

5

The Anatomy of a SharePoint Page

In Part I, you learned about the basic functions of SharePoint Designer, some of the capabilities of SharePoint itself, and how various SharePoint Features were represented in SharePoint Designer. In Part II, you discover how to use SharePoint Designer to customize the look and feel of your site.

Muscle and blood, skin and bones make up the body of a person. This chapter takes you deep inside a SharePoint Page to show you the parts and processes that make up the structure (body). You will learn about:

- ❑ The page assembly process.
- ❑ What it means to customize a page.
- ❑ The roles of styles and Web Parts.
- ❑ How to build the skeleton of your site with Master Pages.

Bits and Pieces

Web pages have always been a bit like a jigsaw puzzle. The page name a user enters into a browser's address bar rarely contains everything that will be displayed in the window by the time the rendering process is completed. Images, scripts, and style sheets are simply stated by reference, and loaded by the browser.

CGI (Common Gateway Interface) and other server-side programming systems like ASP (Active Server Pages), PHP, and ASP.NET made the rendering of Internet sites even less direct. They automate the generation of the HTML, which is used to assemble the pieces for the final page rendering. While these systems have often been used to present information stored in databases, rarely has the combination of database, application code, and physical files been as balanced as that used by SharePoint.

Here, There, and Everywhere

Leaving aside the client-side final assembly by the browser, a SharePoint server normally combines information from a number of locations to create a page.

The first stage is the configuration database. Based upon the URL entered, SharePoint determines which web application and site collection are being requested, and which content database contains files for that site collection. The server then looks up which row in the AllUserData table of that database represents the page in question.

This is where things start to get interesting. Most pages on a SharePoint site are assembled from a combination of a file-based template and instance information from the content database. While this instance information may tell the server to retrieve even more data from other places (such as a SharePoint list), the core structure of the page will still be served from the site definition on the web server's file system. In earlier versions of SharePoint, this two-part file service was called ghosting (a term still used by many people) because the instance information stored in the content database was just a "ghost" — the real page was on the web front-end server.

That is, unless a page had been customized in a tool such as SharePoint Designer. Because SharePoint Designer accesses the file through a web protocol, it has no capability to modify the files on a server directly. (This is a good thing, because if your change were saved to the template file, it could impact every page on every site served by the server.) Instead, after you customize a page and save it, SharePoint breaks the link to the file system, and stores the entire page in the content database. SharePoint gives the warning shown in Figure 5-1 before breaking this link. Because the page being pulled from the database is now real, it was called unghosted.

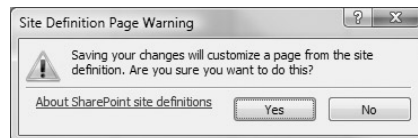


Figure 5-1

No Longer Seeing Ghosts

For Windows SharePoint Services 3.0 and Microsoft Office SharePoint Server 2007, the terminology in the official documentation changed from the prior versions. Ghosted pages are now referred to as *uncustomized*, and unghosted pages are now simply *customized*. Throughout most of this book, the new terms are preferred, though occasionally both may be used where needed for clarity.

Figure 5-2 shows the process by which a page is assembled by a SharePoint server.

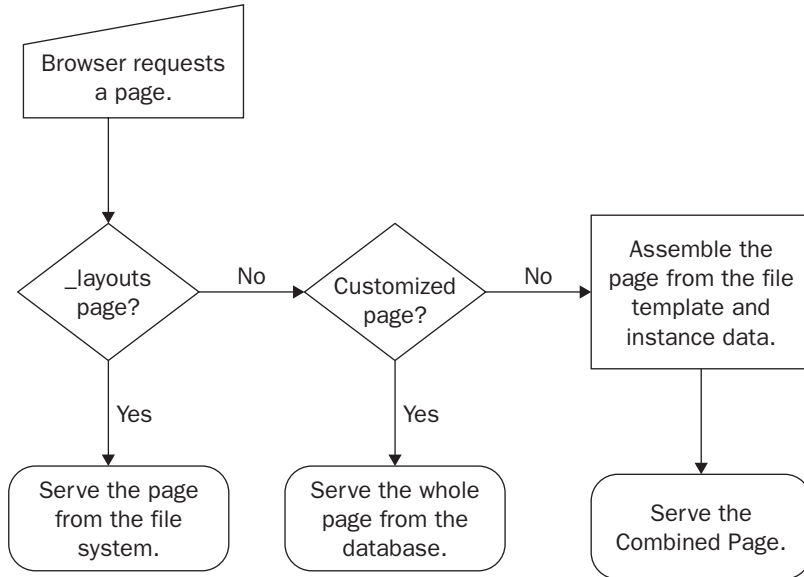


Figure 5-2

Ramifications and Reversion

Naturally, changing the way your page is assembled has a number of ramifications. Most of them are minor, and most are far outweighed by the flexibility you gain from the customizations you make in SharePoint Designer. Nevertheless, there are two key impacts you should understand so that you can make informed choices about when (and when not) to customize pages:

- ❑ Lost ties to the underlying site definition.
- ❑ Potential impacts to performance.

The sections below explain the ramifications of these changes, and show you how to return a page to its uncustomized state.

Lost Ties

The biggest impact from customizing a page is the loss of the link to the template page in the site definition. Once you customize a page, any changes made to the template file will not be reflected in the customized pages.

Part II: Customizing the SharePoint Look and Feel

While, in theory, this could be considered a major loss, in practice it usually is not. There are two main reasons:

1. It is generally considered bad practice to directly modify the template files of the default site definitions. These files are potentially subject to being overwritten by patches and upgrades. Any changes you made to these files would, in those cases, be lost, and you would need to recreate them.
2. Modifying any site definition (default or custom), after sites based on it have been deployed, is not officially supported by Microsoft. The preferred method for changing sites based on existing definitions is to create and activate stapled features. A *stapled* feature is one that is linked to a preexisting site definition. Although not the easiest process, features can be changed/updated at a later date. See the SharePoint SDKs for further information on features and feature staples.

Of course, that doesn't prevent people from making those kinds of changes. In some cases, they can be an expedient way of enforcing company standards across the entire environment. Nevertheless, if you are seriously concerned that you may need to change these server files, you might wish to avoid customizing pages based on them.

Performance

It is a fact of life that retrieving information directly from a local hard disk will be faster than pulling it from a (possibly remote) database. Because customizing a page forces the entire page to be read from the database, there is indeed a small performance penalty for customized files. How small? That depends upon a number of factors, so there is no way to give a definite percentage, but the impact truly is minor. Consider this — whether the whole page is in the database, or just the ghost, *you still need to read the database to find out!* It isn't as if you don't read the database at all when the file is not customized, so the difference is minimal.

In addition, SharePoint is designed with built-in caching mechanisms to ensure that as few database hits as possible are performed under any circumstances. Under all but the most demanding conditions, you don't need to worry about the relative performance of customized and uncustomized pages.

Back to the Future

So, what happens if you have made changes to a page in SharePoint Designer, and for whatever reason, decide you need to get back to the original disk files? Fortunately, there is an easy process for that. Figure 5-3 shows a customized file (note the extra “i,” or customized, icon) upon which the right-click context menu has been summoned. Choosing Reset to Site Definition will make a backup copy of the existing page, and change the entry in the content database so that it is once again served from the file system.

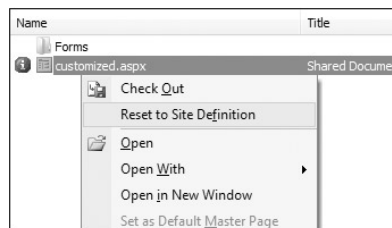


Figure 5-3

Pages created completely from scratch in SharePoint Designer (i.e., not based on one of the SharePoint Page templates) will not have the file-system template to pull from, and therefore are always served from the content database. The Reset to Site Definition option is not available on these files, nor are they flagged with the “customized” icon.

Restoring a page’s link to its site definition does not delete the instance information for the page. It remains in the content database, and continues to inform the rendering of the page. This means, for example, that the page title, and any Web Parts you have placed in Web Part Zones on the page, will remain intact.

The Special Case of the `_layouts` Folder

There is a set of pages to which the above service process does not apply. In fact, they are not held in the content database at all. These are the files that are served from the `_layouts` folder of a site. The `_layouts` folder is the location for various system pages, such as Site Settings, that access the administrative APIs, or perform other tasks at a very deep level in SharePoint. While most `_layouts` files are administrative in nature, and therefore not seen by a typical site user, there are some that may bleed through. The All Site Content Page (`./_layouts/viewlsts.aspx`) is a prime example.

The `_layouts` folder is a *virtualization* of a folder from the 12 Hive, typically `C:\program files\common files\microsoft shared\web server extensions\12\template\layouts\`. Every SharePoint site in the farm serves `_layouts` files directly from this physical path. Layouts files are not assembled from the database, and do not use the local site’s Master Page (discussed below). This folder is read-only from the web sites, and does not appear in folder lists generated via the SharePoint Web API.

Because SharePoint Designer reads and saves SharePoint files via the Web API, you will notice that the `_layouts` folder does not appear when you open a site in SharePoint Designer. You are not able to access or make changes to `_layouts` files. This also means that most changes you make to an individual site will not be reflected in files served from the `_layouts` folder.

There is only one exception. Files served from the `_layouts` folder will respect the local site’s Theme, or a globally applied CSS style sheet. By ensuring that as much of your site’s branding as possible is achieved through the Theme or CSS, you will minimize the discontinuity between those `_layouts` files that are seen and the rest of your site. Use of CSS and Themes in SharePoint will be discussed in detail in chapter 7.

Thinking outside the Box

One of the approved locations to install custom web-based applications (not to be confused with SharePoint Web Applications) which make use of the SharePoint binary API is 12 Hive’s `.\template\layouts` directory.

Files in the `_layouts` folder are served directly from the web front-end server, without going through SharePoint’s safety mechanisms. While this allows great power in deploying an application that is then available to all sites in your SharePoint farm, it is accompanied by corresponding risks. Be very certain of the source and quality of any applications deployed in `.\template\layouts`.

Web Part Pages

You can create many types of pages and files with SharePoint Designer, and use most of them on a SharePoint site. You can even display SharePoint content on some of them (general .ASPX pages, for example); however, in most cases, you need to manually generate many of your own user interface components. While there may be times this is exactly what you want or need to do (and this book assumes you have the fundamental web design skills to do so), most of the time you will be better served by using SharePoint Web Part Pages.

Virtually all of the pages you see in SharePoint are forms of the Web Part Page. Technically, a Web Part Page is any .ASPX page inheriting from the `Microsoft.SharePoint.WebPartPages.WebPartPage` assembly. In practice, most SharePoint Web Part Pages have the following key characteristics:

- ❑ They are .ASPX (ASP.NET) form pages.
- ❑ They use a .NET Master Page that contains SharePoint-related placeholders (including the requisite `WebPartPage` inheritance just mentioned).
- ❑ The content area contains one or more *Web Part Zones*.

