Oracle® WebCenter Framework Interaction

UI Customization Guide,

Release 10*g*

September 2008

The *Oracle WebCenter Framework Interaction UI Customization Guide* provides detailed instructions for extending and customizing the WebCenter Framework Interaction user interface.

ORACLE®

Oracle WebCenter Interaction UI Customization Guide, 10.3.

Primary Author: Jennifer Shipman

Contributing Author:

Contributor:

Contents

3 Using Adaptive Page Layouts

4 Using Adaptive Styles (CSS Customization)

5 Customizing Portal Layout Using CSS - Legacy User Interface

12 Using PEIs

13 Using View Replacement

14 Creating Custom Activity Spaces

Preface

Audience

This document is intended for administrators and developers responsible for customizing the Oracle WebCenter Interaction user interface.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, 7 days a week. For TTY support, call 800.446.2398. Outside the United States, call +1.407.458.2479.

Related Documents

For more information, see the following documents in the Oracle WebCenter Interaction 10.3 documentation set or the Oracle WebCenter Interaction Development Kit (IDK)10.3 documentation set:

- *Administrator Guide for Oracle WebCenter Interaction*

- *Oracle WebCenter Interaction Web Service Development Guide*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Introduction to UI Customization

Oracle WebCenter Interaction provides built-in customization tools that allow you to create a portal that fits the needs of all your company's users. Using the frameworks and tools provided ensures that your customizations can be retained during future upgrades. Most customizations require no custom Java or C# code. This chapter provides an overview of customization options.

For an introduction to the portal UI, see Chapter 2, "Portal Page Layout".

1.1 Customizing Portal Look and Feel

The portal UI is designed for customization. The portal includes a range of built-in solutions for customizing look and feel. These customizations are covered in Part I, "Customizing Portal Look and Feel".

1.1.1 Adding Logo and Branding

The header and footer portlets displayed on most portal pages usually contain the company logo and contact information. For details on building custom portlets, see the *Oracle WebCenter Interaction Web Service Development Guide*. Adaptive page layouts allow you to change the look and feel of the portal user interface using adaptive tags in standard XHTML For details, see Chapter 3, "Using Adaptive Page Layouts".

1.1.2 Modifying Portal Style Sheets

Oracle WebCenter Interaction style sheets are fully customizable. The portal comes with a selection of different options to change the style of portal pages, including a range of color schemes, fonts, and other options. The default style sheets used for each experience definition can be modified using portal administration. For details on using CSS customization with Adaptive Layouts, see Chapter 4, "Using Adaptive Styles (CSS Customization)". You can also use CSS customization with the legacy layouts; for details, see Chapter 5, "Customizing Portal Layout Using CSS - Legacy User Interface".

1.1.3 Customizing Page Layout and Design

Adaptive Page Layouts allow you to customize the entire portal page layout and design using tags . For details, see Chapter 3, "Using Adaptive Page Layouts". The portal CSS template file also allows you to customize the layout of the portal page, including columns, navigation tabs, banners and footers. You can modify the look and feel of individual table controls and form elements, including text box sizing, button colors and fonts. You can also use style sheets to customize portlet content and style. For details, see Chapter 4, "Using Adaptive Styles (CSS Customization)".

1.1.4 Changing Portal Text

All messages displayed in the portal can be customized easily by modifying the portal string files. This is a simple customization that is often overlooked in favor of more complicated methods. All text in the portal is stored in internationalized string files, including login instructions and error messages, with the exception of object names and text generated by portlets. For details and instructions, see Chapter 6, "Using String Replacement".

1.1.5 Creating Customized Experiences for Specific Groups

Experience definitions allow the portal to use different branding for different groups of users, including departments, product teams, or specific customers. By creating multiple experience definitions and communities, you can create focused pages and experiences for distinct groups of portal users. For an introduction to experience definitions, see the *Administrator Guide for Oracle WebCenter Interaction*. For more information on customizing experience definitions, see Chapter 7, "Customizing Experience Definitions".

1.2 Customizing Portal Functionality

Oracle WebCenter Interaction supports customizing and extending all aspects of portal functionality. The most common options are detailed below. These customizations are covered in Part II, "Customizing Portal Functionality".

1.2.1 Customizing Portal Login

The portal login page can be customized for different groups of users. A common customization is to provide different branding on the login page based on the URL used to access the portal. This allows you to provide each group of users with a seamlessly branded portal, including pages viewed as the guest user. This can be implemented easily using Experience Definitions. You can also create a custom login page using Adaptive Layouts. For information, see Chapter 8, "Customizing Portal Login".

1.2.2 Modifying Portal Navigation

Navigation is a key element of the portal page. Experience definitions allow you to add custom links to the navigation pane that point to community pages, documents, and web pages without writing any code. Adaptive Layouts allow you to define the navigation section of the page using tags. For details, see Chapter 9, "Customizing Portal Navigation". You can also use adaptive tags to quickly and easily create a custom navigation scheme in a header for footer portlet. For details, see the *Oracle WebCenter Interaction Web Service Development Guide*.

1.2.3 Adding New Functionality to Portal Pages

The most common way to add functionality to a page is to implement custom portlets. Basic portlets allow you to display custom HTML and content from other applications. You can also use portlets to access portal components, and build portlets that are updated dynamically based on user action and other events. For more information on portlet development, see the*Oracle WebCenter Interaction Web Service Development Guide*.

You can also add functionality to portal components using advanced customizations. For details, see Part III, "Advanced UI Customization".

1.2.4 Customizing and Extending Search

Oracle WebCenter Interaction search indexes and searches all the documents, information, applications, communities, discussions, web sites and other content accessible through the portal. You can customize how search is implemented in the portal, and extend search to include enterprise content. For details, see Chapter 10, "Customizing Portal Search".

1.3 Advanced UI Customization

The basic customizations listed above require little or no custom code. If these options do not provide a solution, you can replace portal components with custom versions. The advanced customizations below require Java or C# coding. These customizations are covered in Part III, "Advanced UI Customization".

For an introduction to the inner workings of the portal UI, see Chapter 11, "Portal UI Architecture".

1.3.1 Adding Functionality Using PEIs

Portal Event Interfaces (PEIs) are used to execute custom actions in many places throughout the portal. For example, you can modify search queries before they are processed, or perform validation when users attempt to create new portal objects. A common PEI implementation is to require users to accept a usage agreement before being allowed to access the portal. For more information, see Chapter 12, "Using PEIs".

1.3.2 Customizing Pages Using View Replacement

You can completely customize the display of portal components by creating a custom version of the associated View class(es). For details, see Chapter 13, "Using View Replacement".

1.3.3 Adding New Pages Using Custom Activity Spaces

Activity Spaces group task-specific actions into logical sets to provide portal developers with base functionality, and combine related pages to create cohesive Model-View-Control (MVC) objects. Everything in the portal is an Activity Space: a MyPage, an administrative editor, even the Directory tree. A custom Activity Space allows you to add new pages to your portal. For details, see Chapter 14, "Creating Custom Activity Spaces". (To change existing code or add new components to existing pages, use View Replacement.)

1.3.4 Using Advanced UI Customization Tools and Components

Oracle WebCenter Interaction includes a collection of useful tools and components to support UI customization. For details, see Chapter 15, "Accessing Portal Objects", Chapter 16, "Adding Custom Images", and Chapter 17, "Using VarPacks (Variable Packages)".

1.4 Internationalizing UI Customizations

Oracle WebCenter Interaction is available in a wide variety of languages. The topics that follow offer step-by-step instructions for internationalizing your web services and portal customizations to make them available to all audiences.

- **Using String Replacement**: All text in the portal is stored in internationalized string files, including login instructions and error messages, with the exception of object names and text generated by portlets. For details, see Chapter 6, "Using String Replacement".
- **Adding Language Style Sheets**: If you add support for an additional language to the portal, you must add the corresponding style sheets for that language. For details, see Chapter 4.10, "Implementing Localized Stylesheets for Adaptive Page Layouts". If you are not using Adaptive Page Layouts, see Chapter 5.3, "Adding New Language Style Sheets".

1.5 Reference Material

For additional resources related to UI customization, see Part IV, "Appendices and Additional References".

Part I

Customizing Portal Look and Feel

The portal UI is designed for customization. The portal includes a range of built-in solutions for customizing look and feel. For an introduction to the layout of the portal page, see Chapter 2, "Portal Page Layout".

If you just want to add your logo and branding to the portal, you can use header and Footer portlets are displayed on most portal pages, and usually contain the company logo and contact information. For details on building custom portlets, see the *Oracle WebCenter Interaction Web Service Development Guide*.

This section contains the following chapters:

- Chapter 3, "Using Adaptive Page Layouts": Adaptive Page Layouts allow you to customize the entire portal page layout and design using tags .
- Chapter 4, "Using Adaptive Styles (CSS Customization)": Oracle WebCenter Interaction style sheets are fully customizable. The portal comes with a selection of different options to change the style of portal pages, including a range of color schemes, fonts, and other options. The portal CSS template file also allows you to customize the layout of the portal page, including columns, navigation tabs, banners and footers. You can modify the look and feel of individual table controls and form elements, including text box sizing, button colors and fonts. You can also use style sheets to customize portlet content and style. **Note**: If you are not using Adaptive Layouts, you can still use CSS to customizing the portal page; for details, see Chapter 5, "Customizing Portal Layout Using CSS - Legacy User Interface"
- Chapter 6, "Using String Replacement": All messages displayed in the portal can be customized easily by modifying the portal string files. This is a simple customization that is often overlooked in favor of more complicated methods. All text in the portal is stored in internationalized string files, including login instructions and error messages, with the exception of object names and text generated by portlets.
- Chapter 7, "Customizing Experience Definitions": Experience definitions allow the portal to use different branding for different groups of users, including departments, product teams, or specific customers. By creating multiple experience definitions and communities, you can create focused pages and experiences for distinct groups of portal users.

2

Portal Page Layout

The portal page is made up of sections. This chapter provides an overview of the portal page and information about customizing each section of the page.

2.1 Top Bar

The Top Bar includes the Search box, Log In/Log Off link and help link.

Some elements of the Top Bar can be customized by changing style sheets and modifying the associated strings in the portal language files. You can design a completely new Top Bar using Adaptive Tags and a custom portlet. For details on using Adaptive Layouts and Adaptive Styles, see Chapter 3, "Using Adaptive Page Layouts" and Chapter 4, "Using Adaptive Styles (CSS Customization)". For details on creating portlets, see the *Oracle WebCenter Interaction Web Service Development Guide*.

2.2 Header and Footer

The header includes branding information for the portal and can also be used to display content.

The footer provides additional content and can be customized to include additional functionality.

These sections of the portal page can be customized using portlets and Adaptive Layouts. For details on using Adaptive Layouts, see Chapter 3, "Using Adaptive Page Layouts". For details on creating portlets, see the *Oracle WebCenter Interaction Web Service Development Guide*.

2.3 Navigation

The navigation section of the page provides access to the different sections of the portal and the current community. The portal comes with a selection of built-in navigation schemes. For details, see the *Administrator Guide for Oracle WebCenter Interaction*. Custom navigations can be built easily using Adaptive Tags and Adaptive Layouts. For details, see Chapter 3, "Using Adaptive Page Layouts" and Chapter 4, "Using Adaptive Styles (CSS Customization)". Advanced navigation customizations can be implemented using the portal navigation framework. For details, see Chapter 9, "Customizing Portal Navigation".

2.4 Banner

The banner includes the top bar, header, and navigation sections of the page.

The entire banner can be easily customized by disabling the navigation and top bar sections and using a header portlet with Adaptive Tags to display all banner content. For details, see the *Oracle WebCenter Interaction Web Service Development Guide*.

2.5 Body

The body is the main section of the portal page, and displays the portlets selected for the page.

