Chapter 21

Designing an Intranet Using Forms Authentication

IN THIS CHAPTER

- Working with specifications
- Creating a database for an intranet
- Creating user- and role-based security from a database
- Creating the Daily Census, a user task

OVER THE NEXT THREE chapters, we are going to work with the Bedframer Corporation, a fictional company that runs rehabilitation facilities in some of the best communities in the United States. While the company is imaginary, the scenarios presented come from real-world situations I have encountered over the past few years.

In this chapter, we build the Bedframer intranet. In the next chapter, we deal with Bedframer's Internet Web site, and in the third, we create an XML Web service for Bedframer's partners that maintains the look and feel of the Bedframer intranet.

These chapters emphasize some of the high points of the Bedframer intranet application and how to map company goals to this application.

This chapter primarily deals with how to set up the security of an intranet site using forms authentication. We'll build the security around setting up a user table in a database rather than using Windows user accounts and groups.

Overview of the Bedframer Intranet

The first problem Bedframer needs to solve is putting up an intranet where its employees can record and track their moving of residents in and out of their rehabilitation facilities. The intranet replaces a paper-based system that was developed quite some time ago by the company from which Bedframer bought the facilities.

Initially, the marketing and accounting departments present the Information Services Department with a few demands for the new system:

- Replace all paper forms. This task includes forms for each task that occurs during a resident's stay, including move-in, move-out, and room change.
- Keep a daily census of all residents. Some residents are long-term, while others are only in the facility for a couple of days. Long-term residents often spend time away in a hospital or on vacation.
- Web-enable a set of reports that currently resides in Excel.

Ultimately, marketing would like to have the entire application moved to an OLAP database where serious analysis can be completed. In the interim, all residents are linked to demographic tables in the database.

The head of Information Services names the intranet site BOW (an acronym for Bedframer Operations Web). After examining the initial needs of the project, the Information Services creates the simple specifications shown in Table 21-1.

TABLE 21-1 SPECIFICATIONS OF THE NEW SYSTEM

Area	Needs
Forms	The following forms need to be created for the Web application: ◆ Daily Census check. ◆ Deposit and Resident Agreement. ◆ Forms to reserve a room, admit a resident, discharge a resident, and change a resident's room.
Reports	The specifications set down a requirement for move-in, move-out, and available-room reports.
Search	The user must be able to search for both resident and community information.
Security	The primary concern with the application is being able to authenticate a user: ◆ The user must be authenticated. ◆ The user must have authorization to use certain resources. ◆ The screen must timeout to accommodate privacy concerns.

In a real-world application, I'd create a full set of functional and technical specifications before moving ahead with the code. The functional specifications would contain use cases for each of the needs determined in Table 21-1. Before writing the full technical specifications, I settle on a platform (.NET), language (Visual Basic .NET), and database platform (SQL Server 2000).

Database Design

Bedframer has purchased a demographic database to use for its city, county, state, and MSA (Metropolitan Statistic Area) information. You can download a database with similar information from the book's companion Web site. If you want to design an application of this nature, consider purchasing this information. Because postal codes change, the data becomes out-of-date, but there are subscription services that can keep you up to date on every change. The data I've compiled for this project will likely be out of date a few months after you start working with it.

You can download the database used in this chapter from this book's companion Web site (`www.wiley.com/extras`). The database file is called `Ch21Data.zip`. The database was culled from a variety of sources on the Internet and is useful as a starting point for your own demographic work. The code for this chapter, in the file `Ch21VS.zip` (Visual Studio .NET version) or `Ch21SDK.zip` (SDK version), contains a smaller subset of this data if you don't want to play with data from the entire United States.

As I mentioned earlier, the database design I'm using for this project is adopted from work I've done on a consulting assignment. Any of the sections of the database that seem a bit different from traditional design are probably rooted in the original project.

Let's first take a look at the demographics portion of the database, which is shown in Figure 21-1. The tables include City, County, State, Country, Metropolitan Statistical Area (MSA), and Nielsen Dominant Market Areas (DMA). The DMA is used to map out radio and television markets.

The main purpose of the demographic tables is for future analysis of data. At some point in time, Bedframer plans to build a data warehouse to aid in its marketing efforts, so these tables are included with this future use in mind. For example, this data might be used for the following:

- Linking a resident to a DMA can help Bedframer determine which radio and television markets might be useful for its advertising efforts.
- The postal code information is useful for mass mailing, as well as determining at which locations to hold seminars to get the best response.
- City data helps to determine newspaper advertising. Because many newspapers print editions tailored to certain areas of town, this can further be broken down using the postal code information.

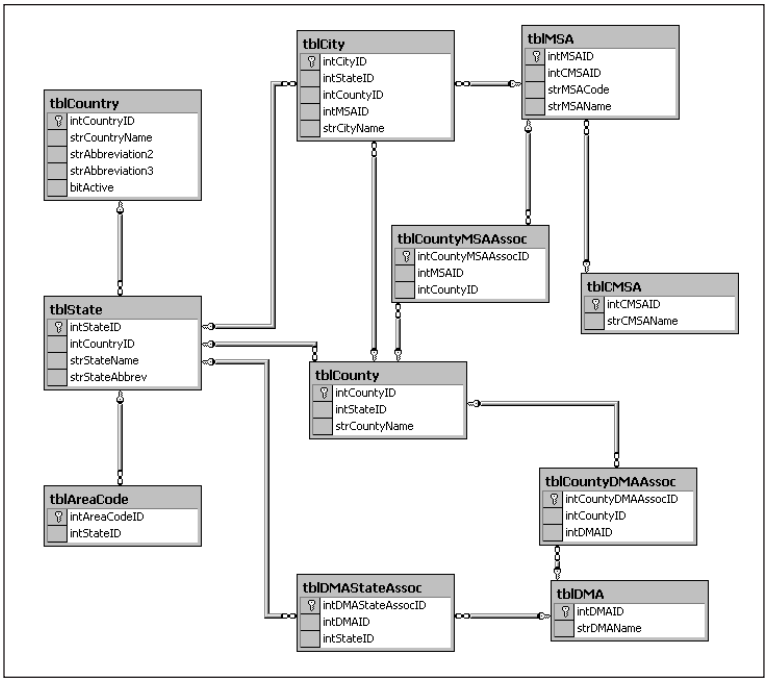

Figure 21-1: Demographics tables (and the tables that join these tables).

As mentioned, these are future uses of the data and not part of the current phase of the project.