There are a number of Web Part Page templates provided with SharePoint, which can be instantiated in a document library by end users. Users can then add and connect Web Parts through the web interface. You can add and connect Web Parts through SharePoint Designer as well. In addition, with SharePoint Designer, you can fully customize the content area of these pages.

Although you can edit pages based on these templates from within SharePoint Designer, you cannot directly create instances of them.

There are also several specialty page types. These specialty types, while still fundamentally Web Part Pages, are not usually referred to as such. In particular, MOSS Publishing pages have another layer to their architecture, which will be described more fully in chapter 8.

Files served from the `_layouts` folder, though they may contain Web Parts, are not Web Part Pages. They cannot contain a Web Part Zone, and therefore cannot have Web Parts added dynamically at run time.

Master of Their Own Destiny

To fully take advantage of the functionality ASP.NET has to offer, all SharePoint-generated pages are derived from Master Pages. A Master Page is essentially a template, which provides elements that are common across a number of pages. These elements can range from navigation to style sheet links, to placeholders for the users' content. The pages that use a Master Page are called *Content Pages*, described in the next section.

As shown in Figure 5-4, you can easily create new Master Page–derived pages as well.

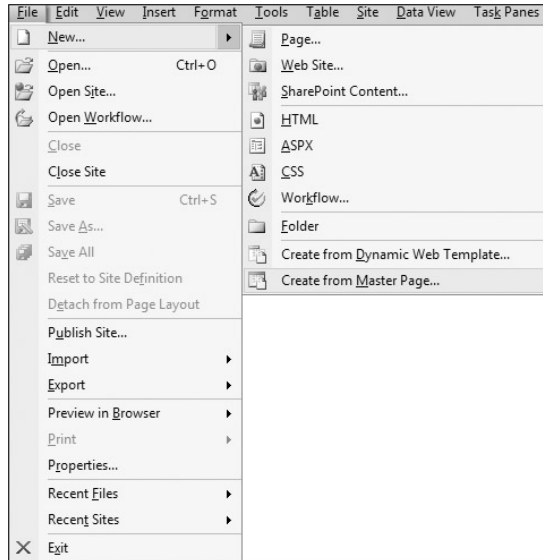


Figure 5-4

The SharePoint server manages several Master Pages, over two of which you have control in SharePoint Designer. You can also create arbitrary Master Pages of your own, and derive new Content Pages from them. Figure 5-5 shows the Master Page selector that results from choosing File → New → Create from Master Page.

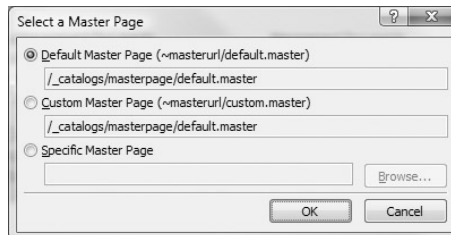


Figure 5-5

The Default Master is used by default. `default.aspx` and most other pages in WSS. The Custom Master is not used as often in WSS, but figures prominently in MOSS Publishing sites. There is no requirement that these point to different Master Pages; they simply give you the option to do so for different sets of pages. You will get an extensive look inside a Master Page later in this chapter.

A Few Words about Dynamic Web Templates

Users of Dreamweaver or FrontPage may have recognized the .DWT extension in earlier figures. DWT stands for either Dynamic Web Template or Dreamweaver Web Template — the file format is the same, though the name is different in the two products. Like Master Pages, .DWT provides a way for web designers to create a common page design, with regions for Content Editors to fill in with the actual page information.

Although it is possible for you to create .DWT files and pages based on .DWT in a SharePoint site, you should *not* use them for creating SharePoint pages: .DWT is provided primarily for working with types of sites other than SharePoint.

Inside a Content Page

Figure 5-6 shows a freshly created Content Page, `NewSPPage.aspx`, which is based upon (or *inherits*) the site's `Default.master`. The Split view lets you see the full “chrome” typical of a SharePoint site in the Design pane, whereas the Code pane on top shows the entire functional code of the page.

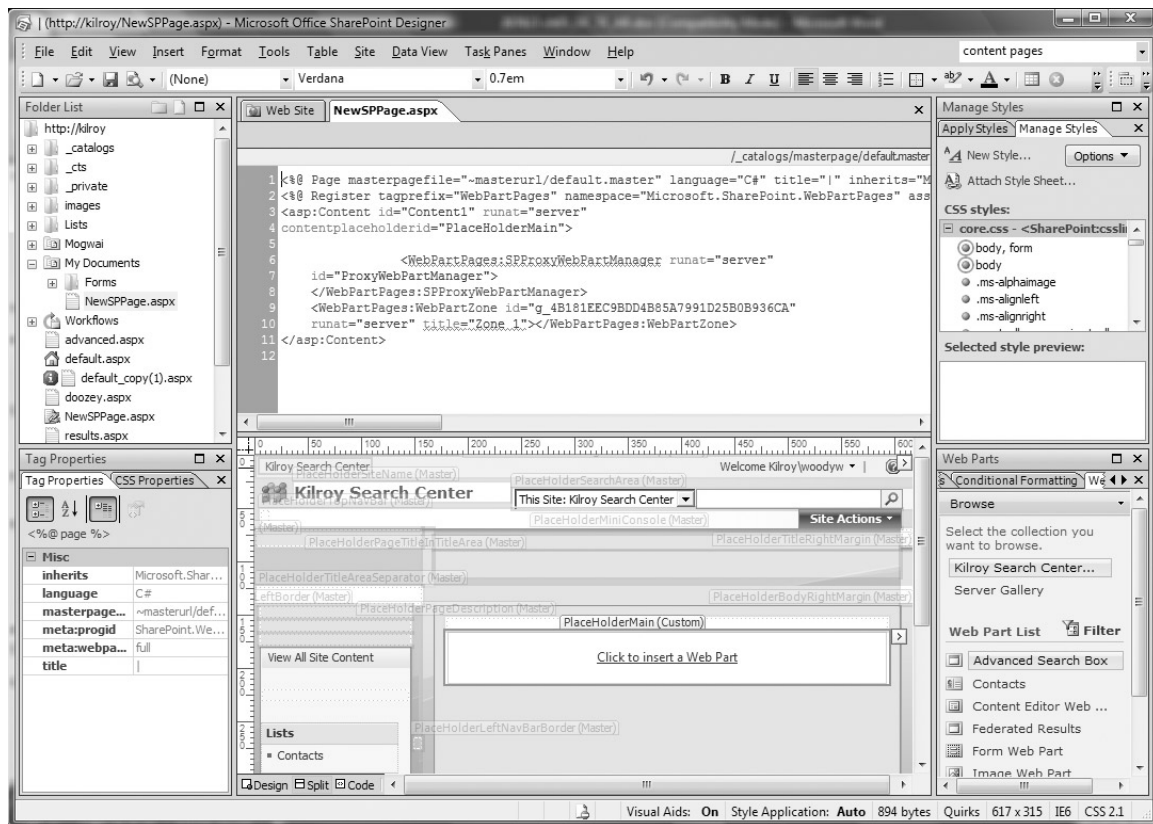


Figure 5-6

There are only ten lines of active code on this page.

Lines 1 and 2 define the Master Page to be used, and declare the page a WebPartPage. The Master Page is usually declared with one of two *tokens*: `~masterurl/default.master` or `~masterurl/custom.master`. This will force the page to use either the site's default or Custom Master Page, as selected previously in Figure 5-5. This is distinct from the page actually named `default.master`.

```
<%@ Page masterpagefile="~masterurl/default.master" language="C#"
  title="|" inherits="Microsoft.SharePoint.WebPartPages.WebPartPage,
Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
  PublicKeyToken=71e9bce11e9429c" meta:webpartpageexpansion="full"
  meta:progid="SharePoint.WebPartPage.Document" %>
<% Register tagprefix="WebPartPages"
  namespace="Microsoft.SharePoint.WebPartPages"
  assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
  PublicKeyToken=71e9bce11e9429c" %>
```

Lines 3 and 5 are empty placeholders for additional page header information.

Lines 6 through 13 define the main content of the page, which in this case consists of a single Web Part Zone, and the Web Part support module. This is held in an `<asp:Content>` tag, and will substitute in the Master Page for the corresponding `"PlaceHolderMain"` `<asp:ContentPlaceholder>`.

```
<asp:Content id="Content2" runat="server"
  contentplaceholderid="PlaceHolderMain">

  <WebPartPages:SPProxyWebPartManager runat="server"
    id="ProxyWebPartManager">
  </WebPartPages:SPProxyWebPartManager>
  <WebPartPages:WebPartZone
    id="g_D519CCB8D15C4984807064C581F37CC3" runat="server"
    title="Zone 1">
  </WebPartPages:WebPartZone>

</asp:Content>
```

When working with content placeholders in Code view, you may see elements underlined by the red squiggles that IntelliSense uses to indicate a potential problem. If you hover over the element, the details will appear to explain how the code is invalid for a particular browser. This is because the editor is interpreting these as client-side elements, when in actuality they will be rendered by the server. You can safely ignore these "errors."

Other than the Web Part Zone, everything that produces the rendering seen in the Design pane comes from the Master Page. Just above the Code pane, on the right-hand edge, SharePoint Designer shows you the address of the Master Page actually in use (expanded from the token above).

*On SharePoint sites, you rarely edit Content Pages with SharePoint Designer, unless you are using a **Data View** Web Part (see chapter 9). Most of your changes will be to the site's Master Pages, style sheets, and layout pages.*

List and Library View Pages

In chapter 4, you saw the folder structure of SharePoint lists and libraries. In the root of each list, and the Forms folder of each library, is a collection of pages used for entering and displaying the information in that list. These are called view pages, and are also Web Part Pages.

End users can create view pages through the web interface, but unlike the generic Web Part Pages mentioned above, view pages can be both edited *and* created in SharePoint Designer. When creating a new page, select the List View Pages section, as shown in Figure 5-7.