Portlets are the building blocks for the body of the portal page. Each portal page is made up of multiple portlets with a range of functionality, each providing specific content and services. The Oracle WebCenter Interaction Development Kit (IDK) provides a wide range of tools for creating dynamic portlets that plug in to the portal. For details, see the*Oracle WebCenter Interaction Web Service Development Guide*.

The body section can be split into multiple panes in a variety of layouts, configured in the My Page or Community Editor. For instructions, see the portal online help. You can also create custom page layouts using Adaptive Layouts. For details, see Chapter 3, "Using Adaptive Page Layouts".

For very advanced customizations to the layout of the portal page, you can use View replacement (not recommended for most customizations). For details, see Chapter 13, "Using View Replacement".

3

Using Adaptive Page Layouts

Adaptive page layouts allow you to change the look and feel of the portal user interface using adaptive tags in standard XHTML

Adaptive page layouts can be used in most areas of the portal. Each page layout type has an associated tag library. The tags in each library only work in the related page layout. The base page library and the standard adaptive tag libraries (pt:common, pt:core, pt:logic, pt:ptdata, pt:ptui, pt:standard and pt:transformer) work on any page with a portal banner and are used in all page layouts.

This chapter provides detailed information on customizing each of the page layout types. For additional information, see the following resources:

- For detailed information on using adaptive tags, see the*Oracle WebCenter Interaction Web Service Development Guide*
- For a complete list of tags, see the tagdocs. (For links to all API documentation, see Appendix C, "Portal API Documentation".)
- For details on configuring adaptive page layouts in the portal, see the *Administrator Guide for Oracle WebCenter Interaction*
- For complete examples of each page layout type, see the default page layouts on the Oracle WebCenter Interaction image service in the \imageserver\plumtree\portal\private\pagelayouts folder.

3.1 Available Adaptive Page Layouts

The following adaptive page layouts are available in Oracle WebCenter Interaction. For more information on customizing these layouts, see the sections that follow.

Page Layout Type	Description	Library
Base Page	Controls the layout for components that are common to each page (header, footer, navigation).	pt:basepage
Profile Page	Controls the layout for components that are common to each user profile page.	pt:basepage
Login Page	Controls the layout of the login page.	pt:ptui
Portlet Page	Controls the layout for the portlet area of the portal page.	pt:portletpage
Knowledge Directory	Controls the layout for the content area of the Directory.	pt:kdpage
Search Results	Controls the layout for the content area of search results.	pt:searchpage

Page Layout Type	Description	Library
Portlet Selection	Controls the layout of the pop-up or fly-out editor used to select portlets on a portlet page. Note: In most implementations, there is no reason to modify this page layout.	pt:portletpageeditor
Join Communities	Controls the layout of the pop-up or fly-out editor used to join communities on a portal page.	pt:joincommunitypageeditor
My Account Page	Controls the layout of the My Account page in the portal.	pt:ptui
Error Page	Controls tha layout of the portal error page. These tags can also be used on any portal page to display standard errors.	pt:ptui
iPhone	Controls the layout of portal pages delivered for the iPhone interface.	(standard libraries)

3.2 Creating a Base Page Adaptive Page Layout

The Base Page layout defines components that are common to each page (header, footer, navigation, content area). The Profile Page layout is almost identical, but uses a user profile search box instead of the standard search box.

The pt:basepage library includes tags to display all common sections of the portal page.

- The `<title>` tag displays the title of the page.
- The `<pt:basepage.pagebody>` tag displays the html body tag and initializes the JavaScript required by the page.
- The `<pt:basepage.navarea>` tag is used to display legacy portal navigation components in the top bar. The `pt:area` parameter defines which part of the navigation is being defined by each section (ABOVEHEADER, BELOWHEADER, ABOVEBODY, LEFTOFBODY, RIGHTOFBODY, BELOWBODY, ABOVEFOOTER, BELOWFOOTER, and TOPBAR). You can also create a customized display of links to portal components using tags from the pt:ptdata library. For details on these tags, see the *Oracle WebCenter Interaction Web Service Development Guide*
- The `<pt:basepage.content>` tag defines the section where the main content of the page is displayed.
- The `<pt:basepage.header>` and `<pt:basepage.footer>` tags define the sections of the page where the header and footer are displayed. The header and footer are implemented in separate HTML files. For examples, see the header.html and footer.html files in the \imageserver\plumtree\portal\private\pagelayouts folder on your portal image service.

The example below uses tags from the pt:basepage library to define common page components, and standard adaptive tags to implement portal links and handle logic. For detailed information on standard adaptive tags, including logic tags and data tags for implementing portal links , see the *Oracle WebCenter Interaction Web Service Development Guide*.

Note: This example has been oversimplified for illustration purposes; for a complete implementation of the base page layout, see the basepagelayout.html file in the \imageserver\plumtree\portal\private\pagelayouts folder on your portal image service.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:pt="http://www.plumtree.com/xmlschemas/ptui/">
```

```
<head>
<title><pt:basepage.title/></title>
<pt:standard.stylesheets/>
<link href="pt://images/plumtree/portal/private/css/mainstyle.css"
rel="stylesheet" type="text/css" />

<!-- Dojo Initialization, create a custom namespace -->
<script type="text/javascript">
        var djConfig = {
            isDebug: false,
            scopeMap :  [ ["dojo", "alidojo"] ]
         };
</script>

<!-- Dojo is used by drag and drop and flyout editor -->
<script type="text/javascript"
src="pt://images/plumtree/portal/private/js/dojo.js"></script>

<!-- This tag displays the html body tag and initializes page JavaScript. -->
<pt:basepage.pagebody marginwidth="0" marginheight="0" topmargin="0">
<div id="ali-header-nav">

<!-- This area is used to build links to portal components displayed in the top
bar -->
<pt:basepage.navarea pt:area="TOPBAR"/>

... banner actions ....

<pt:basepage.navarea pt:area="ABOVEHEADER"/>
<pt:basepage.header/>

<!-- This area is used to build links to portal navigation elements -->
<pt:basepage.navarea pt:area="BELOWHEADER"/>

... navigation links ...

  <pt:basepage.navarea pt:area="LEFTOFBODY"/>
</div>
<div style="float:right; width:200px;" >
  <pt:basepage.navarea pt:area="RIGHTOFBODY"/>
</div>
<pt:basepage.navarea pt:area="ABOVEBODY"/>
<pt:common.error/>
<pt:basepage.content/>
<pt:basepage.navarea pt:area="BELOWBODY"/>
<pt:basepage.navarea pt:area="ABOVEFOOTER"/>
<pt:basepage.footer/>
<pt:basepage.navarea pt:area="BELOWFOOTER"/>
</pt:basepage.pagebody>
</html>
```

3.3 Creating a Login Page Adaptive Page Layout

Login Page layouts allow you to customize the layout of the portal login page. The outer area (header, footer, navigation areas) is based on the Base Page adaptive layout. Use this layout to customize the entire login page (with no portlets); use the Portal Login portlet if you want users to have access to the portal as a guest.

The pt:ptui library includes tags to define all necessary login elements.

All login tags must be contained within a <pt:ptui.loginform> tag. The following tags are used to implement specific elements within the login form:

Tag	Description
ptui:loginusername	Displays the user name text box for the login form.
ptui:loginpassword	Displays the password text box for the login form.
ptui:loginauthsource	Displays the Authentication Source dropdown list. (If AllowDefaultLoginPageAuthsource in portalconfig.xml is set to 3, this tag displays nothing.)
ptui:loginrememberme	Displayts the 'Remember My Password' checkbox. This tag works in two ways: ■ If used without the key attribute, it displays the 'Remember My Password' checkbox. ■ If the 'key' optional attribute is given a value, the tag outputs the name of the inputcontrol as the key parameter's value and displays the body of the html inside it. This tag displays the above outputs only when 'AllowAutoConnect' in portalconfig.xml is enabled.
ptui:createaccount	Displays a "Create Account" link. If this tag is used as a singleton tag, the text "Create Account" will be used. If opening and closing tags are used, the HTML inside the tag will be displayed as the link. (If the Allow Creation of Self Registered Users option in the Portal Settings section of portal Administration is not enabled, this tag will display nothing.)
ptui:loginoptionsenabled	Optional. Conditionally processes inner content based on the option parameter and relevant portal settings. The following values are allowed in the option attribute: authsource, remembermypassword, and createaccount. For example, if the option attribute is set to remembermypassword and the portal setting for remembermypassword is true, then the HTML inside the tag will be displayed.
ptui:loginbutton	Displays the login button.

3.4 Creating a Portlet Adaptive Page Layout

Portlet Page layouts allow you to customize the layout of the portlet header as well as how individual portlets are displayed. The outer area (header, footer, navigation areas) is based on the Base Page adaptive layout.

The pt:portletpage library includes tags to define all areas of the portlet section of the portal page.

- All portlets on the page must be within a <pt:portletpage.portletregiondisplay> tag. This tag defines a container for portlets that specifies where and how portlets inside the region are displayed.
- The <pt:portletpage.portletdisplay> tag displays an entire portlet (header and content), while the <pt:portletpage.portletdisplaycontent> tag displays the content of the portlet without the header.
- Portlet data is accessed via the <pt:portletpage.portletregiondata> tag. The pt:region parameter can be set to any value, as long as it corresponds with

a portletregiondisplay section in the page. The following properties are available for each portlet from the `portletregiondata` tag:

Property	Description
name	The name of the portlet
objid	The ID of the portlet object in the portal.
index	The index of the portlet in the portlet array.
portletidstring	The string identifier of the portlet.
adminprefurl	A link to the associated administrative preference page for the portlet if one exists.
commprefurl	A link to the associated community preference page for the portlet if one exists.
userprefurl	A link to the associated user preference page for the portlet if one exists.
helpurl	A link to the associated help page for the portlet if one exists.
collapseexpandurl	A link for collapsing or expanding the portlet depending on the collapse state of the portlet
iscollapsed	True if the portlet is collapsed, false if it is expanded.
removeurl	A link to remove the portlet from the current page.
hastitlebar	True if portlet title bar is not suppressed, false otherwise.

- The `<pt:portletpage.pagenamebreadcrumbsdata>` tag can be used on MyPages and community pages. This tag contains an ordered list representing the path leading up to the current page. The list contains the name and URL that makes up each breadcrumb.
  - On a MyPage, the breadcrumb will display a "Home" link as the parent that leads back to the user's main page, followed by the name of the MyPage that the user is on.
  - In communities, the breadcrumb will display all the communities leading up to the page. The breadcrumbs will all be hyperlinks except for the last breadcrumb which represents the current page that the user is on.

Additional tags in the pt:portletpage library can be used to display buttons with access to portlet-specific functionality, including refresh, remove, and collapse/expand. The example below uses tags from the pt:portletpage library to define portlet page components, and standard adaptive tags to implement portal links and navigation and handle logic. For detailed information on standard adaptive tags, including logic tags and data tags for implementing portal links , see the *Oracle WebCenter Interaction Web Service Development Guide*.

This example uses the `<pt:portletpageeditor.addportletsflyoutdata>` tag to add the portlet flyout editor to the portlet page. For details, see Section 3.7, "Creating a Portlet Selection Adaptive Page Layout".

Note: This example has been oversimplified for illustration purposes; for a complete implementation of the portlet page layout, see the portletdefaultlayout.html file in the \imageserver\plumtree\portal\private\pagelayouts folder on your portal image service.

```
<!-- Portlet Content Area Begin -->
<span xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
```

```
<pt:portletpage.portletregiondata pt:key="region1" pt:region="1" />
<pt:portletpage.portletregiondata pt:key="region2" pt:region="2" />
<pt:portletpage.portletregiondata pt:key="region3" pt:region="3" />

... breadcrumb display ...

