Advanced SharePoint Services Solutions

SCOT P. HILLIER

Apress®

Advanced SharePoint Services Solutions

Copyright © 2005 by Scot P. Hillier

ISBN (pbk): 1-59059-456-8

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Jim Sumser

Technical Reviewer: Seth Bates

Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Chris Mills, Dominic Shakeshaft, Jim Sumser

Project Manager: Beth Christmas

Copy Edit Manager: Nicole LeClerc

Copy Editor: Liz Welch

Production Manager: Kari Brooks-Copony

Production Editor: Mary Keith Trawick

Compositor: Linda Weidemann, Wolf Creek Press

Proofreader: April Eddy

Indexer: Michael Brinkman

Artist: Kinetic Publishing Services, LLC

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013, and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders@springer-ny.com`, or visit `http://www.springer-ny.com`. Outside the United States: fax +49 6221 345229, e-mail `orders@springer.de`, or visit `http://www.springer.de`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Downloads section.

CHAPTER 4

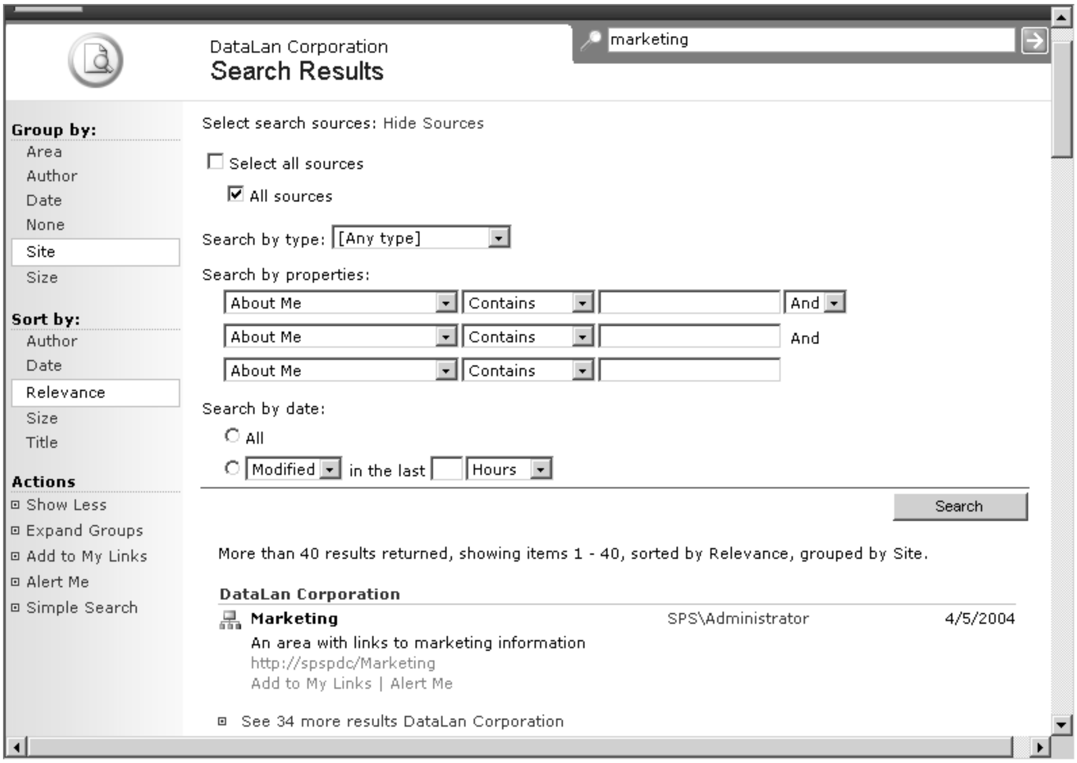
Advanced SharePoint Portal Server Solutions

While many of the techniques and projects I discuss in this book are applicable to both Windows SharePoint Services (WSS) and SharePoint Portal Server (SPS), this chapter is specifically geared toward developers who have deployed SPS as part of their solution and want to customize the portal. SPS provides a number of features that enhance the overall SharePoint infrastructure, including My Site, search, audiences, and profiles. Many of these features are candidates for customization and improvement. In this chapter, I will address the most common issues with SPS development and show you some solutions.

Customizing SharePoint Portal Server Search

If your organization is going to use a SharePoint installation seriously, then a full-strength search engine is critical. Although WSS sites offer a limited search capability, SPS provides a much more robust search mechanism that allows administrators to specify search scopes while giving end users advanced tools to build and execute queries. This enhanced search functionality is one of the primary reasons to deploy SPS alongside WSS sites.

SPS users can take advantage of two different search web parts to build and execute queries. The first web part executes simple keyword queries and the second web part allows an end user to create more complicated queries. When a query is executed using either interface, the results are displayed on the `Search.aspx` page. Once the results are displayed on the page, you can subsequently use associated command links to group and sort the results. Figure 4-1 shows a typical query result in SPS.



More than 40 results returned, showing items 1 - 40, sorted by Relevance, grouped by Site.

DataLan Corporation

Marketing SPS\Administrator 4/5/2004
An area with links to marketing information
http://spspdc/Marketing
Add to My Links | Alert Me

See 34 more results DataLan Corporation

Figure 4-1. *Query results displayed on Search.aspx*

While the standard query functionality and resulting display are fine for general purposes, there are many occasions when you may want to customize the way queries are executed or how the results are displayed. If you are creating a customer extranet, for example, you may not want users to be able to search for other users because this would essentially allow direct access to your customer base. In other cases, you may simply want to make the query more focused or display the results in a compact format.

In this section, I'll present several techniques for customizing search in SPS. These techniques will include ways to build a custom query interface as well as a custom result display. Using these techniques, you will be able to create an appropriate search interface to complement any SPS solution.

Sending Parameters to Search.aspx

Perhaps the simplest way to create a custom search interface is to post queries directly to the `Search.aspx` page. This technique involves encoding a query into a URL. When the query is posted to the `Search.aspx` page, the results of the query are displayed in the standard manner.

While the simplicity of this technique makes it attractive, it can theoretically be used to execute complicated queries as well. The problem, however, is that the documentation provided in the SDK is horrible. As of this writing, there is only a single page in the SDK dedicated to this technique. Therefore, we are left alone to figure out by trial and error what can be accomplished. In this section, I'll cover the most useful parameters and help you understand how to use them.

Using Search Keywords

You can specify keyword searches using the `k` parameter. When you use this parameter alone, the search will be run against all search scopes, and the result will contain all available types such as categories, sites, and documents. You can only specify a single value and wildcards cannot be used. This query will return the same results as if you had typed the keyword into the simple search box. The following code shows how to post the query:

```
http://[server]/Search.aspx?k=[keyword]
```

Specifying Search Scopes

If you would like to narrow the search, you can specify one or more search scopes using the `s` parameter. Search scopes are specified using the name of the scope as it is defined in the portal administration pages. You can also see the names of the available search scopes using the drop-down list of the simple search box, as shown in Figure 4-2.

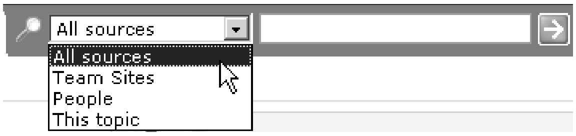


Figure 4-2. *Identifying search scopes*

In my installation, I have created several new search scopes. I have one named Team Sites that includes only WSS team sites. I have also defined one named People for searching only for portal users. Finally, I have one named File System that searches nonportal content. These search scopes make it much easier to return the correct information.

Follow these steps to create a new search scope:

1. Log into SPS as a member of the Administrator Site Group.
2. On the portal home page, click Site Settings.
3. On the Site Settings page, click Search Settings and Indexed Content ➤ Manage Search Scopes.
4. On the Manage Search Scopes page, click New Search Scope.
5. On the Add Search Scope page, name the new scope **Team Sites**.
6. Under the Topics and Areas section, click the option Include No Topic or Area in This Scope.
7. Under the Content Source Groups section, click the option Limit the Scope to the Following Groups of Content Sources.

8. Check the Sites in Site Directory box.
9. Click the OK button.

Once you have defined your search scopes, you can use them when posting queries to `Search.aspx`. The search you create can use one or more of the defined scopes. As an example, the following query searches for people associated with the Marketing department:

```
http://[server]/Search.aspx?s=People&k=Marketing
```

When you include multiple scopes in the query, the scope names must be separated by the special character set `%2c`. The names of the scopes must also be encoded, which usually means replacing the spaces with plus (`+`) signs. As an example, the following query searches for both people and sites that are associated with the Human Resources department:

```
http://[server]/Search.aspx?s=People%2cTeam+Sites&k=Human+Resources
```

Once you are posting queries using the `k` and `s` parameters, you have essentially duplicated the functionality of the standard simple search web part. If you look closely, in fact, you'll notice that the simple search web part will reflect the same scope and keyword that you specify in your query. This shows how tightly query posting is integrated with SPS.

Using Advanced Search Types

Just like you can duplicate the behavior of the simple search web part, you can also duplicate the behavior of the advanced search web part. Along with allowing the user to select one or more scopes, the advanced search capability also allows a user to select a search type. The search type specifies what kind of information should be returned from the search and is defined using the `tp` parameter. This parameter supports the following values: `any`, `announcement`, `category`, `contacts`, `document`, `discussions`, `event`, `listing`, `person`, `picture`, `site`, `stsitems`, and `tasks`.

When using a search type, you can specify one or more scopes as well as a keyword. When you run a query this way, you'll also notice that the advanced search web part will appear and reflect your settings just as the simple search did previously. As an example, the following query searches all team sites for documents that have the keyword `Sales`:

```
http://[server]/Search.aspx?s= Team+Sites&tp=Document&k=Sales
```

Using WHERE Clauses

While the keyword search is useful, it is limited by the fact that you can only specify a single keyword. Furthermore, the `k` parameter doesn't support Boolean algebra or wildcards. In order to add these capabilities to our queries, we will need to specify a `WHERE` clause using the `w` parameter. This parameter will allow you to use the conditional statements available from the SQL full-text search language. For example, the following query searches using two keywords:

```
http://[server]/Search.aspx?s=
All+Sources&w=CONTAINS('Microsoft%20AND%20SharePoint')
```

The `CONTAINS` predicate is a powerful way to customize your queries. Along with Boolean algebra, you can also use it to specify wildcard searches. Be careful with the syntax for wildcard searches because the query expects the wildcard argument to be surrounded by double quotes and then again by single quotes. The following code shows how an asterisk, double quotes, and single quotes are used together to find all items that contain a word beginning with "Micro":

```
http://[server]/Search.aspx?s=All+Sources&w=CONTAINS('"Micro*"')
```

The WHERE clauses you create for your queries can utilize any legal syntax available in the SQL full-text query language. An exhaustive examination of the query syntax is beyond the scope of this book; however, you can access the complete documentation online at http://msdn.microsoft.com/library/en-us/acdata/ac_8_qd_15_3rqg.asp?frame=true.

Using the QueryProvider Class

Although you can support complicated queries by posting to the Search.aspx page, my experience is that this technique is best suited for simple or moderately complex queries. When you really want complete control over query creation and execution, you should utilize the Microsoft.SharePoint.Portal.Search.QueryProvider class. This class will allow you to create custom full-text queries, run them, and return a DataSet object.