The next portion of the database, shown in Figure 21-2, deals with the user and security sections. The center table is the Person table (`tblPerson`). Both the User and Employee tables are subtypes of the Person table. The User table is linked to the Menu tables to set permissions for each of the menu items (contained in `tblSectionMenuItem` and `tblSideMenuItem`). This ensures that the user cannot access menu items he does not have the proper permissions to view. Through the Employee table (`tblEmployee`), the person is linked to the community (`tblCommunity`) in which he works.

The Security and User tables (`tblUser`, `tblSideMenuItem` and `tblSectionMenuItem`) are designed to limit what a user can see based on his logon account. Using the User table (`tblUser`), the user logs on. Based on this logon and the type of user, the user's menu items are limited to only those items he can view (`tblSideMenuItem` and `tblSectionMenuItem`). Some accounts have more permissions than others. By

linking the User table to menu tables, a user can be restricted without a lot of effort. Of course, an incorrect user and password combination result in no access to the site at all.

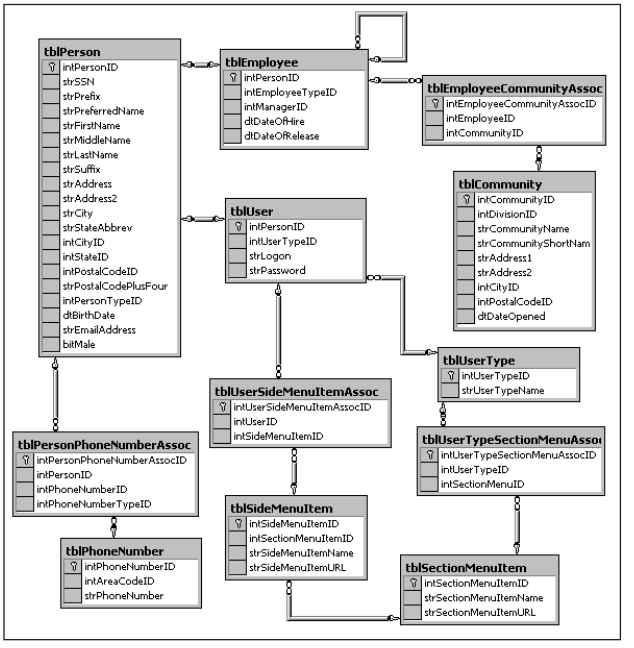

Figure 21-2: Security and User tables: tblCommunity, tblPerson, tblEmployee, tblUser, tblSideMenuItem, and tblSectionMenuItem (and the tables that join them).

The final section of the database, shown in Figure 21-3, deals with the people who stay in the communities and the communities they stay in. Once again the Person table is utilized to store the information about the person. The Resident table is also subtyped from the Person table, just like the User and Employee tables are.

The fact that so many tables are set up as subtypes of the Person table may seem a bit strange at first, but it does simplify a lot of queries because you can bypass tables in your joins. I thought this method of setting up a database was strange the first time I saw it, but I have come to like it in certain situations. This part of the database design is getting to a point where it's more object-oriented than relational; it can be very useful if you design objects for your users in your application.

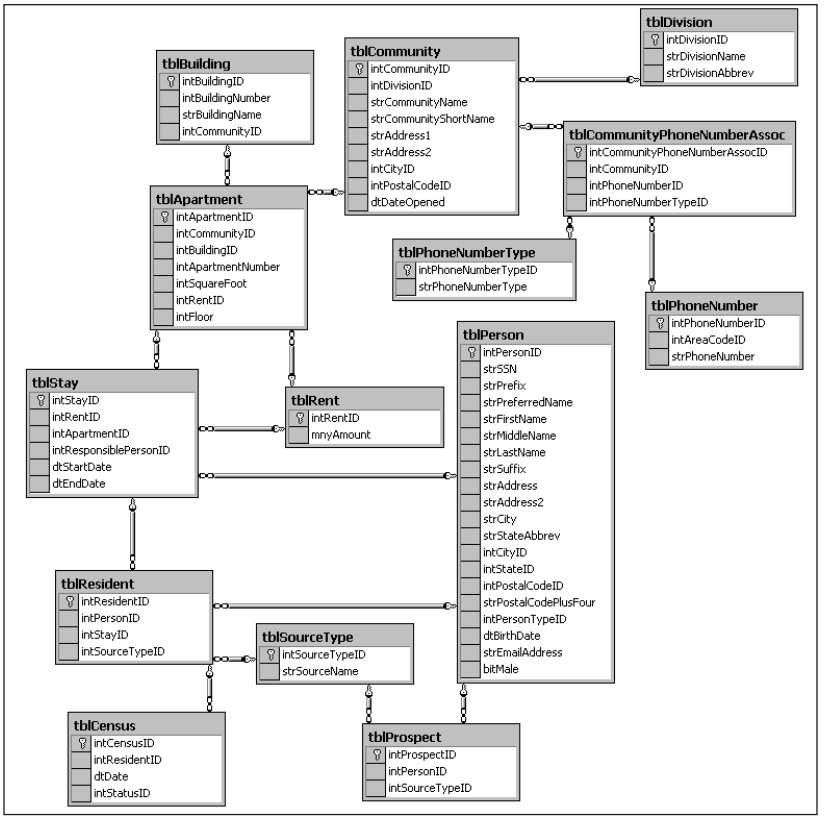

Figure 21-3: Resident and Community tables: tblResident, tblCensus, tblStay, tblApartment (and other supporting tables and join tables).

Security

The primary security method of the Bedframer intranet application is a database-driven logon using forms authentication. Since this is an intranet, Windows authentication was also a possibility but, because many users use the same workstation and do not always have time to completely log out of Windows, the IS staff opted to use forms authentication instead. This allows the ASP.NET application to time out, or the user to use a single-click logoff.

The logon page

The first step in authentication is to log the user into the application. There are two files that must be altered to accomplish this. The first is the `web.config` file, which is used to set the authentication method to forms authentication. The second is the `logon.aspx` file, which is used as the logon form.

Here's how our authentication process works:

1. A user attempts to access a page in the application.
2. The ASP.NET process detects the user is not authenticated and automatically directs the user to the logon.aspx file.
3. The user must log on and provide the appropriate password.
4. If the logon is successful, ASP.NET automatically redirects the user to the page he attempted to access. I assume that the user tried to access the home page in each of the examples in this chapter. Using forms authentication, a user can bookmark any page in the application and be redirected to that page, not the home page.

   If the logon fails, the user is denied access to the page he wanted.

