Programmer to Programmer ™

Building An ASP.NET Intranet

Kourosh Ardestani, Brian Boyce, Chad Hutchinson, Saurabh Nandu, John C Roth, Chandu Thota, Jonathan Walsh, Karli Watson

wrox
Programmer to Programmer

Wrox technical support at: support@wrox.com
Updates and source code at: www.wrox.com
Peer discussion at: p2p.wrox.com

What You Need to Use This Book

The following is a list of recommended system requirements for using this book:

- Windows 2000 or later with IIS installed
- ASP.NET Version 1.0
- SQL Server 2000 or MSDE
- Visual Studio .NET Professional or higher (optional)

This book assumes that you understand the basics of ASP.NET and Visual Basic .NET.

Summary of Contents

The IBuySpy Portal Architecture

In this book, we'll be using the freely available IBuySpy Portal as a starting point for our intranet development. This choice needs justification - and that is what we will do in the first part of this chapter. We'll look at both why we are modifying an existing intranet application rather than creating our own, and why we choose the IBuySpy Portal in particular.

Once we have covered the basics, we'll continue to look in more depth at the IBuySpy Portal, starting with how to install it. Next, we'll take a tour of its features, with an eye on how it fits in with the objectives outlined in the previous chapter.

Moving on from this we'll take a more technical look at the IBuySpy Portal architecture, looking at the files and types it consists of, how they function and fit together, and general principles behind the site.

Finally, we'll take a brief look at how security is handled.

Why use an existing architecture?

In the last chapter, we looked at all the things we are likely to want from an intranet, and there were quite a few of them! However, if we were to design an intranet from scratch, many areas would require a lot of work to obtain even the most basic functionality. There are also many important challenges to overcome, such as deciding how to handle authentication and authorization, what other applications we require on our servers, and so on.

Much of this is solved instantly if we take an existing intranet's architecture as a starting point. If someone else has already worked out how to handle security issues, how to achieve effective data storage, and so on, then we can get up and running much quicker. This doesn't mean that we can just take any existing intranet, though, since some architectures may not fulfill enough of our requirements, or may be very difficult to customize and extend.

What we want, then, is a well-designed intranet architecture that solves problems without creating new ones – an intranet architecture that we can easily tailor to our needs.

What does the IBuySpy Portal Architecture provide?

The IBuySpy Portal architecture is one that makes the grade. As we will see in this and other chapters, it simplifies the creation of our intranet while providing an excellent foundation for adding our own features. It also offers one other crucial advantage - it is free to obtain and no restrictions are placed on our usage of the code it contains!

We'll look at the specific features that are obtained from the IBuySpy Portal later, but first here's a list of key points:

- It makes use of well-established database technology via SQL server or Microsoft Data Engine (MSDE) database access.
- It is simple (and free) to install, with no tiresome and time-consuming registration forms to complete.
- It is built in a modular way, with several useful modules (such as modules for authentication, forums, and news articles) already in place, and it's also extensible, so we can add more as required.
- It is 100% customizable to whatever corporate look and feel you need, with whatever extra functionality you can dream up.

The application architecture follows a traditional n-tier model, with functional separation of classes that deal with presentation, business logic, and data access. However, as with many ASP.NET applications, the business logic does overlap slightly with the presentation, since ASP.NET pages are capable of sophisticated processing without affecting performance.

All database access is achieved via stored procedures, which also improves performance. Stored procedures are SQL queries stored as part of the database, which may be parameterized to add versatility. Since they are part of the database, the database server can perform additional optimization for us, such as using built-in caching. In addition, security is improved by using this method, since we can restrict access to stored procedures only, thus preventing malicious users from executing potentially harmful SQL queries. The result of this, as we will see later, is that we end up with very simple data access components that for the most part simply act as an interface for the stored procedures in the database, via ADO.NET.

To get a better idea of what is provided you can either look at the online version of the portal at http://www.ibuyspyportal.com/ or install it on your server. Since we will spend the rest of the book modifying the code for this application you may as well go right ahead and install it now.

Installing the IBuySpy Portal Architecture

IBuySpy Portal can be obtained from http://www.asp.net/ibuyspy/downloads.aspx. This download page also contains files for another example application, IBuySpy Store, which is a starting point for Internet e-commerce applications. However, we're not interested in this application here.

Before we download the application (or, as it is referred to on the web page for downloading it, the Solution Kit) we should ensure that our server is equipped to deal with it. The most basic requirement here is having the .NET Framework installed. We also need access to either a SQL Server database, or the free (but trickier to administer - see note below) MSDE data engine. MSDE is installed as part of the ASP.NET **quickstart** applications that are part of the .NET Framework.

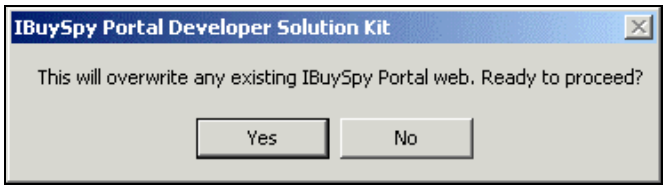
> *If you only have MSDE installed then many administration tasks may only be carried out using command line SQL queries. Here's a tip: get the Enterprise Manager application from the MSDN SQL Server evaluation download - it'll make your life a lot easier and isn't limited by the 120 day trial license that applies to the SQL Server database download itself!*

The IBuySpy Portal also requires an installation of the ASP.NET Mobile Internet Toolkit, a link to which appears on the web page mentioned above. In this book, we won't be looking at the mobile device access of intranets, but it is worth remembering that IBuySpy Portal has been designed with this functionality in mind, and you may want to conduct your own research into this area.

> *For more information on the Microsoft Mobile Internet Toolkit, see ASP.NET Mobile Controls: Tutorial Guide: Adaptive Web Content for Mobile Devices with the MMIT, by Wrox Press, ISBN: 1861005229*

There are four different versions of the download for IBuySpy Portal, two for the VB.NET version of the code and two for the C# version. In this book we're using VB.NET, so we can ignore the two C# versions. The difference between the two VB.NET versions is that one of them (the VS.NET version) is intended for Visual Studio .NET users while the other (the SDK version) isn't. There is no difference in functionality between these versions, only that the VS.NET version uses the 'codebehind' model for all ASP.NET files, where the VB.NET code is contained in separate `.vb` files rather than being included in `.aspx` and `.ascx` files, and includes solution files for loading straight into the VS.NET IDE. The choice of which version you use is entirely up to you depending on which code model you prefer (and on whether you own a copy of VS.NET!). I prefer the codebehind model, so the screenshots etc. in this chapter will reflect this, but don't feel left out if you go for the other version, it really makes very little difference.

Now, a word of caution! When you run the installer program for the IBuySpy Portal Developer Solution Kit (whichever version you choose), it will present a dialog that warns about the risk of installing a new version when an existing version of the portal is present (even if the installer is being run for the first time):



Modifications made to a previous installation can be lost if you ever run one of the four IBuySpy Portal installer programs at a later date. We'll come back to this point shortly, after we've seen what the installer program actually does.

The installer requires very little intervention - all we need to do is to select a folder to install to (the default, `C:\PortalVBVS` for the VS.NET version of the application, is fine) and a database instance to use to store the data used by the application. Since a new database will be added, we need the required database privileges to do this, although I won't go into details about this. Suffice to say that if you are logged in as Administrator, and are using a local SQL Server or MSDE instance you won't encounter any problems here.