<!-- Start Page Action Buttons -->
<pt:core.comment><!-- get the Portlet Selection Flyout URL --></pt:core.comment>
<pt:portletpageeditor.sortpropertiesdata pt:id="sortprops" pt:scope="session"
pt:defaultsort="1"/>

<pt:portletpageeditor.addportletsflyoutdata pt:id="flyoutLink"
pt:sortprops="sortprops" pt:propscope="session" />
<pt:logic.existexpr pt:data="flyoutLink" pt:key="hasFlyout"/>
<pt:logic.if pt:expr="$hasFlyout">
    <pt:logic.iftrue>
      <pt:portletpageeditor.flyoutjs pt:flyoutID="portletSelection"
pt:onclick="openFlyout" pt:headerId="ali-header-nav" pt:url="$flyoutLink.url"
pt:anchorId="ali-pageEdit-anchor" pt:flyoutAnchorText="$#130.ptmsgs_
portalinfrastructure" pt:flyinAnchorText="$#301.ptmsgs_portalcommonmsgs"/>
    </pt:logic.iftrue>
</pt:logic.if>

<pt:core.comment><!-- add javascript for collapsing and expanding portlets
--></pt:core.comment>
<pt:portletpageeditor.collapseexpandjs pt:onclick="sendCollapseExpandRequest"
pt:flyoutID="portletSelection1" />

... page action implementation ....

<table width="100%" height="100%" style="clear:left;">
    <tr>
        <td class="columnOne" valign="top">
  <!-- Vertical Region One -->
  <pt:portletpage.portletregiondisplay class="portletRegion" pt:region="1"
pt:direction="v">
    <pt:logic.foreach pt:data="region1" pt:var="curr">
      <pt:logic.variable pt:key="titleBarData" pt:value="$curr.hastitlebar"/>
      <pt:logic.stringexpr pt:expr="($titleBarData) == false"
pt:key="suppressTitleBar" />

      <pt:logic.variable pt:key="portletDivStyle" pt:value="ali-portlet-regionone
"/>
      <pt:logic.if pt:expr="$curr.ismandatory">
      <pt:logic.iffalse>
        <pt:logic.concat pt:key="portletDivStyle" pt:value1="$portletDivStyle"
pt:value2="dndPortlet"/>
      </pt:logic.iffalse>
      </pt:logic.if>

      <pt:core.html pt:tag="div" class="$portletDivStyle"
id="$curr.portletidstring">


      <pt:logic.if pt:expr="$curr.iscollapsed"><pt:logic.iffalse>
        <pt:logic.variable pt:key="containerclass"
pt:value="ali-portlet-container"/>
      </pt:logic.iffalse><pt:logic.iftrue>
        <pt:logic.variable pt:key="containerclass"
```

```
pt:value="ali-portlet-container-collapsed"/>
      </pt:logic.iftrue></pt:logic.if>
        <pt:core.html pt:tag="div" class="$containerclass">
        <pt:logic.if pt:expr="$suppressTitleBar">
        <pt:logic.iftrue>
          <!-- Suppress portlet toolbar -->
          <div class="ali-portlet-toolbar">
            <div class="ali-portlet-cornerleft"></div>
            <div class="ali-portlet-cornerright"></div>
          </div>
        </pt:logic.iftrue>
        <pt:logic.iffalse>
        <!-- Display Portlet Header -->
        <div class="ali-portlet-toolbar">
          <div class="ali-portlet-cornerleft"></div>
          <div class="ali-portlet-cornerright"></div>
          <div
class="ali-portlet-controlone"><pt:portletpage.portletremovebuttondisplay
pt:datavar="curr" pt:scope="tag"/></div>
          <div
class="ali-portlet-controlone"><pt:portletpage.portletcollapseexpandbuttondisplay
pt:datavar="curr" pt:onclick="sendCollapseExpandRequest" pt:scope="tag"/></div>
          <div
class="ali-portlet-controlone"><pt:portletpage.portletpreferencebuttondisplay
pt:datavar="curr" pt:scope="tag"/></div>
          <div
class="ali-portlet-controlone"><pt:portletpage.portlethelpbuttondisplay
pt:datavar="curr" pt:scope="tag"/></div>
          <div
class="ali-portlet-controlone"><pt:portletpage.portletrefreshbuttondisplay
pt:datavar="curr" pt:scope="tag"/></div>
          <div class="ali-portlet-title"><pt:logic.value
pt:value="$curr.name"/></div>
        </div>
        </pt:logic.iffalse>
        </pt:logic.if>
        <!-- Display Portlet Content Body -->
        <div class="ali-portlet-content">
          <pt:portletpage.portletcontentdisplay pt:datavar="curr" pt:colindex="0"
pt:scope="tag"/>
        </div>
        <div class="ali-portlet-footer">
          <div class="ali-portlet-botleft"></div>
          <div class="ali-portlet-botright"></div>
        </div>
      </pt:core.html>
      </pt:core.html>
    </pt:logic.foreach>
  </pt:portletpage.portletregiondisplay>
    </td>
    <td class="columnTwo" valign="top">

... portlet regions 2 and 3 ....
      </td>
    </tr>
</table>

</span>
```

3.5 Creating a Knowledge Directory Adaptive Page Layout

The Knowledge Directory adaptive page layout defines the content area of the Directory. The outer area (header, footer, navigation areas) is based on the Base Page adaptive layout.

All content in the Directory can be displayed using tags in the pt:kdpage library. The tags in this library can only be used in the Knowledge Directory page layout.

- Data about the folders in the Directory can be accessed using the `<pt:kdpage.currentfolderdata>` and `<pt:kdpage.subfoldersdata>` tags.
- Data about each folder's contents can be accessed using the `<pt:kdpage.documentsdata>` and `<pt:kdpage.relatedresourcesdata>` tags.
- This data can be arranged in a useful way using the `<pt:kdpage.documentcolumnheadersdata>`, `<pt:kdpage.paginationdata>`, `<pt:kdpage.documentfilterdata>`, and `<pt:kdpage.documentsperpagedata>` tags.

The example below uses tags from the pt:kdpage library to define Directory page components, and logic tags to handle iteration and display. For detailed information on standard adaptive tags, including logic tags, see the *Oracle WebCenter Interaction Web Service Development Guide*.

Note: This example has been oversimplified for illustration purposes; for a complete implementation of the Directory page layout, see the knowledgedirectorylayout.html file in the \imageserver\plumtree\portal\private\pagelayouts folder on your portal image service.

The first section of the page retrieves the folder data.

```
<script type="text/javascript">
<!--
    function toggle_visibility(id) {
       var e = document.getElementById(id);
       if(e.style.display == 'block')
          e.style.display = 'none';
       else
          e.style.display = 'block';
    }
//-->
</script>

<div id="content" xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
    <pt:kdpage.currentfolderdata pt:id="currentfolder"/>
    <pt:kdpage.paginationdata pt:id="pagination" pt:pageslist="pageslist"
pt:pagestodisplay="2"/>
    <pt:logic.variable pt:key="currentfolderlevel"
pt:value="$currentfolder.level"/>
    <pt:logic.stringexpr pt:expr="($currentfolderlevel) == 1"
pt:key="isrootfolder"/>
```

After the breadcrumbs section (not shown here), the next section implements the column display. This section defines an index and divid for each column (3 in this example) The code iterates over the subfolders to display each folder in the correct column, using logic tags to divide the subfolder index by the number of columns to determine which column to display each folder. To change the number of columns, you would add or remove folders from the `<pt:logic.collection`

pt:key="foldercolumns"> section and change the divisor in the <pt:logic.intops pt:expr="($subfolderindex) % 3" pt:key="col"/> expression to reflect the correct number of columns.

```
... breadcrumbs section ...

        <!-- Start Root Folder Display -->
        <pt:logic.if pt:expr="$isrootfolder">
        <pt:logic.iftrue>
                <pt:logic.variable pt:key="displayrelatedresource"
pt:value="false"/>
                <div id="ali-kd-main-bar">
                </div>

                <pt:core.comment>This collection contains the meta-data about the
3 subfolder columns.</pt:core.comment>

                <pt:logic.collection pt:key="foldercolumns">
                        <pt:logic.data index="0" divid="ali-kd-main-col1"/>
                        <pt:logic.data index="1" divid="ali-kd-main-col2"/>
            <pt:logic.data index="2" divid="ali-kd-main-col3"/>
                </pt:logic.collection>

                <!-- Displaying Columns Start -->
                <pt:logic.foreach pt:data="foldercolumns" pt:var="foldercolumn">

                        <pt:kdpage.subfoldersdata pt:id="subfolders"/>
                        <pt:logic.collectionlength pt:data="subfolders"
pt:key="flength"/>
                        <pt:logic.intexpr pt:expr="($flength) > 0"
pt:key="hasfolders"/>

                        <pt:logic.if pt:expr="$hasfolders">
                                <pt:logic.iftrue>

                                        <pt:core.html pt:tag="div"
id="$foldercolumn.divid">
                                                <!-- Display Each Folder Start -->
                                                <pt:logic.foreach
pt:data="subfolders" pt:var="subfolder">
                                                        <pt:logic.intops
pt:expr="($subfolderindex) % 3" pt:key="col"/>
                                                        <pt:logic.intexpr
pt:expr="($col) == ($foldercolumn.index)" pt:key="incol"/>
                                                        <pt:logic.if
pt:expr="$incol">
                                                                <pt:logic.iftrue>
                                                                        <div
class="ali-kd-main-header">

<pt:logic.variable pt:key="htmlEncodedName" pt:value="$subfolder.name"
pt:encode="1" />

<pt:core.html pt:tag="a" href="$subfolder.url" title="$htmlEncodedName">

<pt:logic.value pt:value="$subfolder.name"/>

</pt:core.html>
                                                                        </div>
                                                                        <!--
```

```
Display Subfolders Folder Start -->

<pt:logic.variable pt:key="subsubfolderskey"
pt:value="$subfolder.subsubfolderskey"/>

<pt:logic.collectionlength pt:data="$subsubfolderskey" pt:key="flength"/>

<pt:logic.intexpr pt:expr="($flength) > 0" pt:key="hassubsubfolders"/>

<pt:logic.if pt:expr="$hassubsubfolders">

<pt:logic.iftrue>

<div class="ali-kd-main-lists">

<ul>

<pt:logic.foreach pt:data="$subsubfolderskey" pt:var="subsubfolder">

<pt:core.comment><!-- Only display max 5 subfolder --></pt:core.comment>

<pt:logic.intexpr pt:expr="($subsubfolderindex) < 5" pt:key="undermax"/>

<pt:logic.if pt:expr="$undermax">

<pt:logic.iftrue>

<li>

<pt:logic.variable pt:key="htmlEncodedName" pt:value="$subsubfolder.name"
pt:encode="1" />

<pt:core.html pt:tag="a" href="$subsubfolder.url" title="$htmlEncodedName">

<pt:logic.value pt:value="$subsubfolder.name"/>

</pt:core.html>

</li>

</pt:logic.iftrue>

</pt:logic.if>

</pt:logic.foreach>

</ul>

</div>

</pt:logic.iftrue>

</pt:logic.if>
                                                                    </pt:logic.iftrue>
                                                                </pt:logic.if>
                                                            </pt:logic.foreach>
                                                            <!-- Display Each Folder End -->
                                                        </pt:core.html>

                                                    </pt:logic.iftrue>
```

```
                    </pt:logic.if>
                </pt:logic.foreach>
                <!-- End Display Columns -->
            </pt:logic.iftrue>
            <!-- End Root Folder Display -->
```

After the sorting and filtering section (not shown here), the next section displays the document list and content.

```
<!-- Start Document List and content -->
                <div id="ali-kd-content-container">
                        <div id="ali-kd-documents">
                            <div id="ali-kd-docs-showing">
                                <pt:logic.value pt:value="$#1932.ptmsgs_
portalbrowsingmsgs"/>
                                <pt:logic.value pt:value="$pagination.start"/>
                                -
                                <pt:logic.value pt:value="$pagination.end"/>
                                <pt:logic.value pt:value="$#811.ptmsgs_
portalbrowsingmsgs"/>
                                <pt:logic.value pt:value="$pagination.total"/>
                                <pt:logic.value pt:value="$#1043.ptmsgs_
portalbrowsingmsgs"/>
                            </div>

... edit document code ...