Constructing a new QueryProvider object in code requires you to provide the SearchApplicationName for the portal or area to be searched. The SearchApplicationName is a GUID that identifies the portal or area that will be queried by the QueryProvider object. The easiest way to get the SearchApplicationName is to use the Microsoft.SharePoint.Portal.PortalContext object. The PortalContext object allows access to useful information about the portal or area including the SearchApplicationName. The following code shows how to access the PortalContext object and use it to create a new QueryProvider object when a web part loads:

```
protected override void OnLoad(System.EventArgs e)
{
    PortalContext context = PortalApplication.GetContext();
    QueryProvider query = new QueryProvider(context.SearchApplicationName);
}
```

Once the QueryProvider object is created, you may use it to execute a full-text query against the search service. Executing the query is a straightforward process of building a search string and calling the Execute method. This method runs the query and returns a DataSet object, which you can use to display the results. Once again, the challenge is to correctly create the full-text query. Later in this chapter I'll show you how you can create and copy queries using SPS and a custom web part, but for now I've provided a complete function in Listing 4-1 that returns a query string based on a keyword.

Listing 4-1. *Creating a Full-Text Query*

```
private string buildQuery(string keyword)
{
//Create query string from keywords
string queryText =         "SELECT" +
"\"DAV:href\"," +
"\"DAV:displayname\"," +
"\"DAV:contentclass\"," +
"\"DAV:getlastmodified\"," +
"\"DAV:getcontentlength\"," +
"\"DAV:iscollection\"," +
"\"urn:schemas-microsoft-com:sharepoint:portal:profile:WorkPhone\"," +
"\"urn:schemas-microsoft-com:sharepoint:portal:profile:WorkEmail\"," +
"\"urn:schemas-microsoft-com:sharepoint:portal:profile:Title\"," +
"\"urn:schemas-microsoft-com:sharepoint:portal:profile:Department\"," +
"\"urn:schemas.microsoft.com:fulltextqueryinfo:PictureURL\"," +
"\"urn:schemas-microsoft-com:office:office#Author\"," +
"\"urn:schemas.microsoft.com:fulltextqueryinfo:description\"," +
"\"urn:schemas.microsoft.com:fulltextqueryinfo:rank\"," +
"\"urn:schemas.microsoft.com:fulltextqueryinfo:sitename\"," +
"\"urn:schemas.microsoft.com:fulltextqueryinfo:displaytitle\"," +
"\"urn:schemas-microsoft-com:publishing:Category\"," +
"\"urn:schemas-microsoft-com:office:office#ows_CrawlType\"," +
"\"urn:schemas-microsoft-com:office:office#ows_ListTemplate\"," +
"\"urn:schemas-microsoft-com:office:office#ows_SiteName\"," +
"\"urn:schemas-microsoft-com:office:office#ows_ImageWidth\"," +
"\"urn:schemas-microsoft-com:office:office#ows_ImageHeight\"," +
"\"DAV:getcontenttype\"," +
"\"urn:schemas-microsoft-com:sharepoint:portal:area:Path\"," +
"\"urn:schemas-microsoft-com:sharepoint:portal:area:CategoryUrlNavigation\"," +
"\"urn:schemas-microsoft-com:publishing:CategoryTitle\"," +
"\"urn:schemas.microsoft.com:fulltextqueryinfo:sdid\"," +
"\"urn:schemas-microsoft-com:sharepoint:portal:objectid\"" +
" from " +
"( TABLE Portal_Content..Scope() " +
"UNION ALL TABLE Non_Portal_Content..Scope() ) " +
" where " +
"WITH " +
"(\"DAV:contentclass\":0, " +
"\"urn:schemas.microsoft.com:fulltextqueryinfo:description\":0, " +
"\"urn:schemas.microsoft.com:fulltextqueryinfo:sourcegroup\":0, " +
"\"urn:schemas.microsoft.com:fulltextqueryinfo:cataloggroup\":0, " +
"\"urn:schemas-microsoft-com:office:office#Keywords\":1.0, " +
"\"urn:schemas-microsoft-com:office:office#Title\":0.9, " +
"\"DAV:displayname\":0.9, " +
"\"urn:schemas-microsoft-com:publishing:Category\":0.8, " +
"\"urn:schemas-microsoft-com:office:office#Subject\":0.8, " +
"\"urn:schemas-microsoft-com:office:office#Author\":0.7, " +
"\"urn:schemas-microsoft-com:office:office#Description\":0.5, " +
"\"urn:schemas-microsoft-com:sharepoint:portal:profile:PreferredName\":0.2, " +
"contents:0.1,*:0.05) AS #WeightedProps " +
"((\"urn:schemas-microsoft-com:publishing:HomeBestBetKeywords\"" +
"= some array ['" + keyword + "'] " +
" RANK BY COERCION(absolute, 999)) " +
"OR (FREETEXT(" +
"\"urn:schemas-microsoft-com:sharepoint:portal:profile:PreferredName\", '" +
keyword + "')  " +
"OR CONTAINS('\"" + keyword + "\"') " +
"RANK BY COERCION(multiply, 0.01)) " +
"OR FREETEXT(#WeightedProps, '" + keyword + "') ) " +
"ORDER BY \"urn:schemas.microsoft.com:fulltextqueryinfo:rank\" DESC";

return queryText;
}
```

Once you have a good way for building a full-text query, it is relatively easy to create a web part that uses the query to return results. If you use the function in Listing 4-1, then you will be returning just about every column you may be interested in. You can then pick which columns to display. As always, the simplest thing to do is bind the results to a `DataGrid` control. Listing 4-2 shows a complete web part that uses a keyword to return a hyperlink and description for each matching item.

Listing 4-2. *A QueryProvider Web Part*

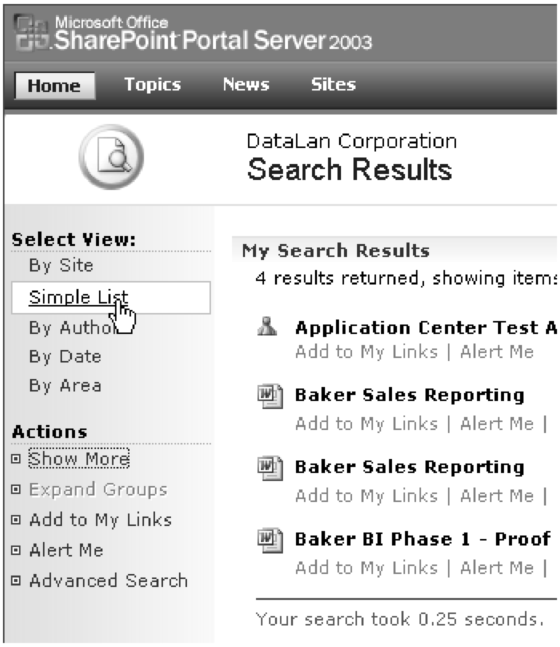
```
using System;
using System.ComponentModel;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml.Serialization;
using Microsoft.SharePoint;
using Microsoft.SharePoint.Portal;
using Microsoft.SharePoint.Utilities;
using Microsoft.SharePoint.WebPartPages;
using Microsoft.SharePoint.WebControls;
using Microsoft.SharePoint.Portal.Search;
using System.Data;
using System.Data.SqlClient;

namespace SPSQueryProvider
{
    [DefaultProperty(""),
    ToolboxData("<{0}:Search runat=server></{0}:Search>"),
    XmlRoot(Namespace="SPSQueryProvider")]
    public class Search : Microsoft.SharePoint.WebPartPages.WebPart
    {

        //GUI elements
        TextBox keywords;
        Button search;
        DataGrid grid;
        Label messages;
```

```
//Search objects
PortalContext context;
QueryProvider query;
DataSet results;

protected override void OnLoad(System.EventArgs e)
{
    try
    {
        //Retrieve context for search
        context = PortalApplication.GetContext();
        query = new QueryProvider(context.SearchApplicationName);
    }
    catch
    {
        context=null;
        query=null;
    }
}

protected override void CreateChildControls()
{
    //Build GUI
    keywords = new TextBox();
    Controls.Add(keywords);

    search = new Button();
    search.Text = "Search";
    search.Click += new EventHandler(search_Click);
    Controls.Add(search);

    grid = new DataGrid();
    grid.AutoGenerateColumns = false;
    grid.GridLines = GridLines.None;
    Controls.Add(grid);

    HyperLinkColumn linkColumn = new HyperLinkColumn();
    linkColumn.DataNavigateUrlField = "DAV:href";
    linkColumn.DataTextField =
    "urn:schemas.microsoft.com:fulltextqueryinfo:displaytitle";
    linkColumn.HeaderText = "Title";
    grid.Columns.Add(linkColumn);

    boundColumn = new BoundColumn();
    boundColumn.DataField =
    "urn:schemas.microsoft.com:fulltextqueryinfo:description";
    boundColumn.HeaderText = "Description";
    grid.Columns.Add(boundColumn);
    messages = new Label();
    Controls.Add(messages);

}

protected override void RenderWebPart(HtmlTextWriter output)
{
    try
    {
        //Bind results
        grid.DataSource = results;
        grid.DataBind();
    }
    catch(Exception x)
    {
        messages.Text += x.Message;
    }

    //Display GUI
    output.Write("<TABLE BORDER=0>");
    output.Write("<TR>");
    output.Write("<TD>");
    keywords.RenderControl(output);
    output.Write("</TD>");
    output.Write("</TR>");
    output.Write("<TR>");
    output.Write("<TD>");
    search.RenderControl(output);
    output.Write("</TD>");
    output.Write("</TR>");
    output.Write("<TR>");
    output.Write("<TD>");
    grid.RenderControl(output);
    output.Write("</TD>");
    output.Write("</TR>");
    output.Write("<TR>");
    output.Write("<TD>");
    messages.RenderControl(output);
    output.Write("</TD>");
    output.Write("</TR>");
    output.Write("</TABLE>");
}

private void search_Click(object sender, EventArgs e)
{
    try
```

```
            {
                //Execute Query
                results = query.Execute(buildQuery());
            }
            catch(Exception x)
            {
                messages.Text += x.Message;
            }
        }

    }
}
```

Customizing Search Results

Using the `QueryProvider` class to execute a query and display results gives you complete control over the searching process, but it also requires you to handle every aspect of displaying, sorting, and grouping the results. This means that you are unable to use the standard sorting and grouping functionality found in the action list on `Search.aspx`. These actions offer significant functionality that you will have to develop yourself when you use a `QueryProvider` class. In many cases, what we want is a way to customize how results are displayed, but still utilize the action links shown in Figure 4-3.



DataLan Corporation
Search Results

My Search Results
4 results returned, showing items

Application Center Test A
Add to My Links | Alert Me

Baker Sales Reporting
Add to My Links | Alert Me |

Baker Sales Reporting
Add to My Links | Alert Me |

Baker BI Phase 1 - Proof
Add to My Links | Alert Me |

Your search took 0.25 seconds.

