of ASP.NET 2.0 and how easy it makes creating projects like this. Gone are the days of hundreds of lines of code to do security logons, create new users, and so on. This chapter employs the new security components of ASP.NET 2.0 to show just how easy it is to create fun, exciting, and useful projects.

The first section takes you through using the diary and its main screens. Then, the "Design of the Online Diary" section walks you through an overview of the system's design. After that you get into the nuts and bolts of the system and how it all hangs together. In the final section, you set up the diary.

Using the Online Diary

Each user has his or her own online diary; to access it requires logging on. Enter username `user5` with the password `123!abc` to log in as a test user. The log on screen is shown in Figure 1-1.

Although the screenshot may suggest lots of controls and lots of code to make the security function, in fact with the new security controls in ASP.NET 2.0 it's very easy and not much work at all.

If you have not registered, a link will take you to the Sign Up page, depicted in Figure 1-2.

Figure 1-1

Figure 1-2

This shows another of the new security controls in ASP.NET 2.0; creating a registration process is now just a matter of adding a control to a form!

If you've forgotten your password, you can click the Forgotten Your Password? link, which directs you to the Password Reminder wizard pages (see Figure 1-3).

Figure 1-3

Having logged on, you arrive at the main diary page, as displayed in Figure 1-4.

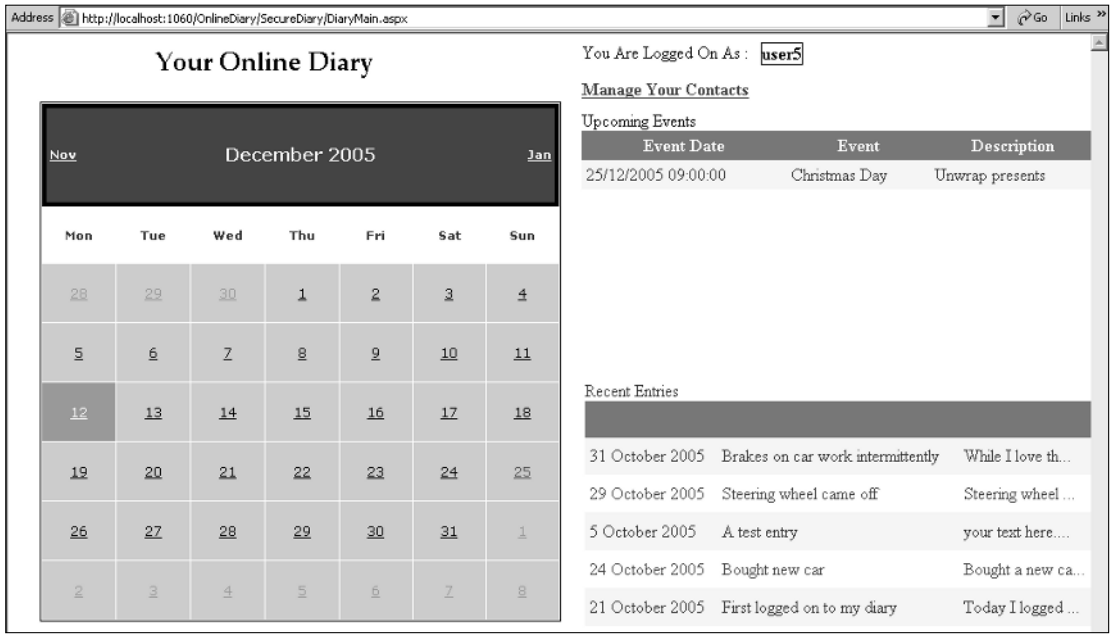

Figure 1-4

On this page you see a monthly calendar. Days with diary entries are marked with a blue background. Days with events are marked in red text. Notice also on the right that upcoming events are highlighted, as are recent diary entries.

Clicking on a day moves you through to the area where you can enter your diary entry for that day; and add, edit, and delete events (see Figure 1-5).

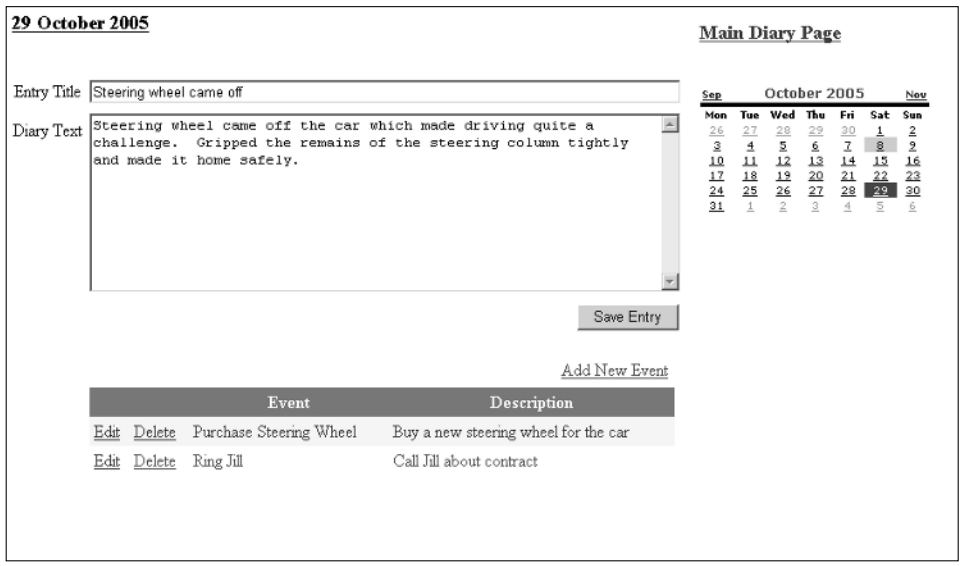

Figure 1-5

You can also navigate your diary from here via the small calendar to the right.

Adding a diary entry simply involves typing in the Entry Title and Diary Text boxes and clicking the Save Entry button.

Events happening on a particular day are listed in the Events table at the bottom-left of Figure 1-5. You can edit and delete events, or click the Add New Event link to add a new event. The Edit and Add event pages are almost identical in look. An example of the Edit Event page is shown in Figure 1-6.

Figure 1-6

In the Edit Event page, you can set the event's name, include a brief description, what time the event starts, and how long it lasts.

Returning to the main diary page (refer to Figure 1-4) you'll see a Manage Your Contacts link, as shown in Figure 1-7.

Figure 1-7

Clicking that link takes you to the Contact Management page (see Figure 1-8).

Figure 1-8

Here you see a list of your contacts, which you can edit and delete by clicking the appropriate link in the Contacts table. You can also add a new contact by clicking the Add New Contact link, which takes you to the New Contact page (no surprise there!), shown in Figure 1-9.

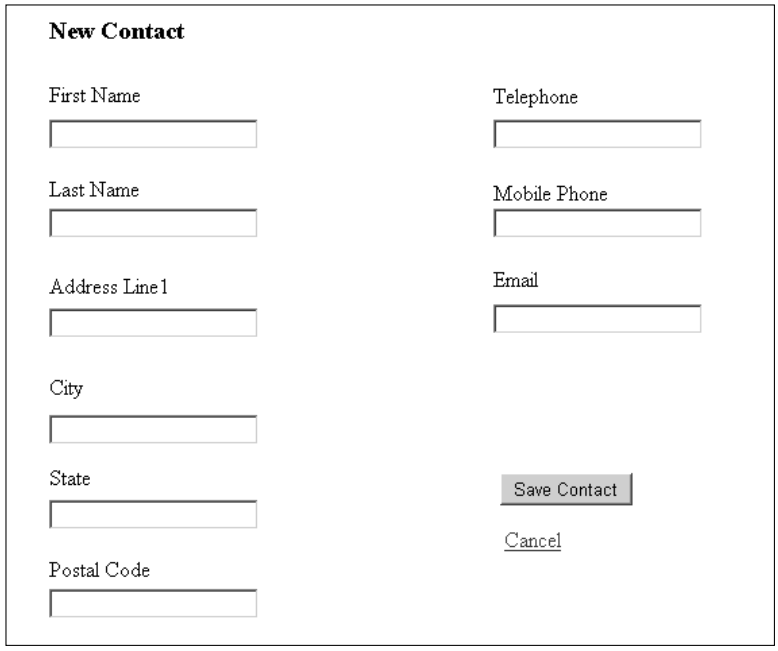

Figure 1-9

Currently the contacts functionality is fairly simple, with such things as linking events and contacts and automatically e-mailing contacts to remind them of an event.