Unlike in ASP, you don't have to create a custom redirect script to facilitate this in ASP.NET. To illustrate the point, Listing 21-1 presents a simple ASP script that is designed to direct a user to a `logon.asp` page. I use this to contrast the methodology employed in ASP.NET (Listings 21-2 through 21-4).

Listing 21-1: ASP Redirection

```
<%
 If Session("loggedon") <> True then
       Response.Redirect("logon.asp")
    End If
%>
```

In this scenario, the user logs on and the `Session("loggedon")` value is set to `True`. You can also facilitate this with cookies if you are using a Web farm for your application. As you will see, forms authentication handles this functionality, along with the redirect back to the page accessed, without any code on your part. You need only code whether a user successfully logged in.

THE WEB.CONFIG FILE

To set up forms authentication, you have to open the `web.config` file and make a few changes:

- Change the `mode` attribute of the authentication tag to read `Forms`.
- Add a `forms` tag that points to the URL (`loginURL`) you'll use for a logon page and set the path that uses this page. I use `/` as a path to ensure that all pages in the entire application redirect to this page.

The code snippet produced from these instructions is shown in Listing 21-2.

Listing 21-2: The web.config File for the Bedframer Operations Web

```
<authentication mode="Forms">
  <forms name="bow" path="/" loginUrl="logon.aspx"
         protection="All" timeout="10" />
</authentication>
```

The authentication method is set to `Forms`. The URL for the logon page is `logon.aspx`. The `protection="All"` is the default setting; it indicates that the cookie is encrypted and the data from the cookie is validated. The `timeout` setting here is 10 minutes.

In addition to `All`, other choices for `protection` are `Encryption` and `Validation`. These options, along with the options for all of the tags in the `web.config` file, are covered in much more detail in Chapter 12.

THE LOGON.ASPX FILE

The next step is to create a logon page. Figure 21-4 shows this Web page, which greets the user whenever he attempts to access any page in the Bedframer Operations Web. This page shows all but the menu system.

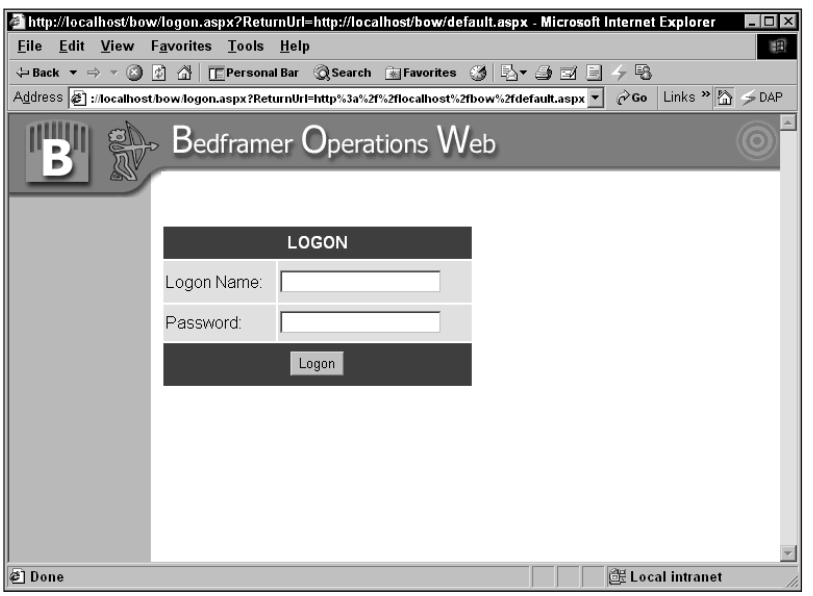

Figure 21-4: The logon page.

The ASPX page contains a couple text box controls, along with a label to tell the user that his username (Logon Name) and password combination were incorrect. The rest of the page is filled with the look and feel of the site. The code for the form is shown in Listing 21-3; the important portions are bold.

Listing 21-3: The logon.aspx Page

```
<form id="Form1" method="post" runat="server">
  <TABLE style="WIDTH: 300px; HEIGHT: 157px" cellSpacing="2" cellPadding="2"

         width="300" border="0">
    <TR>
      <TD colspan="2" bgColor="#333399" style="HEIGHT: 31px" align="middle">
        <font color="#ffffff"><STRONG>LOGON</STRONG></font>
      </TD>
    </TR>
    <TR>
      <TD style="WIDTH: 104px" bgColor="#99ff99">
        Logon Name:
      </TD>
      <TD bgColor="#99ff99">
        <asp:TextBox id="strLogon" runat="server" />
      </TD>
    </TR>
    <TR>
      <TD style="WIDTH: 104px; HEIGHT: 35px" bgColor="#99ff99">
        Password:
      </TD>
      <TD style="HEIGHT: 35px" bgColor="#99ff99">
        <asp:TextBox TextMode="Password" id="strPassword" runat="server" />
      </TD>
    </TR>
    <TR>
      <TD colspan="2" bgColor="#333399" align="middle">
        <asp:Button id="btnLogon" runat="server" Text="Logon" />
      </TD>
    </TR>
  </TABLE>
</form>
<P>
  <asp:Label id="lblLogonProblem" runat="server" />
</P>
```

The code for the `logon.aspx` file is included on the companion Web site (`www.wiley.com/extras`) in this chapter's download file: `Ch21-23VS.zip` (Visual Studio .NET version) or `Ch21-23SDK.zip` (SDK version).

The two text boxes, `strLogon` and `strPassword` are going to be used to input the logon name and password. The reason for using `str` instead of `txt`, which would be normal for a text box in Visual Basic 6, is to keep the names consistent from the page to the `dbBow` database field to which it will map. The button control, `btnLogon`, is used to submit the form, which I cover shortly. Finally, there is a label control, `lblLogonProblem`, which displays error messages when a user has problems logging on.

THE CODEBEHIND FILE: LOGON.ASPX.VB

The magic in the logon routine comes in the CodeBehind file. In this page, the user logon name and password are checked against values in the database. You can also stick these values into the `web.config` file, but I don't do this because I think it's more of a hassle to update a file than a database.

In order to use the database, I add a little bit of code to check the input from the form against the database and redirect the user. Listing 21-4 shows the `btnLogon_Click` event.