             <!-- Start display documents -->
                                            <pt:logic.foreach pt:data="docs"
pt:var="doc">
                            <pt:logic.variable pt:key="htmlEncodedName"
pt:value="$doc.name" pt:encode="1" />
                            <tr>
                                <!-- Document Image Icon -->
                                <td><pt:core.html pt:tag="img" src="$doc.imagesrc"
alt="$htmlEncodedName"/></td>
                                <td>
                                    <!-- Document Title -->
                                    <p class="ali-kd-docs-title">
                                            <pt:logic.variable
pt:key="htmlEncodedName" pt:value="$doc.name" pt:encode="1" />
                                                <pt:core.html pt:tag="a"
href="$doc.url" title="$htmlEncodedName"><pt:logic.value
pt:value="$doc.name"/></pt:core.html>
                                        </p>
                                        <!-- Document Description -->
                                        <p>
                                                <pt:logic.variable
pt:key="description" pt:value="$doc.description"/>
                                            <pt:logic.stringexpr
pt:expr="($description) == EMPTY_STRING" pt:key="nodescription"/>
                                            <pt:logic.if pt:expr="$nodescription">
                                                <pt:logic.iftrue>
                                                    <i><pt:logic.value
pt:value="$#832.ptmsgs_portalbrowsingmsgs"/></i>
                                                </pt:logic.iftrue>
                                                    <pt:logic.iffalse>
                                                        <pt:logic.value
pt:value="$doc.description"/>
                                                </pt:logic.iffalse>
                                            </pt:logic.if>
```

```
                                        </p>
                                    <p class="ali-kd-docs-modified">

... pagination code ...

                <div id="ali-kd-side">
                        <ul id="ali-kd-subfolder">
                    <pt:kdpage.subfoldersdata pt:id="subfolders"/>
                        <pt:logic.collectionlength pt:data="subfolders"
pt:key="flength"/>
                        <pt:logic.intexpr pt:expr="($flength) > 0"
pt:key="hasfolders"/>
                        <pt:logic.if pt:expr="$hasfolders">
                                <pt:logic.iftrue>
                                        <pt:logic.value pt:value="$#1931.ptmsgs_
portalbrowsingmsgs"/>
                                    <pt:logic.foreach pt:data="subfolders"
pt:var="subfolder">
                                        <li>
                                            <pt:logic.variable
pt:key="htmlEncodedName" pt:value="$subfolder.name" pt:encode="1" />
                                                        <pt:core.html pt:tag="a"
href="$subfolder.url" title="$htmlEncodedName"><pt:logic.value
pt:value="$subfolder.name"/></pt:core.html>
                                                    </li>
                                        </pt:logic.foreach>
                                </pt:logic.iftrue>
                        </pt:logic.if>
                        </ul>

...related resource display code ...

        </pt:logic.iffalse>
    </pt:logic.if>
</div>
<!-- End documents view -->
```

3.6 Creating a Search Results Adaptive Page Layout

The Search Results page layout defines the search results content area. The outer area (header, footer, navigation areas) is based on the Base Page adaptive layout.

The pt:searchpage library includes a set of tags to customize search results display. The `<pt:searchpage.searchresultsdata>` tag provides all the data needed to display search results. Each result is a DataObject that includes the following variables:

Variable	Description
name	The name of the result.
description	The description of the result.
rank	The rank of the result.
resulthref	A gatewayed link to the result.
resultonclick	An onclick handler for result link if a handler exists.
resulttarget	The target window name for result link if a target exists.
icon	The URL to the icon for the result.

Variable	Description
iconalttext	Alternate text for the icon associated with the result.
iconwidth	The width of the icon associated with the result, in pixels.
lastmodified	A string in the current locale representing the last modified date.
propertieshref	A link to the properties for the result if a properties link exists.
propertiesonclick	An onclick handler for result link if a properties link exists.
isbestbet	True if the result is a best bet, false otherwise.
isinmultiplefolders	True if the result occurs in more than one Directory folder.
folderpathhref	A link to the Directory folder for the result if one exists.
folderpath	The path to the Directory folder for the result if one exists.
projectname	The name of the Collaboration project containing the result if the result is a Collaboration item.
projectonclick	An onclick handler for the link to the Collaboration project containing the result if the result is a Collaboration item.
lastpublishedby	The name of the last publishing user if the result is a Publisher item.
associatedobjectsonclick	An onclick handler for the link to the associated objects for the result if the result is a Publisher item.

The `<pt:searchpage.paginationdata>` tag provides variables to handle pagination. The `<pt:searchpage.followupformdata>` tag generates the data required to create the follow-up search form shown on the results page. Additional tags define specific form elements.

The example below uses tags from the pt:searchpage library to define Search Results components, and logic tags to iterate over results. For detailed information on standard adaptive tags, including logic tags, see the *Oracle WebCenter Interaction Web Service Development Guide*.

Note: This example has been oversimplified for illustration purposes; for a complete implementation of the search results page layout, see the searchresultslayout.html file in the \imageserver\plumtree\portal\private\pagelayouts folder on your portal image service.

```
<div id="content" xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
        <pt:core.comment>
                <!-- Layout for the search results page.  This layout file renders
the portal search results page.
                        It shows a followup search form and results.
                 -->
        </pt:core.comment>
    <pt:searchpage.searchresultsdata pt:id="groupedresults"/>
        <pt:searchpage.searchsummarydata pt:id="summary"

pt:groups="groups"

pt:spellcorrections="spellcorrections"

pt:breadcrumbs="breadcrumbs"

pt:properties="properties"

pt:collabprojects="collabprojects"
```

```
pt:portlets="portlets"

pt:folders="folders"

pt:communities="communities"

pt:objecttypes="objecttypes"/>

... search modification and sorting implementation ...

        <div id="ali-search-results">
            <pt:logic.collectionlength pt:data="groupedresults"
pt:key="resultslength"/>
            <pt:logic.intexpr pt:expr="($resultslength) > 0" pt:key="hasResults"/>
            <pt:logic.if pt:expr="$hasResults">
        <pt:logic.iftrue>
            <table id="ali-search-results-table">
            <pt:logic.foreach pt:data="groupedresults" pt:var="resultsgroup">
                <pt:logic.variable pt:key="results"
pt:value="$resultsgroup.groupresultskey"/>
                <pt:logic.collectionlength pt:data="$results"
pt:key="resultslength"/>
                <pt:logic.intexpr pt:expr="($resultslength) > 0"
pt:key="hasResults"/>
                <pt:logic.if pt:expr="$hasResults">
                    <pt:logic.iftrue>
                        <pt:logic.foreach pt:data="$results" pt:var="result">
                            <pt:logic.variable pt:key="htmlEncodedName"
pt:value="$result.name" pt:encode="1" />
                            <tr>
                                <td><pt:core.html pt:tag="img" src="$result.icon"
alt="$htmlEncodedName"/></td>
                                <td>
                                    <p class="ali-kd-docs-title">
                                        <pt:logic.existexpr
pt:data="result.resultonclick" pt:key="hasResultOnclick"/>
                                                <pt:logic.if
pt:expr="$hasResultOnclick">
                                                  <pt:logic.iftrue>
                                                <pt:core.html pt:tag="a"
href="$result.resulthref" onclick="$result.resultonclick"
title="$htmlEncodedName"><pt:logic.value pt:value="$result.name"/></pt:core.html>
                                                  </pt:logic.iftrue>
                                                  <pt:logic.iffalse>
                                                        <pt:logic.existexpr
pt:data="result.resulttarget" pt:key="hasResultTarget"/>
                                                        <pt:logic.if
pt:expr="$hasResultTarget">
                                                          <pt:logic.iftrue>
                                                                <pt:core.html
pt:tag="a" href="$result.resulthref" target="$result.resulttarget"
title="$htmlEncodedName"><pt:logic.value pt:value="$result.name"/></pt:core.html>
                                                          </pt:logic.iftrue>
                                                          <pt:logic.iffalse>
                                                                <pt:core.html
pt:tag="a" href="$result.resulthref" title="$htmlEncodedName"><pt:logic.value
pt:value="$result.name"/></pt:core.html>
                                                          </pt:logic.iffalse>
                                                        </pt:logic.if>
                                                </pt:logic.iffalse>
```

```
                                                    </pt:logic.if>
                                                </p>
                                                <p>
                                                    <pt:logic.existexpr
pt:data="result.description" pt:key="hasDesc"/>
                                                    <pt:logic.if pt:expr="$hasDesc">
                                                        <pt:logic.iftrue>
                                                            <pt:core.comment><!-- The
description contains HTML from search, and is safe, so we shouldn't HTML encode
it. --></pt:core.comment>
                                                            <pt:logic.value
pt:value="$result.description" pt:encode="0"/> 
                                                        </pt:logic.iftrue>
                                                        <pt:logic.iffalse>
                                                            <i><pt:logic.value
pt:value="$#832.ptmsgs_portalbrowsingmsgs"/></i>
                                                        </pt:logic.iffalse>
                                                    </pt:logic.if>
                                                </p>
                                                <p class="ali-search-results-modified">
                                                    <pt:core.localize pt:id="1918"
pt:file="ptmsgs_portalbrowsingmsgs" pt:replace0="$result.lastmodified" />
                                                    <pt:logic.existexpr
pt:data="result.propertieshref" pt:key="hasPropsLink"/>
                                                    <pt:logic.if
pt:expr="$hasPropsLink">
                                                        <pt:logic.iftrue>
                                                            <pt:core.html
pt:tag="a" href="$result.propertieshref" onclick="$result.propertiesonclick"
title="$#1657.ptmsgs_portalbrowsingmsgs"><pt:logic.value pt:value="$#31.ptmsgs_
portalbrowsingmsgs"/></pt:core.html>
                                                        </pt:logic.iftrue>
                                                    </pt:logic.if>
                                                </p>
                                            </td>
                                        </tr>
                                    </pt:logic.foreach>
                                </pt:logic.iftrue>
                            </pt:logic.if>
                        </pt:logic.foreach>
                        </table>

... page navigation implementation ...