Figure 4-3. *Sorting and grouping links*

The search results that normally appear on Search.aspx are created by the web part Microsoft.SharePoint.Portal.WebControls.SearchResults while the action links are generated by the web part Microsoft.SharePoint.Portal.WebControls.SearchResultManagement. In order to create our own custom result set that can interact with the SearchResultManagement web part, we must inherit from the SearchResults web part. While it is certainly unusual to inherit directly from web parts in SPS, this approach is what ensures that the resulting web part will interact correctly with the SearchResultManagement web part.

In order to inherit from the SearchResults web part, you create a web part project as normal, but instead of inheriting from Microsoft.SharePoint.WebPartPages.WebPart, the project inherits from Microsoft.SharePoint.Portal.WebControls.SearchResults. Once you inherit from the SearchResults web part, you can override any event in the web part life cycle to customize the presentation of the results. As a simple exercise to get started, you can create a web part that inherits from SearchResults but makes no changes to the output. Listing 4-3 shows the code for such a web part. You should note the call to base.RenderWebPart, which displays the search results in the Search.aspx page.

Listing 4-3. *Inheriting from SearchResults*

```
using System;
using System.ComponentModel;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml.Serialization;
using Microsoft.SharePoint;
using Microsoft.SharePoint.Utilities;
using Microsoft.SharePoint.WebPartPages;

namespace MySearchResults
{

    [DefaultProperty(""),
    ToolboxData("<{0}:ResultSet runat=server></{0}:ResultSet>"),
    XmlRoot(Namespace="MySearchResults")]
    public class ResultSet :
    Microsoft.SharePoint.Portal.WebControls.SearchResults
    {

        protected override void RenderWebPart(HtmlTextWriter output)
        {
            base.RenderWebPart(output);
        }
    }
}
```

Once you create the web part, you must give it a strong name, place it in the `\bin` directory, and register it as safe in the `web.config` file. Then you are ready to add it to the `Search.aspx` page. Unlike most pages, however, the `Search.aspx` page does not have an explicit edit command available in the actions list. Instead, you must manually place the page in edit mode by navigating to the following URL in the browser:

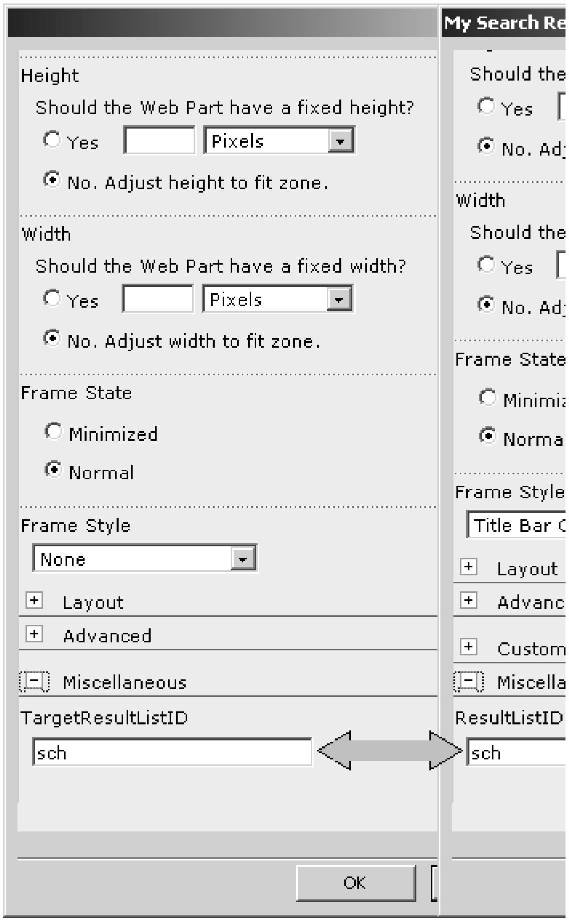
```
http://[server]/Search.aspx?Mode=Edit&PageView=Shared
```

Once the page enters edit mode, you should see the familiar Modify Shared Page menu in the upper-right corner. You can now use this menu to import the new web part. After importing the new web part and running a search, you will see the new web part displaying the same results as the standard implementation of the `SearchResults` web part. At this point, you could go to the Web Parts Maintenance page and close the original `SearchResults` web part, leaving only your custom web part on the page. The Web Parts Maintenance page can be reached by appending `?contents=1` to any page, as shown in this example:

```
http://[server]/[sitename]/default.aspx?contents=1
```

■**Note** When working with web parts on `Search.aspx`, you might find it helpful to add a Links web part to the page that contains a link placing the page in edit mode. You may also want to add a link to the Web Parts Maintenance page, which can be useful for removing malfunctioning web parts during testing.

Although we have created our own web part that shows results, you may notice that it does not yet respond to the sorting and grouping links. This is because the connection between our custom `SearchResults` web part and the `SearchResultManagement` web part has not yet been established. We can connect the web parts by setting the `TargetResultListID` property of the `SearchResultManagement` web part to the same value as the `ResultListID` property of our custom web part. Both of these properties are accessible through the web part properties pane and are located under the Miscellaneous section. However, you will not be able to access the properties for the `SearchResultManagement` web part in SPS. Instead, open `Search.aspx` in Microsoft FrontPage and set the properties to the same value. Figure 4-4 shows the properties pane for each of the web parts.


Height

Should the Web Part have a fixed height?

Yes [ ] Pixels [✓ No. Adjust height to fit zone.]

Width

Should the Web Part have a fixed width?

Yes [ ] Pixels [✓ No. Adjust width to fit zone.]

Frame State

Minimized [✓ Normal]

Frame Style

None

Layout

Advanced

Miscellaneous

TargetResultListID

sch

My Search Re

Should the

Yes [✓ No. Ad]

Width

Should the

Yes [✓ No. Ad]

Frame State

Minimiz [✓ Norma]

Frame Style

Title Bar C

Layout

Advanc

Custom

Miscella

ResultListID

sch

Figure 4-4. *Connecting the web parts in Microsoft FrontPage*

Modifying the Query Behavior

At this point, we have done little more than reproduce the functionality of the original `SearchResults` web part, but we have created a project that will now allow us to override that functionality in several ways. The first changes we can make involve how the query is processed before the results are displayed. It turns out that the `SearchResults` web part does more than just display the query results—it actually processes the query as well.

The `SearchResults` web part maintains a template for creating queries. The template is made up of four parts that are saved as properties of the web part. These properties are `QueryTemplateSelectPart`, `QueryTemplateFromPart`, `QueryTemplateWherePart`, and `QueryTemplateOrderByPart`. These templates are used to generate a full-text query string when the `GenerateQueryString` method is called. This query string is then executed to produce the result set.

The template information, query creation, and execution behavior are contained within the parent `SearchResults` class. If we want to change the query, therefore, we have to override the GenerateQueryString method and modify the query templates. Once we modify the templates, the result set will be based on our new query.

One of the most popular reasons for modifying the query templates is to enable wildcard searching in SPS. This will allow you to put an asterisk after a term in the search box. By default, the query templates do not support wildcards, but we can modify the QueryTemplateWherePart to include this functionality. The templates we want to modify use tokens that are replaced when the query is created, so we have to specify the change using a token that represents the query parameter. The query parameter token is the term keywordinput with two underscores at the beginning and the end. Listing 4-4 shows an example of overriding the GenerateQueryString method to add wildcard functionality to the search. Simply adding this code to the code from Listing 4-3 will enable wildcard searching.

Listing 4-4. *Incorporating Wildcard Searches*

```
protected override string GenerateQueryString(string
strKeyword,System.Collections.ArrayList rgScopeList,string strWhereAndPart,

out string strSavedQuery)
{
    //Add wildcard functionality to the query
    string wherePart = base.QueryTemplateWherePart;
    string firstPart =
        wherePart.Substring(0,wherePart.LastIndexOf("RANK BY COERCION"));
    string lastPart =
        wherePart.Substring(wherePart.LastIndexOf("RANK BY COERCION"));
    wherePart =
         firstPart + " OR CONTAINS('\"%__keywordinput__%\"') " + lastPart;
    base.QueryTemplateWherePart = wherePart;

    //Run query
    string query =
        base.GenerateQueryString(strKeyword,rgScopeList,
                                 strWhereAndPart,out strSavedQuery);
    strSavedQuery = query;
    return query;
}
```

Generating Custom Displays

While the standard result display is useful in most situations, you may want to change the way the results are shown in Search.aspx. In this case, we want to override the rendering behavior of the parent base class and create a custom look. This look may be as simple as binding the results to a DataGrid, or it may be as specialized as using a TreeView control to group results in a compact format. In any case, we must intercept the query results and process them on our own.

Intercepting the query results is accomplished by overriding the IssueQuery method. The IssueQuery method executes the query that was created in the GenerateQueryString method and returns a DataSet. Once we have intercepted the DataSet, we can produce a custom look for the results. Listing 4-5 shows how to override the IssueQuery method.

Listing 4-5. *Intercepting Query Results*

```
DataSet results;

protected override object IssueQuery(string strQuery, int startRowIndex,
int endRowIndex)
{
    results =(System.Data.DataSet)
        base.IssueQuery (strQuery, startRowIndex, endRowIndex);
    return results;
}
```

If you choose to create your own results page, then you will not have to call the RenderWebPart method of the base class. Instead, you just simply process the results yourself. Most of the time this means overriding the CreateChildControls method to add a DataGrid or TreeView to the page and then processing the DataSet to fill the appropriate control. Finally, you should note that creating your own custom display will divorce your web part from the SearchResultManagement web part. If you create your own display, be sure to remove the SearchResultManagement web part from the Search.aspx page.

Customizing SharePoint Portal Server Navigation

One of the complaints I hear regularly about SPS is that the system of topics, areas, and sites is confusing and difficult to navigate. This confusion is caused by several factors that are unique to the way in which SPS was designed and built by Microsoft. Therefore, it is important to understand the design philosophy behind SPS before we can attempt to improve its navigation system.

Technical people tend to think in highly organized hierarchies represented by tree views. This structured approach is valuable when analyzing and solving engineering problems, but often leads to a rigid user interface design. This thinking is so strong that developers initially assume that topics, areas, and sites must have a hierarchical relationship. Therefore, every developer I talk to about SPS navigation wants a hierarchical tree view for navigation similar to the one shown in Figure 4-5.

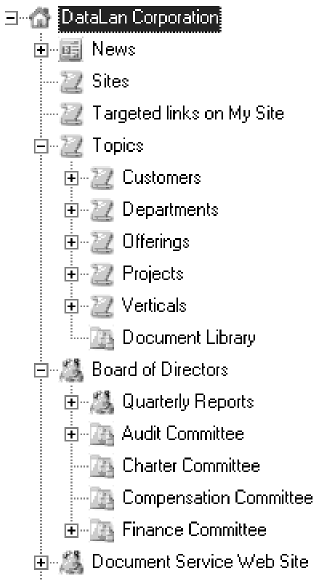
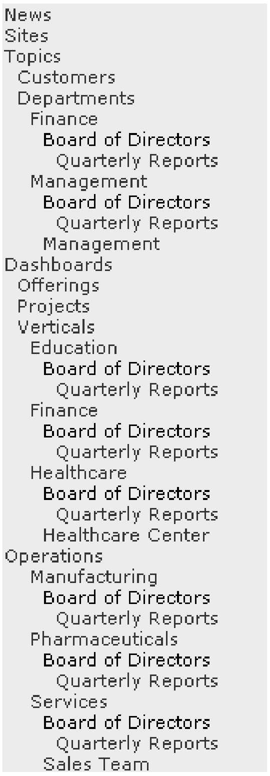


Figure 4-5. *A hierarchy of topics, areas, and sites*

The problem with the monolithic tree view approach to SPS navigation is that the relationship between SPS and WSS is not hierarchical. SPS is, in fact, a site collection and is, therefore, a peer with every other WSS site collection. This means that there is no effective way to create a true hierarchical view of all topics, areas, and sites.