Listing 21-4: Logon Subroutine

```
Private Sub btnLogon_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles btnLogon.Click
  'Values to store into Session
  Dim intPersonID As Integer
  Dim strFirstName As String
  Dim strLastname As String
  Dim intCommunityID As Integer

  Dim strConn As String = Application("ConnectionString")
  Dim objConn As New SQLData.SqlConnection(strConn)

  'Set up the command object to call the logon stored procedure
  Dim objCmd As New SQLData.SqlCommand()
  With objCmd
      .CommandText = "sps_UserLogon"
      .CommandType = CommandType.StoredProcedure
      .Connection = objConn
      .Parameters.Add(New SQLData.SqlParameter("@strLogon", SqlDbType.VarChar, _
                    50, ParameterDirection.Input, False, 0, 0, "strLogon", _
```

```
                    DataRowVersion.Original, strLogon.Text))
        .Parameters.Add(New SQLData.SqlParameter("@strPassword", _
                    SqlDbType.VarChar, 15, ParameterDirection.Input, False, _
                    0, 0, "strPassword", DataRowVersion.Original, _
                    strPassword.Text))
    End With

    Dim objReader As SQLData.SqlDataReader

    Try
        objConn.Open()
        objReader = objCmd.ExecuteReader

        'Get values from the reader
        While objReader.Read
         intPersonID = objReader.Item("intPersonID")
         strFirstName = objReader.Item("strFirstName")
         strLastname = objReader.Item("strLastName")
         intCommunityID = objReader.Item("intCommunityID")
        End While

        objConn.Close()

    Catch objSQLException As SQLData.SqlException
        lblLogonProblem.Text = "Problem contacting database."
    Catch objException As Exception
        lblLogonProblem.Text = "An unexpected error occurred."
    End Try

    'Set session objects (mirrors a current application in ASP)
    Session.Add("intPersonID", intPersonID)
    Session.Add("strFirstName", strFirstName)
    Session.Add("strLastName", strLastname)
    Session.Add("intCommunityID", intCommunityID)

    If intPersonID <> 0 Then
        'Send the user ID to the application as a variable
        SysSecurity.FormsAuthentication.RedirectFromLoginPage(intPersonID, False)
    Else
        lblLogonProblem.Text = "Either the logon or the password are incorrect."
    End If

End Sub
```

The code for the `logon.aspx.vb` file is included on the companion Web site (`www.wiley.com/extras`) in this chapter's download file: `Ch21-23VS.zip` (Visual Studio .NET version) or `Ch21-23SDK.zip` (SDK version).

I bolded the important parts of this routine in the listing, and want to take a little space to cover those sections:

- There are two `Catch` statements in the `Try ... Catch` block. The first is for SQL exceptions, and the second deals with generic exceptions. If you include an exception handler that catches specific exceptions, that `Catch` must come first. If it doesn't, you always get a generic exception message. Here's the syntax for using two `Catch` statements in a `Try ... Catch` block:

```
Try
   'Statements that might cause an error
Catch objSQLException as SQLData.SQLException
   'More specific exception
Catch objException as Exception
   'Less specific exception
End Try
```

- The individual pieces of information are added to the session object. The syntax is a bit different from the syntax in ASP.NET. To store values in `Session`, you use the `Add` method of the `Session` object:

```
Session.Add("intPersonID", intPersonID)
```

  To retrieve the value, use the `Item` collection of the `Session` object:

```
intPersonID = Session.Item("intPersonID")
```

- To redirect to the page the user was attempting to access, use the `RedirectFromLogonPage` method of the `FormsAuthentication` object. You pass it a name to retrieve the cookie (you can also add additional information to the cookie) as well as a Boolean value to indicate whether this cookie should perpetuate beyond this session. In this application, it is assumed the cookie goes out of scope when the user logs off, as shown here:

```
SysSecurity.FormsAuthentication. _
    RedirectFromLoginPage(intPersonID, False)
```

If you only take one piece of information away from this section, it should be the `RedirectFromLoginPage` method.

If you want to create a site that perpetuates a user's credentials once she signs up on your site, you can use forms authentication. Just set the Boolean value for the `RedirectFromLoginPage()` to `True`. Because you can add any additional information you need to personalize your site, this is a rather effective way to set up an application that does not have great security needs. Here's the code snippet to cache a user's credentials in a cookie:

```
SysSecurity.FormsAuthentication. _
    RedirectFromLoginPage(intPersonID, True)
```

If you want to allow the user to decide if this information is persisted, set up a check box for the user to indicate if she'd like to be able to avoid logging in each time. Use this check box to set the Boolean value.

If you get nothing else from this section, you should realize that ASP.NET enables you to create a single logon page for the entire application, without having to create redirectors to the logon page from every other page in your Web site. Because the settings are set up in the `.config` file, you can also create this type of security on a directory-by-directory basis to easily set up public and private content.

Logoff page (logoff.aspx)

The logoff page is shown in Listing 21-5. To simplify this section, I've taken the `Page_Load` event and placed it in the ASPX page. In the actual application, this is not the case, because I stick to the CodeBehind paradigm.

Listing 21-5: The Logoff Page

```
<%@ Page Language="vb"%>
<script language=VB runat=server>
   Private Sub Page_Load(ByVal sender As System.Object, _

                         ByVal e As System.EventArgs) Handles MyBase.Load
        SysSecurity.FormsAuthentication.SignOut()
        Session.Abandon()
        Response.Redirect("default.aspx")
   End Sub
</script>
```

The code from Listing 21-5 is on the companion Web site (`www.wiley.com/extras`) in this chapter's download file: `Ch21-23VS.zip` (Visual Studio .NET version) or `Ch21-23SDDK.zip` (SDK version). Look for the file `logoff.aspx`.

Take a look at the code that is bolded in this listing. The first line—`SysSecurity.FormsAuthentication.SignOut()`—is used to delete the authentication cookie and end the session. Although not required, I also explicitly end the `Session` with the `Abandon` method. This is partially out of habit (from traditional ASP), partially due to my need to be explicit in my code, and partially due to my paranoia that there will be an exception somewhere that leaves a user logged in.

Finally, I redirect to the home page. As you should have guessed, this once again sends the user to the logon.aspx page with a redirect argument to return to `default.aspx` after the next user logs on. Since `default.aspx` is the home page, it is a good page to set this redirect to.

Menus

The final security measure is the way in which menus are built. The user ID is used on every page to build both the top and the side menus. Each of these menus is contained in a user control. Figure 21-5 shows the home page for the application (`default.aspx`) complete with menus.

Figure 21-5: The home page with top and side menus.

The menu horizontally across the top is a sectional menu. It's the primary method of navigation from section to section in the site. The menu on the left side changes in context to the section.

The menus are included in the security section of the database so they can change depending on the user who logs on. Each menu is implemented as a user control to reduce repetitive code.