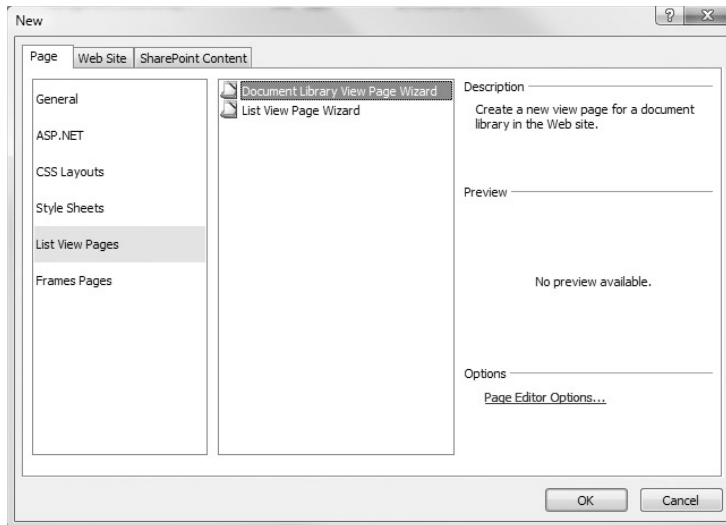


Figure 5-7

Notice that the create options refer to List and Library View Page *Wizards*. As wizards go, these are pretty simple. Figure 5-8 shows the whole wizard. You are asked only to select the (existing) list or library the form will be used for, and to enter the name you want to give the page.

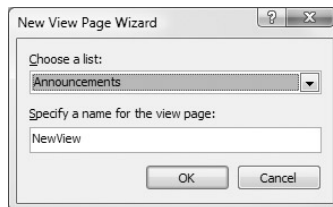


Figure 5-8

The result of the wizard is a Web Part Page in the view folder (either the root of the list, or the `./forms` subfolder of a library, as seen in chapter 4) of the list or library selected in the wizard. This page contains a single Web Part Zone (like the example above), which by default is populated with a Web Part for the default view of the list or library in question. When creating custom view forms in SharePoint Designer, you will almost always discard the default view part, and make extensive use of Data View Web Parts.

SharePoint Master Pages — A Deep Dive

Master Pages are used throughout SharePoint to provide common navigation and apply a consistent look and feel. As stated earlier, a Master Page is a combination of common elements and replaceable *content regions*.

Content regions are represented by `<asp:ContentPlaceHolder>` tags. These tags are assigned various properties and default content. The default content in a region can be replaced on the individual pages based on this Master. You can use the Region Types feature discussed in chapter 4 to restrict the kinds of changes made to content regions in SharePoint Designer.

In addition to the content region placeholders, a SharePoint Master Page contains tags that aren't replaced by content, but rather control page layout, set fonts, or invoke or render a particular piece of SharePoint functionality.

The Default Style Sheets

The default.master Master Page provided in SharePoint is over 400 lines long. Most of these lines are typical HTML tags defining the placement and style of various elements, defining the standard SharePoint look and feel. All of the styles used are defined in a style sheet called `core.css`. The following lines of code load the `core.css` style sheet, and any Theme that may be applied to the site.

```
<SharePoint:CssLink runat="server"/>
<SharePoint:Theme runat="server"/>
```

In Figure 5-9, these two lines have been removed, and `default.master` is saved as `nostyle.master`. The remainder of the Master Page is unchanged. `Nostyle.master` was then applied as the Default Master Page of the same site you have seen before.

Part II: Customizing the SharePoint Look and Feel

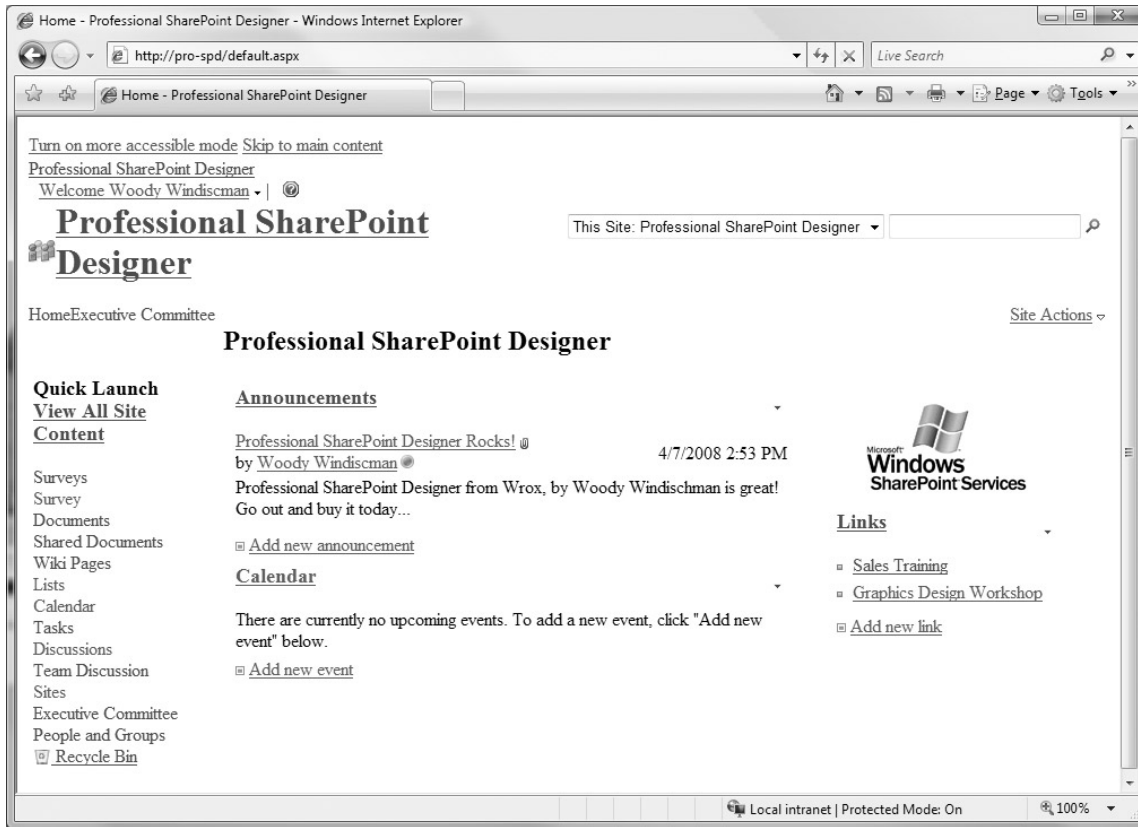


Figure 5-9

Everything still works. Most things are pretty close to the same location as well. This is because default .master uses tables to lay out most of the elements on the page. Different tables, cells, and other elements are assigned CSS classes, which are defined in core.css, and can be overridden by styles in a Theme., or as defined and invoked by you.

In many situations, you can satisfy the branding needs of a client purely with CSS and associated images, and leave all SharePoint's functionality intact. The styles used in Themes and core.css, and the default.master elements they apply to, will be covered in detail in chapter 7.

The Bare Necessities

The remainder of this chapter will focus on the content regions and functional elements that are necessary to the operation of SharePoint. In the process, you will see how to create a pure function or minimal Master Page, which you can then further customize with the branding and other look and feel elements you need.

Remember that such a Master Page will not apply to pages served from the `_layouts` folder.

Content Regions

SharePoint's default.master contains the 31 content regions listed in the following table. These will be described in detail, and in the order they normally appear in the page, later in this section, along with the default content they contain.

Placeholder Name	Default Content/Function
PlaceholderAdditionalPageHead	Placeholder used for any extra markup you may need between the <code><head></code> tags on a particular page.
PlaceholderBodyAreaClass	Extra styles injected at the bottom of a page rendering, thus ensuring they take priority.
PlaceholderBodyLeftBorder	Element left of the page body.
PlaceholderBodyRightMargin	Element right of the page body.
PlaceholderCalendarNavigator	Calendar View Date Navigator/Picker.
PlaceholderFormDigest	Contains encrypted information when using Forms and Digest authentication.
PlaceholderGlobalNavigation	Breadcrumb at the top of the page (usually to the portal site, if any).
PlaceholderHorizontalNav	The tab site navigation element.
PlaceholderLeftActions	Bottom of the left navigation area.
PlaceholderLeftNavBar	Container for the Quick Launch bar.
PlaceholderLeftNavBarBorder	Right side element on the left navigation bar.
PlaceholderLeftNavBarDataSource	Data source for the Quick Launch menu.
PlaceholderLeftNavBarTop	Top of the left navigation area
PlaceholderMain	The main content of the page.
PlaceholderMiniConsole	Certain commands appropriate to a particular page, such as Wiki editing.
PlaceholderNavSpacer	Sets the width of the left navigation area.
PlaceholderPageDescription	Description text from the page's definition.
PlaceholderPageImage	Icon in the page title area.
PlaceholderPageTitle	The title that is shown in the browser's title bar (within the <code><title></code> tag).

(continued)

Part II: Customizing the SharePoint Look and Feel

Placeholder Name	Default Content/Function
PlaceHolderSearchArea	Holds the Search box control.
PlaceHolderSiteName	The site name as set in Site Settings.
PlaceHolderTitleAreaClass	Extra styles injected at the bottom of a page rendering, thus ensuring they take priority.
PlaceHolderTitleAreaSeparator	Below the title area.
PlaceHolderTitleBreadcrumb	Breadcrumb in the page title area.
PlaceHolderTitleInTitleArea	Page title (shown immediately below the breadcrumb in default.master).
PlaceHolderTitleLeftBorder	Element left of the title area.
PlaceHolderTitleRightMargin	Element right of the title area.
PlaceHolderTopNavBar	Top navigation area including the tabs and Site Settings menu.
PlaceHolderUtilityContent	Extra content that needs to be at the bottom of the page.
SPNavigation	Empty by default in Windows SharePoint Services; can be used for additional page-editing controls.
WSSDesignConsole	The page-editing controls when the page is in Edit Page mode (after clicking Site Actions, then Edit Page).

What if you don't want to show all of these regions on your site? Can you just delete the placeholders and their contents? In most cases, the answer is no. This is because of the way a Master-based .ASPX page is built at run time.

Many of the content placeholders contain default content, and many do not. Those that start out empty rely on the Content Page to provide information to inject into them if needed. Placeholders that are not empty will render the default content unless the Content Page provides its own content for that placeholder. In that case, the Content Page's content will override (be substituted for) the default content. While there is no requirement that a Content Page fill all of the potential slots in a Master, the reverse is not the case. A Master Page must have a corresponding placeholder for every element a Content Page tries to inject. If it does not, the ASP.NET engine will generate an error; however, your users will not be subjected to the raw ASP.NET error message. Instead, SharePoint captures the error and produces a page similar to Figure 5-10.