After we select the database instance and click on configure it should only take a minute or two for everything to be set up for us. At this point, we can look at the installation log to ensure everything went smoothly - here's the one I ended up with (I've removed the date stamps for clarity):

```
[Checking system requirements]
[Pass] Detected the NetSDK version of MSDE.
[Pass] Determined SQL Server version (8.00.194).
[Pass] Detected .NET Framework.
[Pass] Detected Internet Information Server (IIS).
[Begin Sample Configuration]
[Pass] Created IIS virtual directory.: PortalVBVS
[Pass] Determined SQL Server version (8.00.194).
[Pass] Added ASPNET user to SQL Server.
[Pass] Executed SQL script.: PortalDB.sql
[Pass] Added aspnet user to database.: Portal
[Done]
```

The installer performs the following steps:

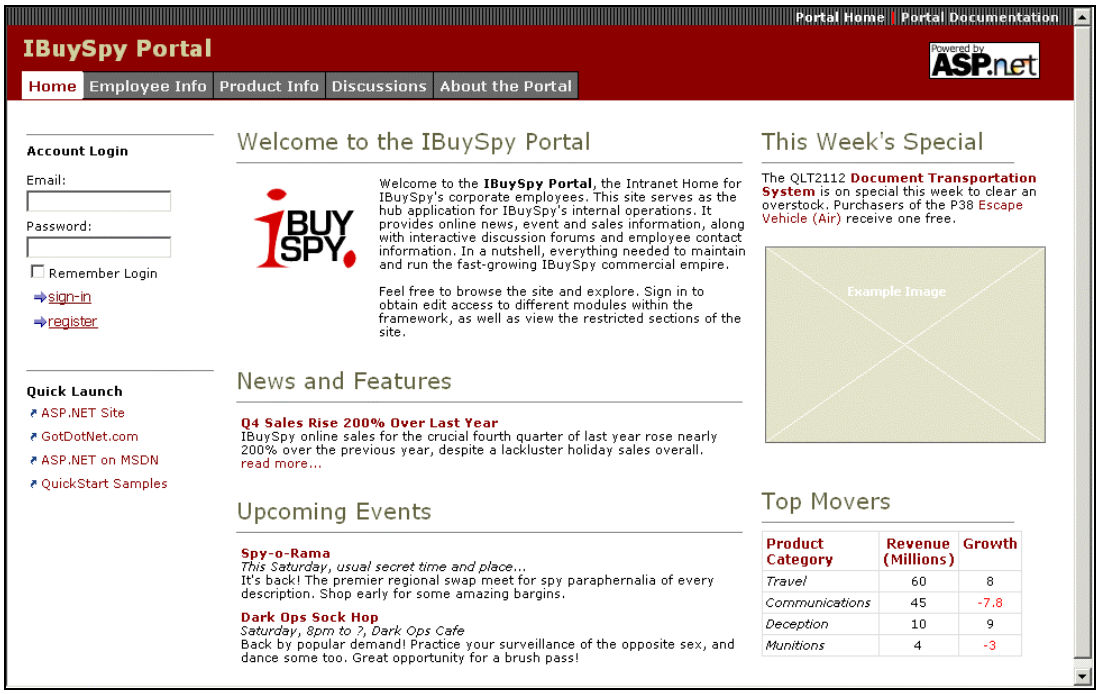
- It creates a new directory to hold the application (`C:\PortalVBVS`) and copies the portal files to it.
- It configures an IIS virtual directory pointing at a subdirectory of the directory created in the above step (`C:\PortalVBVS\PortalVBVS`), with the same name as this directory (`PortalVBVS`).
- It creates a new database in the selected database instance (called `Portal`) and populates it with data.
- It adds the user `(local)\ASPNET` to the database server and to the list of users with permission to access the newly created database (this is essential for ASP.NET applications since it is the account that they run under - the portal wouldn't be able to access the data without this step).

Now we can discuss what will happen if we rerun one of the installation programs at a later date. First, if we select the same directory to install to as before then files may be overwritten. Second, the database creation will fail. The result of this second problem is that the new installation will have access to modified data. Should we wish a completely new database to be installed we have to remove the existing database manually, or move it somewhere else if it contains data we wish to preserve. Unfortunately, no facility exists for specifying the database name to use during installation, which would solve this problem.

The root directory of the portal (`C:\PortalVBVS`) contains many of the files used during installation, including the SQL script used to generate the database, and the resulting log file. It also contains the End User License Agreement (EULA) for the portal code, and a directory with the files that are accessed via the IIS virtual directory for the portal, as mentioned above.

IBuySpy Portal Acclimatization

When you first open the IBuySpy Portal intranet site in a web browser, by either navigating to `http://www.ibyspyportal.com/` or using your local copy at http://localhost/PortalVBVS/, you will be greeted with the following page:



If you are using the online version then the Example Image frame on the right will contain an actual image - many of the bitmaps that make up the example application aren't included in the downloadable version to keep the size of the download to a respectable level. This doesn't affect the functionality at all.

This initial page illustrates much of the functionality of the portal. Starting from the top, we have a title bar containing a main heading, a collection of tabs, and a few other bits and pieces that don't concern us for now. The Home tab is currently selected, and the rest of the page makes up the Home page content.

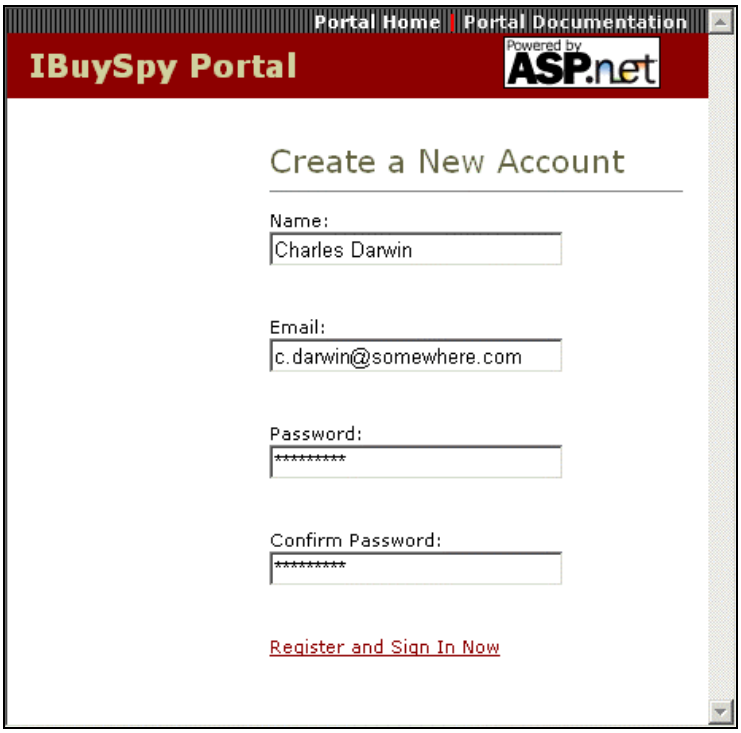
The main body of the page is divided up into three columns - a large one in the middle and smaller ones on either side. Each of these columns contains content that is built up from individual modules, which may not be apparent at first since the modules are designed to use the same styles so they do seem to merge together in places. There are eight modules displayed on this page: two on the left (one for logging in and one for quick navigation), three in the middle (the welcome message, a news section, and an events section), and three on the right (the 'This week's Special' section, an image, and the 'Top Movers' section).

Logging In

The module on the top of the left column allows us to log on, or to create a new login. The IBuySpy Portal is initially configured to handle authentication in this way, using form-based authentication, although it is quite easy to switch to integrated Windows authentication if desired.

> *We'll look at this in a little more depth at the end of this chapter, before covering security issues thoroughly in Chapter 4.*

The initial installation comes with one existing user account called Guest with the e-mail address of Guest and the password of Guest. Users can add themselves to the intranet simply by clicking on the register link in the authentication module. Try this now:



When you click on the Register and Sign in Now link you'll be returned to the Home page with a few differences - the authentication module has disappeared, and the information at the top of the page has changed:

User information now appears, and a new Logoff link enables you to log off.

Once you are logged in you have access to more functionality, such as being able to contribute to discussions and so on, although the specific privileges granted is determined by an administrator. This granting of privileges is known as authorization (not to be confused with authentication, which is the act of verifying login information). In the IBuySpy Portal architecture, this is achieved by privileges based on groups, where a user may belong to any number of groups.