So you've seen what the Online Diary does, now you can look at how it does it! The next section describes the overall design and how the system hangs together. You get a high-level tour of the database setup and each of the classes the system uses.

Design of the Online Diary

The diary system is split into the common three-layer architecture. All data and direct data modifying code are in the data layer, a combination of database tables and stored procedures. The data access layer is examined next.

Above the data access layer is the business layer providing all the rules and intelligence of the system. The business layer has been coded as seven classes, which you tour through shortly.

Finally, the bit the user sees is the presentation layer, consisting of a number of .aspx files that utilize the business and data access layers to create the diary's interface. This layer is discussed in the last part of this section.

The Data Access Layer

The Online Diary uses a SQL Server 2005 Express database. However, there's no reason why this couldn't be changed to work with other databases. If the database supports stored procedures, then in theory all that's needed is a change of connection string and creation of stored procedures matching those in the current SQL Server database. If the database doesn't support stored procedures — for example, MS Access — changes to class code would be necessary but not difficult.

Figure 1-10 shows the tables in the Online Diary database (DiaryDB).

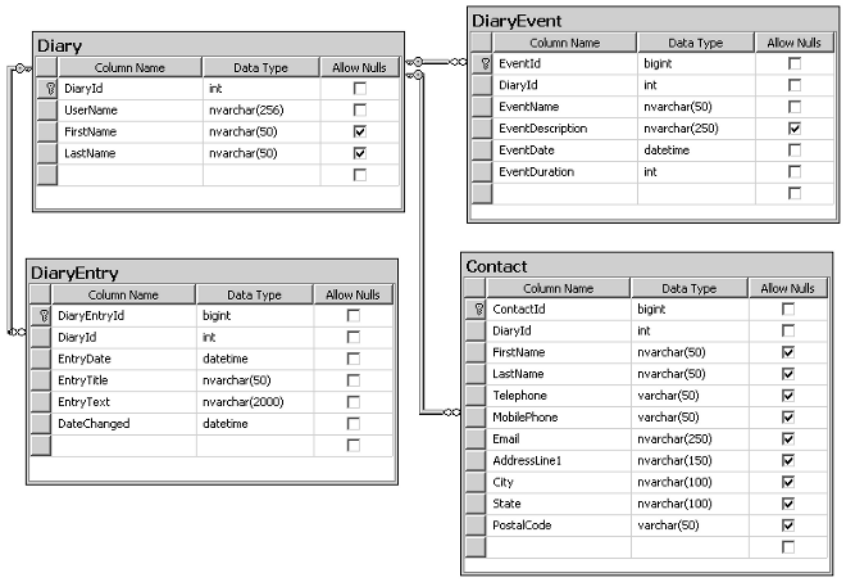

Figure 1-10

The default database created using the new membership features of ASP.NET 2.0 is also used. The database is a SQL Server Express database and not modified from the one created by Visual Studio Express. However, to link the log on and the diary details, the UserName field in the DiaryDB database takes its value originally from the membership database. You go through this in more detail shortly. Membership details are contained in the ASPNETDB database that Visual Web Developer Express creates for you. Although it contains quite a few tables, you never access them via the code in this project. It's accessed exclusively by the new `Login` controls — it does all the hard work behind the scenes!

This project only makes use of the aspnet_Users table, shown in Figure 1-11, to provide log on security checking and provide a username for the main DiaryDB. You may well want to extend the membership database to include extra functionality such as personalizing the user experience or providing different levels of membership (admin, user, operator), among other things.

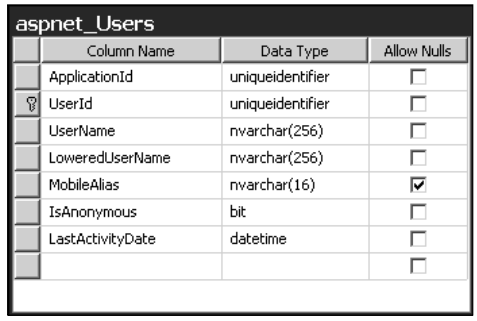

Figure 1-11

The tables of the main Online Diary database and their roles are listed in the following table:

Table Name	Description
Diary	Contains details of all Online Diary users, their DiaryId, and names.
DiaryEntry	Contains all the diary entries for all diary users.
DiaryEvent	Contains all the diary events for all diary users.
Contact	Holds the details of all contacts for the diaries.

The key that links all of the tables together is the DiaryId field. It's the primary key field in the Diary table and a foreign key field in all the other tables. Why not use the UserName field? Basically speed — it's easier and therefore faster for the database to do joins and searches on an integer field than it is on character-based fields.

All access to the database is via a stored procedure. The naming convention is simply as follows:

```
ActionThingThisActionRelatesTo
```

Consider this very simple stored procedure:

```
DeleteContact
```

Rather unsurprisingly, `DeleteContact` deletes a contact from the database. The naming convention means the purpose of each stored procedure doesn't need a lot of explanation. As the code is discussed, you look at the stored procedures in more detail where necessary.

The Business Layer

The business layer is organized into seven classes. The four main classes are as follows:

- `OnlineDiary`
- `DiaryEntry`
- `DiaryEvent`
- `Contact`

These classes do most of the work of temporarily holding diary-related data and retrieving and storing it in the database. There are also three collection classes. The first order of business is the `OnlineDiary` class.

The OnlineDiary Class

This class contains only two shared public methods, detailed in the following table:

Method	Return Type	Description
`InsertDiary(ByVal UserName As String, ByVal FirstName As String, ByVal LastName As String)`	None	Inserts a new diary user into the OnlineDiary database.
`GetDiaryIdFromUserName(ByVal UserName As String)`	Integer	Looks up the UserName in the database and returns the associated DiaryId.

The purpose of the `OnlineDiary` class is simply to provide a couple of handy shared methods relating to an online diary as a whole. It could also be used to expand the diary system and add new functionality that relates to the overall diary system, rather than a specific part such as contacts.

The Contact Class

The `Contact` class objectifies a single contact — a person or thing for which you want to store contact information. It encapsulates everything to do with contacts, including the storing and retrieving of contact information in the database.

It has two constructors, outlined in the following table:

Constructor	Description
`New(ByVal Diaryid as Integer)`	Creates a new `Contact` object with all properties set to their default values.
`New(ByVal ContactId As Long)`	Creates a new `Contact` object with its properties retrieved from the database using the argument `ContactId`.

Having created a `Contact` object, saving it involves simply calling the `Save()` method. The class will work out whether it's a new contact that needs to be inserted into the database, or an existing one that needs to be updated. In addition to the `Save()` method, the `Contacts` class contains two `Delete()` methods, as well as two `GetContacts()` methods, all of which are outlined in the following table:

Method	Return Type	Description
`Save()`	None	Saves a fully populated `Contact` object. If it's a new contact, `Save()` calls `InsertNewContact` sub, and the details are inserted into the database. The new ContactId is returned from the database and entered into `mContactId`. If the contact already exists in the database, `Save()` calls `UpdateContact`, which updates the database values with those in the `Contact` object.
`DeleteContact()`	None	Deletes from the database the `Contact` object with ContactId equal to `mContactId` of the object. `Contact` object's values are re-initialized to their defaults.
`DeleteContact(ByVal ContactId As Long)`	None	Shared method that deletes the `Contact` object from the database with a ContactId value equal to the `ContactId` argument of the method.
`GetContactsByFirstLetter(ByVal DiaryId As Integer,Optional ByVal FirstLetterOfSurname As Char)`	`SqlDataReader`	Shared method that returns a `SqlDataReader`object populated with a list of contacts whose surname's first letter matches the `FirstLetterOfSurname` argument.This argument is optional; if left off, all `Contact` objects regardless of surname's first letter are included in the DataSet's rows.

Table continued on following page

Method	Return Type	Description
`GetContactsByFirstLetterAsCollection(ByVal DiaryId As Integer, Optional ByVal FirstLetterOfSurname As Char)`	`SqlDataReader`	Shared method that returns a `ContactCollection` object populated with `Contact` objects whose surname's first letter matches the `FirstLetterOfSurname` argument. This argument is optional; if left off, all `Contact` objects regardless of surname's first letter are included in the DataSet's rows.