The challenge becomes even greater when you realize that each time a new site collection is created, the author is given the opportunity to select multiple SPS areas that will link to the new site. This means that a hierarchical view of the entire SharePoint infrastructure would contain the same site listed in multiple branches of the tree. Furthermore, it is impossible to know which branch was navigated to arrive at the site. An attempt to create such a view results in a tree similar to the one shown in Figure 4-6, where the Board of Directors site is shown under eight different branches of the tree.



```
News
Sites
Topics
  Customers
  Departments
    Finance
      Board of Directors
        Quarterly Reports
    Management
      Board of Directors
        Quarterly Reports
      Management
Dashboards
  Offerings
  Projects
  Verticals
    Education
      Board of Directors
        Quarterly Reports
    Finance
      Board of Directors
        Quarterly Reports
    Healthcare
      Board of Directors
        Quarterly Reports
      Healthcare Center
Operations
    Manufacturing
      Board of Directors
        Quarterly Reports
    Pharmaceuticals
      Board of Directors
        Quarterly Reports
    Services
      Board of Directors
        Quarterly Reports
      Sales Team
```

Figure 4-6. *A hierarchy with the same site in multiple branches*

Once you encounter the problem of multiple listings for the same site, you begin to understand why Microsoft designed SPS navigation as it did. Furthermore, the realization demands a rethinking of the relationship between SPS and WSS that will bring clarity to developers and end users. This is why I prefer to think of SPS like a Google engine on steroids.

Changing your SPS paradigm from a strict hierarchy to a Google-like search environment is useful because it implies separation between SPS and WSS. Additionally, it is a paradigm that end users understand. No one thinks of the Google search engine as having a hierarchical relationship with the sites that it aggregates. End users know that Google provides links to sites of interest, but that Google itself is not part of those sites. Once you click a link on the Google site, you leave Google and go to the target site. This is the true relationship between SPS and WSS. The issue is how best to reflect this to the end user.

Using Themes to Differentiate Functionality

Perhaps the simplest way to help end users understand the difference between SPS and WSS is to use different themes for each. The themes can be similar in appearance in order to suggest a relationship, but different enough to indicate that the relationship is casual and not strictly hierarchical. WSS ships with several predefined themes that work well for this purpose. For example, you might use the default theme for SPS and choose the Compass theme for all of your WSS sites. This is easily accomplished for WSS sites by selecting Site Settings ➤ Apply Theme to Site.

Unfortunately, it is not possible to use this same technique to change the SPS theme. Once again, this is because of the design choices made by Microsoft. It turns out that sub areas underneath the portal home are specialized versions of WSS sites that do not fit into the existing theme management framework. There is no link available under Site Settings to change the theme for any area in SPS. If you alternately try to change the theme in FrontPage, the operation fails and you are told that the theme is read-only. Fortunately, we can utilize a workaround to make use of the existing theme set.

Follow these steps to change the SPS theme:

1. Open the Windows File Explorer and navigate to `\Program Files\Common Files\Microsoft Shared\web server extensions\60\TEMPLATE\THEMES`.

2. Copy all of the subfolders, which contain the various WSS themes, to the folder `\Program Files\Common Files\Microsoft Shared\web server extensions\60\TEMPLATE\LAYOUTS\`*`culture`*`\STYLES`.

3. Log into SPS as a portal administrator.

4. Select Site Settings from the portal home page.

5. On the Site Settings page, select General Settings ➤ Change Portal Site Properties and SharePoint Site Creation Settings.

6. In the Custom Cascading Style Sheet section, type the address of the new style sheet in the following form:

   ```
   /_layouts/[culture]/styles/[theme folder]/theme.css
   ```

7. Click the OK button to apply the new style.

Using Tree Views Effectively

While there is no way to use a hierarchical tree view to show the entire SharePoint infrastructure, that does not mean that tree views have no place in the navigation system. The key to using tree views effectively is to limit their scope so that the displayed information is meaningful to the end user. This will require us to adopt a slightly different approach for areas than we use for sites.

When displaying a tree view in SPS areas, it is best to show the complete hierarchy of topics and areas as well as the site listings contained in each area. This provides a reasonable view of the portal structure down to the site collection level. Limiting the hierarchy to these elements makes it manageable and provides good navigation information to the end user. Figure 4-7 shows a typical tree view in SPS that replaces the standard topic navigation system.

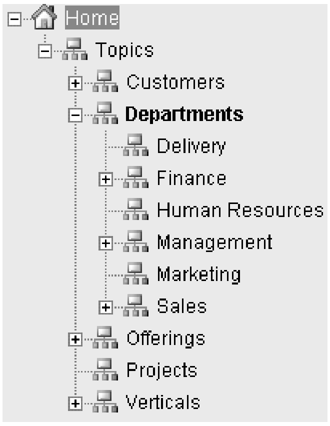


Figure 4-7. *A tree view showing topics, areas, and listings*

The `Microsoft.SharePoint.Portal.SiteData.AreaManager` and `Area` classes are used to build the hierarchical view, which is contained inside a `TreeView` control. For each area, the tree also shows links from the `Listings` collection associated with the area. Because opening a new window can also be an effective way to signal the differences between SPS and WSS, an option on the web part allows you to either navigate directly to a clicked link or open it in a new window.

When presenting the same tree view on a WSS site, I have chosen to expand the tree branch that contains the area where the user originally clicked the site link. This gives the user a sense of continuity. I have also added the site collection hierarchy as a left-justified branch in the tree. Figure 4-8 shows the web part in a WSS site. You will build this web part in its entirety in an exercise at the end of this chapter.

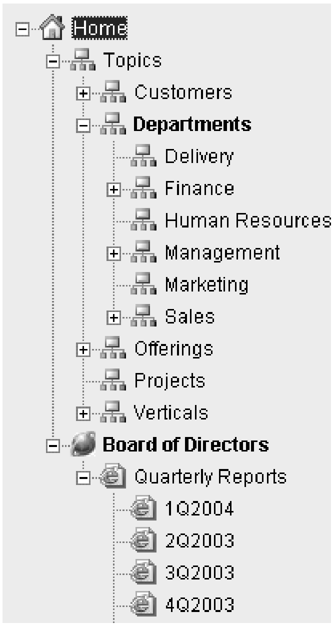


Figure 4-8. *A tree view on a WSS site*

Using Breadcrumb Navigation

Another popular navigation alternative is *breadcrumb* navigation. Breadcrumb navigation uses the same principles as tree view navigation, except it only shows a single branch of the tree. Breadcrumbs are excellent for showing a simplified navigation trail using little screen real estate. Each breadcrumb in the trail is a hyperlink so users can easily return to previously visited pages. Figure 4-9 shows a typical breadcrumb display on a WSS site.

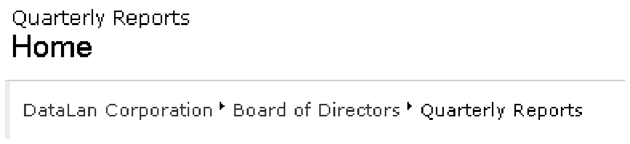

Quarterly Reports
Home
DataLan Corporation ‣ Board of Directors ‣ Quarterly Reports

Figure 4-9. *A breadcrumb navigation web part*

In order to start building the breadcrumb trail, we need to first get a reference to the current web. The trail is then built by recursing backwards through the parent webs until a complete tree branch is created. Listing 4-6 shows how this is done for WSS site collections.

Listing 4-6. *Building a Breadcrumb Trail for a Site Collection*

```
Private strTrail As String = ""

Protected Overrides Sub RenderWebPart( _
ByVal output As System.Web.UI.HtmlTextWriter)

    Try

        Dim objWeb As SPWeb = SPControl.GetContextWeb(Context)
        strTrail = objWeb.Title
        BuildTrail(objWeb)
        output.Write(strTrail)

    Catch x As Exception
        strTrail += "<p>" & x.Message & "</p><br>"
    End Try

End Sub

Private Sub BuildTrail(ByVal objWeb As SPWeb)

    Try
```

```
        If objWeb.IsRootWeb = False Then
            'Show the Parent Web
            strTrail = _
            "<a href='" & objWeb.ParentWeb.Url & "'>" _
            & objWeb.ParentWeb.Title _
            & "</a><img src='/_layouts/images/blk_rgt.gif'>" & strTrail

            'Recurse
            BuildTrail(objWeb.ParentWeb)
        End If

    Catch x As Exception
        strTrail += "<p>" & x.Message & "</p><br>"
    End Try

End Sub
```

Creating a breadcrumb trail is fairly simple if you are only interested in showing the current site collection. If you want to link back to the areas in SPS, however, you face the same problem as with the tree view navigation; the user could have come from any of several areas to arrive at the current site. In order to solve this, I use the `Page.Request.UrlReferrer.AbsolutePath` property to identify the area that referred the user to the current page. This is done by checking it against the collection of areas in SPS until a match is found. Listing 4-7 shows the Visual Basic .NET code for finding an area based on a referring URL.

Listing 4-7. *Finding an Area by URL*

```
Private Function GetAreaByURL( _
ByVal strURL As String, ByVal objAreas As AreaCollection) As Area

    Try

        Dim objReturnArea As Area = Nothing

        'Find the Area matching the given URL
        For Each objSubArea As Area In objAreas
            If objSubArea.AreaTemplate = "SPSTOPIC" AndAlso _
            strURL.IndexOf(objSubArea.WebUrl) = 0 Then
                objReturnArea = objSubArea
                Exit For
            Else
                Dim objSubSubArea = GetAreaByURL(strURL, objSubArea.Areas)
                If Not (objSubSubArea Is Nothing) Then
                    objReturnArea = objSubSubArea
                    Exit For
                End If
            End If
        Next
        Return objReturnArea

    Catch x As Exception
        Return Nothing
    End Try

End Function
```

Handling Performance Issues

The web parts I have presented in this section all share a common potential performance issue. Because these parts must traverse large parts of the SharePoint site structure, they can easily create performance bottlenecks that can lead to slow page loading or even page time-outs. Therefore, this is a good place to discuss some general techniques for improving web part performance. In particular, I will look at two approaches for improving performance: asynchronous data retrieval and data caching.

Understanding Asynchronous Data Retrieval

Whenever pages are loaded that contain web parts, SharePoint uses a single thread to load the page and process the web parts. This means that the time to load a page is increased as the sum of all the data-retrieval processes increase. If you build web parts that require a lot of processing—like filling a tree control with sites—this delay could become unacceptable. One way to mitigate this problem is to build your web parts so that they perform all data-retrieval operations on a separate thread. This type of asynchronous data retrieval is supported directly in the web part framework through the `GetRequiresData` and `GetData` methods.