Initially, only one group is defined for us, a group called Admins whose members are intranet administrators. The user we start with, Guest, is a member of this group. If you log off your new Charles Darwin user and log on as Guest one effect of this is instantly visible:

A new tab has appeared: Admin. Through this tab, which is only available to users in the Admin group, we can perform administrative tasks, such as user account manipulation, and even complete reorganization of the intranet site.

It's not just administrators who can get this customized content; we can add any number of groups and tabs that only certain groups can see.

Exploring the Content

At this point, I'll ask you to put down this book and look around the default installation of the IBuySpy portal. Try to get a feel for the way things work - it's pretty much all self-explanatory. Explore the various tabs and the content on those tabs, add a discussion message or two, perhaps upload a document, and you'll find that the interface is intuitive and exciting. Don't fiddle around too much with the Admin tab for now - you may find yourself accidentally doing something you'll regret later!

When you're ready, come back and we'll take a closer look at how this application meets the objectives outlined in the last chapter.

Addressing our Objectives

In the last chapter, we outlined three broad objectives of an intranet site:

- Centralization of information
- Access to information
- Communication

The IBuySpy Portal certainly proves it's worth here. Information is centralized - document data is stored either in the portal database or through the web server virtual directories. Information is easy to access - the web interface and simple authentication exposes our data to all clients who can run a web browser, and we can expose some or all of the information over the wider Internet for remote access. Communication is made simple - the various modules that exist or can be added make it easy for staff to communicate in a variety of ways, from discussion groups, to articles, to shared documents.

We also pointed out several common features of intranets in the last chapter:

- Content management
- Document management
- Project management
- Forums
- Human Resources Information System
- Shared calendar
- Contact management

Several of these are already available with the existing modules (notably content and document management via the Admin tab, forums via the discussion group module, and contact via the contacts module). The portal architecture makes it possible to add all the rest, which is something we will be looking at in later chapters.

Corporate Identity

The IBuySpy Portal has a specific look and feel about it. This may not be exactly what you are looking for, but don't panic! In the next chapter, we'll see how we can completely customize the look and feel of the portal, using stylesheets and direct code modification. What is there now is purely a demonstration of what is possible, not a limitation on what you can do. The important point is that the backend functionality is solid, and you aren't likely to want to make too many changes to that.

Modules Supplied

The IBuySpy Portal architecture comes with ten modules, which form a starting point for our intranet development. The modules supplied are as follows:

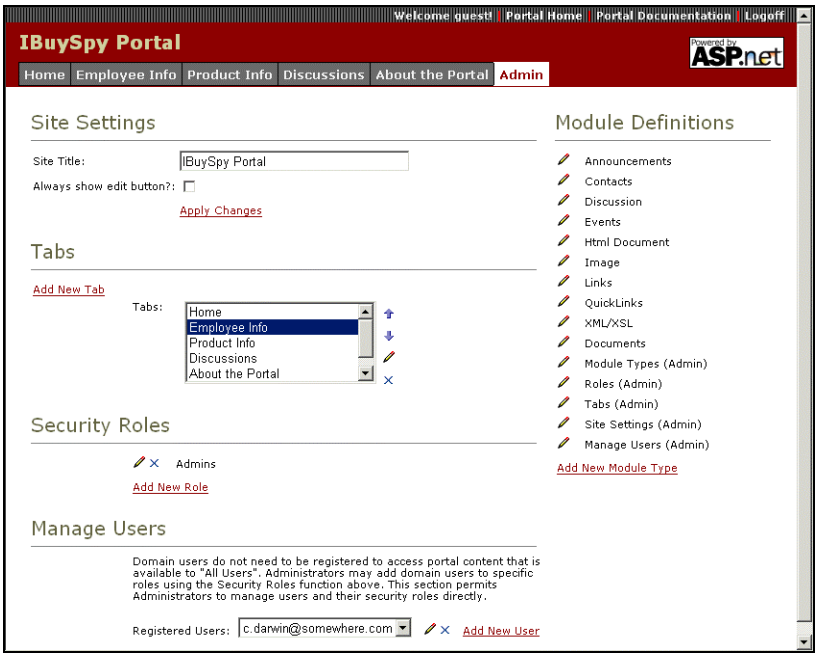
Module Name	Module Description
Announcements	A list of text articles, showing a summary of the article and a read more link to get access to full text.
Contacts	A list of contacts.
Discussion	A discussion forum, divided into threads.
Documents	A list of documents, either uploaded to the server or linked to via URLs.

Module Name	Module Description
Events	A list of events, with date and location information as well as short descriptions.
Html/Text	A general purpose module for displaying HTML.
Image	A simple module for rendering an image.
Links	A list of hyperlinks.
QuickLinks	A list of hyperlinks rendered in a compact way.
Xml/Xsl	A module for displaying XML documents by transforming them with an XSL stylesheet.

These modules may be used several times in a single portal - a single page may even contain multiple instances of a single module. To achieve this, each module added to the portal is assigned a unique ID. This also enables multiple modules of the same type to store their data in the same database table, thus simplifying the data storage structure.

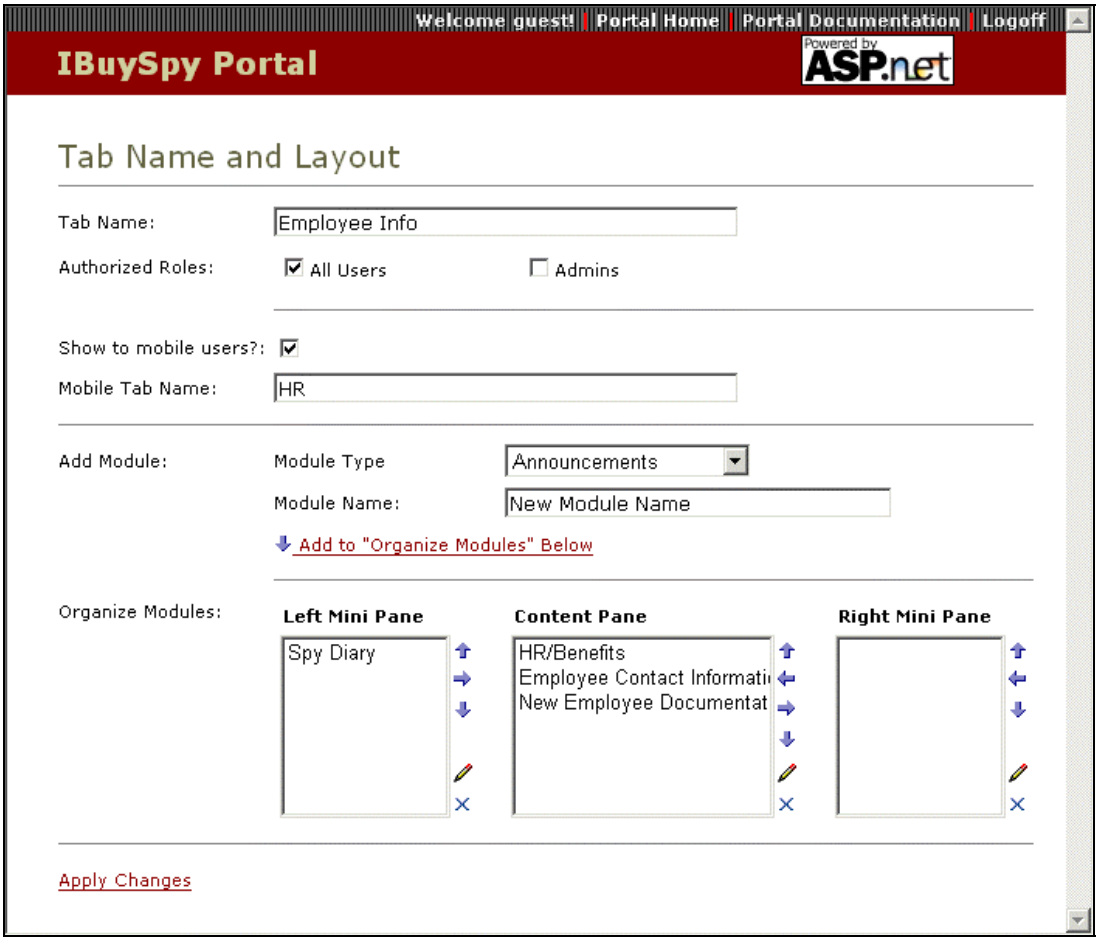
Tab Manipulation

The Admin tab allows administrators to manage the content of the IBuySpy Portal intranet by adding new tabs, deleting old ones, or modifying existing ones. Head to this tab now and edit the Employee Info tab by selecting it and clicking on the pencil icon shown below:

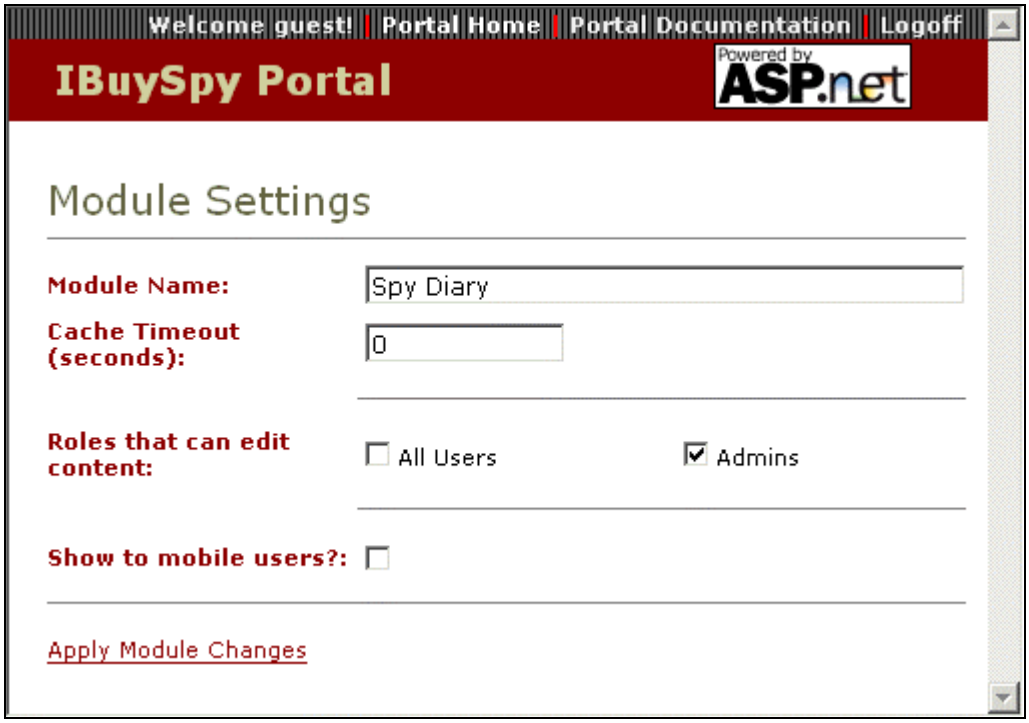


This takes us to a new screen that shows us tab properties as follows:

- The name of the tab
- What roles have access to the tab
- Whether the tab is shown to mobile browsers
- What name to use for the tab for mobile users
- What modules to show on the tab, along with their names and positions



The page also includes the facility to add new modules of a selected type, and edit module properties. If you select a module and click on the pencil icon, you'll be taken to a screen containing general module properties:



Module specific properties and module content can only be modified by subsequently navigating to the tab and editing the module via its own interface. The above module, Spy diary, is only editable by members of the Admin group.

Interestingly, no information is shown concerning the types of modules. Once a module is added the only way to glean this information with the default application code is to navigate to the tab containing the module directly and edit it. However, this information isn't crucial, as it's the functionality of modules that's important once they are added.

A Technical Look at the IBuySpy Portal Architecture

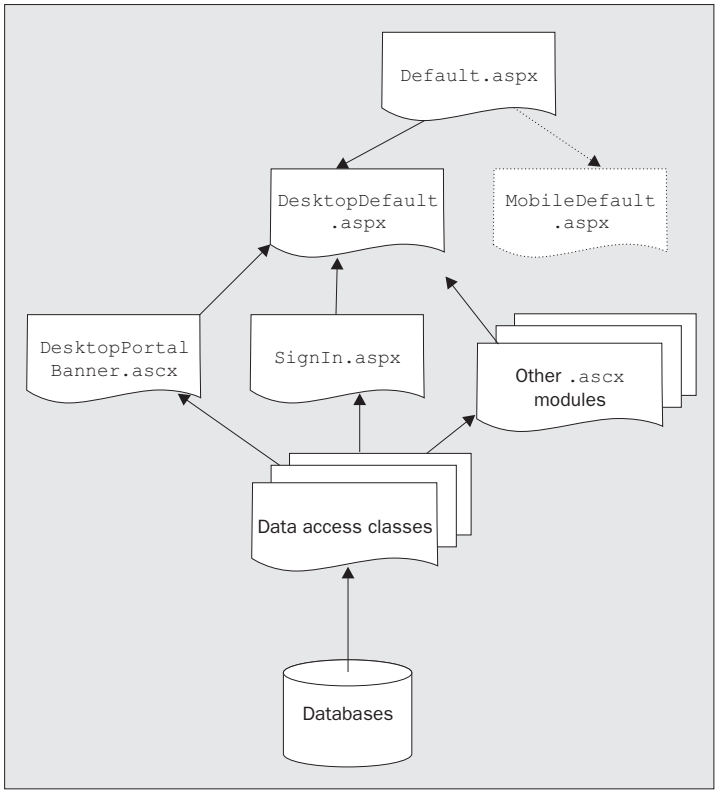
For the remainder of this chapter we will look in a little more depth at how the IBuySpy Portal architecture works, providing an insight into the customization techniques we will be using in the rest of the book.

First, a general overview. Users entering the intranet site are redirected by `default.aspx` to one of two other pages, `DesktopDefault.aspx` or `MobileDefault.aspx`. We'll concentrate on the former, as we're not looking in detail at the mobile aspect of this application in this book.

`DesktopDefault.aspx` is really a placeholder for the various modules that make up individual tabs in the application, which are in fact user controls stored in `.ascx` files. The authentication module, stored in `SignIn.ascx`, is displayed in all tabbed pages, but only if the user has yet to log in. Other modules are added depending on what tab is selected. `DesktopDefault.aspx` also contains one standard user control,- `DeskTopPortalBanner.ascx`, which is displayed at the top of the page. All these user controls are dynamically added to the page that the user sees when they request a given tab, that is, they are added to the content of `DesktopDefault.aspx` as this page is loaded.

Many of the `.ascx` files take their display information from data in the database, or use information in the database in some other way (such as authenticating login information). None of the `.ascx` files perform database access directly; instead they use data access classes stored in other `.vb` files to do this.

The general overview, then, looks like this:



Database Structure

The content and arrangement of content within the IBuySpy Portal application is all stored in the portal database. In order to get to grips with what the code does we should look at this database to see what information it contains.

Table Name	Table Function
`Announcements`	Data used by Announcements modules.
`Contacts`	Data used by Contacts modules.
`Discussion`	Data used by Discussion modules.
`Documents`	Data used by Documents modules.
`Events`	Data used by Events modules.
`HtmlText`	Data used by Html/Text modules.
`Links`	Data used by Links and Quicklinks modules.
`ModuleDefinitions`	A list of the available modules, including a reference to the `.ascx` files that define them.
`Modules`	A list of what modules appear on what tabs, along with information about where they appear, what title to use, who can edit the module content, and so on.
`ModuleSettings`	Additional data for any modules that require it.
`Portals`	A list of the portals defined in the database (initially only one appears here - the default IBuySpy Portal).
`Roles`	The roles that users can be members of.
`Tabs`	The tabs that can be displayed in the portal, and which roles are allowed to view them.
`UserRoles`	A list of links associating users with roles.
`Users`	A list of users, with password and e-mail information.