Finally, the `Contact` class contains the following properties:

Property	Type	Description
`ContactId`	`Long`	Each contact is represented by a unique ID. The ID is auto-generated by the Contact table in the database whenever a new contact is inserted.
`FirstName`	`String`	Contact's first name.
`LastName`	`String`	Contact's surname.
`Email`	`String`	Contact's e-mail address.
`Telephone`	`String`	Contact's telephone number.
`MobilePhone`	`String`	Contact's mobile phone number.
`AddressLine1`	`String`	Contact's house name and street address.
`City`	`String`	Contact's city of residence.
`State`	`String`	Contact's state.
`PostalCode`	`String`	Contact's zip or postal code.

The ContactCollection Class

The `ContactCollection` class inherits from the `System.Collections.CollectionBase` class. The `ContactCollection` class's purpose is simply to store a collection of `Contact` objects. This class gets extensive use in the next chapter, when you create a contacts organizer.

The `ContactCollection` class has only one property:

Property	Type	Description
`Item(ByVal Index As Integer)`	`Integer`	Returns the `Contact` object stored at the position in index in the collection.

The `ContactCollection` class's public methods are as follows:

Method	Return Type	Description
`Add(ByVal NewContact As Contact)`	None	Adds a `Contact` object to the collection held by the `ContactCollection` object.
`Add(ByVal ContactId As Long)`	None	Creates a new `Contact` object. `ContactId` is passed to the `Contact` object's constructor to ensure it's populated with the contact's details from the database. The new `Contact` object is then added to the collection maintained by the `ContactCollection` object.
`Remove(ByVal Index as Integer)`	None	Removes the `Contact` object from the collection at the specified index.

That deals with the `Contact` classes; now take a look at the two classes dealing with diary entries.

The DiaryEntry Class

The `DiaryEntry` class objectifies a single entry in a diary. It encapsulates everything to do with diary entries, including creating, updating, and retrieving diary entry data. It handles all the database access for diary entries.

It has three constructors, outlined in the following table:

Constructor	Description
`New(ByVal DiaryId as Integer)`	Creates a new `DiaryEntry` object with all properties set to their default values.
`New(ByVal DiaryEntryId As Long)`	Creates a new `DiaryEntry` object with its properties retrieved from the database using the argument `DiaryEntryId`.
`New(ByVal DiaryId AS Integer, ByVal EntryDate As Date)`	Creates a new `DiaryEntry` object with its properties retrieved from the database using the arguments `DiaryId` and `EntryDate`.

Having created a `DiaryEntry` object, saving it involves simply calling the `Save()` method. As with the `Save()` method of the `Contacts` class, the `DiaryEntry` class will work out whether it's a new diary entry that needs to be inserted into the database, or an existing entry that needs to be updated. As well as enabling retrieval of one diary entry's details, the `DiaryEntry` class provides additional methods for getting details of a number of diary entries as either a collection or as a DataSet by returning a `sqlDataReader` object. The methods of this class are explained in the following table:

Method	Return Type	Description
`Save()`	None	Saves a fully populated `DiaryEntry` object. If it's a new entry, `Save()` calls `InsertNewDiaryEntry` sub and the details are inserted in to the database. The new `DiaryEntryId` is returned from the database and entered in to `mDiaryEntryId`. If the entry already exists in the database, `Save()` calls `UpdateContact`, which updates the database values with those in the `DiaryEntry` object.
`GetDaysInMonthWithEntries(ByVal DiaryId As Integer, ByVal Month As Integer, ByVal Year As Integer)`	Boolean Array	Shared method that returns a Boolean array detailing which days have a diary entry associated with them. The array index matches with the day of the month (1 is the first of the month, 2 the second, and so on).
`GetDiaryEntriesByDate(ByVal DiaryId As Integer, ByVal FromDate As Date, ByVal ToDate As Date)`	`SqlDataReader`	Shared method that returns a `SQLDataReader` object populated with rows from the database detailing diary entries between the `FromDate` and `ToDate` arguments.

Method	Return Type	Description
`GetDiaryEntriesByDateAsCollection(ByVal DiaryId As Integer, ByVal FromDate As Date, ByVal ToDate As Date)`	`DiaryEntryCollection`	Creates a new `DiaryEntryCollection` object and populates it with `DiaryEntry` objects whose `EntryDate` is between the `FromDate` and `ToDate` arguments.
`GetDiaryEntriesRecentlyChanged(ByVal DiaryId As Integer)`	`SqlDataReader`	Returns a `SqlDataReader` containing a DataSet of diary entries recently created.

In addition to the constructors and methods, the `DiaryEntry` class contains the following properties:

Property	Type	Description
`EntryTitle`	`String`	Title for the day's diary entry.
`EntryText`	`String`	Text of the day's diary entry.
`EntryDate`	`Date`	Date the entry was posted.

The other class dealing with diary entries is the `DiaryEntryCollection` class, which is explained next.

The DiaryEntryCollection Class

The `DiaryEntryCollection` class inherits from the `System.Collections.CollectionBase` class. Its purpose is simply to store a collection of `DiaryEntry` objects.

This class contains only one property, described in the following table:

Property	Type	Description
`Item(ByVal Index As Integer)`	`Integer`	Returns the `DiaryEntry` object stored at the specified position in index in the collection.

Along with the `Item()` property, the `DiaryEntryCollection` class has three public methods:

Method	Return Type	Description
`Add(ByVal New DiaryEntry As DiaryEntry)`	None	Adds a `DiaryEntry` object to the collection held by the `DiaryEntryCollection` object.
`Add(ByVal DiaryEntryId As Long)`	None	Creates a new `DiaryEntry` object. `DiaryEntryId` is passed to the `DiaryEntry` object's constructor to ensure it's populated with the diary entry's details from the database. The new `DiaryEntry` object is then added to the collection maintained by the `DiaryEntryCollection` object.
`Remove(ByVal Index as Integer)`	None	Removes the `DiaryEntry` object from the collection at the specified index.

So far the classes dealing with contacts and diary entries have been discussed. The next section discusses the diary events.

The DiaryEvent Class

The `DiaryEvent` class objectifies a single entry in a diary. It encapsulates everything to do with diary entries, including creating, updating, and retrieving diary events data. It handles all the database access for diary events.

The `DiaryEvent` class has three constructors, outlined as follows:

Constructor	Description
`New(ByVal Diaryid as Integer)`	Creates a new `DiaryEvent` object with all properties set to their default values.
`New(ByVal EntryId As Long)`	Creates a new `DiaryEvent` object with its properties retrieved from the database using the argument `EventId`.
`New(ByVal DiaryId AS Integer, ByVal EventDate As Date)`	Creates a new `DiaryEvent` object with its properties retrieved from the database using a combination of the arguments `DiaryId` and `EventDate`.

Having created a `DiaryEvent` object, saving it involves simply calling the `Save()` method. The class will work out whether it's a new diary event to insert into the database, or an existing one in need of updating. The `DiaryEvent` class also has two `Delete()` methods. One is a shared method and therefore doesn't require a `DiaryEvent` to be created, and requires an `EventId` parameter. It's used by some of the built-in data access components provided with ASP.NET 2.0. The second is an object method that deletes the event referenced by the current `DiaryEvent` object. As well as enabling the details of one diary entry to be retrieved, the `DiaryEvent` class provides additional methods for getting details of a number of diary events as either a collection or as a DataSet by returning a `SqlDataReader` object.

The following table explains these methods in detail:

Method	Return Type	Description
`Save()`	None	Saves a fully populated `DiaryEvent` object. If it's a new entry, `Save()` calls `InsertNew DiaryEvent` sub and the details are inserted into the database. The new `EventId` is returned from the database and entered in to `mEventId`. If the entry already exists in the database, `Save()` calls `UpdateDiaryEvent`, which updates the database values with those in the `DiaryEvent` object.
`GetDaysInMonthWithEvents(ByVal DiaryId As Integer, ByVal Month As Integer, ByVal Year As Integer)`	Boolean Array	Shared method that returns a Boolean array detailing which days have events associated with them. The array index matches with the day of the month (1 is the first of the month, 2 the second, and so on).
`GetDiaryEventsByDate(ByVal DiaryId As Integer, ByVal FromDate As Date, ByVal ToDate As Date)`	`SqlDataReader`	Shared method that returns a `SqlDataReader` object populated with rows from the database detailing diary events between the `FromDate` and `ToDate` arguments.
`GetDiaryEventsByDateAsCollection(ByVal DiaryId As Integer, ByVal FromDate As Date, ByVal ToDate As Date)`	`DiaryEventCollection`	Creates a new `Diary EventCollection` object and populates it with `DiaryEvent` objects whose `EntryDate` is between the `FromDate` and `ToDate` arguments.