                <pt:logic.iffalse>
                    <table id="ali-search-results-table">
                        <tr><td>
                            <p><pt:logic.value pt:value="$#848.ptmsgs_
portalbrowsingmsgs"/></p>
                        </td></tr>
                    </table>
                    <br />
                </pt:logic.iffalse>
                    </pt:logic.if>
            </div>
        </div>
        </div>
```

3.7 Creating a Portlet Selection Adaptive Page Layout

Portlet Selection page layouts allow you to customize the portlet flyout editor used to add and remove portlets from a page.

The pt:portletpageeditor library contains tags to implement a custom portlet flyout editor (DHTML). Many of the tags in this library are intended for use in adaptive portlet layout pages that include the portlet flyout editor.

- The `<pt:portletpageeditor.sortpropertiesdata>` tag sets a collection of properties used to specify portlet sort order. This tag must be displayed before other pt:portletpageeditor tags so the page can be initialized properly.
- The `<pt:portletpageeditor.addportletsflyoutdata>` tag provides a URL to the flyout editor.
- The `<pt:portletpageeditor.flyoutjs>` tag adds the JavaScript required to create a flyout effect.
- The `<pt:portletpageeditor.collapseexpandjs>` tag dds the JavaScript required to collapse and expand portlets through AJAX.

The rest of the tags in the library are used to create the flyout editor in the Portlet Selection adaptive page layout.

- The `<pt:portletpageeditor.portletjs>` tag generates the JavaScript functions required for portlet preview and invitation.
- The `<pt:portletpageeditor.portletdata>` tag generates the data required to show the list of portlets in the editor, and stores it in memory using the variable name specified by the id attribute. Each item in the list is a DataObject with the following variables:

Variable	Description
name	The name of the portlet
description	The description of the portlet.
id	The ID of the portlet object in the portal.
isonpage	True if portlet is on the current page, false otherwise.
type	The portlet type (narrow, wide, or bundle). This variable is used as the div style class suffix and passed in to the preview JavaScript function.
previewenabled	True if preview is enabled for the portlet, false otherwise.
invitationid	The ID to be used in the invitation JavaScript function. If the value is -1, the invitation is disabled.
lastmodified - mandatory - the last modified date of the portlet	The last modified date of the portlet.
mandatory	True if the portlet is mandatory for the current page, false otherwise.

- The `<pt:portletpageeditor.objectrenamehelper>` tag provides a rename div for use with the portlet name. This tag sets the `pt:isEditable` variable to true or false depending on whether the user has edit access to the current page.
- The `<pt:portletpageeditor.portletsearchform>` tag generates the form and hidden inputs necessary to search for portlets. It does not generate the text

input or search button. The text input must to be defined in the `in_tx_query` parameter. The `<pt:portletpageeditor.paginationdata>` tag generates the data for pagination links for the search results.

- The `<pt:portletpageeditor.portletbrowsemode>` tag displays the JavaScript necessary to switch to browse mode. The `<pt:portletpageeditor.browsesubfoldersdata>` tag stores a list of subfolders of the current folder in memory (only populated if the page is in browse mode). The `<pt:portletpageeditor.browsebreadcrumbsdata>` tag stores a list of subfolders of the current folder in memory.
- The `<pt:portletpageeditor.currsortpropid>` sets a data value with the current portlet selection sort by property. The `<pt:portletpageeditor.sortpropertyentry>` tag can be used to specify a sort property entry if the `<pt:portletpageeditor.sortpropertiesdata>` tag has been implemented.

The example below uses tags from the pt:portletpageeditor library to define portlet flyout editor components, and logic tags to iterate through the portlets displayed in the editor. For detailed information on standard adaptive tags, including logic tags, see the *Oracle WebCenter Interaction Web Service Development Guide*.

```
<div xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
<pt:core.comment><!-- this input is required so that IE doesn't strip out the
javacript tags when we add it to a div's innerHTML --></pt:core.comment>
<input type='hidden'/>
<pt:portletpageeditor.portletjs pt:add="addPortlet" pt:remove="removePortlet"
pt:preview="previewPortlet" pt:invitation="invite" pt:addbundle="addBundle"
pt:openbundle="openBundle" pt:orderbyprop="updateOrderByProperty"
pt:flyoutID="portletSelection"/>
<pt:core.comment><!-- NOTE: many tags must be initialized at the top so they can
be used in the rest of the page. --></pt:core.comment>
<pt:portletpageeditor.portletdata pt:id="portlets"/>
<pt:logic.collectionlength pt:data="portlets" pt:key="plength"/>
<pt:logic.intexpr pt:expr="($plength) > 0" pt:key="hasportlets"/>

<pt:portletpageeditor.portletbrowsemode pt:flyoutID="portletSelection"
pt:id="browseMode"/>
<div id="ali-edit-container">
<div id="ali-edit-toolbar">
<div id="ali-edit-cornerleft"></div>
<div id="ali-edit-title"><pt:logic.value pt:value="$#301.ptmsgs_
portalcommonmsgs"/></div>
<div id="ali-edit-cornerright"></div>
<div class="ali-portlet-controlone"><a onclick="try
{bea.PortalPageDnD.dndToggle(); PTFlyoutportletSelection.openFlyout(); return
false;} catch (e) {return true;}" href=""><img
src="pt://images/plumtree/portal/private/img/action_portlet_remove.gif"
/></a></div>
</div>
<div id="ali-edit-tabs-container">
    <div id="ali-edit-rename">
    <pt:ptdata.currpagedata pt:id="currpage" />
    <pt:logic.foreach pt:data="currpage" pt:var="page" >
      <pt:logic.intexpr pt:expr="($page.classid) == 518" pt:key="isMyPage" />
      <pt:logic.if pt:expr="$isMyPage">
        <pt:logic.iftrue>
          <pt:core.comment><!-- Page data object is a MyPage --></pt:core.comment>
          <pt:portletpageeditor.objectrenamehelper pt:objectname="$page.title"
pt:objectid="$page.objectid" pt:classid="$page.classid" pt:in_prefix="page_
```

```
rename"/>
        </pt:logic.iftrue>
        <pt:logic.iffalse>
          <pt:core.comment><!-- Page data object is a Community Page.  ObjectID is
for the community, not the page, sooverride it here. --></pt:core.comment>
          <pt:portletpageeditor.objectrenamehelper pt:objectname="$page.title"
pt:objectid="$page.childid" pt:classid="514" pt:in_prefix="page_rename"/>
        </pt:logic.iffalse>
      </pt:logic.if>
      <pt:core.comment><!-- pt:portletpageeditor.objectrenamehelper tag sets the
isEditable variable to true or false depending on whether the user has edit access
to tha page. --></pt:core.comment>
      <pt:logic.if pt:expr="$isEditable"><pt:logic.iftrue>
        <pt:logic.value pt:value="$#1907.ptmsgs_portalbrowsingmsgs"/>
        <pt:logic.variable pt:key="htmlEncodedTitle" pt:value="$page.title"
pt:encode="1" />
        <pt:core.html pt:tag="input" type="text" class="ali-edit-rename-textbox"
id="$input_id" value="$htmlEncodedTitle" size="20" maxlength="255"/>
      </pt:logic.iftrue></pt:logic.if>
    </pt:logic.foreach>
    </div>

    <!-- Start "Go To Advanced Editor" Link -->
    <pt:core.comment>Check whether current page is a My Page or Community Page and
get the link for the Page Editor.</pt:core.comment>
    <pt:ptdata.currcommunitypagesdata pt:id="currCommPagesData" />
    <pt:logic.existexpr pt:data="currCommPagesData"
pt:key="hasCurrCommPagesData"/>
    <pt:ptdata.communityactionsdata pt:id="commActionLinks" />
    <pt:ptdata.mypageactionsdata pt:id="myPageActionLinks" />
    <pt:logic.existexpr pt:data="commActionLinks" pt:key="hasCommActionLinks"/>
    <pt:logic.existexpr pt:data="myPageActionLinks"
pt:key="hasMyPageActionLinks"/>
    <pt:logic.variable pt:key="commActionEdit" pt:value="$#308.ptmsgs_
portalcommonmsgs"/>
    <pt:logic.variable pt:key="myPageActionEdit" pt:value="$#301.ptmsgs_
portalcommonmsgs"/>

        <pt:logic.if pt:expr="$hasCurrCommPagesData">
      <pt:logic.iftrue>
        <pt:core.comment>Loop through list of Community action URLs and find the
Edit Link.</pt:core.comment>
        <pt:logic.foreach pt:data="commActionLinks" pt:var="link">
          <pt:logic.stringexpr pt:expr="($link.title) == ($commActionEdit)"
pt:key="addCommEditLink" />
          <pt:logic.if pt:expr="$addCommEditLink">
            <pt:logic.iftrue>
              <div class="ali-edit-tabs-right">
                <pt:core.html pt:tag="a" href="$link.url" ><pt:logic.value
pt:value="$#1934.ptmsgs_portalbrowsingmsgs"/></pt:core.html>
              </div>
            </pt:logic.iftrue>
          </pt:logic.if>
        </pt:logic.foreach>
      </pt:logic.iftrue>
      <pt:logic.iffalse>
        <pt:core.comment>Loop through list of My Page action URLs and find the
Edit Link.</pt:core.comment>
        <pt:logic.foreach pt:data="myPageActionLinks" pt:var="link">
          <pt:logic.stringexpr pt:expr="($link.title) == ($myPageActionEdit)"
```

```
pt:key="addMyPageEditLink" />
          <pt:logic.if pt:expr="$addMyPageEditLink">
            <pt:logic.iftrue>
              <div class="ali-edit-tabs-right">
                <pt:core.html pt:tag="a" href="$link.url" ><pt:logic.value
pt:value="$#1934.ptmsgs_portalbrowsingmsgs"/></pt:core.html>
              </div>
            </pt:logic.iftrue>
          </pt:logic.if>
        </pt:logic.foreach>
      </pt:logic.iffalse>
    </pt:logic.if>
    <!-- End "Go To Advanced Editor" Link -->
</div>
<pt:core.comment><!-- pt:portletpageeditor.objectrenamehelper tag sets the
isEditable variable to true or false depending on whether the user has edit access
to tha page. --></pt:core.comment>
<pt:logic.if pt:expr="$isEditable"><pt:logic.iftrue>
    <div id="ali-edit-portlets">
        <div id="ali-edit-search-container">
            <div id="ali-edit-portlets-text"><pt:logic.value
pt:value="$#1908.ptmsgs_portalbrowsingmsgs"/></div>
        </div>
        <div id="ali-edit-sorting-bar">
            <pt:logic.if pt:expr="$browseMode"><pt:logic.iffalse>
            <div id="ali-edit-sort">
                <form name="sort">
                        <pt:logic.value pt:value="$#1936.ptmsgs_
portalbrowsingmsgs"/>
                    <select
onchange="updateOrderByProperty(options[selectedIndex].value); return false;">
                    <pt:portletpageeditor.currsortpropid pt:id="currsortpropid"/>
                    <pt:logic.foreach pt:data="sortprops" pt:var="curr">
                        <pt:logic.stringexpr pt:expr="($curr.id) ==
($currsortpropid)" pt:key="iscurrpropid"/>
                        <pt:logic.if pt:expr="$iscurrpropid">
                        <pt:logic.iftrue>
                            <pt:core.html pt:tag="option" selected="true"
value="$curr.id"><pt:logic.value pt:value="$curr.title"/></pt:core.html>
                        </pt:logic.iftrue>
                        <pt:logic.iffalse>
                            <pt:core.html pt:tag="option"
value="$curr.id"><pt:logic.value pt:value="$curr.title"/></pt:core.html>
                        </pt:logic.iffalse>
                        </pt:logic.if>
                    </pt:logic.foreach>

                    </select>
                </form>
            </div>
            </pt:logic.iffalse></pt:logic.if>
            <div
id="ali-edit-portlets-search"><pt:portletpageeditor.portletsearchform
pt:name="portletSearch" pt:id="portletSearch" pt:submit="submitPortletSearch"
pt:flyoutID="portletSelection" pt:defaulttext="$#1914.ptmsgs_portalbrowsingmsgs">
              <pt:logic.concat pt:key="portletsearchonfocus" pt:value1="if
(this.value == '" pt:value2="$#1914.ptmsgs_portalbrowsingmsgs"/><pt:logic.concat
pt:key="portletsearchonfocus" pt:value1="$portletsearchonfocus" pt:value2="')
this.value='';"/>
              <pt:logic.value pt:value="$#715.ptmsgs_portalbrowsingmsgs"/>
```

```
<pt:core.html pt:tag="input" type="text" class="edit-portlets-search-box"
name="in_tx_query" value="$#1914.ptmsgs_portalbrowsingmsgs" size="20"
onkeydown="return handleSearchEvent(event);"
onfocus="$portletsearchonfocus"/><pt:core.html pt:tag="input" name="Search"
type="button" onClick="return submitPortletSearch();" value="$#1913.ptmsgs_
portalbrowsingmsgs" class="edit-portlets-search-button"/>
            </pt:portletpageeditor.portletsearchform></div>
            <pt:logic.if pt:expr="$browseMode"><pt:logic.iffalse>
                  <div id="ali-edit-browse-portlets"><a href="#"
onclick="openFolder(1);return false;"><pt:logic.value pt:value="$#1917.ptmsgs_
portalbrowsingmsgs"/></a></div>
                </pt:logic.iffalse></pt:logic.if>
        </div>