The database also includes 66 stored procedures used by the data access objects in the application, for all functionality. This includes adding records, extracting records, and so on.

Initially, the sample portal content is configured as follows:

- The IBuySpy Portal is given an ID in the `Portals` table.
- A single role, `Admin`, is set up in the `Roles` table, and a single user, `guest`, is set up in the `Users` table. The `guest` user is set up as a member of the `Admin` role via an entry in the `UserRoles` table.

- The `ModuleDefinitions` table contains a list of the 15 modules: the ten sample modules listed earlier and five for administration.
- The `Tabs` table contains entries for each of the six tabs available to users.
- The `Modules` table contains information about all of the modules present on each of the six tabs in the `Tabs` table.
- The `ModuleSettings` table contains module-specific configurations.
- The remaining tables contain the data used by each module instance (remember that just one of these tables, `Announcements` say, contains data for all of the `Announcements` modules in the portal, identified by Module ID values).

All of the data in the database can be edited by hand if required, although there is nothing there that can't be modified using the tools on the Admin tab of the portal.

We shouldn't remove any of the tables themselves, but we might well add our own tables as we customize the portal.

Keeping Track of Things: the PortalSettings Class

The basic scheme described above centers on the fact that the `.ascx` user controls are added dynamically to a basic framework. In order for this to work in practice the user controls must be aware of certain information, such as the logged in user (if any). The `DesktopPortalBanner.ascx` control also needs to know what tabs to display, as well as which one is selected. Some other controls also have custom information that will be required when they render themselves.

To facilitate this, contextual information is stored whenever the portal receives an HTTP request. This information can then be accessed by any of the user controls, or other classes that make up the application. User information is stored in the standard `User` member of the `Context` object that is accessible by ASP.NET code, and other information is stored in an instance of a class called `PortalSettings`, which is defined as part of the application.

This information storage is carried out in the code for `global.asax`, which is the file that contains application, session, and request level event handlers. One event handler, `Application_AuthenticateRequest`, deals with authentication (as its name suggests!), and will be covered in more detail in Chapter 4. Another event handler, `Application_BeginRequest`, deals with creating a `PortalSettings` instance, placing it in the `Items` collection of the `Context` object for retrieval by whatever code requires it during request processing.

When an instance of the `PortalSettings` class is created, it extracts tab information from the database, including the names of the tabs, which user roles have access to each tab, and so on. It also uses information passed to it in its constructor to determine the currently selected tab (which is detected in `Application_BeginRequest` by looking at the `querystring` accompanying the web request - selecting a tab results in this information being sent to the server). Settings for individual modules are extracted from the database as and when required - individual modules can use the shared `PortalSettings.GetModuleSettings` method for this.

The code for `Application_BeginRequest` is as follows:

```
Sub Application_BeginRequest(ByVal sender As Object, ByVal e As EventArgs)

   Dim tabIndex As Integer = 0
   Dim tabId As Integer = 0

   ' Get TabIndex from querystring
   If Not (Request.Params("tabindex") Is Nothing) Then
      tabIndex = CInt(Request.Params("tabindex"))
   End If

   ' Get TabID from querystring
   If Not (Request.Params("tabid") Is Nothing) Then
      tabId = CInt(Request.Params("tabid"))
   End If

   Context.Items.Add("PortalSettings", New PortalSettings(tabIndex, tabId))

End Sub
```

Here you can see that the two `querystring` parameters `tabindex` and `tabid` (referring to the position and type of tab selected respectively) are extracted and passed to the `PortalSettings` constructor to initialize the object. The newly created object is stored as an item called `PortalSettings` in the `Context` object.

The `PortalSettings` class exposes the following public fields:

```
Public PortalId As Integer
Public PortalName As String
Public AlwaysShowEditButton As Boolean
Public DesktopTabs As New ArrayList()
Public MobileTabs As New ArrayList()
Public ActiveTab As New TabSettings()
```

Here `PortalId` and `PortalName` are taken from the `querystring` parameters, and the rest are extracted from the database. The global `AlwaysShowEditButton` field is used by many modules when they render, the `DesktopTabs` and `MobileTabs` fields are lists of properties of tabs that exist (including what roles they are available for, used when determining which tabs to display), and the `ActiveTab` field contains information about the currently selected tab.

`DesktopTabs` and `MobileTabs` are `ArrayList` fields that contain `TabStripDetails` instances. This simple class is as follows:

```
Public Class TabStripDetails

   Public TabId As Integer
   Public TabName As String
   Public TabOrder As Integer
   Public AuthorizedRoles As String

End Class
```

Each tab in the `Tabs` table is placed in an instance of this class, with fields as follows:

TabStripDetails Field Name	Description
`TabId`	The ID of the tab in the `Tabs` table.
`TabName`	The text to display for the tab.
`TabOrder`	Tabs with lower order number are displayed first.
`AuthorizedRoles`	The roles with access to the tab, in the form of a semicolon separated list of role names.

`ActiveTab` is an instance of the `TabSettings` class, defined as follows:

```
Public Class TabSettings

   Public TabIndex As Integer
   Public TabId As Integer
   Public TabName As String
   Public TabOrder As Integer
   Public MobileTabName As String
   Public AuthorizedRoles As String
   Public ShowMobile As Boolean
   Public Modules As New ArrayList()

End Class
```

Some of these fields are duplicates of the ones examined above. The rest of the fields are as follows:

TabSettings Field Name	Description
`TabIndex`	The position of the tab in the title bar.
`MobileTabName`	The text to display for the tab on mobile devices.
`ShowMobile`	Whether this tab should be displayed on a mobile device.
`Modules`	Which modules appear on the tab page.

The `Modules` field is very important, since it contains information about all the modules that exist on a tab. Without this data the page couldn't be rendered. `ActiveTab.Modules` is an `ArrayList` containing a `ModuleSettings` object for every object on the tab. The class definition of `ModuleSettings` is as follows:

```
Public Class ModuleSettings

   Public ModuleId As Integer
   Public ModuleDefId As Integer
   Public TabId As Integer
   Public CacheTime As Integer
```

```
        Public ModuleOrder As Integer
        Public PaneName As String
        Public ModuleTitle As String
        Public AuthorizedEditRoles As String
        Public ShowMobile As Boolean
        Public DesktopSrc As String
        Public MobileSrc As String

      End Class
```

All this information is extracted from the `Modules` table, using the current tab ID to get the relevant modules, except for `DesktopSrc` and `MobileSrc`, which come from the `ModuleDefintions` table for associated module type. The fields contain the following information:

ModuleSettings Field Name	Description
`ModuleId`	The ID of the module in the `Modules` table.
`ModuleDefId`	The corresponding ID of the module in the `ModuleDefinitions` table.
`TabId`	The ID of the tab that the module belongs to.
`CacheTime`	The amount of time to cache the module output for (0 in all cases for the default installation).
`ModuleOrder`	Modules with a lower number here are added to the tab page first.
`PaneName`	Where to display the module on the page (see next section).
`ModuleTitle`	The title to display for the module, if any.
`AuthorizedEditRoles`	Which roles have access to edit the content of the modules.
`ShowMobile`	Whether this module can be displayed on a mobile device.
`DesktopSrc`	The `.ascx` user control source code for the module when displaying on a desktop web browser.
`MobileSrc`	The `.ascx` user control source code for the module when displaying on a mobile web browser, if any.

Now we know what information is available to the rest of the application we can delve deeper into what happens when the main page is loaded.