Table continued on following page

Method	Return Type	Description
`DeleteEvent()`	None	Deletes from the database the event with `EventId` equal to `mEventId` of the object. The `DiaryEvent` object's values are re-initialized to their defaults.
`DeleteEvent(ByVal EventId As Long)`	None	Shared method that deletes the event from the database with an `EventId` value equal to the `EventId` argument of the method.

In addition to the constructors and public methods, the `DiaryEvent` class has these four properties:

Property	Type	Description
`EventDescription`	String	Description of the event.
`EventName`	String	Short name for the event.
`EventDate`	Date	Date the event starts.
`EventDuration`	Integer	Length of time in minutes that the event lasts.

One more class to go. The next section looks at the diary collection handling class: `DiaryEventCollection`.

The DiaryEventCollection Class

The `DiaryEventCollection` class inherits from the `System.Collections.CollectionBase` class. Its purpose is simply to store a collection of `DiaryEvent` objects. The class employs the following methods:

Method	Return Type	Description
`Add(ByVal NewDiaryEvent As DiaryEvent)`	None	Adds a `DiaryEvent` object to the collection held by the `DiaryEventCollection` object.
`Add(ByVal DiaryEventId As Long)`	None	Creates a new `DiaryEvent` object. `DiaryEventId` is passed to the `DiaryEvent` object's constructor to ensure it's populated with the event's details from the database. The new `DiaryEvent` object is then added to the collection maintained by the `DiaryEventCollection` object.
`Remove(ByVal Index As Integer)`	None	Removes the `DiaryEvent` object from the collection at the specified index.

This class contains only one property:

Property	Type	Description
`Item(ByVal Index As Integer)`	Integer	Returns the `DiaryEvent` object stored at the position in index in the collection.

That completes an overview of all the classes and their design, methods, and properties. The next section takes a more in-depth look at the code and the .aspx pages dealing with presentation.

Code and Code Explanation

This section digs into each of the important pages and shows you how they interact with each other, as well as how they use the classes in the business layer. This section doesn't cover every single line of every page, but rather it takes a general overview of how the application works and dives a bit deeper where necessary.

Discussion of the project is approached in a functionality-based way. Instead of discussing a specific page and what it does, the following sections discuss a process — such as registration — and how it's achieved.

It begins with an overview of the files and file structure.

File Structure

An overview of the file structure is shown in Figure 1-12.

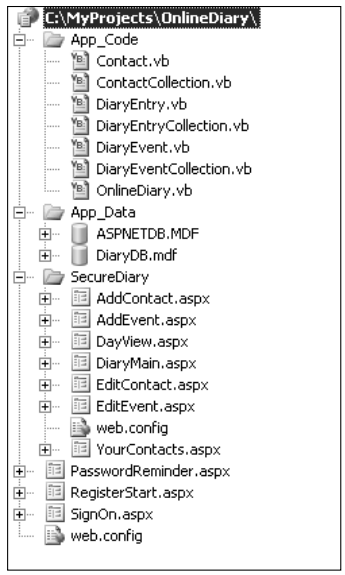

Figure 1-12

Each of the seven class files is stored in the App_Code directory (at the top of the figure). The App_Data directory contains the two databases: the login database (ASPNETDB.MDF) and the Online Diary database (DiaryDB.mdf). Pages that require you to log in before viewing are stored separately in the SecureDiary directory. Finally, the root directory contains login pages, registration pages, and password reminder pages; basically anything that requires you to be logged in to view.

Registration, Logging On, and Security

The Online Diary application uses the new `Login` controls to provide the diary's user handing features, including new user registration, log in, and password reminder.

The `Login` controls are a real time saver, allowing a lot of sophisticated functionality to be added with just a little work and hardly any code! ASP.NET 2.0 has seven new security or login controls:

- `Login`: Enables users to log in and verifies username and password.
- `LoginView`: Enables the display of different templates depending on whether a user is logged in and also his or her role membership.
- `PasswordRecovery`: Provides password reminder functionality for users who forget their password.
- `LoginStatus`: Displays whether a user is logged in or out.
- `LoginName`: Displays currently logged-in username.
- `CreateUserWizard`: Creates a new user wizard — registration of a new user in simple steps.
- `ChangePassword`: Enables users to change their password.

The Online Diary project, however, use only the `Login`, `LoginName`, `CreateUserWizard`, and `ChangePassword` controls.

Logging On

The SignOn.aspx page contains a `Login` control. The user database is created using the web site administration tools. This goes through the steps needed one by one, and once it's finished a new database called ASPNETDB.MDF appears in the App_Data directory of the diary project.

The markup for the `Login` control is shown here:

```
<asp:Login ID="Login1" runat="server" BackColor="#F7F6F3" BorderColor="#E6E2D8"
BorderPadding="4"
                        BorderStyle="Solid" BorderWidth="1px" CreateUserText="Not
registered?  Click here to register now."
                        CreateUserUrl="~/RegisterStart.aspx"
DestinationPageUrl="~/SecureDiary/DiaryMain.aspx" Font-Names="Verdana"
                        Font-Size="0.8em" ForeColor="#333333" Height="197px"
PasswordRecoveryText="Forgotten your password?"
                        PasswordRecoveryUrl="~/PasswordReminder.aspx" Style="z-
index: 100; left: 78px;
                        position: absolute; top: 55px" Width="315px">
                        <LoginButtonStyle BackColor="#FFFBFF" BorderColor="#CCCCCC"
BorderStyle="Solid" BorderWidth="1px"
```

```
                              Font-Names="Verdana" Font-Size="0.8em"
ForeColor="#284775" />
                          <TextBoxStyle Font-Size="0.8em" />
                          <TitleTextStyle BackColor="#5D7B9D" Font-Bold="True" Font-
Size="0.9em" ForeColor="White" />
                          <InstructionTextStyle Font-Italic="True" ForeColor="Black"
/>

                      </asp:Login>
```

Important attributes to note are `DestinationPageUrl`, which determines where the user is navigated to if he or she enters a valid username and password. In the Online Diary project it's the Diarymain.aspx page, the center of the Online Diary's interface.

To enable new users to register, the `CreateUserText` has been set to a friendly "register here" message; the URL for registering is specified in `CreateUserUrl`.

Finally, just in case the user has already registered but forgotten his or her password, the `PasswordRecoveryText` attribute displays a "Forgotten your password?" message and `PasswordRecoveryUrl` sets the URL the users are navigated to if they need to find out their password.

The only code you need to write is in the `Login` control's `LoggedIn` event, which fires if the user successfully enters a username and password:

```
    Protected Sub Login1_LoggedIn(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Login1.LoggedIn
        Dim DiaryId As Integer = GetDiaryIdFromUserName(Login1.UserName)
        Session("DiaryId") = DiaryId
    End Sub
```

This uses the supplied username to look up the user's `DiaryId` in the Online Diary database. This is then stored in the session variable.

The SignOn.aspx page also allows new users to register.