While not heavily covered in this book, because I consider them just another type of object, user controls are a very powerful tool in your ASP.NET arsenal. In addition to menus, there are a variety of snippets that you repeat from page to page throughout your application. User controls

- Enable you to cache the user control without caching the entire page as a single unit. This allows you to write applications that perform and scale better.
- Remove repetitive code from each page and place it in one central location. This helps with reuse and makes the site more maintainable.
- Allow you to share your controls with other ASP.NET developers. This reduces the amount of repeat code from a development team.

User controls are covered in Chapter 3, in the section "Creating Custom User Controls."

TOP MENU

The top menu is the sectional menu for the Bedframer intranet application. The application is divided into eight sections (home, community, census, events, forms, reports, search, and admin), whose links appear in the top menu. At the right end of the menu is a logoff button for the user to end his session.

The top menu spans across the top of the page. The ASPX page uses a `Repeater` control to make a simple list of hyperlinks, using a pipe character (|) as a separator. The code for this page is shown in Listing 21-6. Notice that the method of binding in a repeater template is through the use of the `Container.DataItem("ItemName")` lines of code.

The `Repeater` control is explained in Chapter 8, in the section "Data Controls," and is used in Chapter 15 in the section "Using a DataReader." Consult these chapters if you need a bit of guidance on this control.

Listing 21-6: The ASCX Page for the Sectional Menu

```
<table>
  <tr>
    <td>
      <asp:Repeater id="TopMenuRepeater" runat="server">
```

Continued

Listing 21-6 *(Continued)*

```
        <ItemTemplate>
          <a href='<%# Container.DataItem("strSectionMenuItemURL") %>'>
            <%# Container.DataItem("strSectionMenuItemName") %>
          </a>
        </ItemTemplate>
        <SeparatorTemplate>
           | 
        </SeparatorTemplate>
      </asp:Repeater>
    </td>
  </tr>
</table>
```

All of the code to fill this sectional menu is contained in the CodeBehind page. There's nothing really stellar about this page – you've seen this type of example numerous times in the Part III of this book. The only new item is pulling the user ID, which was placed in session in the logon page, and using it to create the menu. In this page, I've chosen to pull the user ID from session in the control. The CodeBehind file is shown in Listing 21-7. I bolded the line where I pull the user ID.

Listing 21-7: The CodeBehind File for the Top Menu

```
Private Sub Page_Load(ByVal sender As System.Object, _

                      ByVal e As System.EventArgs) Handles MyBase.Load
    Dim strConn As String = Application("ConnectionString")
    Dim objConn As New SQLData.SqlConnection(strConn)

    Dim strSQL = "sps_GetMenu"
    Dim objCmd As New SQLData.SqlCommand(strSQL, objConn)
    Dim intPersonID = Session.Item("intPersonID")

    With objCmd
        .CommandType = CommandType.StoredProcedure
        .Parameters.Add(New SQLData.SqlParameter("@intPersonID",

                        SqlDbType.Int, 4, ParameterDirection.Input, False, 0,

                        0, "intUserID", DataRowVersion.Original, intPersonID))
    End With

    Try
        objConn.Open()
        Dim objReader As SQLData.SqlDataReader = objCmd.ExecuteReader()
        TopMenuRepeater.DataSource = objReader
```

```
        TopMenuRepeater.DataBind()
        objConn.Close()
    Catch objSQLException As SQLData.SqlException
        'TODO: Handle SQL Exceptions
    Catch objException As Exception
        'TODO: Handle general exceptions
    End Try
End Sub
```

You can find the code for Listings 21-6 and 21-7 on the companion Web site (`www.wiley.com/extras`) in this chapter's download file: `Ch21-23VS.zip` (Visual Studio .NET version) or `Ch21-23SDDK.zip` (SDK version). Look for `SectionMenu.ascx` and its CodeBehind file, `SectionMenu.ascx.vb`.

The procedure that fills this page is rather simple. It takes the user's ID, which is `intPersonID` (remember the subtyping of the Person table through any tables that have a persona?), and uses it to pull the menu items to which the user has access. The stored procedure is shown in Listing 21-8.

Listing 21-8: The sps_GetMenu Stored Procedure

```
CREATE PROCEDURE sps_GetMenu
(
    @intPersonID    int
)
AS

SELECT
       smi.strSectionMenuItemName,
       smi.strSectionMenuItemURL
FROM
       tblSectionMenuItem smi
JOIN
       tblUserTypeSectionMenuAssoc utsma
ON
       smi.intSectionMenuItemID = utsma.intSectionMenuID
JOIN
       tblUser u
ON
       utsma.intUserTypeID = u.intUserTypeID
WHERE
u.intPersonID = @intPersonID
```

The stored procedure from Listing 21-8 is on the companion Web site (`www.wiley.com/extras`) in this chapter's download file: `Ch21-23VS.zip` (Visual Studio .NET version) or `Ch21-23SDDK.zip` (SDK version). Look in the data directory. There are also SQL scripts to create the entire SQL database.

SIDE MENU

The side menu is almost identical to the sectional menu. One noticeable difference is that you need the section ID to put this menu in context. Since the user does not change, the side menu pulls the user ID from session. The section ID, however, does change with each page. To facilitate this, I create a `WriteOnly` property for the section in the user control. The value of this `WriteOnly` property is fed from a constant placed in the CodeBehind page for each ASPX page. Since the session ID is intimately joined with the page, this makes sense. There are a couple of other methods I could use to determine the section, but as much as I dislike hard coding, the simpler approach prevails here — I can't envision a place where the page and section would not be tightly linked.

The user control (ASCX page) for the side menu differs from the top menu pretty much in direction. I once again use a repeater control. For completeness, I include it here in Listing 21-9. I am using simple binding here to bind the section name and the links; these statements are highlighted in bold.

Listing 21-9: The Side Menu User Control

```
<asp:Repeater id="SideMenuRepeater" runat="server">
  <HeaderTemplate>
    <table cellpadding="2" cellspacing="2">
      <tr>
        <td>
          <strong>
            <%# SectionName %>
          </strong>
        </td>
      </tr>
  </HeaderTemplate>
  <ItemTemplate>
    <tr>
      <td>
        <a href='<%# Container.DataItem("strSideMenuItemURL") %>'>
          <%# Container.DataItem("strSideMenuItemName") %>
        </a>
      </td>
    </tr>
  </ItemTemplate>
```

```
  <FooterTemplate>
    </table>
  </FooterTemplate>
</asp:Repeater>
```

The code for Listing 21-9 is on the companion Web site in this chapter's download file: `Ch21-23VS.zip` (Visual Studio .NET version) or `Ch21-23SDDK.zip` (SDK version). Look for the file `SideMenu.ascx`.