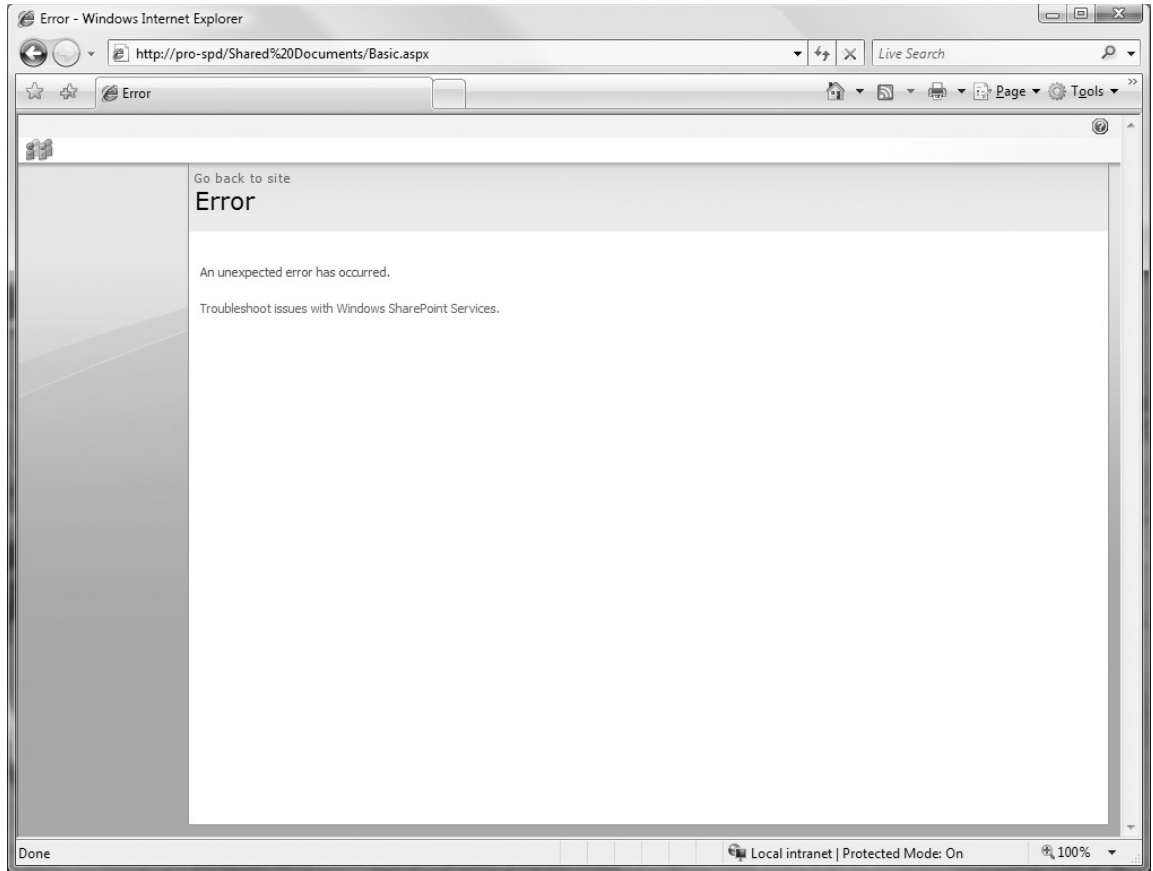


Figure 5-10

One way to suppress display of the content regions you aren't planning to use, but still allow the page to work, is to include empty versions of the placeholders in an `<asp:panel>` control, with the `visible` property set to `false`, as shown below.

```
<asp:Panel visible="false" runat="server">
<!-- The ContentPlaceHolders in this group are required to
    exist (although they don't have to be visible) in any master
    used by a SharePoint page. --%>
<asp:ContentPlaceholder id="PlaceholderAdditionalPageHead" runat="server" />
<asp:ContentPlaceholder id="PlaceholderPageTitle" runat="server" />
<asp:ContentPlaceholder id="PlaceholderSearchArea" runat="server" />
<asp:ContentPlaceholder id="PlaceholderTitleBreadcrumb" runat="server" />
<asp:ContentPlaceholder id="PlaceholderPageTitleInTitleArea"
    runat="server" />
<asp:ContentPlaceholder id="PlaceholderLeftNavBar" runat="server" />
<asp:ContentPlaceholder ID="PlaceholderPageImage" runat="server" />
<asp:ContentPlaceholder ID="PlaceholderBodyLeftBorder" runat="server" />
<asp:ContentPlaceholder ID="PlaceholderNavSpacer" runat="server" />
```

Part II: Customizing the SharePoint Look and Feel

```
<asp:ContentPlaceHolder ID="PlaceHolderTitleLeftBorder" runat="server"/>
<asp:ContentPlaceHolder ID="PlaceHolderTitleAreaSeparator" runat="server"/>
<asp:ContentPlaceHolder ID="PlaceHolderMiniConsole" runat="server"/>
<asp:ContentPlaceHolder id="PlaceHolderCalendarNavigator" runat="server" />
<asp:ContentPlaceHolder id="PlaceHolderLeftActions" runat="server"/>
<asp:ContentPlaceHolder id="PlaceHolderPageDescription" runat="server"/>
<asp:ContentPlaceHolder id="PlaceHolderBodyAreaClass" runat="server"/>
<asp:ContentPlaceHolder id="PlaceHolderTitleAreaClass" runat="server"/>
<asp:ContentPlaceHolder id="PlaceHolderBodyRightMargin" runat="server" />
</asp:Panel>
```

The ContentPlaceHolder highlighted below is not absolutely required for MOSS Publishing pages; however, it is used by various other SharePoint pages (e.g., library instantiated Web Part Pages). You should include it on any general-purpose Master for use in Team Collaboration sites.

```
<asp:ContentPlaceHolder id="PlaceHolderBodyRightMargin" runat="server" />
<asp:ContentPlaceHolder ID="PlaceHolderTitleRightMargin" runat="server" />
</asp:Panel>
```

The items in this final group of ContentPlaceHolders are not required in every case, but encapsulate certain pieces of SharePoint functionality that may be useful in your environment. If you do not use this functionality, you may totally remove them from your Master Page, or just include the elements you need.

```
<asp:ContentPlaceHolder id="PlaceHolderFormDigest" runat="server">
<asp:ContentPlaceHolder id="PlaceHolderGlobalNavigation" runat="server">
<asp:ContentPlaceHolder id="PlaceHolderHorizontalNav" runat="server">
<asp:ContentPlaceHolder id="PlaceHolderLeftNavBarBorder" runat="server">
<asp:ContentPlaceHolder id="PlaceHolderLeftNavBarDataSource"
  runat="server" />
<asp:ContentPlaceHolder id="PlaceHolderLeftNavBarTop" runat="server"/>
<asp:ContentPlaceHolder id="PlaceHolderTopNavBar" runat="server">
<asp:ContentPlaceHolder id="PlaceHolderUtilityContent" runat="server"/>
<asp:ContentPlaceHolder ID="SPNavigation" runat="server">
<asp:ContentPlaceHolder ID="WSSDesignConsole" runat="server">
<asp:ContentPlaceHolder id="PlaceHolderPageTitle" runat="server"/>
</asp:Panel>
```

Thinking outside the Box

If you don't want to modify the Master Pages on your own, there are a number of alternative SharePoint Master Pages available.

- ❑ Microsoft has made a variety of Custom Master Pages for Windows SharePoint Services, along with accompanying CSS style sheets, available for you to download. Although these were designed primarily to accompany the Fab 40 custom application templates, they are applicable to any WSS-based site. An interesting characteristic of these Masters is that, when applied, they appear to be mild variations on the WSS default.master, but in actuality they completely reformulate the layout to use <DIV> tags and CSS instead of tables.

- ❑ Several people have published their own interpretations of minimal Master Pages. Most of these are designed for MOSS, though they can be adapted to WSS. Those posted by SharePoint MVP Heather Solomon are particularly popular, and can be accessed at the following site:

<http://www.heathersolomon.com/blog/articles/BaseMasterPages.aspx>

The Very Least You Can Do

The following listing is a Master Page called `wssminimum.master`. It provides a bare shell around `PlaceHolderMain`, which is the primary content placeholder in SharePoint. It does not include the default style sheet, site navigation, utility JavaScript, or any kind of “chrome” around the content area. Nor does it provide any way to log in if you have enabled anonymous access. However, if you manually navigate to any standard SharePoint Content Page, this Master Page will render it without an error.

The first group of lines are directives to identify this page as a Master Page, and to register the various assemblies and namespaces required to implement SharePoint functionality.

```
<%@Master language="C#" %>
<%@ Register Tagprefix="SharePoint"
    Namespace="Microsoft.SharePoint.WebControls"
    Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
    PublicKeyToken=71e9bce111e9429c" %>
<%@ Register Tagprefix="Utilities"
    Namespace="Microsoft.SharePoint.Utilities"
    Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
    PublicKeyToken=71e9bce111e9429c" %>
<%@ Import Namespace="Microsoft.SharePoint" %>
<%@ Import Namespace="Microsoft.SharePoint.ApplicationPages" %>
<%@ Register Tagprefix="WebPartPages"
    Namespace="Microsoft.SharePoint.WebPartPages"
    Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
    PublicKeyToken=71e9bce111e9429c" %>
```

Next comes the HTML itself. The HTML head section includes a mandatory tag to prevent the SharePoint search engine from crawling the code on SharePoint Content Pages. The page body includes a Web Part management component, and the placeholder for the actual page content. Finally, the panel control described earlier is included, and the HTML is closed.

Part II: Customizing the SharePoint Look and Feel

```
<html dir="ltr">
  <head runat="server">
    <SharePoint:RobotsMetaTag runat="server"/>
  </head>
<BODY onload="javascript:if (typeof(_spBodyOnLoadWrapper) != 'undefined')
  _spBodyOnLoadWrapper();">
  <WebPartPages:SPWebPartManager runat="server"/>
  <form runat="server" onsubmit="return _spFormOnSubmitWrapper();">
    <asp:ContentPlaceHolder id="PlaceholderMain" runat="server" />
    <asp:Panel visible="false" runat="server">
    <asp:ContentPlaceHolder id="PlaceholderAdditionalPageHead" runat="server" />
    <asp:ContentPlaceHolder id="PlaceholderPageTitle" runat="server" />
    <asp:ContentPlaceHolder id="PlaceholderSearchArea" runat="server"/>
    <asp:ContentPlaceHolder id="PlaceholderTitleBreadcrumb" runat="server"/>
    <asp:ContentPlaceHolder id="PlaceholderPageTitleInTitleArea"
      runat="server"/>
    <asp:ContentPlaceHolder id="PlaceholderLeftNavBar" runat="server"/>
    <asp:ContentPlaceHolder ID="PlaceholderPageImage" runat="server"/>
    <asp:ContentPlaceHolder ID="PlaceholderBodyLeftBorder" runat="server"/>
    <asp:ContentPlaceHolder ID="PlaceholderNavSpacer" runat="server"/>
    <asp:ContentPlaceHolder ID="PlaceholderTitleLeftBorder" runat="server"/>
    <asp:ContentPlaceHolder ID="PlaceholderTitleAreaSeparator" runat="server"/>
    <asp:ContentPlaceHolder ID="PlaceholderMiniConsole" runat="server"/>
    <asp:ContentPlaceHolder id="PlaceholderCalendarNavigator" runat="server" />
    <asp:ContentPlaceHolder id="PlaceholderLeftActions" runat="server"/>
    <asp:ContentPlaceHolder id="PlaceholderPageDescription" runat="server"/>
    <asp:ContentPlaceHolder id="PlaceholderBodyAreaClass" runat="server"/>
    <asp:ContentPlaceHolder id="PlaceholderTitleAreaClass" runat="server"/>
    <asp:ContentPlaceHolder id="PlaceholderBodyRightMargin" runat="server" />
    <asp:ContentPlaceHolder ID="PlaceholderTitleRightMargin" runat="server" />
  </asp:Panel>
</form>
</body>
</html>
```

Figure 5-11 shows the home page with `wssminimum.master` applied. This provides a clean slate for you to build virtually your entire user interface from scratch.