New User Registration

The RegisterStart.aspx. page deals with the registration of a new user. As with SignOn.aspx, this page also uses one of the new `Login` controls, this time the `CreateUserWizard` control. The markup for the `CreateUserWizard` control is shown in the following code:

```
        <asp:CreateUserWizard ID="CreateUserWizard1" runat="server"
BackColor="#F7F6F3" BorderColor="#E6E2D8"
            BorderStyle="Solid" BorderWidth="1px" Font-Names="Verdana" Font-
Size="0.8em"
            Style="z-index: 100; left: 66px; position: absolute; top: 43px"
Height="164px" Width="300px" FinishDestinationPageUrl="~/SignOn.aspx">
            <SideBarStyle BackColor="#5D7B9D" BorderWidth="0px" Font-Size="0.9em"
VerticalAlign="Top" />
            <SideBarButtonStyle BorderWidth="0px" Font-Names="Verdana"
ForeColor="White" />
```

```
            <NavigationButtonStyle BackColor="#FFFBFF" BorderColor="#CCCCCC"
BorderStyle="Solid"
                BorderWidth="1px" Font-Names="Verdana" ForeColor="#284775" />
            <HeaderStyle BackColor="#5D7B9D" BorderStyle="Solid" Font-Bold="True"
Font-Size="0.9em"
                ForeColor="White" HorizontalAlign="Left" />
            <CreateUserButtonStyle BackColor="#FFFBFF" BorderColor="#CCCCCC"
BorderStyle="Solid"
                BorderWidth="1px" Font-Names="Verdana" ForeColor="#284775" />
            <ContinueButtonStyle BackColor="#FFFBFF" BorderColor="#CCCCCC"
BorderStyle="Solid"
                BorderWidth="1px" Font-Names="Verdana" ForeColor="#284775" />
            <StepStyle BorderWidth="0px" />
            <TitleTextStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White"
/>
            <WizardSteps>
                <asp:CreateUserWizardStep runat="server">
                </asp:CreateUserWizardStep>
                <asp:WizardStep ID="personalDetailsStep" runat="server" Title="User
Details">
                        <table border="0" style="font-size: 100%; font-family:
Verdana; z-index: 100; left: 0px; position: absolute; top: 0px;">
                            <tr>
                                <td align="center" colspan="2" style="font-weight:
bold; color: white; background-color: #5d7b9d">
                                    Your Personal Details</td>
                            </tr>
                            <tr>
                                <td align="right" style="height: 26px">
                                    <label for="UserName">
                                        Your First Name:</label></td>
                                <td style="width: 179px; height: 26px">
                                    <asp:TextBox ID="firstNameTextBox"
runat="server" CausesValidation="True"></asp:TextBox> 
                                </td>
                            </tr>
                            <tr>
                                <td align="right">
                                    <label for="Password">
                                        Your Last Name:</label></td>
                                <td style="width: 179px">
                                    <asp:TextBox ID="lastNameTextBox"
runat="server" CausesValidation="True"></asp:TextBox> 
                                </td>
                            </tr>
                            <tr>
                                <td align="center" colspan="2" style="height:
18px">
                                     </td>
                            </tr>
                            <tr>
                                <td align="center" colspan="2" style="color: red">
                                     </td>
                            </tr>
```

```
                        </table>
                    </asp:WizardStep>
                    <asp:CompleteWizardStep runat="server">
                        <ContentTemplate>
                            <table border="0" style="font-size: 100%; width: 383px;
font-family: Verdana; height: 164px">
                                <tr>
                                    <td align="center" colspan="2" style="font-weight:
bold; color: white; background-color: #5d7b9d">
                                        Complete</td>
                                </tr>
                                <tr>
                                    <td>
                                        Your account has been successfully
created.</td>
                                </tr>
                                <tr>
                                    <td align="right" colspan="2">
                                        <asp:Button ID="ContinueButton" runat="server"
BackColor="#FFFBFF" BorderColor="#CCCCCC"
                                            BorderStyle="Solid" BorderWidth="1px"
CausesValidation="False" CommandName="Continue"
                                            Font-Names="Verdana" ForeColor="#284775"
Text="Continue" ValidationGroup="CreateUserWizard1" />
                                    </td>
                                </tr>
                            </table>
                        </ContentTemplate>
                    </asp:CompleteWizardStep>
                </WizardSteps>
            </asp:CreateUserWizard>
```

Most of the markup and attributes relate to style settings. However, one essential attribute is the `FinishDestinationPageUrl`. This is where the user is taken once the registration process is completed; in the Online Diary it's the SignOn.aspx page.

You've probably noticed a number of `WizardStep` tags in the markup, such as this one:

```
<asp:WizardStep ID="personalDetailsStep" runat="server" Title="User Details">
```

The `CreateUserWizard` works on a step-by-step basis. There must be least one step that allows the user to choose a username and password and various security questions (see Figure 1-13).

This step and its style can be modified, but Figure 1-13 shows its default value. The control takes care of inserting the new user data into the user database.

A second step, shown in Figure 1-14, is displayed after the user is created.

Figure 1-13

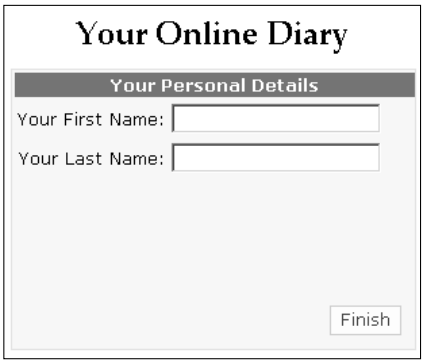

Figure 1-14

This screen asks users for their first name and last name. This time it's up to you to store the data somewhere, and you do that in the `CreateUserWizard` control's `FinishButtonClick` event:

```
    Protected Sub CreateUserWizard1_FinishButtonClick(ByVal sender As Object, ByVal
e As System.Web.UI.WebControls.WizardNavigationEventArgs) Handles
CreateUserWizard1.FinishButtonClick
        Dim myTextBox As TextBox
        Dim UserName, FirstName, LastName
        myTextBox = CreateUserWizard1.FindControl("firstNameTextBox")
        FirstName = myTextBox.Text
        myTextBox = CreateUserWizard1.FindControl("lastNameTextBox")
        LastName = myTextBox.Text
        UserName = CreateUserWizard1.UserName
        OnlineDiary.InsertDiary(UserName, FirstName, LastName)
    End Sub
```

This step creates a new diary for users and stores their first and last names. The `UserName` comes from the `CreateUserWizard` control's `UserName` property, and then uses the shared method `InsertDiary()` to insert the new user in the Online Diary's database.

Being human, sometimes people forget their passwords. Fortunately, ASP.NET 2.0 comes with the capability to refresh overloaded memories.

Password Reminder

Again with virtually no code, you can create a fully functional password reminder feature for the Online Diary, this time courtesy of the `PasswordRecovery` control. Virtually all of its settings are at the default values or simply related to style. Even better, there's just one line of code and that's in the `SendingMail` event:

```
Protected Sub PasswordRecovery1_SendingMail(ByVal sender As Object, ByVal e As
System.Web.UI.WebControls.MailMessageEventArgs) Handles
PasswordRecovery1.SendingMail
        returnToLogOnHyperLink.Visible = True
    End Sub
```

The `SendingMail` event fires when the user presses the Send Email button and simply displays the Return to Main Page link, rather than leaving the user guessing as to where to go next.

The main work involved is configuring the SMTP server settings that'll be used to actually send the password reminder e-mail. Visual Web Developer doesn't come with an SMTP server. However, if you are using Windows XP or 2000, all you need to do to install one is go to the Start➪Settings➪Control Panel➪Add or Remove Programs. From there, select Add/Remove Windows Components. Select the Internet Information Server (IIS) option and click Details at the bottom-right of the dialog. In the resulting dialog box, you'll see a list. Check the box next to SMTP Service and click OK. Then click Next to install an SMTP service.

Once the SMTP service is installed, add the following shaded code between the `<configuration>` tags in the Web.config file:

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
 <connectionStrings>
  <add name="DiaryDBConnectionString" connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\DiaryDB.mdf;Integrated
Security=True;User Instance=True"
   providerName="System.Data.SqlClient" />
 </connectionStrings>
  <system.web>
    <roleManager enabled="true" />
  <authentication mode="Forms"/>
        <compilation debug="true"/></system.web>
 <system.net>
  <mailSettings>
   <smtp from="system@diary-system.com">
    <network host="localhost" password="" userName="" />
   </smtp>
  </mailSettings>
 </system.net>
</configuration>
```

Viewing the Online Calendar

The DiaryMain.aspx page is the central hub of the application. It displays a calendar of the current month, showing which days have events or diary entries associated with them. It also displays a list of upcoming events and diary entries for the current month.

To display when a day has events or a diary entry, the `OnDayRender` event of the `Calendar` control is used:

```
    Protected Sub Calendar1_OnDayRender(ByVal sender As Object, ByVal e As
System.Web.UI.WebControls.DayRenderEventArgs) Handles Calendar1.DayRender
        If Not e.Day.IsOtherMonth Then
            If entryArrayOfDays Is Nothing Then
                entryArrayOfDays = GetDaysInMonthWithEntries(Session("DiaryId"),
e.Day.Date.Month, e.Day.Date.Year)
            End If