Looking at the code, you can see that I bound a property from the CodeBehind page to the `HeaderTemplate` section of the `Repeater` control. When I show you the CodeBehind page, you'll notice that I use the section ID to also set the section name.

Because the section ID changes from page to page, I have to use a property in the side menu user control. To set the properties in the page, I utilize a code line like this one taken from the home page (`default.aspx`):

```
<bow:SideMenu id="SideMenu" runat="Server" Section="<%# m_intSectionID %>">
```

The bolded section shows I am setting the `Section` property of the user control. To use this user control, as with all user controls, I have to register it on the page using a line like this:

```
<%@ Register TagPrefix="bow" TagName="SideMenu" src="SideMenu.ascx" %>
```

The `@ Register` line is located at the top of the page with other `@` directives. Notice that the `TagPrefix` and `TagName` map to the tag used in the body of the page, `<bow:SideMenu>`. Listing 21-10 shows the `Section` property in the user control, which is set with the `Section=""` attribute of the user control tag.

Listing 21-10: Properties of the Side Menu User Control

```
Private m_intSection As Integer
Private m_strSectionName As String

Public WriteOnly Property Section() As String
    Set(ByVal Value As String)
        m_intSection = CType(Value, Integer)

        Select Case CType(Value, Integer)
            Case 1
                SectionName = "Home"
            Case 2
```

Continued

Listing 21-10 *(Continued)*

```
                SectionName = "Community"
            Case 3
                SectionName = "Census"
            Case 4
                SectionName = "Events"
            Case 5
                SectionName = "Forms"
            Case 6
                SectionName = "Reports"
            Case 7
                SectionName = "Search"
            Case 8
                SectionName = "Admin"
        End Select

    End Set
End Property

Public Property SectionName() As String
    Set(ByVal Value As String)
        m_strSectionName = Value
    End Set
    Get
        Return m_strSectionName
    End Get
End Property
```

"Okay, wait a second, Greg! You are hard-coding values in a page?" you ask, incredulous. Normally, I'd be as skeptical as you, but follow my logic for a second:

- Unless the entire application is data-driven (perhaps XML- and XSLT-driven), I am going to have more than one page in my Web application.
- It is unlikely the logon page will become a report page. The content may change, based on user input or data pulled from the database, but the page purpose is unlikely to change.
- If the page purpose is unlikely to change, why not have the page "tell" the application which page it is?

This methodology is a bit outside of the box, but it makes sense if you think about it. Since the Events page is always an Events page, and never a Reports page, why not let it help us render our side menu?

The `Page_Load` event is fairly similar to the `Page_Load` event in the sectional menu user control, so there's no need to elaborate here on how to fill a repeater.

MENUS AND SECURITY

To illustrate the security aspects of the top and side menus, you have to log in with two different user IDs. I've included an Excel file in the download file for this chapter; it has all of the users of the system and their logons and passwords. The file also tells their position in the company. In the system, users with the position "maintenance" are the most restricted.

The Excel file is called `users.xls`. You'll find it on the companion Web site (`www.wiley.com/extras`) in this chapter's download file: `Ch21-23VS.zip` (Visual Studio .NET version) or `Ch21-23SDK.zip` (SDK version).

For the first run, you will log in as William Charles, the president of Bedframer Corporation. In this database the logon is always first initial and last name, while the password is first name and last initials. The logon would therefore be WCharles with a password of WilliamC. When you log on as William Charles, you see the menus shown in Figure 21-6.

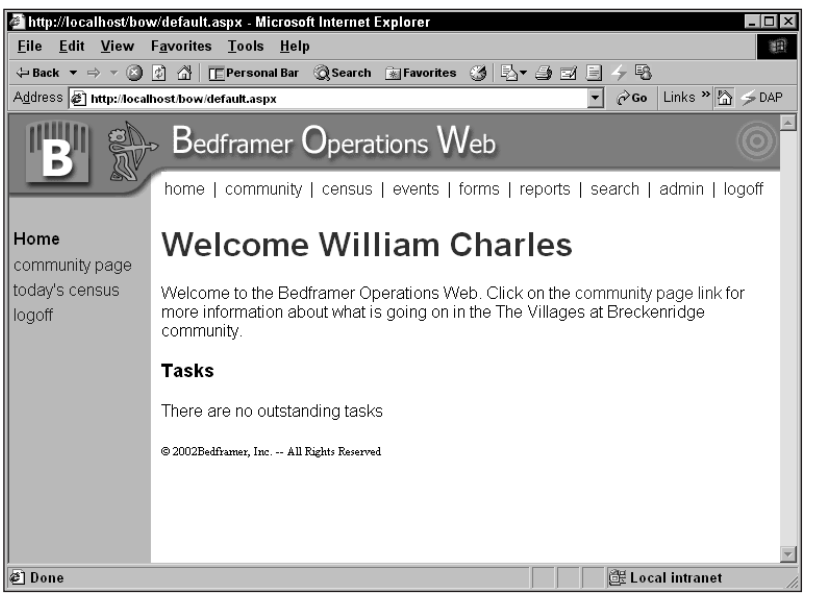

Figure 21-6: Menus when logged on as William Charles.

Now, log off and log on as Richard Grayson: logon is RGrayson with a password of RichardG. Notice the different menus (shown in Figure 21-7).

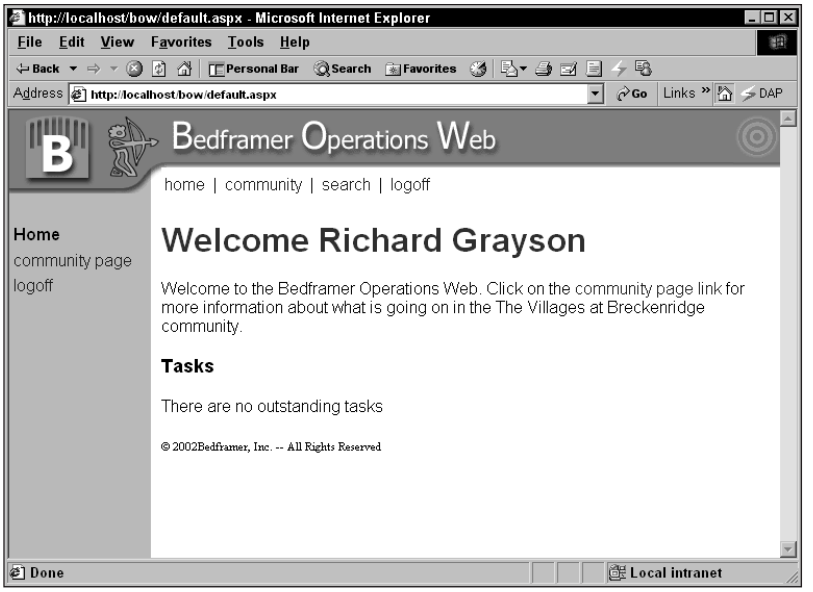

Figure 21-7: Menus when logged on as Richard Grayson.