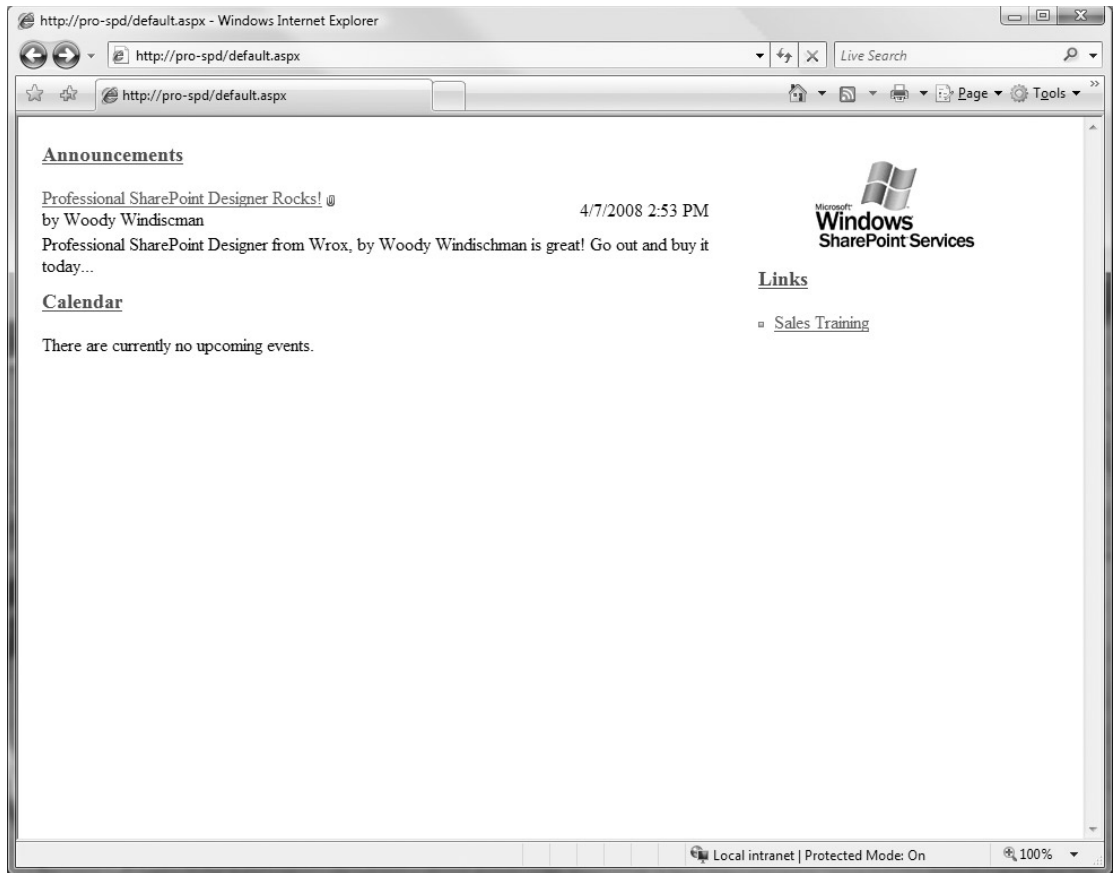


Figure 5-11

The downside of such a minimal Master Page is that you *must* build your entire user interface from scratch. The SharePoint content placeholders and web controls described later provide a rich toolbox that can help you provide a consistent user experience.

MOSS Publishing Pages require several additional directives in their Master Pages, which should not be used unless MOSS is installed. These are detailed in chapter 8.

SharePoint: Functional Stuff

In addition to, and often within, the content regions are a number of tags that invoke or include particular pieces of SharePoint functionality on a page. These tags are generally in the form `<SharePoint:somefunction>`, and invoke SharePoint Web Controls (the SharePoint tag prefix is defined in one of the directives at the top of the page). You have already seen the two simple SharePoint: controls that load the style sheets used on a typical SharePoint page.

Unlike the content region placeholders, which either stand alone or wrap default content, `<SharePoint: tags` are not always simple. Because they are instantiating real controls, they will often

Part II: Customizing the SharePoint Look and Feel

include an array of parameters appropriate to the function being invoked. The default.master includes 38 SharePoint Web Controls.

A Not-So-Minimal Master, from Head to Toe

There is a lot of ground between the wssminimum.master described above and the default.master provided with SharePoint. This section will walk you through a page called purefunction.master. This Master Page contains all of the functionality provided in default.master, but, for ease of reading, strips out all of the tables used to create the layout.

Some of the removed table components have classes and names that are styled by core.css, and contribute styles to the retained elements. This can cause the rendering of the child elements to be suboptimal.

Naturally, unless you want your site to look like Figure 5-12, you will never use this Master Page as is. Rather, you will either use the actual default.master, or wrap your own layout markup around the components described later. Any further screenshots in this section will assume this to be the case, and show the components in an appropriate context.

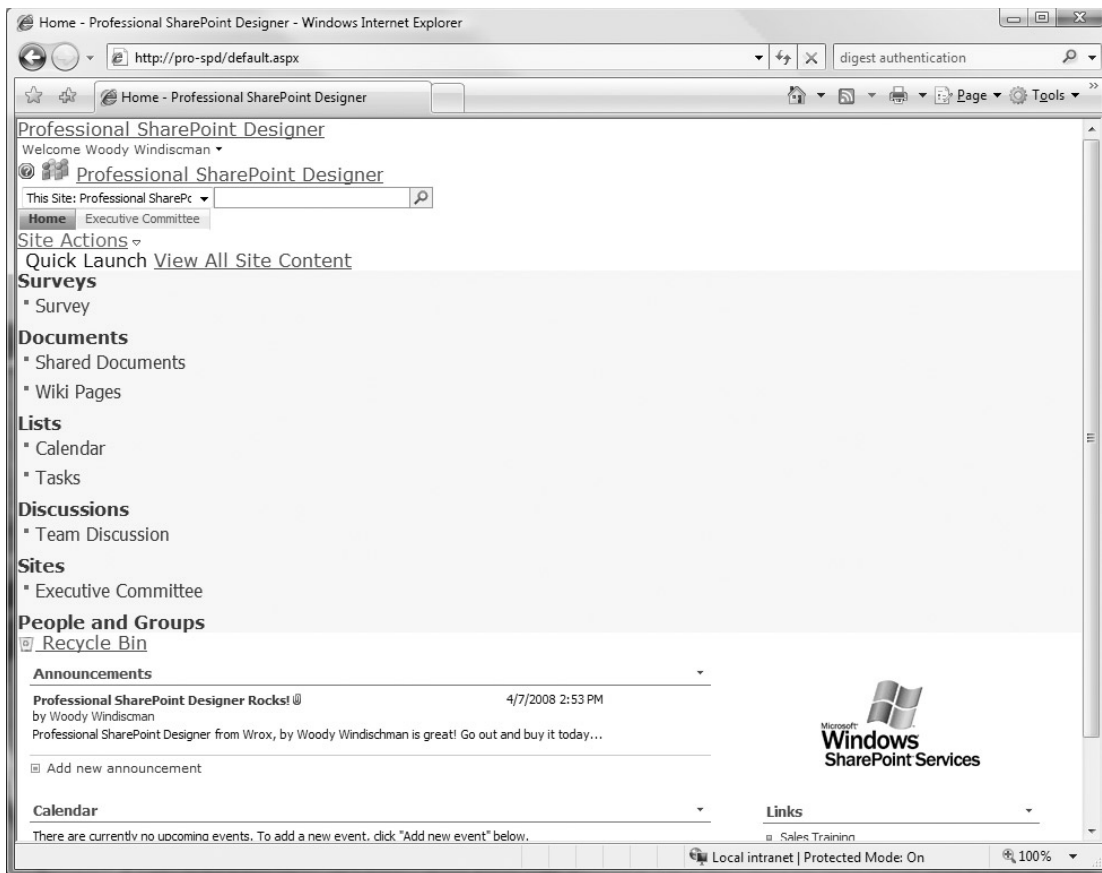


Figure 5-12

Unlike the empty placeholders in the minimal Master Page, the placeholders and web control definitions in this Master Page often span many lines. You may want to save any of these components you find useful as code snippets in SharePoint Designer.

Just as a Content Page could use a token to select a shared Master Page, the Master Page itself uses tokens to access certain controls. As you look through the code, note the presence of the tilde (~) in paths. This indicates that the control being referenced may be substituted dynamically based upon the particular site and configuration options.

Sight Unseen

Many of the lines at the top of this Master Page file are the same as those you saw in the minimal Master Page. These are two sections of content that are not actually seen by your user:

- ❑ The declarations and directives, which appear before any “normal” HTML.
- ❑ The <head> section of the HTML.

The new directives are `TagPrefix` registrations for two `wssuc` components — the `Welcome.ascx` token and the `DesignModeConsole` token. These tokens provide references to controls that exist in both MOSS and WSS, but are implemented differently. This tokenization allows the same Master Page to be used in both environments, and rendered appropriately in context.

An element that is used throughout the Master Page, but which makes its first appearance in this code block, is the *resource reference*. For example, in the <HTML tag, you see the segment `<%=Resources:wss,multipages_direction_dir_value%>`. The resources are stored in RESX files on the web server’s file system. A `.resx` file is an XML listing of resource name/value pairs. If you have multiple language resources installed, a resource reference will pull the name’s associated value from the file appropriate to the site’s language. This allows you to avoid hard-coding common strings into your Master Page. The RESX files are generally kept in the Web Root folder for the web application.

In the head section are several SharePoint Web controls. The `CssLink`, `Theme`, and `RobotsMetaTag` controls you have already seen. There are also two script-loading controls. Just as the style controls load default and custom styles respectively, the first script link control loads the primary script file for SharePoint (`core.js`), and the second is there to allow the loading of any custom scripts that need to be common to every page.