            If eventArrayOfDays Is Nothing Then
                eventArrayOfDays = GetDaysInMonthWithEvents(Session("DiaryId"),
e.Day.Date.Month, e.Day.Date.Year)
            End If

            If entryArrayOfDays(CInt(e.Day.DayNumberText)) Then
                e.Cell.BackColor = Drawing.Color.Blue
            End If

            If eventArrayOfDays(CInt(e.Day.DayNumberText)) Then
                e.Cell.ForeColor = Drawing.Color.Red
            End If

        End If
    End Sub
```

The first `If` block in the preceding event code deals with ensuring `entryArrayOfDays` and `eventArrayOfDays` are populated with details of which days have an associated event or diary entry. They are both Boolean arrays; if a day has an event or entry, the array element for that day contains `True`. Arrays are populated by the `DiaryEnty` and `DiaryEvent` classes' shared functions `GetDaysInMonthWithEntries()` and `GetDaysInMonthWithEvents()`.

In the second `If` block of the event the code checks to see whether the day of the month being rendered has a diary event or diary entry. If there's an event, the day's text is set to red. If there's a diary entry the day's background is rendered in blue.

As well as a `Calendar` control, the main page also has two `GridView` controls (discussed a bit later). The upper one displays upcoming events; the lower one displays recent diary entries. Both `GridView` controls get their data from an `ObjectDataSource` control, new to ASP.NET 2.0. In the past, data source controls have interacted directly with the database. They are nice and easy to use — put on one a page, set a few properties, drop in a few data-aware controls, and away you go. However, that's not actually good coding practice. Splitting up the data access, business, and presentation layers is generally considered good practice, but means leaving behind nice and easy-to-use data source controls.

However, the new ObjectDataSource lets you have the best of both: easy-to-use data controls and use of classes to separate business, data, and presentation layers. Instead of connecting directly to a database, the `ObjectDataSource` takes its data from one of the classes. `diaryEntriesObjectDataSource` on DiaryMain.aspx, for example, takes its data from the `GetDiaryEntriesRecentlyChanged()` method of the `DiaryEntry` class, whose markup is shown here:

```
        <asp:ObjectDataSource ID="diaryEntriesObjectDataSource" runat="server"
SelectMethod="GetDiaryEntriesRecentlyChanged"
            TypeName="DiaryEntry">
            <SelectParameters>
                <asp:SessionParameter DefaultValue="-1" Name="DiaryId"
SessionField="DiaryId" Type="Int32" />
            </SelectParameters>
        </asp:ObjectDataSource>
```

The `TypeName` attribute specifies the class name to use, and the `SelectMethod` attribute specifies which method of that class will provide the data. `GetDiaryEntriesRecentlyChanged()` is a shared method, shown here:

```
    Public Shared Function GetDiaryEntriesRecentlyChanged(ByVal DiaryId As Integer)
As SqlDataReader
        Dim diaryDBConn As New SqlConnection(conString)
        Dim sqlString As String = "GetRecentDiaryEntries"
        Dim sqlCmd As New SqlCommand(sqlString, diaryDBConn)
        sqlCmd.CommandType = CommandType.StoredProcedure

        sqlCmd.Parameters.AddWithValue("@DiaryId", DiaryId)

        diaryDBConn.Open()
        Dim entrySQLDR As SqlDataReader =
sqlCmd.ExecuteReader(CommandBehavior.CloseConnection)
        sqlCmd = Nothing
        Return entrySQLDR
    End Function
```

The method returns a `SqlDataReader` object populated with the data the `ObjectDataSource` control will use.

Actually displaying the data is then just a matter of pointing a data-aware control at the `ObjectDataSource`:

```
            <asp:GridView ID="recentEntriesGridView" runat="server"
AutoGenerateColumns="False"
                Caption="Recent Entries" CaptionAlign="Left" CellPadding="4"
DataSourceID="diaryEntriesObjectDataSource"
                ForeColor="#333333" GridLines="None" Style="z-index: 105; left:
535px; position: absolute;
                top: 321px" Width="476px" Height="208px">
                <FooterStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White"
/>
                <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />
                <Columns>
                    <asp:BoundField DataField="EntryDate" />
                    <asp:BoundField DataField="EntryTitle" />
```

```
                    <asp:BoundField DataField="EntryText" />
                </Columns>
                <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>
```

In the `GridView` control's markup, the `DataSourceID` attribute specifies the source of the data, which is the `ObjectDataSource` control. In addition, the markup specifies which columns to display by setting `AutoGenerateColumns` to `False`. A final step is to create a list of columns:

```
                <Columns>
                    <asp:BoundField DataField="EntryDate" />
                    <asp:BoundField DataField="EntryTitle" />
                    <asp:BoundField DataField="EntryText" />
                </Columns>
```

As well as enabling the display of data, the `ObjectDataSource` control can also update, insert, and delete records from a database, as demonstrated shortly.

Creating, Editing, and Viewing a Diary Entry

The DayView.aspx page allows for diary editing. This page contains a simple form allowing you to enter title and diary entry details. It also displays any existing diary entry.

All of the hard work is done by use of the `DiaryEntry` class. Its `Page_Load` event creates a new `DiaryEntry` class, passing its constructor the current user's `DiaryId` and also the date the page refers to:

```
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
        mDiaryEntry = New DiaryEntry(CInt(Session("DiaryId")),
CDate(dayShownLabel.Text))
        changeDayCalendar.SelectedDate = CDate(dayShownLabel.Text)
        changeDayCalendar.VisibleDate = changeDayCalendar.SelectedDate
        If Not IsPostBack Then
            entryTextTextBox.Text = mDiaryEntry.EntryText
            entryTitleTextBox.Text = mDiaryEntry.EntryTitle
        End If
    End Sub
```

`mDiaryEntry` is a global variable used to hold the `DiaryEntry` object relating to the day being edited.

The constructor, shown in the following code, does all the hard work of actually getting the data:

```
    Public Sub New(ByVal DiaryId As Integer, ByVal EntryDate As Date)
        mDiaryId = DiaryId
        If mDiaryId > 0 Then
            Try

                Dim diaryDBConn As New SqlConnection(conString)
                Dim sqlString As String = "GetDiaryEntryByDate"
                Dim sqlCmd As New SqlCommand(sqlString, diaryDBConn)
                sqlCmd.CommandType = CommandType.StoredProcedure

                sqlCmd.Parameters.AddWithValue("@DiaryId", mDiaryId)
                sqlCmd.Parameters.AddWithValue("@EntryFromDate", EntryDate)
                sqlCmd.Parameters.AddWithValue("@EntryToDate", EntryDate)


                diaryDBConn.Open()
                Dim diaryEntrySQLDR As SqlDataReader =
sqlCmd.ExecuteReader(CommandBehavior.CloseConnection)
                sqlCmd = Nothing
                If diaryEntrySQLDR.Read() Then
                    mDiaryEntryId = CLng(diaryEntrySQLDR("DiaryEntryId"))
                    mEntryDate = CDate(diaryEntrySQLDR("EntryDate"))
                    mEntryTitle = diaryEntrySQLDR("EntryTitle").ToString
                    mEntryText = diaryEntrySQLDR("EntryText").ToString
                Else
                    mDiaryEntryId = -1
                    mEntryDate = EntryDate
                End If

                diaryEntrySQLDR.Close()
                diaryEntrySQLDR = Nothing
                diaryDBConn.Close()
                diaryDBConn = Nothing

            Catch ex As Exception
                mDiaryEntryId = -1
            End Try

        End If
    End Sub
```

The `GetDiaryEntryByDate` stored procedure is called to get the data. If there isn't an existing entry for that day, `mDiaryEntryId` is set to `-1` and all the other properties are left at their default values. Otherwise they are populated with the data from the database.

When the diary title or entry boxes are changed, `mDiaryEntry` is updated:

```
    Protected Sub entryTitleTextBox_TextChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles entryTitleTextBox.TextChanged
        mDiaryEntry.EntryTitle = entryTitleTextBox.Text
    End Sub

    Protected Sub entryTextTextBox_TextChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles entryTextTextBox.TextChanged
        mDiaryEntry.EntryText = entryTextTextBox.Text
    End Sub
```

Saving changes occurs when you click the Save button:

```
    Protected Sub saveDiaryEntryButton_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles saveDiaryEntryButton.Click
        mDiaryEntry.Save()
    End Sub
```

All that's involved is calling the `Save()` method of the `DiaryEntry` object:

```
    Public Sub Save()
        If mDiaryEntryId = -1 Then
            InsertNewDiaryEntry()
        Else
            UpdateDiaryEntry()
        End If
    End Sub
```

Based on whether or not `mDiaryEntryId` is `-1`, the method either inserts a new entry into the database or updates an existing one. The private method `InsertNewDiaryEntry()` inserts a new diary entry:

```
    Private Sub InsertNewDiaryEntry()
        If mDiaryId <> -1 Then
            Dim diaryDBConn As New SqlConnection(conString)
            Dim sqlString As String = "InsertDiaryEntry"
            Dim sqlCmd As New SqlCommand(sqlString, diaryDBConn)
            sqlCmd.CommandType = CommandType.StoredProcedure

            sqlCmd.Parameters.AddWithValue("@DiaryId", mDiaryId)
            sqlCmd.Parameters.AddWithValue("@EntryDate", mEntryDate)
            sqlCmd.Parameters.AddWithValue("@EntryTitle", mEntryTitle)
            sqlCmd.Parameters.AddWithValue("@EntryText", mEntryText)
            sqlCmd.Parameters.Add("@NewDiaryEntryId", SqlDbType.BigInt)
            sqlCmd.Parameters("@NewDiaryEntryId").Direction =
ParameterDirection.ReturnValue

            diaryDBConn.Open()
            sqlCmd.ExecuteNonQuery()
            mDiaryEntryId = CLng(sqlCmd.Parameters("@NewDiaryEntryId").Value())

            diaryDBConn.Close()
            sqlCmd = Nothing
```

```
            diaryDBConn = Nothing
        End If
    End Sub
```

The private method `UpdateDiaryEntry()` updates it:

```
    Private Sub UpdateDiaryEntry()
        If mDiaryEntryId <> -1 Then
            Dim diaryDBConn As New SqlConnection(conString)
            Dim sqlString As String = "UpdateDiaryEntry"
            Dim sqlCmd As New SqlCommand(sqlString, diaryDBConn)
            sqlCmd.CommandType = CommandType.StoredProcedure

            sqlCmd.Parameters.AddWithValue("@DiaryEntryId", mDiaryEntryId)
            sqlCmd.Parameters.AddWithValue("@EntryDate", mEntryDate)
            sqlCmd.Parameters.AddWithValue("@EntryTitle", mEntryTitle)
            sqlCmd.Parameters.AddWithValue("@EntryText", mEntryText)

            diaryDBConn.Open()
            sqlCmd.ExecuteNonQuery()
            diaryDBConn.Close()
            sqlCmd = Nothing
            diaryDBConn = Nothing
        End If
    End Sub
```

Moving on, the next section discusses aspects of the code dealing with editing, viewing, and deleting events.

Creating, Editing, and Viewing Diary Events

Events are created by clicking the Add New Event link on the DayView.aspx page. This takes you to a simple form on the AddEvent.aspx page. When the Save button is clicked, the button's `click` event creates a new `DiaryEvent` object, populates its properties from the form, and then calls its `Save()` method. The code flow is much the same as for the `DiaryEvent` object's `Save()` method. Where the functionality is similar or the same, the names of methods on different objects have been kept the same. It reduces confusion and makes your life easier.

All events relating to a particular day are shown on the DayView.aspx page. An `ObjectDataSource` control on the DayView.aspx page draws its data from the `DiaryEvent` object's `GetDiaryEventsByDate()` shared method. The markup for the `ObjectDataSource` control is shown here:

```
        <asp:ObjectDataSource ID="eventsObjectDataSource" runat="server"
SelectMethod="GetDiaryEventsByDate"
            TypeName="DiaryEvent" DeleteMethod="DeleteEvent">
            <SelectParameters>
                <asp:SessionParameter DefaultValue="-1" Name="DiaryId"
SessionField="DiaryId" Type="Int32" />
                <asp:ControlParameter ControlID="dayShownLabel" DefaultValue=""
Name="FromDate" PropertyName="Text"
                    Type="DateTime" />
                <asp:ControlParameter ControlID="dayShownLabel" DefaultValue=""
Name="ToDate" PropertyName="Text"
```

```
                    Type="DateTime" />
                <asp:Parameter DefaultValue="0" Name="MaxRows" Type="Int32" />
            </SelectParameters>
            <DeleteParameters>
                <asp:Parameter Name="EventId" Type="Int64" />
            </DeleteParameters>
        </asp:ObjectDataSource>
```

Notice that the `SelectParameters` and the `DeleteParameters` are set to specify the data passed to the `GetDiaryEventsByDate()` method used to pull back the data, and the `DeleteEvent()` method is used to delete diary events.

A `GridView` control is hooked to the `ObjectDataSource` in the code above:

```
        <asp:GridView ID="eventsGridView" runat="server"
AutoGenerateColumns="False" CellPadding="4"
            DataSourceID="eventsObjectDataSource" ForeColor="#333333"
GridLines="None" Height="1px"
            PageSize="5" Style="z-index: 108; left: 78px; position: absolute; top:
357px"
            Width="542px" DataKeyNames="EventId">
            <FooterStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
            <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />
            <Columns>
                <asp:HyperLinkField DataNavigateUrlFields="EventId" Text="Edit"
DataNavigateUrlFormatString="~/SecureDiary/EditEvent.aspx?EventId={0}" />
                <asp:CommandField ShowDeleteButton="True" />
                <asp:BoundField DataField="EventName" HeaderText="Event" />
                <asp:BoundField DataField="EventDescription"
HeaderText="Description" />
            </Columns>
            <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>
```

Again, the `AutoGenerateColumns` parameter is set to `False`, and the columns are specified as follows:

```
            <Columns>
                <asp:HyperLinkField DataNavigateUrlFields="EventId" Text="Edit"
DataNavigateUrlFormatString="~/SecureDiary/EditEvent.aspx?EventId={0}" />
                <asp:CommandField ShowDeleteButton="True" />
                <asp:BoundField DataField="EventName" HeaderText="Event" />
                <asp:BoundField DataField="EventDescription"
HeaderText="Description" />
            </Columns>
```

Notice the hyperlink and field that when clicked will take the user to the EditEvent.aspx page, and the URL contains data passed to the `EventId` in the URL by way of the `EventId` querystring parameter. It's set to be `{0}`, which at run time will be substituted by the value of the first column for each row in the DataSet.

In addition, the code specifies a Delete button on each row in the grid:

```
<asp:CommandField ShowDeleteButton="True" />
```

When you click the Delete button, the `GridView` control asks the `ObjectDataSource` control to call the specified delete method of the data providing class. In this case it's the `DeleteEvent()` method of the `DiaryEvent` class. The `DataKeyNames` attribute in the `GridView` control's markup specifies the primary key field that needs to be used to delete the row.

Returning to editing the event: When you click the Edit link you are taken to the EditEvent.aspx page. The clicked Edit link's `EventId` is passed as a URL parameter. The EditEvent.aspx page is virtually identical to the AddEvent.aspx page discussed previously. The main difference is when the page initializes. The `Page_Init` event handler is shown in the following code, and it's here that the event details are entered into the form:

```
    Protected Sub Page_Init(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Init

        Dim EventBeingEdited As New
DiaryEvent(CLng(Request.QueryString("EventId")))
        eventNameTextBox.Text = EventBeingEdited.EventName
        eventDescriptionTextBox.Text = EventBeingEdited.EventDescription
        dayShownLabel.Text = EventBeingEdited.EventDate.Day & " " &
MonthName(EventBeingEdited.EventDate.Month) & " " & EventBeingEdited.EventDate.Year


        Dim NewListItem As ListItem, HourCount, MinuteCount

        For HourCount = 0 To 23
            If HourCount < 10 Then
                NewListItem = New ListItem("0" & HourCount, HourCount.ToString)
            Else
                NewListItem = New ListItem(HourCount.ToString, HourCount.ToString)
            End If
            If EventBeingEdited.EventDate.Hour = HourCount Then
                NewListItem.Selected = True
            End If
            StartHourDropDownList.Items.Add(NewListItem)
        Next

        For MinuteCount = 0 To 59
            If MinuteCount < 10 Then
                NewListItem = New ListItem("0" & MinuteCount.ToString,
MinuteCount.ToString)
            Else
                NewListItem = New ListItem(MinuteCount.ToString,
MinuteCount.ToString)
            End If
            If EventBeingEdited.EventDate.Minute = MinuteCount Then
                NewListItem.Selected = True
            End If
            StartMinuteDropDownList.Items.Add(NewListItem)
        Next
        Dim itemToSelect As ListItem
```