When you are logged on as Richard Grayson, there are very few menu items that you have access to. If this were a real application, the maintenance man (Richard Grayson) would certainly be restricted more than the president (William Charles) of the company, but he would probably not be this restricted. The example is a bit exaggerated, but illustrates the point nicely.

The Census Page

One of the most important tasks for the Bedframer Operations Web is a daily accounting of all of the residents. This is done through the Daily Census. To ensure that the census is completed each day, every user is informed if the census for his or her primary community is still undone.

Showing the task

The Daily Census page is the page that most employees who regularly use the system will look at each day. It's where the status of the residents is placed. The first step is to look at the home page again. When the census is not done for the day, the home page has a Daily Census task, as shown in Figure 21-8.

The determination of whether the census has been done is made by running the Daily Census stored procedure to check for all items that have not been confirmed. If there are any that are unconfirmed, the Daily Census task shows up on the home page.

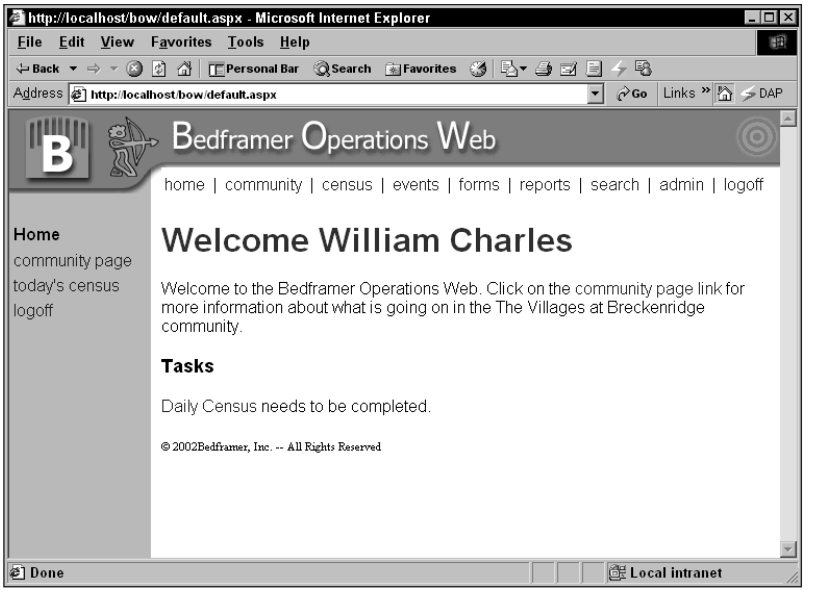

Figure 21-8: Home page when Daily Census is not done.

At some time in the future, other tasks will also appear on the front page, but the census is all for now. The code to determine if a census needs to be done for the day is shown in Listing 21-11. There are two bolded lines in the code sample. The first shows that I am using `ExecuteScalar()` to only return the first name of one of the residents, while the second shows where I test for NULL.

Listing 21-11: The Census Task Event

```
Protected Function GetTasks() As String

  Dim intCommunityID As Integer = Session("intCommunityID")
  Dim dtDate As Date = FormatDateTime(Date.Now, DateFormat.ShortDate)
  Dim strFirstName As String
  Dim bitConfirmed As Boolean = False

  Dim objConn As New SQLData.SqlConnection(Application("ConnectionString"))
  Dim objCmd As New SQLData.SqlCommand()

  With objCmd
    .CommandText = "sps_CensusByCommunity"
    .CommandType = CommandType.StoredProcedure
    .Connection = objConn
.Parameters.Add(New SQLData.SqlParameter("@intCommunityID", _
                    SqlDbType.Int, 4, ParameterDirection.Input, False, _
```

Continued

Listing 21-11 *(Continued)*

```
    0, 0, "intCommunityID", DataRowVersion.Original, _
                    intCommunityID))
    .Parameters.Add(New SQLData.SqlParameter("@dtDate", _
                    SqlDbType.SmallDateTime, 4, ParameterDirection.Input, _
                    False, 0, 0, "dtDate", DataRowVersion.Original, dtDate))
    .Parameters.Add(New SQLData.SqlParameter("@bitConfirmed", SqlDbType.Bit, _
                    1, ParameterDirection.Input, True, 0, 0, "bitConfirmed",
                    DataRowVersion.Original, bitConfirmed))
  End With

  Try
    objConn.Open()
    strFirstName = CType(objCmd.ExecuteScalar(), String)
    objConn.Close()

  Catch objSQLException As SQLData.SqlException
    'TODO: Add SQL Server exception handler
  Catch objException As Exception
    'TODO: Add generic exception handler
  End Try

  If IsDBNull(strFirstName) Then
    Return "There are no outstanding tasks."
  Else
    Return "<a href=""census.aspx?dtDate=" & dtDate & """>Daily Census" & _
         "</a>" needs to be completed."
  End If

End Function
```

The code from Listing 21-11 is on the companion Web site (`www.wiley.com/extras`) in this chapter's download file: `Ch21-23VS.zip` (Visual Studio .NET version) or `Ch21-23SDDK.zip` (SDK version). Look for `default.aspx` and its CodeBehind file, `default.aspx.vb`.

Here is a quick run-through of the code in Listing 21-11:

- I store today's date in a variable. I'll use this both for the stored procedure, as well as to create a link to the census if the stored procedure returns a value:

  ```
  Dim dtDate As Date = FormatDateTime(Date.Now, _
  DateFormat.ShortDate)
  ```

This format is a shortcut that is a throwback to Visual Basic 6 days. You can also use `Now()` method, which is an even greater throwback:

```
Dim dtDate as Date = FormatDateTime(Now, _
    DateFormat.ShortDate)
```

And, finally, you can actually create the `DateTime` structure and use the `Now` property:

```
Dim objDate as new System.DateTime(1,1,1)
Dim dtDate as Date = FormatDateTime(objDate.Now, _
    DateFormat.ShortDate)
```

In each case, I have added the `FormatDateTime()` function to make sure I am getting a short date returned. This is important since SQL Server returns nothing if I send the time in as well. Remember that a SQL Server datetime field stores a date sent in without a time as 1/1/01 00:00:00 AM, which equates to midnight on the date in question. When you send in a date without time to a stored procedure, it is converted in the same manner to compare against the field. Until Microsoft adds a separate date and time field, this is going to be the case.