DesktopDefault.aspx Architecture

We can get a better understanding of `DesktopDefault.aspx` by looking at the ASP.NET code for this page. The important section is the `<table>` element in the form body:

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">
   <tr valign="top">
```

```
        <td colspan="2">
           <portal:Banner id="Banner" SelectedTabIndex="0" runat="server" />
        </td>
     </tr>
     <tr>
        <td>
           <br>
           <table width="100%" cellspacing="0" cellpadding="4" border="0">
              <tr height="*" valign="top">
                 <td width="5">
                     
                 </td>
                 <td id="LeftPane" Visible="false" Width="170" runat="server">
                 </td>
                 <td width="1">
                 </td>
                 <td id="ContentPane" Visible="false" Width="*" runat="server">
                 </td>
                 <td id="RightPane" Visible="false" Width="230" runat="server">
                 </td>
                 <td width="10">
                     
                 </td>
              </tr>
           </table>
        </td>
     </tr>
  </table>
```

Here the `<portal:Banner>` element is mapped to the `DesktopPortalBanner.ascx` user control (which we'll look at in more detail in the next section), and you can see the three columns in the form of three table cells with the names `LeftPane`, `ContentPane`, and `RightPane` respectively:

```
                 <td id="LeftPane" Visible="false" Width="170" runat="server">
                 </td>
                 <td width="1">
                 </td>
                 <td id="ContentPane" Visible="false" Width="*" runat="server">
                 </td>
                 <td id="RightPane" Visible="false" Width="230" runat="server">
                 </td>
```

These columns are populated as part of the `Page_Init` event handler in the code for this page. The code here uses the `PortalSettings` instance created earlier to get the information it needs. One of the first tasks of the code, then, is to retrieve this object from the `Context` object:

```
        ' Obtain PortalSettings from Current Context
        Dim _portalSettings As PortalSettings =
           CType(HttpContext.Current.Items("PortalSettings"), PortalSettings)
```

Next, the code handles some security issues, namely checking to see if the current user has been authenticated and if so whether access to the current tab is permitted. We won't look at this for now since we're covering security issues later.

The currently selected tab determines what modules are displayed. As we saw in the last section, information about these modules is stored in the `Modules` field of the `ActiveTab` field of the `PortalSettings` object that has just been extracted. The rest of the code examines this information and populates the three columns accordingly:

```
            ' Dynamically Populate the Left, Center and Right pane sections of the
            ' portal page
            If _portalSettings.ActiveTab.Modules.Count > 0 Then

               ' Loop through each entry in the configuration system for this tab
               Dim _moduleSettings As ModuleSettings
               For Each _moduleSettings In _portalSettings.ActiveTab.Modules

                  Dim parent As Control = Page.FindControl(_moduleSettings.PaneName)

                  ' If no caching is specified, create the user control instance and
                  ' dynamically inject it into the page.  Otherwise, create a cached
                  ' module instance that may or may not optionally inject the module
                  ' into the tree
                  If _moduleSettings.CacheTime = 0 Then

                     Dim portalModule As PortalModuleControl = _
                        CType(Page.LoadControl(_moduleSettings.DesktopSrc), _
                              PortalModuleControl)

                     portalModule.PortalId = _portalSettings.PortalId
                     portalModule.ModuleConfiguration = _moduleSettings

                     parent.Controls.Add(portalModule)

                  Else

                     Dim portalModule As New CachedPortalModuleControl()

                     portalModule.PortalId = _portalSettings.PortalId
                     portalModule.ModuleConfiguration = _moduleSettings

                     parent.Controls.Add(portalModule)

                  End If

                  ' Dynamically inject separator break between portal modules
                  parent.Controls.Add(New LiteralControl("<" + "br" + ">"))
                  parent.Visible = True

               Next _moduleSettings

            End If

         End Sub
```

An interesting point, with consequences for customization as we will see in the next chapter, is that the `PaneName` information for a module is one of `LeftPane`, `ContentPane`, or `RightPane`, that is, it matches one of the columns on the page. The correct column is found using the `Page.FindControl` method, and used to add the module to.

When the module to add has a `CacheTime` setting of `0`, the user control for the module is loaded using the `Page.LoadControl` method, which returns a `UserControl`, then converted to a `PortalModuleControl` (the base class for all modules user controls as we will see later). It is then initialized and added to the child controls collection of the column that the module will be displayed in. If `CacheTime` is not `0` then a `CachedPortalModuleControl` is used instead, and initialized with enough information to create the required module and cache rendering results.

> *The CacheTime setting actually means 'the amount of time, in seconds, to cache the HTML returned by this control'.*

This pretty much completes the discussion of `DesktopDefault.aspx`. The rest of the functionality of the portal is encapsulated in the user controls that are added to this page.

Tabs and the DesktopPortalBanner.ascx Control

The `DecsktopPortalBanner.ascx` user control appears at the top of `DesktopDefault.aspx`, and displays tabs along with some other information. This control uses the `PortalSettings` object stored in `Context` to find the tabs to display, and user login information to determine whether to display a Log Off link etc. The site title is displayed in a label called `siteName`:

```
<asp:label id="siteName" CssClass="SiteTitle" EnableViewState="false"
           runat="server" />
```

The ASP.NET code for the control also includes an `<asp:datalist>` control for displaying tabs:

```
<asp:datalist id="tabs" cssclass="OtherTabsBg" repeatdirection="horizontal"
              ItemStyle-Height="25" SelectedItemStyle-CssClass="TabBg"
              ItemStyle-BorderWidth="1" EnableViewState="false" runat="server">
   <ItemTemplate>
       
      <a href='<%= Request.ApplicationPath %>/DesktopDefault.aspx?tabindex=
               <%# Container.ItemIndex %>&tabid=<%# Ctype(Container.DataItem,
               TabStripDetails).TabId %>'
         class="OtherTabs">
         <%# Ctype(Container.DataItem, TabStripDetails).TabName %>
      </a>
       
   </ItemTemplate>
   <SelectedItemTemplate>
       
      <span class="SelectedTab">
         <%# Ctype(Container.DataItem, TabStripDetails).TabName %>
      </span>
       
   </SelectedItemTemplate>
</asp:datalist>
```

This control, called `tabs`, displays an array of `TabStripDetails` objects. It uses the position of the tab (as represented by `Container.ItemIndex`) and the ID of the tab (from `TabStripDetails.TabId`, extracted from `Container.DataItem`) to create a URL that users can use to navigate to the tab page. This URL is of the form:

```
(AppPath)/DesktopDefault.aspx?tabindex=(TabIndex)&tabid=(TabId)
```

This is the correct format used by the `Application_BeginRequest()` event handler we saw earlier, when these two parameters are used to initialize a `PortalSettings` object.

The code for this user control is very short, and only consists of a `Page_Load()` event handler. As before, this code needs to extract the `PortalSettings` object:

```
' Obtain PortalSettings from Current Context
Dim _portalSettings As PortalSettings = _
   CType(HttpContext.Current.Items("PortalSettings"), PortalSettings)
```

The name of the site is extracted from this object and used as the `Text` property of the `siteName` label we saw above:

```
' Dynamically Populate the Portal Site Name
siteName.Text = _portalSettings.PortalName
```

Next, we have some security code to check what other information to display on the banner (which we won't look at here), and finally the code to obtain an `ArrayList` of `TabStripDetails` objects to bind to the `tabs` datalist:

```
' Dynamically render portal tab strip
If ShowTabs = True Then

   tabIndex = _portalSettings.ActiveTab.TabIndex

   ' Build list of tabs to be shown to user
   Dim authorizedTabs As New ArrayList()
   Dim addedTabs As Integer = 0

   Dim i As Integer
   For i = 0 To _portalSettings.DesktopTabs.Count - 1

      Dim tab As TabStripDetails = _
         CType(_portalSettings.DesktopTabs(i), TabStripDetails)

      If PortalSecurity.IsInRoles(tab.AuthorizedRoles) Then
         authorizedTabs.Add(tab)
      End If

      If addedTabs = tabIndex Then
         tabs.SelectedIndex = addedTabs
      End If

      addedTabs += 1
```

```
            Next i

            ' Populate Tab List at Top of the Page with authorized tabs
            tabs.DataSource = authorizedTabs
            tabs.DataBind()

         End If

      End Sub
```

This code simply extracts the `TabStripDetails` objects stored in the `PortalSettings` object and adds them to an `ArrayList` called `authorizedTabs` if the current user is in a role that the tab applies to. This `ArrayList` is then databound to the tabs datalist.

Apart from the security code, which we'll leave to Chapter 4, this functionality is pretty straightforward, and results in the tabbed banner we've seen several times already.

Module Structure

Earlier we saw that `DesktopDefault.aspx` added user control modules to three table cells called `LeftPane`, `ContentPane`, and `RightPane` using the following code:

```
Dim portalModule As PortalModuleControl = _
   CType(Page.LoadControl(_moduleSettings.DesktopSrc), _
         PortalModuleControl)

portalModule.PortalId = _portalSettings.PortalId
portalModule.ModuleConfiguration = _moduleSettings

parent.Controls.Add(portalModule)
```

The code in the `.ascx` file for the control is used to create a `PortalModuleControl` object, which then has two of its properties initialized before being added to the controls on the page. Specifically, the `PortalId` property is set to the `PortalId` value extracted from the `PortalSettings` instance used to create the page, and the `ModuleConfiguration` property is set to the `ModuleSettings` object that has previously been extracted from the `PortalSettings` object for this module.

These actions link the module controls into the IBuySpy Portal framework. Since all modules inherit from this `PortalModuleControl` class we should examine this class (which is stored in the `DesktopControls.vb` file) in more detail.

PortalModuleControl

The two properties used by the `DesktopDefault.aspx` above, `PortalId` and `ModuleConfiguration`, are simple - they simply wrap two private fields, an integer value in `_portalId` for `PortalId`, and a `ModuleSettings` value in `_moduleConfiguration` for `ModuleConfiguration`.

`PortalModuleControl` has three other properties: `ModuleId`, `Settings`, and `IsEditable`. The simplest of these, `ModuleId`, simply returns the `ModuleId` field of the stored `ModuleSettings` object. This could just as easily be achieved via the `ModuleConfiguration` property, but since this is a commonly used value, the IBuySpy Portal designers have seen fit to expose it directly.

`Settings` uses the shared `PortalConfiguration.GetModuleSettings` method discussed earlier to extract custom module configuration information from the database, in the form of a `HashTable`.This method is only called once, the first time the `Settings` property is accessed, at which point the returned `HashTable` is stored in the private field `_settings`. Subsequent attempts to access `Settings` have access to this cached version. This mode of operation ensures that as few database accesses as possible are used - modules that don't have custom configuration information won't even access the database once. The code to do this is as follows:

```
<Browsable(False), _
DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)> _
Public ReadOnly Property Settings() As Hashtable

   Get

      If _settings Is Nothing Then
         _settings = PortalSettings.GetModuleSettings(ModuleId)
      End If

      Return _settings
   End Get

End Property
```

`IsEditable` is a Boolean property that is used to determine whether the module is currently editable, a fact that is used by many controls to render an Edit button. There are two situations when this property will be `True`: either the current user will have edit permission granted directly on account of role membership, or the global `PortalSettings.AlwaysShowEditButton` is `True`. The latter case doesn't necessarily mean that the user can edit the content, they still require the correct permission to do this, but it does mean that the Edit button will be visible.

The code for this property makes use of a private `_isEditable` field to cache the result of the calculation of this value:

```
<Browsable(False), _
DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)> _
Public ReadOnly Property IsEditable() As Boolean

   Get

      ' Perform tri-state switch check to avoid having to perform a security
      ' role lookup on every property access (instead caching the result)
      If _isEditable = 0 Then

         ' Obtain PortalSettings from Current Context
         Dim _portalSettings As PortalSettings = _
          CType(HttpContext.Current.Items("PortalSettings"), PortalSettings)
```

```
                If _portalSettings.AlwaysShowEditButton = True Or _
                   PortalSecurity.IsInRoles( _
                      _moduleConfiguration.AuthorizedEditRoles) Then
                   _isEditable = 1
                Else
                   _isEditable = 2
                End If
             End If

             Return _isEditable = 1
          End Get

       End Property
```

Initially, `_isEditable` is set to `0`, so the logic in the middle of the above code is executed. This code results in `_isEditable` being set either to `1` if the `PortalSettings.AlwaysShowEditButton` property is `True` or the user is authorized to edit the module, or `2` otherwise. If `_isEditable` is set to 1 then the property returns a value of `True`, otherwise it returns `False`.

The advantage of doing things this way is that most of the code shown above only executes once, since `_isEditable` cannot be 0 after it has finished processing, only 1 or 2.

Editable modules generally display an Edit button using the `DesktopModuleTitle.ascx` user control.

DesktopModuleTitle.ascx

In order to get a common look and feel for modules, many of them use this user control to display a title and an Edit button. The ASP.NET code for this simple control is as follows:

```
<table width="98%" cellspacing="0" cellpadding="0">
   <tr>
      <td align="left">
         <asp:label id="ModuleTitle" cssclass="Head" EnableViewState="false"
                    runat="server" />
      </td>
      <td align="right">
         <asp:hyperlink id="EditButton" cssclass="CommandButton"
                    EnableViewState="false" runat="server" />
      </td>
   </tr>
   <tr>
      <td colspan="2">
         <hr noshade size="1">
      </td>
   </tr>
</table>
```

A label control (`ModuleTitle`) is used to display the title itself, and a hyperlink (`EditButton`) shows the Edit button. This button only appears if the control is editable, a fact that is determined in the `Page_Load` event handler for the control:

```
Private Sub Page_Load(ByVal sender As System.Object, _
                      ByVal e As System.EventArgs) Handles MyBase.Load

   ' Obtain PortalSettings from Current Context
   Dim _portalSettings As PortalSettings = _
      CType(HttpContext.Current.Items("PortalSettings"), PortalSettings)

   ' Obtain reference to parent portal module
   Dim portalModule As PortalModuleControl = _
      CType(Me.Parent, PortalModuleControl)

   ' Display Modular Title Text and Edit Buttons
   ModuleTitle.Text = portalModule.ModuleConfiguration.ModuleTitle

   ' Display the Edit button if the parent portalmodule has configured the
   ' PortalModuleTitle User Control to display it -- and the current client
   ' has edit access permissions
   If _portalSettings.AlwaysShowEditButton = True Or _
      (PortalSecurity.IsInRoles( _
          portalModule.ModuleConfiguration.AuthorizedEditRoles) And _
      Not (EditText Is Nothing)) Then

      EditButton.Text = EditText
      EditButton.NavigateUrl = EditUrl + "?mid=" + _
         portalModule.ModuleId.ToString()
      EditButton.Target = EditTarget
   End If

End Sub
```

This code sets the `Text` property of `ModuleTitle` label to a value extracted from the `ModuleSettings` object obtained from the parent module, and the `Text`, `NavigateUrl`, and `Target` properties of `EditButton` to values taken prom the parameters used in the ASP.NET code used to insert the control. The URL for `EditButton` also has an additional querystring parameter added, `mid` - which is set to the module `ID`.

User Controls

The code for the various module user controls varies quite a lot for the ten modules included with the IBuySpy Portal download, but they share several common features:

- They all inherit from `PortalModuleControl`.
- Many of them use `DesktopModuleTitle.ascx` to render a title and edit link.
- Many of them use a separate data object for accessing database-stored data.

To illustrate all these features, let's look at a control that exhibits all three of the above: `Announcements.ascx`. The complete ASP.NET code for this control is as follows:

```
<%@ Control language="vb" Inherits="ASPNetPortal.Announcements"
            CodeBehind="Announcements.ascx.vb" AutoEventWireup="false" %>
<%@ Register TagPrefix="Portal" TagName="Title" Src="~/DesktopModuleTitle.ascx"%>
<portal:title EditText="Add New Announcement"
            EditUrl="~/DesktopModules/EditAnnouncements.aspx" runat="server" />
<asp:DataList id="myDataList" CellPadding="4" Width="98%" EnableViewState="false"
            runat="server">
   <ItemTemplate>
      <asp:HyperLink id="editLink" ImageUrl="~/images/edit.gif"
            NavigateUrl='<%# "~/DesktopModules/EditAnnouncements.aspx?ItemID=" &
            DataBinder.Eval(Container.DataItem,"ItemID") & "&mid=" & ModuleId %>'
            Visible="<%# IsEditable %>" runat="server" />
      <span class="ItemTitle">
         <%# DataBinder.Eval(Container.DataItem,"Title") %>
      </span>
      <br>
      <span class="Normal">
          <%# DataBinder.Eval(Container.DataItem,"Description") %>
           