            <div id="ali-edit-breadcrumb-container">
            <pt:core.comment><!-- Browse All Folders (Switching to browse mode
opens root folder [ID 1]) --></pt:core.comment>
            <pt:logic.variable pt:value="false" pt:key="isrootfolder"/>
            <pt:logic.if pt:expr="$browseMode"><pt:logic.iftrue>
            <!-- Start breadcrumbs path -->
              <pt:portletpageeditor.browsebreadcrumbsdata pt:id="breadcrumbs"/>
              <div id="ali-edit-breadcrumb"><ul>
                    <pt:logic.foreach pt:data="breadcrumbs" pt:var="breadcrumb">
                            <pt:logic.variable pt:key="breadcrumburl"
pt:value="$breadcrumb.url"/>
                            <pt:logic.stringexpr pt:expr="($breadcrumburl) ==
EMPTY_STRING" pt:key="iscurrentfolder"/>
                            <pt:logic.if
pt:expr="$iscurrentfolder"><pt:logic.iftrue>
                              <li><pt:logic.value
pt:value="$breadcrumb.name"/></li>
                              <pt:logic.intexpr pt:expr="($breadcrumb.id) == 1"
pt:key="isrootfolder"/>
                            </pt:logic.iftrue><pt:logic.iffalse>
                              <pt:logic.concat pt:key="openFolder"
pt:value1="openFolder(" pt:value2="$breadcrumb.id"/>
                              <pt:logic.concat pt:key="openFolder"
pt:value1="$openFolder" pt:value2="); return false;"/>
                              <pt:logic.variable pt:key="htmlEncodedName"
pt:value="$breadcrumb.name" pt:encode="1" />
                              <li><pt:core.html pt:tag="a" href="#"
onclick="$openFolder" title="$htmlEncodedName"><pt:logic.value
pt:value="$breadcrumb.name"/></pt:core.html></li>
                            </pt:logic.iffalse></pt:logic.if>
                            <pt:logic.separator><li></li></pt:logic.separator>
                    </pt:logic.foreach>
              </ul></div>
            </pt:logic.iftrue></pt:logic.if>
            </div>
            <!-- End breadcrumbs path -->

            <pt:logic.if pt:expr="$browseMode"><pt:logic.iftrue>
                    <pt:core.comment><!-- Figure out if page is in browse mode and
has subfolders --></pt:core.comment>
                    <pt:portletpageeditor.browsesubfoldersdata
pt:id="subfolders"/>
                    <pt:logic.collectionlength pt:data="subfolders"
pt:key="flength"/>
                    <pt:logic.intexpr pt:expr="($flength) > 0"
pt:key="hasfolders"/>
```

```
            </pt:logic.iftrue></pt:logic.if>

            <!-- Start Root Folder Display -->
            <pt:logic.if pt:expr="$browseMode"><pt:logic.iftrue>
            <pt:logic.if pt:expr="$isrootfolder"><pt:logic.iftrue>
            <pt:core.comment><!-- folder info has been determined above.
--></pt:core.comment>
            <pt:logic.if pt:expr="$hasfolders"><pt:logic.iftrue>
            <div id="ali-edit-table-container">
                <table id="ali-edit-browse-table">
                    <pt:logic.foreach pt:data="subfolders" pt:var="subfolder">
                                <pt:logic.intops pt:expr="($subfolderindex) % 3"
pt:key="col"/>
                                <pt:logic.intexpr pt:expr="($col) == 0"
pt:key="addTR"/>
                                <pt:logic.if pt:expr="$addTR"><pt:logic.iftrue>
                                    <tr>
                                </pt:logic.iftrue></pt:logic.if>
                        <td class="ali-edit-browse-folder"><img
src="pt://images/plumtree/portal/private/img/icon_folder_24px.gif"
alt="folder"></td>
                        <td class="ali-edit-browse-description"><p
class="ali-edit-portlets-title">
                            <pt:logic.concat
pt:key="openFolder"pt:value1="openFolder(" pt:value2="$subfolder.id"/>
                                <pt:logic.concat pt:key="openFolder"
pt:value1="$openFolder" pt:value2="); return false;"/>
                                <pt:logic.variable pt:key="htmlEncodedName"
pt:value="$subfolder.name" pt:encode="1" />
                                <pt:core.html pt:tag="a" href="#"
onclick="$openFolder" title="$htmlEncodedName">
                                <pt:logic.value pt:value="$subfolder.name"/>
                                </pt:core.html>
                        </p></td>
                        <pt:logic.intexpr pt:expr="($col) == 2" pt:key="endTR"/>
                                <pt:logic.if pt:expr="$endTR"><pt:logic.iftrue>
                                    </tr>
                                </pt:logic.iftrue></pt:logic.if>
                    </pt:logic.foreach>
                        <tr>
                            <td colspan="6">
                            <div id="ali-edit-close">
                                    <pt:core.html name="Close" pt:tag="input"
type="button" class="edit-portlets-close-button" id="ali-closeButton"
value="$#1945.ptmsgs_portalbrowsingmsgs" onclick="try
{bea.PortalPageDnD.dndToggle(); PTFlyoutportletSelection.openFlyout(); return
false;} catch (e) {return true;}"/>
                            </div>
                        </td>
                    </tr>
                    </table>
                </div>
            </pt:logic.iftrue></pt:logic.if>
            </pt:logic.iftrue></pt:logic.if>
            </pt:logic.iftrue></pt:logic.if>
            <!-- End Root Folder Display -->

            <!-- End Browse Mode Display -->

    <pt:logic.variable pt:value="true" pt:key="show3columns"/>
```

```
    <pt:logic.if pt:expr="$browseMode"><pt:logic.iftrue>
            <pt:logic.if pt:expr="$isrootfolder"><pt:logic.iffalse>
                    <pt:logic.if pt:expr="$hasfolders"><pt:logic.iftrue>
                            <pt:logic.variable pt:value="false"
pt:key="show3columns"/>
                    </pt:logic.iftrue></pt:logic.if>
            </pt:logic.iffalse></pt:logic.if>
    </pt:logic.iftrue></pt:logic.if>

    <pt:core.comment>This collection and variable contain the metadata about the
portlet columns.</pt:core.comment>
    <pt:logic.collection pt:key="columns">
            <pt:logic.data index="0" divid="ali-edit-portlets-column1"/>
            <pt:logic.data index="1" divid="ali-edit-portlets-column2"/>
            <pt:logic.if pt:expr="$show3columns"><pt:logic.iftrue>
                    <pt:logic.data index="2" divid="ali-edit-portlets-column3"/>
            </pt:logic.iftrue></pt:logic.if>
    </pt:logic.collection>
    <pt:logic.collectionlength pt:data="columns" pt:key="portletmod"/>

    <div id="ali-edit-table-container">
        <!-- Start browse mode folder display (except for root landing page) -->
        <pt:logic.if pt:expr="$browseMode"><pt:logic.iftrue>
            <pt:logic.if pt:expr="$isrootfolder"><pt:logic.iffalse>
                    <pt:core.comment><!-- folder info has been determined above.
--></pt:core.comment>
                    <pt:logic.if pt:expr="$hasfolders"><pt:logic.iftrue>
                    <table id="ali-edit-browse-table">
                        <pt:logic.foreach pt:data="subfolders" pt:var="subfolder">
                            <pt:logic.intops pt:expr="($subfolderindex) % 3"
pt:key="col"/>
                            <pt:logic.intexpr pt:expr="($col) == 0"
pt:key="addTR"/>
                            <pt:logic.if pt:expr="$addTR"><pt:logic.iftrue>
                                <tr>
                            </pt:logic.iftrue></pt:logic.if>
                            <td class="ali-edit-browse-folder"><img
src="pt://images/plumtree/portal/private/img/icon_folder_24px.gif"
alt="folder"></td>
                            <td class="ali-edit-browse-description"><p
class="ali-edit-portlets-title">
                                <pt:logic.concat
pt:key="openFolder"pt:value1="openFolder(" pt:value2="$subfolder.id"/>
                                <pt:logic.concat pt:key="openFolder"
pt:value1="$openFolder" pt:value2="); return false;"/>
                                <pt:logic.variable pt:key="htmlEncodedName"
pt:value="$subfolder.name" pt:encode="1" />
                                <pt:core.html pt:tag="a" href="#"
onclick="$openFolder" title="$htmlEncodedName">
                                <pt:logic.value pt:value="$subfolder.name"/>
                                </pt:core.html>
                            </p></td>
                            <pt:logic.intexpr pt:expr="($col) == 2"
pt:key="endTR"/>
                            <pt:logic.if pt:expr="$endTR"><pt:logic.iftrue>
                                </tr>
                            </pt:logic.iftrue></pt:logic.if>
                        </pt:logic.foreach>
                    </table>
                    </pt:logic.iftrue></pt:logic.if>
```

```
            </pt:logic.iffalse></pt:logic.if>
        </pt:logic.iftrue></pt:logic.if>
        <!-- End browse mode folder display (except for root landing page)  -->
        <pt:core.comment><!-- No portlets to display. --></pt:core.comment>
        <pt:logic.if pt:expr="$hasportlets"><pt:logic.iffalse>
                <pt:logic.if pt:expr="$isrootfolder"><pt:logic.iffalse>
                    <table id="ali-edit-table">
                        <tr>
                            <td class="ali-edit-table-description"><pt:logic.value
pt:value="$#1939.ptmsgs_portalbrowsingmsgs"/></td>
                        </tr>
                        <tr>
                            <td colspan="6">
                            <div id="Div1">
                                    <pt:core.html name="Close" pt:tag="input"
type="button" class="edit-portlets-close-button" id="ali-closeButton"
value="$#1945.ptmsgs_portalbrowsingmsgs" onclick="try
{bea.PortalPageDnD.dndToggle(); PTFlyoutportletSelection.openFlyout(); return
false;} catch (e) {return true;}"/>
                            </div>
                        </td>
                    </tr>
                    </table>
                </pt:logic.iffalse></pt:logic.if>
        </pt:logic.iffalse>
        <pt:logic.iftrue>

        <table id="ali-edit-table">
            <pt:logic.foreach pt:data="columns" pt:var="column">
                <tr>
                    <pt:core.comment>In each column, loop over all the portlets to
find the ones for this column.</pt:core.comment>
                    <pt:logic.foreach pt:data="portlets" pt:var="portlet">
                        <pt:logic.intops pt:expr="($portletindex) % ($portletmod)"
pt:key="col"/>
                        <pt:logic.intexpr pt:expr="($col) == ($column.index)"
pt:key="incol"/>
                        <pt:logic.if pt:expr="$incol">
                        <pt:logic.iftrue>
                            <pt:logic.stringexpr pt:expr="($portlet.type) ==
bundle" pt:key="isbundle"/>
                            <pt:logic.if
pt:expr="$isbundle"><pt:logic.iftrue><pt:core.comment><!-- This is a portlet
bundle --></pt:core.comment>
                                <pt:logic.concat pt:key="onclickadd"
pt:value1="addBundle(" pt:value2="$portlet.id"/>
                                <pt:logic.concat pt:key="onclickadd"
pt:value1="$onclickadd" pt:value2=");"/>
                                <pt:logic.variable pt:key="onclickremove"
pt:value="$onclickadd"/>
                            </pt:logic.iftrue>
                            <pt:logic.iffalse><pt:core.comment><!-- This is a
portlet --></pt:core.comment>
                                <pt:logic.concat pt:key="onclickremove"
pt:value1="removePortlet(" pt:value2="$portlet.id"/>
                                <pt:logic.concat pt:key="onclickremove"
pt:value1="$onclickremove" pt:value2=");"/>

                                <pt:logic.concat pt:key="onclickadd"
pt:value1="addPortlet(" pt:value2="$portlet.id"/>
```