The `ContentPlaceHolders` in this section implement the page title as seen in the browser’s title bar, and allow for the addition of custom page headers if needed.

```
<%@Master language="C#" %>
<%@ Register Tagprefix="SharePoint"
    Namespace="Microsoft.SharePoint.WebControls"
    Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
    PublicKeyToken=71e9bce111e9429c" %>
<%@ Register Tagprefix="Utilities"
    Namespace="Microsoft.SharePoint.Utilities"
    Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
    PublicKeyToken=71e9bce111e9429c" %>
<%@ Import Namespace="Microsoft.SharePoint" %>
```

Part II: Customizing the SharePoint Look and Feel

```
<%@ Import Namespace="Microsoft.SharePoint.ApplicationPages" %>
<%@ Register Tagprefix="WebPartPages"
  Namespace="Microsoft.SharePoint.WebPartPages"
  Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
  PublicKeyToken=71e9bce111e9429c" %>

<%@ Register TagPrefix="wssuc" TagName="Welcome"
  src="~/_controltemplates/Welcome.ascx" %>
<%@ Register TagPrefix="wssuc" TagName="DesignModeConsole"
  src="~/_controltemplates/DesignModeConsole.ascx" %>
<HTML dir="<%=Resources:wss,multipages_direction_dir_value%"
  runat="server" xmlns:o="urn:schemas-microsoft-com:office:office"
  __expr-val-dir="ltr">

<HEAD runat="server">

<META Name="GENERATOR" Content="Microsoft SharePoint">
<META Name="progid" Content="SharePoint.WebPartPage.Document">
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=utf-8">
<META HTTP-EQUIV="Expires" content="0">

<SharePoint:RobotsMetaTag runat="server" />

<Title ID=onetidTitle><asp:ContentPlaceHolder id=PlaceholderPageTitle
  runat="server" /></Title>

<SharePoint:CssLink runat="server" />
<SharePoint:Theme runat="server" />

<SharePoint:ScriptLink language="javascript" name="core.js"
  Defer="true" runat="server" />
<SharePoint:CustomJSUrl runat="server" />
<SharePoint:SoapDiscoveryLink runat="server" />
<asp:ContentPlaceHolder id="PlaceholderAdditionalPageHead" runat="server" />
<SharePoint:DelegateControl runat="server"
  ControlId="AdditionalPageHead" AllowMultipleControls="true" />
</HEAD>
```

The Global Navigation Bar

This code block includes the start of the body, which begins the .ASPX form and instantiates the `SPWebPartManager` control. This is followed by the `PlaceholderGlobalNavigation` placeholder. This placeholder is filled with default content, including:

- Some hidden links to control accessibility.
- The Global breadcrumb.
- The Welcome menu.
- The SharePoint Help link.

As shown in Figure 5-13, the accessibility controls are normally not visible on the page.



Figure 5-13

```
<BODY scroll="yes" onload="javascript:if
  (typeof(_spBodyOnLoadWrapper) != 'undefined') _spBodyOnLoadWrapper();">
<form runat="server" onsubmit="return _spFormOnSubmitWrapper();">
<WebPartPages:SPWebPartManager id="m" runat="Server"/>
<asp:ContentPlaceHolder id="PlaceHolderGlobalNavigation" runat="server">
<span id="TurnOnAccessibility" style="display:none">
  <a href="#" class="ms-skip"
    onclick="SetIsAccessibilityFeatureEnabled(true);UpdateAccessibilityUI();
    return false;">
  <SharePoint:EncodedLiteral runat="server"
    text="<%%$Resources:wss,master_turnonaccessibility%"
    EncodeMethod="HtmlEncode"/></a>
</span>
<A href="javascript:;" onclick="javascript:this.href='#mainContent';"
  class="ms-skip" AccessKey="<%%$Resources:wss,maincontent_accesskey%"
  runat="server">
<SharePoint:EncodedLiteral runat="server"
  text="<%%$Resources:wss,mainContentLink%" EncodeMethod="HtmlEncode"/></A>
<span id="TurnOffAccessibility" style="display:none">
  <a href="#" class="ms-acclink"
    onclick="SetIsAccessibilityFeatureEnabled(false);
    UpdateAccessibilityUI();return false;">
  <SharePoint:EncodedLiteral runat="server"
    text="<%%$Resources:wss,master_turnoffaccessibility%"
    EncodeMethod="HtmlEncode"/></a>
</span>
```

The left breadcrumb is implemented as the default content of the `PlaceHolderGlobalNavigationSiteMap` content placeholder. This is a standard ASP.NET breadcrumb control, connected to SharePoint's Site Map data source. It will show either one or two levels, depending on whether or not a portal site connection is configured in the site collection administration page. If a portal site is configured, the leftmost link will be to that site, and the right link will be to the root of the current site collection. If no portal connection is specified, only the site collection root link is shown.

Notice the CSS class designator, `NodeStyle-CssClass="ms-sitemapdirectional"`. The class being assigned, like all of the classes you will see throughout this listing, is defined in `core.css`. Some SharePoint controls have hard-coded style references, while others allow you to specify their style in the Master Page. Generally, you want to retain the `core.css` class reference, and use your own Theme or CSS file to override the default styling, but this is not required.

There are four delegate controls on the right side of the global navigation bar, only one of which is normally visible in WSS — the Welcome control mentioned above.

Finally, there is a link to the help system. This is built up with JavaScript and several resource strings, as it needs to be available in many languages, and opens into its own customized window.

```
<asp:ContentPlaceHolder id="PlaceholderGlobalNavigationSiteMap"
  runat="server">
  <asp:SiteMapPath SiteMapProvider="SPSiteMapProvider"
    id="GlobalNavigationSiteMap" RenderCurrentNodeAsLink="true"
    SkipLinkText="" NodeStyle-CssClass="ms-sitemapdirectional" runat="server"/>
</asp:ContentPlaceHolder>
<SharePoint:DelegateControl runat="server" ControlId="GlobalSiteLink0"/>
<wssuc:Welcome id="IdWelcome" runat="server" EnableViewState="false">
</wssuc:Welcome>
<SharePoint:DelegateControl ControlId="GlobalSiteLink1" Scope="Farm"
  runat="server"/>
<SharePoint:DelegateControl ControlId="GlobalSiteLink2" Scope="Farm"
  runat="server"/>
<a href="javascript:TopHelpButtonClick('NavBarHelpHome') "
  AccessKey="<%=Resources:wss,multipages_helplink_accesskey%>"
  id="TopHelpLink" title="<%=Resources:wss,multipages_helplinkalt_text%>"
  runat="server">" runat="server"></a>
</asp:ContentPlaceHolder>
```

Site (Global) Title Area

The site title area, Figure 5-14, contains three primary elements:

- ❑ The `SiteLogoImage` web control, which defaults to the pawn icon, will instead render the image defined on the Site Settings Title, Description, and Icon page if it is set.
- ❑ The `PlaceholderSiteName` content placeholder, whose default content consists of the Site Title itself, linked to the home page of the site. The title is set at creation time, and can be edited through the same settings page as the `SiteLogoImage`.
- ❑ The `PlaceholderSearchArea` content placeholder. This defaults to the `SmallSearchInputBox` delegated web control, which is configured based upon the version of SharePoint and the site administrator's option choices.



Figure 5-14

```
<SharePoint:SiteLogoImage id="onetidHeadbnnr0"
  LogoImageUrl="/_layouts/images/titlegraphic.gif" runat="server"/></td>
<asp:ContentPlaceHolder id="PlaceholderSiteName" runat="server">
  <SharePoint:SPLinkButton runat="server" NavigateUrl="~site/"
    id="onetidProjectPropertyTitle">
  <SharePoint:ProjectProperty Property="Title" runat="server" />
  </SharePoint:SPLinkButton>
</asp:ContentPlaceHolder>
<asp:ContentPlaceHolder id="PlaceholderSearchArea" runat="server">
  <SharePoint:DelegateControl runat="server"
    ControlId="SmallSearchInputBox"/>
</asp:ContentPlaceHolder>
```

Top (Tab) Navigation Area

The top navigation area is defined by the `PlaceHolderTopNavBar` content placeholder, and contains two menus. Although (by default) they share a content placeholder and a tablike inactive state, these menus are very different from one another, both operationally and in how they are generated.

The TopNavigationMenu (Tab Bar)

The top tab menu is based on two controls — the `SharePoint:AspMenu` and a delegate data source control. You can see from the many parameters that you have a great deal of control over how this menu is presented. Again, many of the styles used are drawn from the `core.css` and applied Theme or custom .CSS files.

This is one of the two standard menus your site administrator can control the elements in. The delegate data source control is populated differently in MOSS Publishing sites and WSS collaboration sites, in that MOSS allows a full multilevel hierarchy for navigation elements, whereas WSS is limited to a single level for the top menu.

Notice the `StaticDisplayLevels` and `MaximumDynamicDisplayLevels` parameters in the `AspMenu` control. You can determine how many levels are presented directly versus as dropdowns. In fact, the control offers full flexibility to provide vertical or horizontal menus, as may be appropriate in your design. By leveraging this control (and its Quick Launch counterpart described later), you can both control the look and feel, and keep administrative overhead to a minimum.

You can leverage SharePoint's `AspMenu` control with other data sources as well, although without the built-in Administrative UI.

```
<asp:ContentPlaceHolder id="PlaceHolderTopNavBar" runat="server">
  <asp:ContentPlaceHolder id="PlaceHolderHorizontalNav" runat="server">
    <SharePoint:AspMenu
      ID="TopNavigationMenu"
      Runat="server"
      DataSourceID="topSiteMap"
      EnableViewState="false"
      AccessKey="<%=Resources.wss_navigation_accesskey%>"
      Orientation="Horizontal"
      StaticDisplayLevels="2"
      MaximumDynamicDisplayLevels="1"
      DynamicHorizontalOffset="0"
      StaticPopoutImageUrl="/_layouts/images/menudark.gif"
      StaticPopoutImageTextFormatString=" "
      DynamicHoverStyle-BackColor="#CBE3F0"
      SkipLinkText=" "
      StaticSubMenuIndent="0"
      CssClass="ms-topNavContainer">
    <StaticMenuStyle/>
    <StaticMenuItemStyle CssClass="ms-topnav" ItemSpacing="0px"/>
    <StaticSelectedStyle CssClass="ms-topnavselected" />
    <StaticHoverStyle CssClass="ms-topNavHover" />
    <DynamicMenuStyle BackColor="#F2F3F4" BorderColor="#A7B4CE"
      BorderWidth="1px"/>
  </asp:ContentPlaceHolder>
</asp:ContentPlaceHolder>
```

```
<DynamicMenuItemStyle CssClass="ms-topNavFlyOuts" />
<DynamicHoverStyle CssClass="ms-topNavFlyOutsHover" />
<DynamicSelectedStyle CssClass="ms-topNavFlyOutsSelected" />
</SharePoint:AspMenu>
<SharePoint:DelegateControl runat="server"
  ControlId="TopNavigationDataSource">
  <Template_Controls>
    <asp:SiteMapDataSource
      ShowStartingNode="False"
      SiteMapProvider="SPNavigationProvider"
      id="topSiteMap"
      runat="server"
      StartingNodeUrl="sid:1002" />
  </Template_Controls>
</SharePoint:DelegateControl>
</asp:ContentPlaceHolder>
```

The Site Actions Menu

The other menu in the top menu bar is the Site Actions dropdown, shown in Figure 5-15. There are several things to notice about this menu:

- ❑ Site Actions is not enclosed in its own placeholder (though it is within the `PlaceholderTopNavBar` placeholder along with the horizontal tab bar).
- ❑ The items in the menu are defined explicitly, meaning you can easily add your own arbitrary items to the menu.
- ❑ The menu uses a verbose display format, including icons and descriptions.
- ❑ There is a `PermissionsString` parameter, allowing you to trim the menu to users with particular rights on the site.