- Create a connection object and a command object. In this example, I don't have to create a `DataAdapter`, `DataReader`, or `DataSet`, because only one value is returned.

```
Dim objConn As New _
  SQLData.SqlConnection(Application("ConnectionString"))
Dim objCmd As New SQLData.SqlCommand()
```

- Add parameters to the command object. In the stored procedure, I use the community ID, the date, and a Boolean for whether the census has been confirmed. In this instance, I only want to return those items that have not been confirmed, because the user confirms each person's status as he goes through the census. Here are the parameters:

```
With objCmd
  .Parameters.Add(New _
     SQLData.SqlParameter("@intCommunityID", _
     SqlDbType.Int, 4, ParameterDirection.Input, False, _
     0, 0, "intCommunityID", DataRowVersion.Original, _
     intCommunityID))
  .Parameters.Add(New SQLData.SqlParameter("@dtDate", _
     SqlDbType.SmallDateTime, 4, ParameterDirection.Input, _
     False, 0, 0, "dtDate", DataRowVersion.Original, dtDate))
  .Parameters.Add(New _
     SQLData.SqlParameter("@bitConfirmed", SqlDbType.Bit, _
     1, ParameterDirection.Input, True, 0, 0, _
     "bitConfirmed", DataRowVersion.Original, bitConfirmed))
End With
```

If I had written a separate stored procedure for this task, which I'd probably do except for the illustration of reuse of code, I'd only need the community ID passed, because the confirmation bit could always be set to 0 in the stored procedure (Boolean false in my Visual Basic .NET code) and the date could always be hard-coded to the current date.

- Because my only duty here is to check if there are any records, I use `ExecuteScalar` to return one value:

```
strFirstName = CType(objCmd.ExecuteScalar(), String)
```

- Finally, I check if the first name is empty. If it is, I add a link to the census page.

```
If IsDBNull(strFirstName) Then
  Return "There are no outstanding tasks."
Else
  Return "<a href=""census.aspx?dtDate=" & dtDate & """>" & _
         "Daily Census</a>" needs to be completed."
End If
```

As mentioned previously, the stored procedure used to create this page is not necessarily the most efficient. It would seem to be more efficient to write a stored procedure that simply returns a count of records. This is a fair assessment, but the code here was written to serve two purposes. The same stored procedure is used to create the complete census on the census page. With only one stored procedure to produce a census, or tell a user if there is a census, I have made an application that is easier to maintain. If I find I need a bit more performance, I can always alter this page to provide it.

Building the census page

It's fairly easy, following the logic in the home page, for you to develop your own census page, so I'm not going to take the time to explain everything about that page. I include this section to show how easy it is to reuse code by changing a couple of lines.

You already have a good deal of code for the census page. The stored procedure that creates the census page is the same stored procedure that helps you create the task hyperlink in Listing 21-11. The main difference here is you are going to use more than just a single value returned by `ExecuteScalar()`.

Listing 21-12 shows the code for the `RenderCensus()` subroutine. The changes from the `GetTasks` function are bolded.

Listing 21-12: The Routine to Render the Census

```
Protected Sub RenderCensus()

    Dim intCommunityID As Integer = Session("intCommunityID")
    Dim dtDate As Date = FormatDateTime(Date.Now, DateFormat.ShortDate)
    Dim strFirstName As String
```

```
    Dim bitConfirmed As Boolean = False

    Dim objConn As New SQLData.SqlConnection(Application("ConnectionString"))
    Dim objCmd As New SQLData.SqlCommand()

    With objCmd
        .CommandText = "sps_CensusByCommunity"
        .CommandType = CommandType.StoredProcedure
        .Connection = objConn
        .Parameters.Add(New SQLData.SqlParameter("@intCommunityID", _
         SqlDbType.Int, 4, ParameterDirection.Input, False, _
                        0, 0, "intCommunityID", DataRowVersion.Original, _
                        intCommunityID))
        .Parameters.Add(New SQLData.SqlParameter("@dtDate", _
                        SqlDbType.SmallDateTime, 4, ParameterDirection.Input, _
                        False, 0, 0, "dtDate", DataRowVersion.Original, dtDate))
        .Parameters.Add(New SQLData.SqlParameter("@bitConfirmed", SqlDbType.Bit, _
                1, ParameterDirection.Input, True, 0, 0, "bitConfirmed", _
                DataRowVersion.Original, bitConfirmed))
    End With

    Dim objDataReader As SQLData.SqlDataReader

    Try
        objConn.Open()
        objDataReader = objCmd.ExecuteReader()
        objConn.Close()

    Catch objSQLException As SQLData.SqlException
        'TODO: Add SQL Server exception handler
    Catch objException As Exception
        'TODO: Add generic exception handler
    End Try

    dgCensus.DataSource = objDataReader
    dgCensus.DataBind()

End Sub
```

You can find the code for Listing 21-12 on the companion Web site (`www.wiley.com/extras`) in the download file `Ch21-23VS.zip` (Visual Studio .NET version) or `Ch21-23SDDK.zip` (SDK version). Look for `census.aspx` and its CodeBehind file, `census.aspx.vb`.

As you can see in Listing 21-12, the only differences from the code in Listing 21-11 are as follows:

- I created a DataReader object to hold the data returned from the command object:

```
Dim objDataReader As SQLData.SQLDataReader
```

- I use ExecuteReader, as opposed to ExecuteScalar, to fill the DataReader object:

```
objDataReader = objCmd.ExecuteReader()
```

- The DataReader is bound to the DataGrid on the page, which I have named dgCensus:

```
dgCensus.DataSource = objDataReader
dgCensus.DataBind()
```

Summary

The intranet application for Bedframer Corporation centers on forms authentication, because different users use the same machine. In this chapter, I detailed the security model for the Bedframer. I also showed how each user is mapped only to sections he is able to see.

There is additional documentation for the Bedframer Operations Web, along with the code for the entire application, on this book's companion Web site.

In the next chapter, I expand on the Bedframer application by adding some XML and XSLT to render the look of the site. This functionality is then used in the extranet part of our project, in Chapter 23.