The `GetRequiresData` method is called during the web part life cycle, and determines whether or not asynchronous data retrieval will be implemented. Normally, this method returns `False`, which means that none of the standard web parts implement asynchronous data retrieval. You may, however, override this method and return `True`, which will signal your intention to use asynchronous data retrieval. The following code shows how this is done:

```
public override bool GetRequiresData()
{
    return true;
}
```

If the `GetRequiresData` method returns `True`, then the web part framework will call the `GetData` method. The `GetData` method is used to register a callback method, which ultimately will perform the data retrieval on a different thread. Therefore, you must first define a callback method that conforms to the type `System.Threading.WaitCallback`. The function signature for this type has no return value and takes a single argument of type `object`. The argument is used to pass any data you want to the callback method. The following code shows an example callback method to build a simple list of subsites. Note how the object data type is cast to the correct data type before it is used for retrieval.

```
public void buildTree(object data)
{
    //get child webs
    SPWeb web = (SPWeb)data;
    SPWebCollection webs = web.Webs;

    foreach(SPWeb subweb in webs)
    {
        tree += "<p>" + subweb.Title + "</p>";
    }
}
```

Registering the callback method is accomplished using the `RegisterWorkItemCallback` method. This method is called from the `GetData` method, where the arguments are created and passed to the callback method. The following code shows an example using the callback method I showed earlier:

```
public override void GetData()
{
    SPSite site = SPControl.GetContextSite(Context);
    SPWeb web = site.OpenWeb();
    tree = "<p>" + web.Title + "</p>";
    RegisterWorkItemCallback(new WaitCallback(buildTree),web);
}
```

When callback methods are used to retrieve data, the web part framework will track their work and wait for them to complete before trying to render the web part. The timeout for asynchronous data retrieval is controlled by the `WebPartWorkItem` element in the `web.config` file and is set to 7 seconds by default. If an asynchronous data-retrieval operation takes longer than this time, the `RenderWorkItemTimeout` method is called. Using this method, you can create a simple HTML message to indicate that the web part has timed out. The following code shows an example:

```
protected override void RenderWorkItemTimeout(HtmlTextWriter writer)
{
    writer.Write("<p>Web part timed out</p>");
}
```

Understanding Data Caching

While asynchronous data retrieval may lead to somewhat improved performance in multiprocessor machines, data caching will typically yield even bigger performance improvements. When you create web parts, you can make use of either the SharePoint or ASP.NET cache. The cache that is utilized is determined by the value of the `Storage` attribute of the `WebPartCache` element in the `web.config` file. This element is set to `CacheObject` by default, which utilizes the ASP.NET cache. Setting this attribute to `Database` utilizes the SharePoint database as a cache.

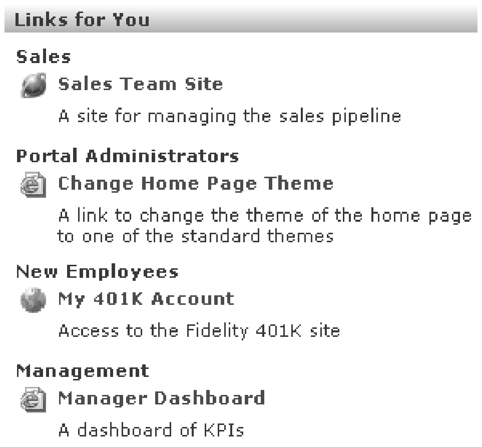
The `PartCacheWrite` method is used to write data to the cache and the `PartCacheRead` method is used to read data from the cache. Typically, a web part will read from the cache and check to see if the return value is `null`. If the value is `null`, then the web part will process normally and write the results to the cache for future use. The `PartCacheInvalidate` method is used to clear the cache, which functions to force a refresh of the data. Listing 4-8 shows a complete web part that creates a simple HTML list of subsites while using the cache to improve performance.

Listing 4-8. *Caching Web Part Data*

```
namespace SPSCacheTreeView
{

    [DefaultProperty(""),
        ToolboxData("<{0}:Builder runat=server></{0}:Builder>"),
        XmlRoot(Namespace="SPSCacheTreeView")]
    public class Builder : Microsoft.SharePoint.WebPartPages.WebPart
    {

        string tree;
        int i = 0;
        LinkButton button;

        public void buildTree()
        {
            SPSite site = SPControl.GetContextSite(Context);
            SPWeb web = site.OpenWeb();
            tree = web.Title + "<br>";
            i++;
            addChildWebs(web);
            i--;
          PartCacheWrite(Storage.Shared,"tree", tree, TimeSpan.FromSeconds(10));
        }

        public void addChildWebs(SPWeb parent)
        {
            try
            {

                //get child webs
                SPWebCollection webs = parent.Webs;

                foreach(SPWeb subweb in webs)
                {
                    //add to tree
                    tree += subweb.Title.PadLeft(subweb.Title.Length + i,
                       "-".ToCharArray()[0]) + "<br>";
                    i++;
                    addChildWebs(subweb);
                    i--;
                }
            }
            catch(Exception x)
            {
                tree += "<p>" + x.Message + "</p>";
            }
        }

        protected override void CreateChildControls()
        {
            button = new LinkButton();
            button.Text = "Refresh";
            button.Click +=new EventHandler(Refresh);
            Controls.Add(button);
        }

        protected override void RenderWebPart(HtmlTextWriter output)
        {
            //display tree
            if(PartCacheRead(Storage.Shared,"tree") == null) buildTree();

            output.Write("<br>");
            output.Write(tree);
            button.RenderControl(output);
        }

        private void Refresh(object sender, EventArgs e)
        {
            PartCacheInvalidate(Storage.Shared, "tree");
        }
    }
}
```

Improving Presentation with Audiences

If you have used SPS in any way, then you have undoubtedly experienced the use of *audiences* to target content. To review briefly, audiences are groups of users that are interested in the same information. These groups can be based on almost any relationship, such as department, team membership, or job role. SPS allows you to specify audience membership based on profile attributes. The important thing to remember about audiences is that they are used for presentation only and have no ability to secure information. Figure 4-10 shows a classic use of audiences to drive the Links for You web part that appears on the SPS home page.


Links for You

Sales
Sales Team Site
A site for managing the sales pipeline

Portal Administrators
Change Home Page Theme
A link to change the theme of the home page to one of the standard themes

New Employees
My 401K Account
Access to the Fidelity 401K site

Management
Manager Dashboard
A dashboard of KPIs

Figure 4-10. *Using audiences to target content*

While audiences are useful for determining when links and web parts should be displayed, we may also want to make use of them within web parts that we create. Using audiences in this way allows your web part to render targeted content, which can significantly improve the end user experience. This strategy also allows you to introduce audience functionality into WSS sites, where it is normally not available.

The entry point for all audience-based functionality is the `Microsoft.SharePoint.Portal.Audience.AudienceManager` class. This class allows you to retrieve audience definitions for the current user or for the entire portal. You can also perform several administrative functions such as adding new audiences to the portal, but our focus will be on using the existing audience definitions.

When you create web parts that access audience information, you will quite often be concerned with identifying the audience membership of the current user. This will allow your web part to determine the appropriate content for targeting. You can easily return a collection of audiences to which the current user belongs by calling the `GetUserAudienceIDs` method of the `AudienceManager` class.

The `GetUserAudienceIDs` method returns an `ArrayList` of `AudienceNameID` objects. The `AudienceNameID` objects can be used to access the GUID for the audience and the name of the audience and to determine if the audience is currently valid. Listing 4-9 shows a complete web part that lists the audiences to which the current user belongs.

Listing 4-9. *Listing Audience Membership*

```
using System;
using System.ComponentModel;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml.Serialization;
using Microsoft.SharePoint;
using Microsoft.SharePoint.Utilities;
using Microsoft.SharePoint.WebPartPages;
using Microsoft.SharePoint.Portal;
using Microsoft.SharePoint.Portal.Audience;
using System.Collections;

namespace SPSMyAudiences
{

    [DefaultProperty(""),
        ToolboxData("<{0}:Membership runat=server></{0}:Membership>"),
        XmlRoot(Namespace="SPSMyAudiences")]
    public class Membership : Microsoft.SharePoint.WebPartPages.WebPart
    {

        protected override void RenderWebPart(HtmlTextWriter output)
        {

            try
            {
                //Get the portal context
                SPSite portal =
                new SPSite(Page.Request.Url.GetLeftPart(UriPartial.Authority));
                PortalContext context = PortalApplication.GetContext(portal.ID);

                //Get the list of audiences for the user
                AudienceManager manager = new AudienceManager(context);
                ArrayList audienceIDList = manager.GetUserAudienceIDs();

                if(audienceIDList != null)
                {
                   IEnumerator audienceNameIDs = audienceIDList.GetEnumerator();

                    output.Write("<div class=\"ms-sectionheader\">");
                    while(audienceNameIDs.MoveNext())
                    {
                        output.Write(
                          ((AudienceNameID)audienceNameIDs.Current).AudienceName
                          + "<br>");
                    }
                    output.Write("</div>");
                }
            }

            catch (Exception x)
            {
                output.Write("<p>" + x.Message + "</p>");
            }
        }
    }
}
```

Along with determining the membership for the current user, you will often want to retrieve the complete set of audiences for the portal. This information is useful for assigning targeted content to individual audiences. You can either use the information within the web part to change the display, or you can use it in a tool part to target the entire web part. At the end of this chapter, you will find an exercise that uses such information internally to display list items organized by audience. For this discussion, we'll look at using audience information to target an entire web part on a WSS site. This functionality will reproduce the audience targeting capabilities found in SPS, but not normally available in WSS. Figure 4-11 shows a typical web part property in SPS used to target audiences.

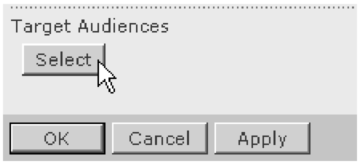


Figure 4-11. *Targeting a web part in SPS*

In order to re-create the audience targeting functionality found in SPS, we must construct a custom tool part. Custom tool parts are similar to web parts, but they are intended to be displayed in the properties pane and not in the site itself. This allows us to create complex interfaces for setting web part properties. Because I covered tool part creation in detail in my first book, *Microsoft SharePoint: Building Office 2003 Solutions*, I'm going to assume some familiarity with the concept and cover just the highlights. In our example, we will create a tool part that lists all of the available audiences and allows an administrator to select the ones to associate with the web part. We will then use the information to allow only users who are members of the selected audiences to see the web part.

As I noted, tool parts are similar to web parts in their construction and support nearly the same life-cycle events. This means that the code we write will be similar to any web part. Listing 4-10 shows the basics of our tool part, which fills a list with the set of all available audiences in SPS.