```
                                <pt:logic.concat pt:key="onclickadd"
pt:value1="$onclickadd" pt:value2=");"/>
                            </pt:logic.iffalse></pt:logic.if>
                            <pt:logic.concat pt:key="src1"
pt:value1="pt://images/plumtree/portal/private/img/icon_portlet_"
pt:value2="$portlet.type"/>
                            <pt:logic.concat pt:key="src" pt:value1="$src1"
pt:value2=".gif"/>
                            <pt:logic.variable pt:key="swapsrc"
pt:value="pt://images/plumtree/portal/private/img/icon_portlet_chosen.gif"/>
                            <pt:logic.stringexpr pt:expr="($portlet.isonpage) ==
true" pt:key="isonpage"/>
                            <pt:logic.if pt:expr="$isonpage"><pt:logic.iftrue>
                                <pt:logic.variable pt:key="swapsrc"
pt:value="$src"/>
                                <pt:logic.variable pt:key="src"
pt:value="pt://images/plumtree/portal/private/img/icon_portlet_chosen.gif"/>
                                <pt:logic.variable pt:key="onclickstring"
pt:value="$onclickremove"/>
                            </pt:logic.iftrue>
                            <pt:logic.iffalse>
                                <pt:logic.variable pt:key="onclickstring"
pt:value="$onclickadd"/>
                            </pt:logic.iffalse></pt:logic.if>
                            <pt:logic.concat pt:key="onclickimg"
pt:value1="$onclickstring" pt:value2=" return false;"/>
                            <pt:logic.concat pt:key="divid" pt:value1="portlet"
pt:value2="$portlet.id"/>
                            <pt:logic.stringexpr pt:expr="($portlet.mandatory) ==
true" pt:key="ismandatory"/>
                            <pt:logic.if pt:expr="$ismandatory"><pt:logic.iftrue>
                                <td><pt:core.html pt:tag="img" id="$divid"
src="$src" border="0"/></td>
                            </pt:logic.iftrue><pt:logic.iffalse>
                                <td><pt:core.html pt:tag="a" href="#"
onclick="$onclickimg" onclickstring="$onclickstring"><pt:core.html pt:tag="img"
id="$divid" src="$src" swapsrc="$swapsrc" border="0"/></pt:core.html></td>
                            </pt:logic.iffalse></pt:logic.if>
                            <td class="ali-edit-table-description"><p
class="ali-edit-portlets-title"><pt:logic.value pt:value="$portlet.name"/></p>
                                <pt:core.comment><!-- The description contains
HTML from search, and is safe, so there is no need to HTML encode it.
--></pt:core.comment>
                                <p><pt:logic.value pt:value="$portlet.description"
pt:encode="0"/></p>
                                <p>
                                    <pt:logic.stringexpr pt:expr="($portlet.type)
== bundle" pt:key="isbundle"/>
                                    <pt:logic.if
pt:expr="$isbundle"><pt:logic.iftrue><pt:core.comment><!-- This is a portlet
bundle - include add/open links --></pt:core.comment>
                                        <pt:logic.concat pt:key="addurl"
pt:value1="addBundle(" pt:value2="$portlet.id"/>
                                        <pt:logic.concat pt:key="addurl"
pt:value1="$addurl" pt:value2="); return false;"/>
                                        <pt:core.html pt:tag="a" href="#"
onclick="$addurl"><pt:logic.value pt:value="$#1273.ptmsgs_
portalbrowsingmsgs"/></pt:core.html>

                                        <pt:logic.concat pt:key="openurl"
```

```
pt:value1="openBundle(" pt:value2="$portlet.id"/>
                                        <pt:logic.concat pt:key="openurl"
pt:value1="$openurl" pt:value2="); return false;"/>
                                        - <pt:core.html pt:tag="a" href="#"
onclick="$openurl"><pt:logic.value pt:value="$#1915.ptmsgs_
portalbrowsingmsgs"/></pt:core.html>
                                        <p
class="ali-edit-portlets-modified"><pt:core.localize pt:id="1918" pt:file="ptmsgs_
portalbrowsingmsgs" pt:replace0="$portlet.lastmodified" /></p>
                                    </pt:logic.iftrue>
                                    <pt:logic.iffalse><pt:core.comment><!-- This
is a portlet - include add/remove/preview/invite links --></pt:core.comment>
                                        <pt:logic.if
pt:expr="$ismandatory"><pt:logic.iffalse>
                                            <pt:logic.if
pt:expr="$isonpage"><pt:logic.iftrue>
                                                <pt:logic.variable
pt:key="addstyle" pt:value="display:none;"/>

                                                <pt:logic.variable
pt:key="removestyle" pt:value="display:visible;"/>
                                            </pt:logic.iftrue>
                                            <pt:logic.iffalse>
                                                <pt:logic.variable
pt:key="addstyle" pt:value="display:visible;"/>
                                                <pt:logic.variable
pt:key="removestyle" pt:value="display:none;"/>
                                            </pt:logic.iffalse></pt:logic.if>
                                            <pt:logic.concat pt:key="addid"
pt:value1="add-portlet" pt:value2="$portlet.id"/>
                                            <pt:logic.concat pt:key="removeid"
pt:value1="remove-portlet" pt:value2="$portlet.id"/>

                                            <pt:logic.concat pt:key="removeurl"
pt:value1="removePortlet(" pt:value2="$portlet.id"/>
                                            <pt:logic.concat pt:key="removeurl"
pt:value1="$removeurl" pt:value2="); return false;"/>
                                            <pt:core.html pt:tag="a" href="#"
onclick="$removeurl" id="$removeid" style="$removestyle"><pt:logic.value
pt:value="$#1910.ptmsgs_portalbrowsingmsgs"/></pt:core.html>

                                            <pt:logic.concat pt:key="addurl"
pt:value1="addPortlet(" pt:value2="$portlet.id"/>
                                            <pt:logic.concat pt:key="addurl"
pt:value1="$addurl" pt:value2="); return false;"/>
                                            <pt:core.html pt:tag="a" href="#"
onclick="$addurl" id="$addid" style="$addstyle"><pt:logic.value
pt:value="$#1909.ptmsgs_portalbrowsingmsgs"/></pt:core.html> -
                                        </pt:logic.iffalse></pt:logic.if>

                                        <pt:logic.stringexpr
pt:expr="($portlet.previewenabled) == true" pt:key="previewEnabled"/>
                                        <pt:logic.if
pt:expr="$previewEnabled"><pt:logic.iftrue>
                                            <pt:logic.concat pt:key="previewid"
pt:value1="preview-portlet" pt:value2="$portlet.id"/>
                                            <pt:logic.concat pt:key="previewurl"
pt:value1="previewPortlet(" pt:value2="$portlet.id"/>
                                            <pt:logic.concat pt:key="previewurl"
pt:value1="$previewurl" pt:value2=", '"/>
```

```
                                        <pt:logic.concat pt:key="previewurl"
pt:value1="$previewurl" pt:value2="$portlet.type"/>
                                        <pt:core.comment><!-- We can't just do
this.getAttribute('onpage'), because that doesn't work in IE.
--></pt:core.comment>
                                        <pt:logic.concat pt:key="previewurl"
pt:value1="$previewurl" pt:value2="', this.attributes.isonpage.nodeValue); return
false;"/>
                                        <pt:core.html pt:tag="a" href="#"
onclick="$previewurl" id="$previewid" isonpage="$portlet.isonpage"><pt:logic.value
pt:value="$#1911.ptmsgs_portalbrowsingmsgs"/></pt:core.html>
                                    </pt:logic.iftrue></pt:logic.if>

                                    <pt:logic.intexpr
pt:expr="($portlet.invitationid) != -1" pt:key="invitationEnabled"/>
                                    <pt:logic.if
pt:expr="$invitationEnabled"><pt:logic.iftrue>
                                        <pt:logic.concat pt:key="inviteurl"
pt:value1="invite(" pt:value2="$portlet.invitationid"/>
                                        <pt:logic.concat pt:key="inviteurl"
pt:value1="$inviteurl" pt:value2="); return false;"/>
                                        - <pt:core.html pt:tag="a" href="#"
onclick="$inviteurl"><pt:logic.value pt:value="$#1912.ptmsgs_
portalbrowsingmsgs"/></pt:core.html>
                                    </pt:logic.iftrue></pt:logic.if>
                                    <p
class="ali-edit-portlets-modified"><pt:core.localize pt:id="1918" pt:file="ptmsgs_
portalbrowsingmsgs" pt:replace0="$portlet.lastmodified" /></p>
                                </pt:logic.iffalse></pt:logic.if>
                            </p>
                        </td>
                    </pt:logic.iftrue>
                    </pt:logic.if>
                </pt:logic.foreach>
            </tr>
        </pt:logic.foreach>
        <tr>
            <td colspan="6">
                <div id="ali-edit-close">
                    <pt:core.html name="Close" pt:tag="input" type="button"
class="edit-portlets-close-button" id="ali-closeButton" value="$#1945.ptmsgs_
portalbrowsingmsgs" onclick="try {bea.PortalPageDnD.dndToggle();
PTFlyoutportletSelection.openFlyout(); return false;} catch (e) {return true;}"/>
                </div>
                <pt:core.comment><!-- In an AJAX flyout, so wrap the
pagination URLs in an AJAX method. --></pt:core.comment>
                <pt:portletpageeditor.paginationdata pt:id="pagination"
pt:pageslist="pageslist" pt:pagestodisplay="2" pt:flyoutID="portletSelection"/>
                <div id="ali-edit-portlets-pagenav">
                    <ul>
                      <pt:logic.existexpr pt:data="pagination.previousurl"
pt:key="linkToPrevious"/>
                      <pt:logic.if pt:expr="$linkToPrevious"><pt:logic.iftrue>
                          <pt:logic.concat pt:key="paginatefunc"
pt:value1="paginate('" pt:value2="$pagination.previousurl"/>
                          <pt:logic.concat pt:key="paginatefunc"
pt:value1="$paginatefunc" pt:value2="'); return false;"/>
                          <li><pt:core.html pt:tag="a" href="#"
onclick="$paginatefunc" title="$#34.ptmsgs_portalbrowsingmsgs"><pt:logic.value
pt:value="$#207.ptmsgs_portalinfrastructure"/></pt:core.html></li>
```

```
                            </pt:logic.iftrue><pt:logic.iffalse>
                              <li><pt:logic.value pt:value="$#207.ptmsgs_
portalinfrastructure"/></li>
                            </pt:logic.iffalse></pt:logic.if>

                            <pt:logic.existexpr pt:data="pagination.firstpageurl"
pt:key="displayFirstPageLink"/>
                            <pt:logic.if
pt:expr="$displayFirstPageLink"><pt:logic.iftrue>
                              <pt:logic.concat pt:key="paginatefunc"
pt:value1="paginate('" pt:value2="$pagination.firstpageurl"/>
                              <pt:logic.concat pt:key="paginatefunc"
pt:value1="$paginatefunc" pt:value2="'); return false;"/>
                              <li class="ali-edit-number"><pt:core.html pt:tag="a"
href="#" onclick="$paginatefunc">1</pt:core.html></li>
                            </pt:logic.iftrue></pt:logic.if>