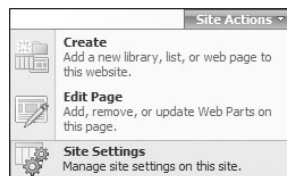


Figure 5-15

The order in which the elements are rendered is not based upon their order in the code. Rather the `MenuGroupId` and `Sequence` parameters are used. Items within the same menu group are ordered by the `Sequence`. The groups are ordered by their `MenuGroupId` and separated by a bar. Both orderings are smallest to largest.

```
<SharePoint:SiteActions runat="server"
  AccessKey="<%=Resources:wss,tb_SiteActions_AK%>" id="SiteActionsMenuMain"
  PrefixHtml="&lt;div&gt;&lt;div&gt;"
  SuffixHtml="&lt;/div&gt;&lt;/div&gt;"
  MenuNotVisibleHtml="&nbsp;"
  <CustomTemplate>
    <SharePoint:FeatureMenuTemplate runat="server"
      FeatureScope="Site"
      Location="Microsoft.SharePoint.StandardMenu"
      GroupId="SiteActions"
      UseShortId="true"
    >
    <SharePoint:MenuItemTemplate runat="server" id="MenuItem_Create"
      Text="<%=Resources:wss,viewlsts_pagetitle_create%>"
      Description="<%=Resources:wss,siteactions_createdescription%>"
      ImageUrl="/_layouts/images/Actionscreate.gif"
      MenuGroupId="100"
      Sequence="100"
      UseShortId="true"
      ClientOnClickNavigateUrl="~site/_layouts/create.aspx"
      PermissionsString="ManageLists, ManageSubwebs"
      PermissionMode="Any" />
    <SharePoint:MenuItemTemplate runat="server" id="MenuItem_EditPage"
      Text="<%=Resources:wss,siteactions_editpage%>"
      Description="<%=Resources:wss,siteactions_editpagedescription%>"
      ImageUrl="/_layouts/images/ActionsEditPage.gif"
      MenuGroupId="100"
      Sequence="200"
      ClientOnClickNavigateUrl="javascript:MSOLayout_ChangeLayoutMode(false);"
    />
    <SharePoint:MenuItemTemplate runat="server" id="MenuItem_Settings"
      Text="<%=Resources:wss,settings_pagetitle%>"
      Description="<%=Resources:wss,siteactions_sitesettingsdescription%>"
      ImageUrl="/_layouts/images/ActionsSettings.gif"
      MenuGroupId="100"
      Sequence="300"
      UseShortId="true"
      ClientOnClickNavigateUrl="~site/_layouts/settings.aspx"
      PermissionsString="EnumeratePermissions, ManageWeb, ManageSubwebs,
        AddAndCustomizePages, ApplyThemeAndBorder, ManageAlerts, ManageLists,
        ViewUsageData"
      PermissionMode="Any" />
  </SharePoint:FeatureMenuTemplate>
</CustomTemplate>
</SharePoint:SiteActions>
</asp:ContentPlaceholder>
```

Page Edit Bar

The Page Edit Bar (DesignModeConsole and PublishingConsole) is visible in the browser when you are editing the content area of a page. On WSS pages (Figure 5-16) the bar is very simple — little more than a status line and close box. As shown in Figure 5-17, however, MOSS Publishing pages have a variety of tools for content management in their edit bar.

Part II: Customizing the SharePoint Look and Feel

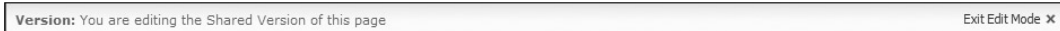


Figure 5-16

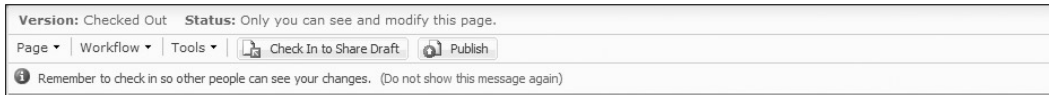


Figure 5-17

The WSSDesignConsole is still present, but it is overlaid by the PublishingConsole. Removing the PublishingConsole delegate control allows the simple WSS bar to show through.

```
<asp:ContentPlaceholder ID="WSSDesignConsole" runat="server">
  <wssuc:DesignModeConsole id="IdDesignModeConsole" runat="server" />
</asp:ContentPlaceholder>
<asp:ContentPlaceholder ID="SPNavigation" runat="server">
  <SharePoint:DelegateControl runat="server" ControlId="PublishingConsole"
    PrefixHtml="&lt;tr&gt;&lt;td colspan=&quot;4&quot;
      id=&quot;mpdmconsole&quot; class=&quot;ms-consolemptablerow&quot;&gt;&gt;"
    SuffixHtml="&lt;/td&gt;&lt;/tr&gt;">
  </SharePoint:DelegateControl>
</asp:ContentPlaceholder>
```

Page Title Area

The title area spans the width of the page, and contains elements for both content and layout. The four functional components, as shown in Figure 5-18, are:

- The Page Image.
- The primary site breadcrumb.
- The PageTitleInTitleArea version of the Page Title.
- The mini console.

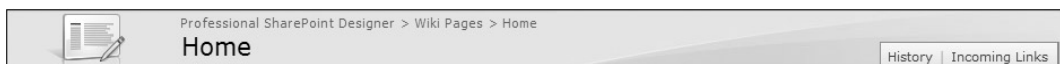


Figure 5-18

The layout elements are used to help keep the title area consistent in style and alignment with the left navigation and page body sections below it. In particular, the Page Image aligns with the left navigation. The breadcrumb and title align with the body area. The mini console is rendered at run time in a relatively positioned `<div>`, and normally aligned to the upper-right corner of the main body element below it.

```
<asp:ContentPlaceHolder id="PlaceholderPageImage" runat="server" />
<asp:ContentPlaceHolder id="PlaceholderTitleLeftBorder" runat="server">
</asp:ContentPlaceHolder>
<asp:ContentPlaceHolder id="PlaceholderTitleBreadcrumb" runat="server">
<asp:SiteMapPath SiteMapProvider="SPContentMapProvider" id="ContentMap"
  SkipLinkText="" NodeStyle-CssClass="ms-sitemapdirectional"
  runat="server" /> &nbsp;
</asp:ContentPlaceHolder>
<asp:ContentPlaceHolder id="PlaceholderPageTitleInTitleArea"
  runat="server" />
<asp:ContentPlaceHolder id="PlaceholderMiniConsole" runat="server" />
<asp:ContentPlaceHolder id="PlaceholderTitleRightMargin" runat="server">
</asp:ContentPlaceHolder>
<asp:ContentPlaceHolder id="PlaceholderTitleAreaSeparator" runat="server" />
```

Left Navigation Bar

The left navigation bar contains placeholders and controls for a variety of navigation elements. Not all of these elements are available at all times.

Navigation Data Source and Calendar Control

The first control is not a visible element. It is a placeholder for an alternate data source for the left Quick Launch navigation bar.

The second control is the placeholder for the calendar navigator, shown in Figure 5-19. This date picker is only injected into this placeholder on Calendar view pages. If the parent calendar is in month view, it displays the 12 months (as shown), otherwise it shows the days in the current month for day and week views.

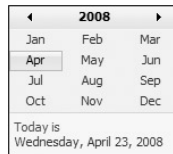


Figure 5-19

```
<asp:ContentPlaceHolder id="PlaceholderLeftNavBarDataSource"
  runat="server" />
<asp:ContentPlaceHolder id="PlaceholderCalendarNavigator" runat="server" />
```

Quick Launch

The Quick Launch bar is assembled from several components:

- The Quick Launch title label (hidden in default.master).
- The View All Site Content link.
- The Quick Launch menu itself.
- The Quick Launch data source delegate control.

Part II: Customizing the SharePoint Look and Feel

Notice the `SPSecurityTrimmedControl` wrapper around the View All Site Content link. This wrapper class can be useful any time you have content you wish to restrict to users with a particular right. In this case, the link is only displayed to users who have the `ViewFormPages` right.

Like the `TopNavigationMenu`, the Quick Launch is an `ASPMenu` fed by a data source maintained by SharePoint, and customizable by the site owner. Unlike the top menu, this data source is hierarchical in WSS as well as MOSS — for example, the data source has multiple levels. In the case of WSS, this is two levels (listed as link and heading), but MOSS is more flexible in defining site hierarchies.

You can take advantage of this by changing the values for `StaticDisplayLevels` and `MaximumDynamicDisplayLevels` from their defaults of 2 and 0 to create fly-out menus such as that shown in Figure 5-20.