Listing 4-10. *Listing Audiences in a Tool Part*

```
public class AudienceTool : ToolPart
{
    //variables
    protected ListBox audienceList;
    protected TextBox audienceString;
    protected TextBox cancelString;

    protected override void OnLoad(EventArgs e)
    {
        //Get the portal context
        SPSite portal =
        new SPSite(Page.Request.Url.GetLeftPart(UriPartial.Authority));
        PortalContext context = PortalApplication.GetContext(portal.ID);
        //Get the list of audiences
        AudienceManager manager = new AudienceManager(context);

        EnsureChildControls();
        if (audienceList.Items.Count==0)
        {
            Container webpart = (Container)this.ParentToolPane.SelectedWebPart;
            audienceString.Text = webpart.Audiences;
            cancelString.Text = webpart.Audiences;

            //Fill the ListBox
            foreach(Audience audience in manager.Audiences)
            {
                audienceList.Items.Add(audience.AudienceName);
            }
        }
    }

    protected override void CreateChildControls()
    {
            audienceList = new ListBox();
            audienceList.SelectionMode = ListSelectionMode.Multiple;
            Controls.Add(audienceList);

            audienceString = new TextBox();
            Controls.Add(audienceString);

            cancelString = new TextBox();
            cancelString.Visible=false;
            Controls.Add(cancelString);

    }

    protected override void RenderToolPart(HtmlTextWriter output)
    {
            //Draw list
            audienceList.RenderControl(output);
            output.Write("<br>");
            audienceString.RenderControl(output);
            output.Write("<br>");
            cancelString.RenderControl(output);
    }
}
```

The main difference between tool parts and web parts is that tool parts must respond when the user clicks the OK, Apply, or Cancel button in the properties pane. In our design, the tool part will then build a semicolon-delimited string that contains the selected audiences and pass it to the parent web part. The parent web part can then parse the string and determine if the current user is a member of the designated audiences.

Our tool part will receive a call to the ApplyChanges function whenever the OK or Apply button is clicked. The CancelChanges function is called when the Cancel button is clicked. The SyncChanges function is called after any operation in order to synchronize the web part with the tool part. In our example, we are assuming that the tool part is being used by a web part named Container. Listing 4-11 shows the code for setting the Audience property of the web part based on the selections made in the tool part.

Listing 4-11. *Setting the Web Part Property*

```
public override void ApplyChanges()
{
    //Build audience list
    EnsureChildControls();
    cancelString.Text = audienceString.Text;
    audienceString.Text="";

    foreach(ListItem item in audienceList.Items)
    {
        if(item.Selected==true)
            audienceString.Text += item.Text + ";";
    }

    //Update web part property
    Container webpart = (Container)ParentToolPane.SelectedWebPart;
    webpart.Audiences = audienceString.Text;

}

public override void SyncChanges()
{
    //Update web part property
    EnsureChildControls();
    Container webpart = (Container)this.ParentToolPane.SelectedWebPart;
    audienceString.Text = webpart.Audiences;
    cancelString.Text = webpart.Audiences;
}

public override void CancelChanges()
{
    EnsureChildControls();
    audienceString.Text = cancelString.Text;
    Container webpart = (Container)ParentToolPane.SelectedWebPart;
    webpart.Audiences = cancelString.Text;
}
```

Once the tool part is complete, it is associated with the parent web part by overriding the GetToolParts function. We can then use the audience information for targeting the web part.

For this example, the `Container` web part is a page viewer that creates a `Frameset` element for viewing another web page. Listing 4-12 shows how this web part uses the information from the tool part to decide whether to show the targeted web page.

Listing 4-12. *Targeting a Web Page*

```
[DefaultProperty("URL"),
ToolboxData("<{0}:Container runat=server></{0}:Container>"),
XmlRoot(Namespace="SPSPageView")]
public class Container : Microsoft.SharePoint.WebPartPages.WebPart
{
    //NOTE: URL and PageHeight property code not shown
    protected String m_audiences="All portal users;";

    [Browsable(false),Category("Miscellaneous"),
    DefaultValue("All portal users;"),
    WebPartStorage(Storage.Shared),
    FriendlyName("Audiences"),Description("The selected audiences.")]
    public string  Audiences
    {
        get
        {
            return m_audiences;
        }

        set
        {
            m_audiences = value;
        }
    }

    public override ToolPart[] GetToolParts()
    {
        WebPartToolPart webPartToolPart = new WebPartToolPart();
        CustomPropertyToolPart customToolPart = new CustomPropertyToolPart();
        AudienceTool audienceTool = new AudienceTool();

        ToolPart[] toolParts = new ToolPart[3]
        {webPartToolPart,customToolPart,audienceTool};
        return toolParts;
     }

    protected override void RenderWebPart(HtmlTextWriter output)
    {
        //Get the portal context
        SPSite portal =
        new SPSite(Page.Request.Url.GetLeftPart(UriPartial.Authority));
        PortalContext context = PortalApplication.GetContext(portal.ID);
        //Get the list of audiences for the user
        AudienceManager manager = new AudienceManager(context);
        ArrayList audienceIDList = manager.GetUserAudienceIDs();

        //Get the list of authorized audiences
        String [] audiences = Audiences.Split(";".ToCharArray());
        Boolean authorized = false;

        //Check the lists
        if(audienceIDList != null)
        {
            IEnumerator audienceNameIDs = audienceIDList.GetEnumerator();

            while(audienceNameIDs.MoveNext())
            {
                for(int i=audiences.GetLowerBound(0);
                i<=audiences.GetUpperBound(0);i++)
                {
                    if(audiences[i]==
                    ((AudienceNameID)audienceNameIDs.Current).AudienceName)
                    authorized=true;
                }
            }
        }

        //Draw web part
        if(authorized==true)
            output.Write("<div><iframe height='"
            + PageHeight + "' width='100%' src='" + URL + "'></iframe></div>");
    }
}
```

Exercise 4-1. Tree View Navigation

Many developers and end users find that the default navigation system associated with SPS provides them with insufficient information and capabilities to effectively navigate a SharePoint installation. They would prefer a more inclusive explorer that allows them to better understand the SharePoint topics, areas, and sites without having to leave the current page. In this exercise, you will build a tree view web part with improved navigation capabilities.

Prerequisites

Before beginning this exercise, be sure that you have properly built and deployed the assembly `IEWebControls.dll` according to the instructions given in Chapter 3. The assembly should be strongly named and located in the `Inetpub\wwwroot\bin` directory of your WSS server. This exercise also assumes that you have SPS installed as part of your SharePoint infrastructure.

Starting the Project

This exercise will use the TreeView web control presented in Chapter 3. For simplification, however, we will use standard web part development techniques as opposed to the Web Forms User Control techniques described in Chapter 3. Therefore, you will use the standard web part template for this project.

Follow these steps to start the project:

1. In Visual Studio .NET, select File ➤ New ➤ Project from the menu.
2. In the New Project dialog, select the Visual Basic Projects folder in the Project Types list.
3. In the Templates list, select Web Part Library.
4. Name the project **SPSTreeNav** and click the OK button.
5. In the Solution Explorer, rename WebPart1.vb to **SPSTreeNav.vb**.
6. In the Solution Explorer, rename WebPart1.dwp to **SPSTreeNav.dwp**.
7. Open the file Manifest.xml for editing in Visual Studio .NET.
8. In the DwpFiles section, change the web part description filename to **SPSTreeNav.dwp**.
9. Save and close the file.
10. Open the file SPSTreeNav.dwp for editing in Visual Studio .NET.
11. Change the file contents to appear as shown in Listing 4-13.
12. Save and close the file.

Listing 4-13. *The Web Part Description File*

```
<?xml version="1.0" encoding="utf-8"?>
<WebPart xmlns="http://schemas.microsoft.com/WebPart/v2" >
    <Title>Tree Navigation</Title>
    <Description>Collapsable Treeview Navigation control</Description>
    <Assembly>SPSTreeNav</Assembly>
    <TypeName>SPSTreeNav.Lister</TypeName>
    <FrameType>None</FrameType>
</WebPart>
```

Coding the Web Part

Once the project is started, you are ready to code the web part. The code for the web part is moderately complex primarily due to the recursion required to build the tree. Before you get started on the code, however, you will need to set some references and specify imported libraries.

Follow these steps to add references and libraries:

1. In Visual Studio .NET, select Project ➤ Add Reference from the menu.
2. In the Add Reference dialog, click the Browse button and locate the IEWebControls.dll assembly.

3. Click the OK button to add these references to the project.
4. Open `SPSTreeNav.vb` for editing in Visual Studio. NET.
5. Add the following `Imports` statements to the top of the file:

```
Imports Microsoft.SharePoint.WebControls
Imports Microsoft.SharePoint.Portal
Imports Microsoft.SharePoint.Portal.Topology
Imports Microsoft.SharePoint.Portal.SiteData
```

6. Change the name of the class to **Lister** and the default property to **WSSMode** so that your code looks like Listing 4-14.

Listing 4-14. *The Web Part Code*

```
Imports System
Imports System.ComponentModel
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.Xml.Serialization
Imports Microsoft.SharePoint
Imports Microsoft.SharePoint.Utilities
Imports Microsoft.SharePoint.WebPartPages
Imports Microsoft.Web.UI.WebControls
Imports Microsoft.SharePoint.Portal
Imports Microsoft.SharePoint.Portal.Topology
Imports Microsoft.SharePoint.Portal.SiteData
Imports System.Web
Imports Microsoft.SharePoint.WebControls

<DefaultProperty("WSSMode"), ToolboxData("<{0}:Lister _
runat=server></{0}:Lister>"), XmlRoot(Namespace:="SPSTreeNav")> _
Public Class Lister
    Inherits Microsoft.SharePoint.WebPartPages.WebPart

    Protected Overrides Sub RenderWebPart( _
    ByVal output As System.Web.UI.HtmlTextWriter)

    End Sub

End Class
```

Coding the Properties

The web part uses several properties to customize its look and behavior. The most important of these properties is the `WSSMode` property. This property determines what branch on the tree expands automatically. When set to `False`, the default, the tree expands to show the area that the end user is currently viewing. When set to `True`, the tree expands to show the area from which the user just navigated. The web part is set to `False` when placed on an area and `True` when placed on a WSS site.

The other properties of the web part control the background color and font style for the tree. These properties are useful in helping to match the look of the web part to the current site theme. There is also a property you can set to specify whether a new window is opened when a user clicks on a node in the tree. Add the code from Listing 4-15 to your web part to code the properties.