          <asp:HyperLink id="moreLink"
             NavigateUrl='<%# DataBinder.Eval(Container.DataItem,"MoreLink") %>'
             Visible='<%# DataBinder.Eval(Container.DataItem,"MoreLink") <>
             String.Empty %>' runat="server">
                read more...
          </asp:HyperLink>
      </span>
      <br>
   </ItemTemplate>
</asp:DataList>
```

Near the top of this code you can see the `<portal:title>` control that is used for the `DesktopModuleTitle.ascx` control detailed in the last section, with appropriate parameters used to provide a link to the edit page for the module. We won't look at `EditAnnouncements.aspx` here since the code is simple and makes use of data access objects in the same way as this user control, so we won't learn anything new there.

The main body of this module is a `DataList`, which is set to display the announcements obtained from the database. We'll come back to the specifics of this control shortly, once we've seen the data it is used to display.

The `DataList` control (`myDataList`) is initialized in the `Page_Load` event handler for the module (which constitutes 100% of the code for the user control):

```
        Private Sub Page_Load(ByVal sender As System.Object, _
                              ByVal e As System.EventArgs) Handles MyBase.Load

           ' Obtain announcement information from Announcements table
           ' and bind to the datalist control
           Dim announcements As New ASPNetPortal.AnnouncementsDB()
```

```
            ' DataBind Announcements to DataList Control
            myDataList.DataSource = announcements.GetAnnouncements(ModuleId)
            myDataList.DataBind()

        End Sub
```

This code uses a data object, `AnnouncementsDB`, to obtain data for filling the `DataList`. The `AnnouncementsDB.GetAnnouncements` method gets a `DataSet` filled with the announcements that correspond with this particular module (as specified by the ID of the module). This `DataSet` is then databound to the `DataList` control.

Since we haven't looked in detail at one of the data objects in this application, this is a good opportunity to do so. The method used, `GetAnnouncements`, is as follows:

```
        Public Function GetAnnouncements(ByVal moduleId As Integer) As DataSet

            ' Create Instance of Connection and Command Object
            Dim myConnection As _
             New SqlConnection(ConfigurationSettings.AppSettings("connectionString"))
            Dim myCommand As New SqlDataAdapter("GetAnnouncements", myConnection)

            ' Mark the Command as a SPROC
            myCommand.SelectCommand.CommandType = CommandType.StoredProcedure

            ' Add Parameters to SPROC
            Dim parameterModuleId As New SqlParameter("@ModuleId", SqlDbType.Int, 4)
            parameterModuleId.Value = moduleId
            myCommand.SelectCommand.Parameters.Add(parameterModuleId)

            ' Create and Fill the DataSet
            Dim myDataSet As New DataSet()
            myCommand.Fill(myDataSet)

            ' Return the DataSet
            Return myDataSet

        End Function
```

The connection to the database is initialized using the `connectionString` value from the `AppSettings` of the application (stored in `web.config`). The `moduleId` parameter obtained from the module using this data object is then used as a parameter for the `GetAnnouncements` SQL stored procedure to obtain the required data.

The `GetAnnouncements` stored procedure is as follows:

```
CREATE PROCEDURE GetAnnouncements
(
    @ModuleID int
)
AS

SELECT
```

```
    ItemID,
    CreatedByUser,
    CreatedDate,
    Title,
    MoreLink,
    MobileMoreLink,
    ExpireDate,
    Description

FROM
    Announcements

WHERE
    ModuleID = @ModuleID
  AND
    ExpireDate > GetDate()

GO
```

This stored procedure wraps a simple SQL `SELECT` statement, returning all columns except `ModuleId` from the `Announcements` table for records with the selected `ModuleId` value.

The `AnnouncementsDB` data object also has methods for getting individual announcements, for deleting announcements, and for adding new announcements. These are used by the `EditAnnouncements.aspx` page.

Back in the ASP.NET code for the `Announcements.ascx` user control, we can see that the `<ItemTemplate>` starts with a link for editing the announcement item:

```
<asp:DataList id="myDataList" CellPadding="4" Width="98%" EnableViewState="false"
              runat="server">
   <ItemTemplate>
      <asp:HyperLink id="editLink" ImageUrl="~/images/edit.gif"
             NavigateUrl='<%# "~/DesktopModules/EditAnnouncements.aspx?ItemID=" &
             DataBinder.Eval(Container.DataItem,"ItemID") & "&mid=" & ModuleId %>'
             Visible="<%# IsEditable %>" runat="server" />
```

The URL of the page for editing the item is hardcoded here, and is combined with an `ItemID` querystring parameter taken from the `ItemID` field of the announcement being displayed and a `mid` querystring parameter taken from the `PortalModuleControl.ModuleId` property. This link is only displayed if the module is currently editable, making use of the `IsEditable` property of `PortalModuleControl` we saw earlier.

Next, we have the announcement title:

```
      <span class="ItemTitle">
         <%# DataBinder.Eval(Container.DataItem,"Title") %>
      </span>
```

This uses the `Title` field of the current announcement. Similarly, the `Description` field of the announcement is used to give the announcement summary, and `MoreLink` is used to display a link to more text if the field is present:

```
        <br>
        <span class="Normal">
            <%# DataBinder.Eval(Container.DataItem,"Description") %>
             
            <asp:HyperLink id="moreLink"
               NavigateUrl='<%# DataBinder.Eval(Container.DataItem,"MoreLink") %>'
               Visible='<%# DataBinder.Eval(Container.DataItem,"MoreLink") <>
               String.Empty %>' runat="server">
                  read more...
            </asp:HyperLink>
        </span>
        <br>
    </ItemTemplate>
</asp:DataList>
```

That is all that is required for a IBuySpy Portal module. In the next chapter we'll look at how we can make our own modules using the framework described here, recapping much of this information.

Security

As mentioned earlier in the chapter, the IBuySpy Portal works with both forms-based and Windows authentication.

Forms-based authentication is an independent authentication method that works equally well for local and remote users of the intranet, since login information is requested from the user as part of the web application, using the authentication module we looked at earlier.

Windows integrated authentication involves tying the intranet authentication in with existing network accounts. This is great for local network intranets, since users only need to log on once, when they log on to their Windows domain account. The account information is then passed on to the intranet code when they access the site, so that they don't need to enter additional information. This has the effect of streamlining things, and makes the intranet feel more like a corporate application. The main benefit is that all user account information is centralized.

There are, however, problems with Windows integrated authentication. For a start, we are assuming that all of our users are using Windows operating systems. If our organization uses other operating systems as well, such as Linux or Macintosh consoles, then it can be a bit tricky to configure the network users and groups correctly, resulting in many annoyed users being unable to connect to the intranet. I'm not saying that this is impossible - it's just more trouble than it's worth in most cases. In addition, remote users may find it difficult to log on to their network account, or might not even have such an account. There are also security concerns, since transmitting this information over the Internet is by definition less secure than doing so over a local network.

Security is covered in more depth in Chapter 4, where we will look at how the forms authentication system works and how we can configure IBuySpy Portal-based intranets for Windows authentication if desired.

Summary

In this chapter, we have taken a fairly in-depth tour of the IBuySpy Portal intranet architecture. We started out in general terms, looking at what is provided via the interface created for us. Once we got past the 'wow' stage, we proceeded to look at things in more depth, and at how things fit together. We covered the backend for the database structure, and the important user controls and types that encapsulate the intranet functionality. We saw how the framework works and how individual modules communicate with the framework as a whole.

The rest of this book is concerned with customizing this architecture, in both simple and advanced ways. Now we have a good understanding of how things work, we can start to do more interesting things with the code. In the next chapter, we'll look at the basic principles behind this.