```
        itemToSelect =
eventDurationDropDownList.Items.FindByValue(EventBeingEdited.EventDuration.ToString
())
        itemToSelect.Selected = True

        EventBeingEdited = Nothing
    End Sub
```

The `EventId` is extracted from the URL parameters and used to create a new `DiaryEvent` object. Populating the event text boxes is easy enough, but the details of time and duration of the event involve populating the Hour and Minute drop-down boxes and ensuring the correct value is selected. This is achieved by looping through hours from 0 to 23 and then minutes from 0 to 59. If the hour to be added to the list is the same as the hour about to be added to the list box, make sure it's the default selected one. The same goes for the minute list box population.

Managing Contacts

Managing contacts is the last aspect of the Online Diary you'll examine, and uses many of the same principles as the other sections. YourContacts.aspx is the central contact management page. Here a list of current contacts is displayed, and the option to add, edit, and delete contacts is possible.

All contacts are displayed using a `DataObjectSource` and a `GridView` control; the principles being identical to the displaying, deleting, and editing of the diary events. This time the `Contact` class is used for editing and display contact details, but otherwise the code is very similar to the events code.

The main page for displaying contacts is YourContacts.aspx, which contains a `GridView` control in which all current contacts are listed:

```
       <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
CellPadding="4"
            DataSourceID="ObjectDataSource1" ForeColor="#333333" GridLines="None"
Style="z-index: 101;
            left: 36px; position: absolute; top: 137px" DataKeyNames="ContactId">
            <FooterStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
            <Columns>
                <asp:CommandField ShowDeleteButton="True" />
                <asp:HyperLinkField DataNavigateUrlFields="ContactId"
DataNavigateUrlFormatString="~/SecureDiary/EditContact.aspx?ContactId={0}"
                    Text="Edit" />
                <asp:BoundField DataField="LastName" HeaderText="Last Name" />
                <asp:BoundField DataField="FirstName" HeaderText="First Name" />
                <asp:BoundField DataField="Telephone" HeaderText="Telephone" />
                <asp:BoundField DataField="Email" HeaderText="Email Address" />
            </Columns>
            <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />
            <EditRowStyle BackColor="#999999" />
            <SelectedRowStyle BackColor="#E2DED6" Font-Bold="True"
ForeColor="#333333" />
            <PagerStyle BackColor="#284775" ForeColor="White"
HorizontalAlign="Center" />
            <HeaderStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
            <AlternatingRowStyle BackColor="White" ForeColor="#284775" />
        </asp:GridView>
```

It gets its data from the `ObjectDataSource` control `ObjectDataSource1`, which in turn connects to the `Contact` class's `GetContactByFirstLetter()` shared method:

```
        <asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
SelectMethod="GetContactsByFirstLetter"
            TypeName="Contact" DeleteMethod="DeleteContact">
            <SelectParameters>
                <asp:SessionParameter DefaultValue="6" Name="DiaryId"
SessionField="DiaryId" Type="Int32" />
                <asp:Parameter Name="FirstLetterOfSurname" Type="Char" />
            </SelectParameters>
            <DeleteParameters>
                <asp:ControlParameter ControlID="GridView1" Name="ContactId"
PropertyName="SelectedValue"
                    Type="Int64" />
            </DeleteParameters>
        </asp:ObjectDataSource>
```

The `ObjectDataSource` control's `DeleteMethod` parameter is also hooked to the `Contact` class's `DeleteContact`. The `GridView` control has been set to show a link to delete each contact, and it's this method that does the actual deleting:

```
    Public Shared Sub DeleteContact(ByVal ContactId As Long)
        Dim diaryDBConn As New SqlConnection(conString)
        Dim sqlString As String = "DeleteContact"
        Dim sqlCmd As New SqlCommand(sqlString, diaryDBConn)
        sqlCmd.CommandType = CommandType.StoredProcedure

        sqlCmd.Parameters.AddWithValue("@ContactId", ContactId)
        diaryDBConn.Open()
        sqlCmd.ExecuteNonQuery()
        diaryDBConn.Close()
        sqlCmd = Nothing
        diaryDBConn = Nothing
    End Sub
```

The `GridView` also includes an Edit link, which when clicked navigates the user to the EditContact.aspx page:

```
                <asp:HyperLinkField DataNavigateUrlFields="ContactId"
DataNavigateUrlFormatString="~/SecureDiary/EditContact.aspx?ContactId={0}"
                    Text="Edit" />
```

The corresponding `ContactId` is passed in the URL as URL data.

Adding a new user involves clicking the Add Contact link on the YourContacts.aspx page. This takes you to a basic form for adding contact information such as name, e-mail, phone number, and so on. This page and the EditContact.aspx page are identical in operation except for one important detail: The EditContact.aspx page retrieves the details of the contact to be edited using the `Contact` class. This happens in the `Page_Load` event:

```
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        If IsPostBack Then
            Dim currentContact As New
Contact(CLng(Request.QueryString("ContactId")))
            currentContact.FirstName = firstNameTextBox.Text
            currentContact.LastName = lastNameTextBox.Text
            currentContact.AddressLine1 = addressLine1TextBox.Text
            currentContact.City = cityTextBox.Text
            currentContact.PostalCode = postalCodeTextBox.Text
            currentContact.State = stateTextBox.Text
            currentContact.Telephone = telephoneTextBox.Text
            currentContact.MobilePhone = mobilePhoneTextBox.Text
            currentContact.Email = emailTextBox.Text
            currentContact.SaveContact()
            currentContact = Nothing
            Response.Redirect("YourContacts.aspx")
        Else
            Dim currentContact As New
Contact(CLng(Request.QueryString("ContactId")))
            firstNameTextBox.Text = currentContact.FirstName
            lastNameTextBox.Text = currentContact.LastName
            addressLine1TextBox.Text = currentContact.AddressLine1
            cityTextBox.Text = currentContact.City
            postalCodeTextBox.Text = currentContact.PostalCode
            stateTextBox.Text = currentContact.State
            telephoneTextBox.Text = currentContact.Telephone
            mobilePhoneTextBox.Text = currentContact.MobilePhone
            emailTextBox.Text = currentContact.Email
            currentContact = Nothing
        End If
    End Sub
```

The `If` statement determines whether this is a postback (the form has been submitted to itself) or whether the page has just been loaded. If it's a postback, you need to save the data and then move back to the main contacts section. If it's a new page load, it's necessary to create a new `Contact` object, and use the data from that to populate the form fields with the contact information.

The AddContact.aspx page is identical except there's no need to populate with existing contact data, because a new contact has no prior data!

Setting up the Online Diary

One of the great things about ASP.NET 2.0 is how easy it is to set up web applications created on one machine onto another. To install the application on your PC, simply copy the entire directory and files from the accompanying CD-ROM (or download it from `www.wrox.com`) onto a directory on your PC (for example, `C:\Websites`). In VWD, all you have to do is choose File➪Open Web Site and browse to the folder where you copied the files. Then press F5 to run it.

Alternatively, if you have IIS installed make the `OnlineDiary` directory you copied over a virtual directory and then simply browse to SignOn.aspx.

To find out how to modify the Online Diary application, visit `www.wrox.com` and download this chapter's code, or you can grab it from the companion CD-ROM in the back of the book.

Summary

In this chapter you've seen how to create a fully functioning diary and contacts management system, all with only a little code thanks to ASP.NET 2.0's new controls and functionality. The new security controls in particular help save a lot of time and coding. In this chapter they've been used to create users and login control. However, they can also help provide a lot more functionality like creating different types of user roles, which then allows you to specify what users can and cannot do based on their role. Or you can let users determine the look and feel of their pages using their account details and ASP.NET 2.0's new login and role controls.

Another great control you discovered in this chapter is the `ObjectDataSource` control. In the past data source controls have made life nice and easy. But they were quick and dirty, which meant poor code design, and you had to wave goodbye to a three-tier architecture. Now with the `ObjectDataSource` control you can have quick and dirty and three-tier architecture — great news for creating easily maintainable, well-designed projects.

In the next chapter you will be creating a file sharing project and learning some more about ASP.NET 2.0's great new features.