Listing 4-15. *The Web Part Properties*

```
'Member Variables
Private objTree As TreeView
Private lblMessages As Label
Private objCurrentWeb As SPWeb
Private m_WSSMode As Boolean = False
Private m_Red As Integer = 234
Private m_Green As Integer = 234
Private m_Blue As Integer = 234
Private m_FontName As String = "arial"
Private m_FontSize As String = "9pt"
Private m_NewWindow As Boolean = False

<Browsable(True), Category("Tree View"), DefaultValue(False), _
WebPartStorage(Storage.Shared), FriendlyName("WSSMode"), Description( _
"Sets the display mode of the tree.")> _
Property WSSMode() As Boolean
    Get
        Return m_WSSMode
    End Get

    Set(ByVal Value As Boolean)
        m_WSSMode = Value
    End Set
End Property

<Browsable(True), Category("Tree View"), DefaultValue(234), _
WebPartStorage(Storage.Shared), FriendlyName("Background Red"), Description( _
"Red portion of RGB background color.")> _
Property Red() As Integer
    Get
        Return m_Red
    End Get

    Set(ByVal Value As Integer)
        m_Red = Value
    End Set
End Property
```

```
<Browsable(True), Category("Tree View"), DefaultValue(234), _
WebPartStorage(Storage.Shared), FriendlyName("Background Green"), _
Description("Green portion of RGB background color.")> _
Property Green() As Integer
    Get
        Return m_Green
    End Get

    Set(ByVal Value As Integer)
        m_Green = Value
    End Set
End Property

<Browsable(True), Category("Tree View"), DefaultValue(234), _
WebPartStorage(Storage.Shared), FriendlyName("Background Blue"), _
Description("Blue portion of RGB background color.")> _
Property Blue() As Integer
    Get
        Return m_Blue
    End Get

    Set(ByVal Value As Integer)
        m_Blue = Value
    End Set
End Property

<Browsable(True), Category("Tree View"), DefaultValue("arial"), _
WebPartStorage(Storage.Shared), FriendlyName("Font Name"), Description( _
"Name of the display font.")> _
Property FontName() As String
    Get
        Return m_FontName
    End Get

    Set(ByVal Value As String)
        m_FontName = Value
    End Set
End Property

<Browsable(True), Category("Tree View"), DefaultValue("9pt"), _
WebPartStorage(Storage.Shared), FriendlyName("Font Size"), Description( _
"Size of the display font.")> _
Property FontSize() As String
    Get
        Return m_FontSize
    End Get
    Set(ByVal Value As String)
        m_FontSize = Value
    End Set
End Property

<Browsable(True), Category("Tree View"), DefaultValue(False), _
WebPartStorage(Storage.Shared), FriendlyName("New Window"), _
Description( _
"Determines iF a new window is opened when a listing link is clicked.")> _
Property NewWindow() As Boolean
    Get
        Return m_NewWindow
    End Get

    Set(ByVal Value As Boolean)
        m_NewWindow = Value
    End Set
End Property
```

Creating the Child Controls

Creating the set of child controls for the web part is straightforward. Along with the `TreeView`, the web part also contains a `Label` control, which is used to display any error messages that occur during processing. Add the code from Listing 4-16 to create the child controls.

Listing 4-16. *Creating the Child Controls*

```
Protected Overrides Sub CreateChildControls()
    'Add Tree
    objTree = New TreeView
    objTree.BackColor = System.Drawing.Color.FromArgb(Red, Green, Blue)
    objTree.DefaultStyle.Add("font-family", FontName)
    objTree.DefaultStyle.Add("font-size", FontSize)
    Controls.Add(objTree)

    'Add message label
    lblMessages = New Label
    Controls.Add(lblMessages)
End Sub
```

Adding Areas and Listings

Whenever the tree displays an area, it must also list the associated subareas and listings. Because we cannot predict how many listings and subareas will be underneath the current area, we must write a function that is called recursively until all of the child information is collected.

Along with building the tree, we also want to make a decision as to whether or not the branch we are building should be expanded. This process is based on the value of the `WSSMode` property as I described previously. Along with expanding the appropriate branch, the web part also makes the key nodes bold so that they stand out to the end user. Add the code from Listing 4-17 to build the areas and listings into the tree.

Listing 4-17. *Adding Areas and Listings*

```
Private Function AddSubAreasAndListings( _
ByVal objParentNode As TreeNode, ByVal objParentArea As Area) As Boolean

    Try

        Dim blnReturn As Boolean = False

        'Add the subareas under the parent area
        For Each objSubArea As Area In objParentArea.Areas
            If objSubArea.AreaTemplate = "SPSTOPIC" Then
                Dim objAreaNode As New TreeNode
                With objAreaNode
                    .Text = objSubArea.Title
                    .NavigateUrl = _
                    System.Web.HttpUtility.UrlPathEncode(objSubArea.WebUrl)
                    .ImageUrl = "/_layouts/images/cat.gif"
                    If WSSMode = False _
                    AndAlso Page.Request.Url.AbsolutePath.IndexOf( _
                    HttpUtility.UrlPathEncode(objSubArea.WebUrl)) = 0 Then
                        objAreaNode.Expanded = True
                        objAreaNode.DefaultStyle.Add("font-weight", "bold")
                        blnReturn = True
                    ElseIf WSSMode = True _
                    AndAlso Page.Request.UrlReferrer.AbsolutePath.IndexOf( _
                    HttpUtility.UrlPathEncode(objSubArea.WebUrl)) = 0 Then
                        objAreaNode.Expanded = True
                        objAreaNode.DefaultStyle.Add("font-weight", "bold")
                        blnReturn = True
                    End If

                End With
                objParentNode.Nodes.Add(objAreaNode)

                'Recursively add additional sub areas and listings
                If AddSubAreasAndListings(objAreaNode, objSubArea) = True Then
                    objAreaNode.Expanded = True
                    blnReturn = True
                End If
            End If
        Next
        'Add the portal listings for this area
        For Each objListing As AreaListing In objParentArea.Listings
            If objListing.Type = ListingType.Site _
            Or objListing.Type = ListingType.TeamSite Then
                Dim objListingNode As New TreeNode
                With objListingNode
                    .Text = objListing.Title
                    .NavigateUrl = objListing.URL
                    .ImageUrl = "/_layouts/images/link.gif"
                    If NewWindow = True Then .Target = "_BLANK"
                End With
                objParentNode.Nodes.Add(objListingNode)
            End If
        Next

        Return blnReturn

    Catch x As Exception
        lblMessages.Text += x.Message
    End Try

End Function
```

Add Sites and Subsites

Just like we want to add areas and listings to the tree, we also want to add sites and subsites. Unlike areas, however, sites are only added if the tree is in WSSMode. This means that the sites should not appear on an area page, but only when the tree is on a team site. Add the code from Listing 4-18 to put the sites and subsites in the tree.

Listing 4-18. *Adding Sites and Subsites*

```
Private Sub AddSubSites( _
ByVal objParentNode As TreeNode, ByVal objParentWeb As SPWeb)

    Try

        Dim objWebs As SPWebCollection = objParentWeb.GetSubwebsForCurrentUser()

        For Each objWeb As SPWeb In objWebs

            'Add Node
            Dim objNode As New TreeNode
            With objNode
                .Text = objWeb.Title
                .ImageUrl = "/_layouts/images/asp16.gif"
                .NavigateUrl = objWeb.Url
            End With
            objParentNode.Nodes.Add(objNode)
```

```
                'Recurse sub nodes
                AddSubSites(objNode, objWeb)

            Next

        Catch x As Exception
            lblMessages.Text += x.Message
        End Try

End Sub
```

Rendering the Web Part

Once all of the helper functions are coded, you are ready to render the web part. The tree is built by first creating the portal home as the root and then adding areas, listings, sites, and subsites as appropriate. Add the code from Listing 4-19 to render the web part.

Listing 4-19. *Rendering the Web Part*

```
Protected Overrides Sub RenderWebPart( _
ByVal output As System.Web.UI.HtmlTextWriter)
    Try

        'Get portal home Area
        Dim objPortalSite As New _
        SPSite(Page.Request.Url.GetLeftPart(UriPartial.Authority))
        Dim objContext As PortalContext = _
        PortalApplication.GetContext(objPortalSite.ID)
        Dim PortalGuid As Guid = _
        AreaManager.GetSystemAreaGuid(objContext, SystemArea.Home)
        Dim objPortalArea As Area = AreaManager.GetArea(objContext, PortalGuid)

        'Add the portal home Area to the tree
        Dim objPortalNode As New TreeNode
        With objPortalNode
            .Text = objPortalArea.Title
            .NavigateUrl = objPortalArea.WebUrl
            .ImageUrl = "/_layouts/images/sphomesm.gif"
            .Expanded = True
        End With
        objTree.Nodes.Add(objPortalNode)

        'Add subareas and listings
        If AddSubAreasAndListings(objPortalNode, objPortalArea) = True Then
            objPortalNode.Expanded = True
        End If
```

```
            'Add the WSS Site Collection to the tree
            If WSSMode = True Then

                'Get this web
                Dim objSite As SPSite = SPControl.GetContextSite(Context)
                Dim objSiteWeb As SPWeb = objSite.OpenWeb()

                'Add it to tree
                Dim objSiteNode As New TreeNode
                With objSiteNode
                    .Text = objSiteWeb.Title
                    .NavigateUrl = objSiteWeb.Url
                    .ImageUrl = "/_layouts/images/globe.gif"
                    .DefaultStyle.Add("font-weight", "bold")
                    .Expanded = True
                End With
                objPortalNode.Nodes.Add(objSiteNode)

                'Add subsites
                AddSubSites(objSiteNode, objSiteWeb)

            End If

        Catch x As Exception
            lblMessages.Text += x.Message
        End Try

        'Draw Controls
        objTree.RenderControl(output)
        lblMessages.RenderControl(output)

    End Sub
```

Deploying the Web Part

Deploying the web part requires a strongly named assembly. Therefore, you should edit the `AssemblyInfo` file to include a reference to a key pair file. Once this is done, you should be able to build the solution. Copy the built assembly to the `\Inetpub\wwwroot\bin` directory and edit the `web.config` file to mark the assembly as safe. These are steps you should be familiar with from other web part deployments, so I will not cover them in detail here. Once the assembly is deployed, you can use it on any page as normal, but if you want to deploy it in place of the standard navigation tree, then you must perform some special steps.

Follow these steps to replace the standard navigation tree:

1. In Microsoft FrontPage 2003, select File ➤ Open Site from the menu.
2. In the Open Site dialog, type the URL of the SPS home page and click the Open button.
3. When the site opens, select View ➤ Folders from the menu.

4. Each area in your SPS installation will appear as a folder in FrontPage. Select one of the areas and double-click it.
5. When the area folder opens, double-click the `default.aspx` file to edit the page.
6. In Design View, right-click the standard navigation tree and select Cut from the context menu. Be sure to completely remove all pieces of the standard navigation tree.
7. Select View ➤ Task Pane to open the FrontPage Task Pane.
8. In the Task Pane, select Web Parts from the drop-down menu.
9. Click the New Web Part Zone button located on the Task Pane.
10. Select File ➤ Save from the menu to save the changes.
11. Close Microsoft FrontPage.
12. Return to SPS and use the standard techniques to place the new tree navigation web part in the zone you created in FrontPage.

Exercise 4-2. Grouping List Items by Audience

Audience functionality is a great way to target content at end users. Unfortunately, audiences are associated with SPS and cannot normally be used with WSS sites. What's more, SPS usually limits audience functionality to targeting the entire web part. In this exercise, we'll overcome these limitations by creating a web part that can consume any list on a site and target the individual list items to selected audiences.

Starting the Project

This exercise makes use of a connected web part to consume a list on a WSS site and target the content. Because I covered connected web parts in detail in my first book, *Microsoft SharePoint: Building Office 2003 Solutions*, I will assume some familiarity with the concept and focus instead on the audience aspects of the project.