Figure 5-20

Just as in the top menu, you have full flexibility both in setting the style classes used by the menu elements and, of course, in setting the values of those classes.

```
<asp:ContentPlaceHolder id="PlaceHolderLeftNavBarTop" runat="server" />
<asp:ContentPlaceHolder id="PlaceHolderLeftNavBar" runat="server">
  <SharePoint:EncodedLiteral runat="server"
    text="<%=Resources:wss,quicklnch_pagetitle%>" EncodeMethod="HtmlEncode" />
  <Sharepoint:SPSecurityTrimmedControl runat="server"
    PermissionsString="ViewFormPages">
    <SharePoint:SPLinkButton id="idNavLinkViewAll" runat="server"
      NavigateUrl="~/site/_layouts/viewlsts.aspx"
      Text="<%=Resources:wss,quicklnch_allcontent%>"
      AccessKey="<%=Resources:wss,quicklnch_allcontent_AK%>" />
  </SharePoint:SPSecurityTrimmedControl>
  <Sharepoint:SPNavigationManager
    id="QuickLaunchNavigationManager"
    runat="server"
    QuickLaunchControlId="QuickLaunchMenu"
    ContainedControl="QuickLaunch"
    EnableViewState="false">
  <div>
    <SharePoint:DelegateControl runat="server"
      ControlId="QuickLaunchDataSource">
      <Template_Controls>
        <asp:SiteMapDataSource
```

```
SiteMapProvider="SPNavigationProvider"
ShowStartingNode="False"
id="QuickLaunchSiteMap"
StartingNodeUrl="sid:1025"
runat="server"
/>
</Template_Controls>
</SharePoint:DelegateControl>
<SharePoint:AspMenu
id="QuickLaunchMenu"
DataSourceId="QuickLaunchSiteMap"
runat="server"
Orientation="Vertical"
StaticDisplayLevels="2"
ItemWrap="true"
MaximumDynamicDisplayLevels="0"
StaticSubMenuIndent="0"
SkipLinkText=""
>
<LevelMenuItemStyles>
<asp:MenuItemStyle CssClass="ms-navheader" />
<asp:MenuItemStyle CssClass="ms-navitem" />
</LevelMenuItemStyles>
<LevelSubMenuStyles>
<asp:SubMenuStyle CssClass="ms-navSubMenu1" />
<asp:SubMenuStyle CssClass="ms-navSubMenu2" />
</LevelSubMenuStyles>
<LevelSelectedStyles>
<asp:MenuItemStyle CssClass="ms-selectednavheader" />
<asp:MenuItemStyle CssClass="ms-selectednav" />
</LevelSelectedStyles>
</SharePoint:AspMenu>
</div>
</Sharepoint:SPNavigationManager>
```

Site Hierarchy (Tree View)

In addition to (or instead of) the traditional Quick Launch view of a site, site administrators can select a full tree view. This view allows the user to browse and navigate to any site, list, or library (or even folders within them) in the site collection, as shown in Figure 5-21.

Part II: Customizing the SharePoint Look and Feel

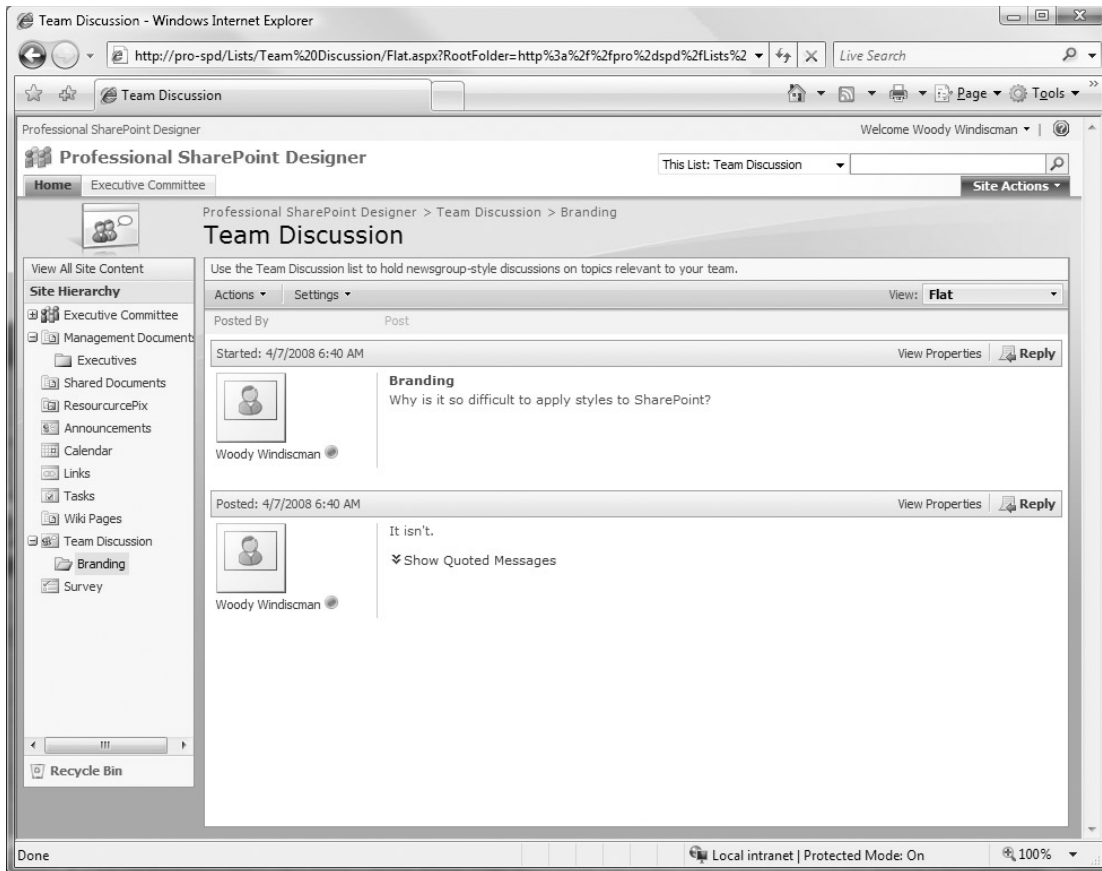


Figure 5-21

As with all of the SharePoint navigation controls, you can define many aspects of the control's look through its parameters, from the styles used for the text to the specific images used for the navigation icons.

```
<Sharepoint:SPNavigationManager
  id="TreeViewNavigationManager"
  runat="server"
  ContainedControl="TreeView">
<SharePoint:SPLinkButton runat="server"
  NavigateUrl=~site/_layouts/viewlsts.aspx" id="idNavLinkSiteHierarchy"
  Text="<%%$Resources:wss,treeview_header%"
  AccessKey="<%%$Resources:wss,quiklnch_allcontent_AK%" />
<SharePoint:SPHierarchyDataSourceControl
  runat="server"
  id="TreeViewDataSource"
  RootContextObject="Web"
  IncludeDiscussionFolders="true"
 />
```

```
<SharePoint:SPRememberScroll runat="server" id="TreeViewRememberScroll"
  onscroll="javascript:_spRecordScrollPositions(this);" Style="overflow:
  auto;height: 400px;width: 150px; ">
<Sharepoint:SPTreeView
  id="WebTreeView"
  runat="server"
  ShowLines="false"
  DataSourceId="TreeViewDataSource"
  ExpandDepth="0"
  SelectedNodeStyle-CssClass="ms-tvselected"
  NodeStyle-CssClass="ms-navitem"
  NodeStyle-HorizontalPadding="2"
  SkipLinkText=""
  NodeIndent="12"
  ExpandImageUrl="/_layouts/images/tvplus.gif"
  CollapseImageUrl="/_layouts/images/tvminus.gif"
  NoExpandImageUrl="/_layouts/images/tvblank.gif"
  >
</Sharepoint:SPTreeView>
</Sharepoint:SPRememberScroll>
</Sharepoint:SPNavigationManager>
```

Recycle Bin and Left Actions

Access to the Recycle bin is implemented via a `SPLinkButton` element. This implementation is slightly different from the View All Site Content link above, and leverages more of the `SPLinkButton`'s native functionality. Notice that it allows an image to be associated, as well as providing for its own security trimming, without the `SPSecurityTrimmedControl` wrapper.

The `PlaceHolderLeftActions` placeholder is used primarily by Wiki pages to display the recent items menu.

```
<SharePoint:SPLinkButton runat="server"
  NavigateUrl="~site/_layouts/recyclebin.aspx" id="idNavLinkRecycleBin"
  ImageUrl="/_layouts/images/recyclebin.gif"
  Text="<%=Resources:wss,StsDefault_RecycleBin%>"
  PermissionsString="DeleteListItems" />
</asp:ContentPlaceHolder>
<asp:ContentPlaceHolder id="PlaceHolderLeftActions"
  runat="server"></asp:ContentPlaceHolder>
<asp:ContentPlaceHolder id="PlaceHolderNavSpacer" runat="server">
  <IMG SRC="/_layouts/images/blank.gif" width=138 height=1 alt="">
</asp:ContentPlaceHolder>
<asp:ContentPlaceHolder id="PlaceHolderLeftNavBarBorder"
  runat="server"></asp:ContentPlaceHolder>
```

Main Body Area

There are only two functional placeholders in this block. The first is a placeholder for the page description, if any, for the current page. The other is the actual content of the page. `PlaceHolderMain` is required to display any SharePoint content, and is substituted as appropriate for each page.

Part II: Customizing the SharePoint Look and Feel

```
<asp:ContentPlaceHolder id="PlaceHolderBodyLeftBorder" runat="server">
<IMG SRC="/_layouts/images/blank.gif" width=10 height=1 alt="">
</asp:ContentPlaceHolder>
<PlaceHolder id="MSO_ContentDiv" runat="server">
<A name="mainContent"></A>
<asp:ContentPlaceHolder id="PlaceHolderPageDescription" runat="server"/>
<asp:ContentPlaceHolder id="PlaceHolderMain" runat="server">
</asp:ContentPlaceHolder>
</PlaceHolder>
<asp:ContentPlaceHolder id="PlaceHolderBodyRightMargin" runat="server">
</asp:ContentPlaceHolder>
```

More No-See-Ums

At the end of the page are several more placeholders for items that are not normally visible. The `FormDigest` is used in extranet and Internet scenarios to store encrypted ID information when you are using forms and digest authentication. Utility content is a place for extra things you may need to put at the bottom of a page, such as script, or a company footer/disclaimer. The two `Class` placeholders are for CSS that may need to guarantee it is loaded last in order to override any other style sheets used on the page.

```
<asp:ContentPlaceHolder id="PlaceHolderFormDigest" runat="server">
<SharePoint:FormDigest runat=server/>
</asp:ContentPlaceHolder>
<input type="text" name="__spDummyText1" style="display:none;" size=1/>
<input type="text" name="__spDummyText2" style="display:none;" size=1/>
</form>
<asp:ContentPlaceHolder id="PlaceHolderUtilityContent" runat="server"/>
<asp:ContentPlaceHolder id="PlaceHolderBodyAreaClass" runat="server"/>
<asp:ContentPlaceHolder id="PlaceHolderTitleAreaClass" runat="server"/>
</BODY>
</HTML>
```

Summary

This chapter explained how a typical SharePoint Page is constructed. You saw the power of Master Pages, and discovered many of the building blocks available to you to change the overall feel of your site. You also found out:

- ❑ SharePoint Pages are built up from elements retrieved from many locations.
- ❑ You are not stuck with the layout or structure provided out-of-the-box.
- ❑ SharePoint Designer cannot directly reach into the server's file system.
- ❑ Even the canned navigational elements provided by SharePoint allow for significant customization.

This chapter touched briefly on the importance of CSS styles in SharePoint. The next two chapters will introduce you to SharePoint Designer's powerful Style Sheet Editor, and how CSS is used by SharePoint's Theme mechanism, giving you full control over the look, to make it go along with the feel of your site.