Follow these steps to start the project:

1. In Visual Studio .NET, select File ➤ New ➤ Project from the menu.
2. In the New Project dialog, select the Visual C# Projects folder in the Project Types list.
3. In the Templates list, select Web Part Library.
4. Name the project **SPSListByAudience** and click the OK button.
5. In the Solution Explorer, rename `WebPart1.cs` to **SPSListByAudience.cs**.
6. In the Solution Explorer, rename `WebPart1.dwp` to **SPSListByAudience.dwp**.
7. Open the file `Manifest.xml` for editing in Visual Studio .NET.
8. In the `DwpFiles` section, change the web part description filename to **SPSListByAudience.dwp**.

9. Save and close the file.
10. Open the file `SPSListByAudience.dwp` for editing in Visual Studio .NET.
11. Change the file contents to appear as shown in Listing 4-20.
12. Save and close the file.

Listing 4-20. *The Web Part Description File*

```
<?xml version="1.0" encoding="utf-8"?>
<WebPart xmlns="http://schemas.microsoft.com/WebPart/v2" >
    <Title>Display List By Audience</Title>
    <Description>Displays a connected list by Audience</Description>
    <Assembly>SPSListByAudience</Assembly>
    <TypeName>SPSListByAudience.Grouper</TypeName>
</WebPart>
```

Coding the Web Part

Once the project is started, you are ready to create the web part. The coding for this project is complex because of the functionality required to implement the web part connections. Before we start on the connections, however, you must set references to some key libraries.

Follow these steps to add references and libraries:

1. In Visual Studio .NET, select Project ➤ Add Reference from the menu.
2. In the Add Reference dialog, choose `Microsoft.SharePoint.Portal.dll` and `System.Data` from the Component Name list and click the Select button.
3. Click the OK button to add these references to the project.
4. Open `SPSListByAudience.cs` for editing in Visual Studio .NET.
5. Add the following `using` statements to the top of the file:

```
using Microsoft.SharePoint.Portal;
using Microsoft.SharePoint.Portal.Audience;
using Microsoft.SharePoint.WebPartPages.Communication;
using System.Security;
using System.Data;
using System.Collections;
```

6. Change the name of the class to **Grouper** and clear the default property so that your code looks like Listing 4-21.

Listing 4-21. *The Initial Web Part Code*

```
using Microsoft.SharePoint.Portal;
using Microsoft.SharePoint.Portal.Audience;
using Microsoft.SharePoint.WebPartPages.Communication;
using System.Security;
using System.Data;
using System.Collections;
using System;
using System.ComponentModel;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml.Serialization;
using Microsoft.SharePoint;
using Microsoft.SharePoint.Utilities;
using Microsoft.SharePoint.WebPartPages;

namespace SPSListByAudience
{

    [DefaultProperty(""),
        ToolboxData("<{0}:Grouper runat=server></{0}:Grouper>"),
        XmlRoot(Namespace="SPSListByAudience")]
    public class Grouper :
    Microsoft.SharePoint.WebPartPages.WebPart,
    Microsoft.SharePoint.WebPartPages.Communication.IListConsumer
    {
    }
}
```

Creating the Child Controls

This project uses an HTML table to render the consumed list grouped by audience; however, it also uses a simple `Label` control to display any error messages that might occur. Therefore, we must code a `CreateChildControls` function within the `Grouper` class. Add the code from Listing 4-22 to create the message label.

Listing 4-22. *Creating the Child Controls*

```
//member variables
protected DataTable list;
protected String [] displayNames;
protected Label messages;
protected Boolean isConnected;
protected int colSpan;

protected override void CreateChildControls()
{
    messages = new Label();
    Controls.Add(messages);
}
```

Coding the Web Part Connection

The `Grouper` class implements the `IListConsumer` interface so that it can connect to any list on a WSS site. Therefore, we must code the appropriate connection life cycle for the web part. As I stated earlier, I am not going to focus on the mechanics of connecting web parts because that is beyond the scope of this chapter. The most important thing to know about this code is that the consumed list is passed to the `Grouper` class when the `ListReady` function is called by the web part infrastructure. Add the code from Listing 4-23 to implement the web part connection.

Listing 4-23. *Connecting the Web Part to a List*

```
public override void EnsureInterfaces()
{
    try
    {
       RegisterInterface("AudienceGrouper",
       "IListConsumer",
       WebPart.LimitOneConnection,
       ConnectionRunAt.Server,
       this,"","Get a list from...",
       "Receives a list for grouping.");
    }

    catch(SecurityException e)
    {
        messages.Text += "<p>" + e.Message + "</p>";
    }
}

public override ConnectionRunAt CanRunAt()
{
    return ConnectionRunAt.Server;
}

public override void PartCommunicationConnect(string interfaceName,
    WebPart connectedPart, string connectedInterfaceName, ConnectionRunAt runAt)
{
    if(runAt==ConnectionRunAt.Server)
    {
        EnsureChildControls();
        if(interfaceName=="AudienceGrouper")isConnected=true;
    }
}
```

```
public void ListProviderInit(object sender,
ListProviderInitEventArgs listProviderInitEventArgs)
{
    //Get the display names of the fields
    displayNames = listProviderInitEventArgs.FieldDisplayList;
    colSpan = displayNames.GetLength(0);
}

public void PartialListReady(object sender,
PartialListReadyEventArgs partialListReadyEventArgs)
{
    // Nothing to do
}

public void ListReady(object sender, ListReadyEventArgs listReadyEventArgs)
{
    list = listReadyEventArgs.List;
}
```

Rendering the Web Part

Once the list has been passed to the `Grouper` class, we are free to render it in any way that we want. Our web part assumes that one of the fields passed in from the consumed list is named "Audience". Our web part uses this value to filter and group the individual list items by audience. This technique is better than creating a separate list view based on the Audience field because we will only show list items that correspond to the audiences associated with the current user. This is a nice way to simplify lists and target the individual items to the appropriate users. Add the code in Listing 4-24 to render the consumed list.

Listing 4-24. *Rendering the List*

```
protected override void RenderWebPart(HtmlTextWriter output)
{
    try
    {
        if(isConnected==true)
        {

            //Write out the column display names
            output.Write("<table border=0 width=\"100%\">");
            output.Write("  <tr>");
            for(int i=displayNames.GetLowerBound(0);
            i<=displayNames.GetUpperBound(0);i++)
            {
                if(displayNames[i]!="Audience")
                output.Write("      <th>" + displayNames[i] + "</th>");
            }
```

```
//Get the portal context
SPSite portal =
new SPSite(Page.Request.Url.GetLeftPart(UriPartial.Authority));
PortalContext context = PortalApplication.GetContext(portal.ID);

//Get the list of audiences for the user
AudienceManager manager = new AudienceManager(context);
ArrayList audienceIDList = manager.GetUserAudienceIDs();

if(audienceIDList != null)
{
    IEnumerator audienceNameIDs = audienceIDList.GetEnumerator();

    while(audienceNameIDs.MoveNext())
    {

        //Get the set of items for this audience
        String audienceName =
            ((AudienceNameID)audienceNameIDs.Current).AudienceName;
        DataView dataView = new DataView(list,
            "Audience='" + audienceName + "'",
            "",DataViewRowState.CurrentRows);

        if (dataView.Count!=0)
        {
            //Write out the audience name
            output.Write(
                "   <tr><td class=\"ms-sectionheader\" colspan=\""
                + colSpan.ToString() + "\">" + audienceName
                + "</td></tr>");

            //Write out the rows
            foreach (DataRowView rowView in dataView)
            {
                output.Write("   <tr>");

                //Write out columns
                IEnumerator columns
                     = rowView.Row.ItemArray.GetEnumerator();
                 while(columns.MoveNext())
                 {
                     if(columns.Current.ToString()!=audienceName)
                     output.Write("      <td>"
                         + columns.Current.ToString() + "</td>");
                 }
```

```
                            output.Write("   </tr>");
                        }

                    }
                }
            }

            //close the table
            output.Write("</table>");
        }
        else
        {
            messages.Text += "<p>Connect this web part to a list.</p>";
        }

        //Show messages
        messages.RenderControl(output);

    }
    catch(Exception x)
    {
        output.Write("<p>" + x.Message + "</p>");
    }
}
```

Deploying the Web Part

Follow all the normal steps necessary to build and deploy the web part. Once it's deployed, you can add it to a page. If successful, the web part should display a message asking you to connect it to a list. You can connect the web part to any list that is displayed on the same page. In order to work successfully, the connected list must have a custom field named "Audience". Figure 4-12 shows an example of the web part connected to a standard document library.

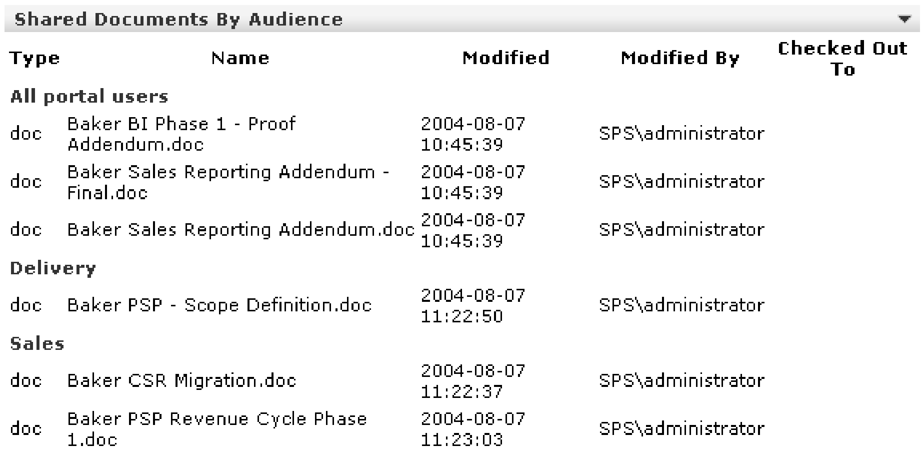
Follow these steps to connect a list:

1. Navigate to any WSS site as a member of the Administrators Site Group.
2. On the site, click the Create link.
3. On the Create page, select the option to create a new Document Library.
4. Follow the steps necessary to create the library. When complete, click the Modify Settings and Columns link.
5. On the Customize page, select to add a new column named **Audience**.
6. Make the new column a single line of text with a default value of **All portal users**, and make the column part of the default view.
7. Add several documents to the library.

8. Return to the site home page and add the new Document Library to the home page as a web part.

9. Select a view for the web part that contains the new Audience column.

10. Connect the new Document Library to the `Grouper` web part.



Type	Name	Modified	Modified By	Checked Out To
All portal users				
doc	Baker BI Phase 1 - Proof Addendum.doc	2004-08-07 10:45:39	SPS\administrator	
doc	Baker Sales Reporting Addendum - Final.doc	2004-08-07 10:45:39	SPS\administrator	
doc	Baker Sales Reporting Addendum.doc	2004-08-07 10:45:39	SPS\administrator	
Delivery				
doc	Baker PSP - Scope Definition.doc	2004-08-07 11:22:50	SPS\administrator	
Sales				
doc	Baker CSR Migration.doc	2004-08-07 11:22:37	SPS\administrator	
doc	Baker PSP Revenue Cycle Phase 1.doc	2004-08-07 11:23:03	SPS\administrator	

Figure 4-12. *The completed project*