                            <pt:logic.collectionlength pt:data="pageslist"
pt:key="pageslength"/>
                            <pt:logic.intops pt:expr="($pageslength) - 1"
pt:key="lastIndex"/>
                            <pt:logic.foreach pt:data="pageslist" pt:var="page">
                              <pt:logic.intexpr pt:expr="($pageindex) == 0"
pt:key="firstDisplayedPage"/>
                              <pt:logic.if
pt:expr="$firstDisplayedPage"><pt:logic.iftrue>
                              <pt:logic.intexpr pt:expr="($page.number) > 2"
pt:key="displayElipses"/>
                              <pt:logic.if
pt:expr="$displayElipses"><pt:logic.iftrue>
                                <li><pt:logic.value pt:value="$#137.ptmsgs_
infrastructure"/></li>
                              </pt:logic.iftrue></pt:logic.if>
                                </pt:logic.iftrue></pt:logic.if>
                              <pt:logic.existexpr pt:data="page.url"
pt:key="linkToPage"/>
                              <pt:logic.if pt:expr="$linkToPage"><pt:logic.iftrue>
                                <pt:logic.concat pt:key="paginatefunc"
pt:value1="paginate('" pt:value2="$page.url"/>
                                <pt:logic.concat pt:key="paginatefunc"
pt:value1="$paginatefunc" pt:value2="'); return false;"/>
                                <li class="ali-edit-number"><pt:core.html
pt:tag="a" href="#" onclick="$paginatefunc"><pt:logic.value
pt:value="$page.number"/></pt:core.html></li>
                              </pt:logic.iftrue><pt:logic.iffalse>
                                <li class="ali-edit-number-off"><pt:logic.value
pt:value="$page.number"/></li>
                              </pt:logic.iffalse></pt:logic.if>
                              <pt:logic.intexpr pt:expr="($pageindex) ==
($lastIndex)" pt:key="lastDisplayedPage"/>
                                <pt:logic.if
pt:expr="$lastDisplayedPage"><pt:logic.iftrue>
                                  <pt:logic.intops pt:expr="($pagination.lastpage)
- 1" pt:key="secondToLastPage"/>
                                  <pt:logic.intexpr pt:expr="($page.number) <
($secondToLastPage)" pt:key="displayElipses"/>
                                  <pt:logic.if
pt:expr="$displayElipses"><pt:logic.iftrue>
                                    <li><pt:logic.value pt:value="$#137.ptmsgs_
infrastructure"/></li>
```

```
                                </pt:logic.iftrue></pt:logic.if>
                                  </pt:logic.iftrue></pt:logic.if>
                              </pt:logic.foreach>

                              <pt:logic.existexpr pt:data="pagination.lastpageurl"
pt:key="displayLastPageLink"/>
                              <pt:logic.if
pt:expr="$displayLastPageLink"><pt:logic.iftrue>
                                <pt:logic.concat pt:key="paginatefunc"
pt:value1="paginate('" pt:value2="$pagination.lastpageurl"/>
                                <pt:logic.concat pt:key="paginatefunc"
pt:value1="$paginatefunc" pt:value2="'); return false;"/>
                                <li class="ali-edit-number">
                                  <pt:core.html pt:tag="a" href="#"
onclick="$paginatefunc"><pt:logic.value
pt:value="$pagination.lastpage"/></pt:core.html>
                                </li>
                              </pt:logic.iftrue></pt:logic.if>

                              <pt:logic.existexpr pt:data="pagination.nexturl"
pt:key="linkToNext"/>
                              <pt:logic.if pt:expr="$linkToNext"><pt:logic.iftrue>
                                  <pt:logic.concat pt:key="paginatefunc"
pt:value1="paginate('" pt:value2="$pagination.nexturl"/>
                                  <pt:logic.concat pt:key="paginatefunc"
pt:value1="$paginatefunc" pt:value2="'); return false;"/>
                                  <li><pt:core.html pt:tag="a" href="#"
onclick="$paginatefunc" title="$#35.ptmsgs_portalbrowsingmsgs"><pt:logic.value
pt:value="$#208.ptmsgs_portalinfrastructure"/></pt:core.html></li>
                                </pt:logic.iftrue><pt:logic.iffalse>
                                  <li><pt:logic.value pt:value="$#208.ptmsgs_
portalinfrastructure"/></li>
                                </pt:logic.iffalse></pt:logic.if>
                            </ul>
                        </div>
                    </td>
                </tr>
            </table>
            </pt:logic.iftrue></pt:logic.if>
        </div>
        <div id="ali-edit-footer">
        <div id="ali-edit-botleft"></div>
        <div id="ali-edit-botright"></div>
        </div>
        </div>
</pt:logic.iftrue><pt:logic.iffalse>
        <div id="ali-edit-portlets">
            <div id="ali-edit-search-container">
                <div id="ali-edit-portlets-text"><pt:logic.value
pt:value="$#1946.ptmsgs_portalbrowsingmsgs"/></div>
            </div>
            <div id="ali-edit-footer">
            <div id="ali-edit-botleft"></div>
            <div id="ali-edit-botright"></div>
            </div>
        </div>
</pt:logic.iffalse></pt:logic.if>
</div>
</div>
```

3.8 Creating a Community Selection Adaptive Page Layout

Community Selection layouts allow you to customize the Join Communities flyout editor used to join communities from a portal page.

The pt:joincommunitypageditor library contains the `<pt:joincommunitypageeditor.addcommunitiesflyoutdata>` tag to add a custom community selection flyout editor (DHTML) to a portal page.

The rest of the tags in the library are used to create the flyout editor in the Community Selection adaptive page layout.

- The `<pt:joincommunitypageeditor.communityjs>` tag generates the JavaScript functions required for community preview and invitation.
- The `pt:joincommunitypageeditor.communitydata` tag generates the data required to show a list of communities for the page editor and stores it in memory using the variable name specified by the `id` attribute. Each result is a DataObject with the following variables:

Variable	Description
name	The name of the community.
description	The description of the community.
id	The ID of the community object in the portal.
isalreadyjoined	True the user has already joined the community.
invitationid	The ID to be used in the invitation JavaScript function. If the value is -1, the invitation is disabled.
lastmodified - mandatory - the last modified date of the portlet	The last modified date of the community.
mandatory	True if the community is mandatory for the current user.

- The `<pt:joincommunitypageeditor.joincommunitysearchform>` tag generates the form and hidden inputs necessary to search for communities. It does not generate the text input or search button. The text input must to be defined in the `in_tx_query` parameter. The `<pt:joincommunitypageeditor.paginationdata>` tag generates the data for pagination links for the search results.
- The `<pt:joincommunitypageeditor.communitybrowsemode>` tag displays the JavaScript necessary to switch to browse mode. The `<pt:joincommunitypageeditorbrowsesubfoldersdata>` tag stores a list of subfolders of the current folder in memory (only populated if the page is in browse mode). The `<pt:joincommunitypageeditor.browsebreadcrumbsdata>` tag stores a list of subfolders of the current folder in memory.

These tags are used in the same order as the Portlet Selection adaptive page layout; for an example implementation, see the previous section. For an example of a Community Selection page layout, see the templates included with the portal installation.

3.9 Creating a My Account Adaptive Page Layout

The My Account tags in the pt:ptui library allow you to create a customized My Account page or use My Account data within other adaptive page layouts.

- The `pt:ptui.myaccount` tag displays the My Account link to the account settings page if the user is logged in as a non-guest user. If this tag is used as a singleton tag, the text "My Account" will be used. If opening and closing tags are included, the HTML inside the tag will be used.
- The `pt:ptui.myaccountdata`tag stores the list of My Account setting items. Each `SettingItem` object contains three variables: name, description and url.

The example below uses tags from the pt:ptui brary to define My Account components, and logic tags to iterate through the setting items. For detailed information on standard adaptive tags, including logic tags, see the *Oracle WebCenter Interaction Web Service Development Guide*.

```
<span xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
<table align="left" border="0" cellpadding="5" cellspacing="0" width="100%">
<tr class="dirHeaderBg">
<td colspan="1" align="left" valign="top">
<span class="dirHeader">
<pt:logic.value pt:value="$#1604.ptmsgs_portalbrowsingmsgs"/>
</span>
</td>
</tr>
<pt:ptui.myaccountdata pt:id="mylinks" />
<pt:logic.foreach pt:data="mylinks" pt:var="mylink">
<tr>
<td colspan="1" class="menuText">
<pt:core.html pt:tag="a" href="$mylink.url" title="$mylink.name"><pt:logic.value
pt:value="$mylink.name"/></pt:core.html>
<br/>
<pt:logic.value pt:value="$mylink.description"/>
</td>
</tr>
<tr>
<td>
<br/>
</td>
</tr>
</pt:logic.foreach>
</table>
</span>
```

3.10 Creating an Error Page Adaptive Page Layout

The Error tags in the pt:ptui library allow you to create a customized Error page or use error data within other adaptive page layouts. These tags display the error text, which can be formatted as desired. If these tags are displayed on a page, errors will no longer be displayed in the normal error location.

- The `pt:ptui.error` tag displays errors on the page. If the `errortext` tag is included inside this tag, the contents of this tag will only be processed if there is an error. If the child tag is not present, error messages will be formatted and displayed from this tag in the same style as used by the portal.
- The `pt:ptui.errortext` tag displays the current error text on the page. Only the first error message will be displayed. Other errors, as well as exception stack

traces and extended error messages will be ignored. Note: This tag does not display the contents of the tag and should only be used as a singleton tag, rather than as a tag with both an open and close tag.

- The `pt:ptui.errorextendedmessage` tag displays the extended error text on the page. Only the first error message will be displayed. Other errors, as well as exception stack traces will be ignored. Note: This tag does not display the contents of the tag and should only be used as a singleton tag, rather than as a tag with both an open and close tag.

The example below uses tags from the pt:ptui brary to define error display, and additional adaptive tags to access images and portal message strings. For detailed information on standard adaptive tags, see the *Oracle WebCenter Interaction Web Service Development Guide*.

```
<span xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
<pt:ptui.error>
<table border="0" cellpadding="5" cellspacing="0" width="100%">
<tbody>
<tr class="alertBg">
<td colspan="1" class="alertErrorTitle" align="center" width="80">
<pt:core.html pt:tag="img" src="pt://images/plumtree/portal/public/img/icon_
error.gif" alt="$#624.ptmsgs_portalbrowsingmsgs" border="0" height="20"
width="20"/>
</td>
<td colspan="1" class="alertErrorTitle" align="left" width="100%">
<span class="alertErrorTitle" >
<pt:logic.value pt:value="$#624.ptmsgs_portalbrowsingmsgs"/>
<pt:logic.value pt:value=" - "/>
<pt:ptui.errortext/>
<pt:logic.value  pt:encode="0" pt:value="<!--"/>
<pt:logic.value pt:value="$#1949.ptmsgs_portalbrowsingmsgs" />
<pt:ptui.errorextendedmessage/>
<pt:logic.value pt:encode="0" pt:value="-->" />
</span>
</td>
<td colspan="1" align="right" width="0">
<!-- Comment -->
</td>
</tr>
</tbody>
</table>
</pt:ptui.error>
</span>
```

3.11 Creating an iPhone Adaptive Page Layout

iPhone adaptive page layouts use the same tags as the adaptive page layouts described in the previous sections, but reference a different style sheet and JavaScript file. For examples of iPhone layouts, see the iphone* templates provided with the portal installation.

4

Using Adaptive Styles (CSS Customization)

Oracle WebCenter Interaction includes a UI customization framework based on Adaptive Page Layouts. If you are using Adaptive Page Layouts in your portal implementation, additional UI customization options are available through the portal CSS file. For details on Adaptive Page Layouts, see Chapter 3, "Using Adaptive Page Layouts" and the *Administrator Guide for Oracle WebCenter Interaction*.

This chapter provides details on the types of CSS elements available, examples of customizing page and portlet style and layout, and information on creating localized stylesheets. All customizations are made in the **mainstyle.css** file located in the %PT_HOME%\ptimages\imageserver\plumtree\common\private\css\ folder on the portal image service. The portal CSS template file follows standard CSS syntax rules. For details on CSS, see http://www.w3.org/Style/CSS/.

Note: If you are not using Adaptive Page Layouts, you can still customize the portal page layout using CSS; see Chapter 5, "Customizing Portal Layout Using CSS - Legacy User Interface".

4.1 Adaptive Styles Base Page Elements

Base page elements control style and layout the basic part of the portal page, including the action bar and banner. This is not a complete list; for all available elements, see the mainstyle.css file.

Element	Page Component	Example
body	The main body of the page (any text-based content not inside another element).	`body { font-family: Verdana, Arial, Helvetica, sans-serif; font-size:100%; color: #FFFFFF; margin: 0px; }`
a:hover	Links in the main body of the page on mouse-over.	`a:hover { text-decoration:underline; }`

Element	Page Component	Example
#ali-actionbar	The action bar at the top of the page.	`#ali-actionbar { width:100%; height:22px; background-color: #0A2F66; background-image: url(../img/banner_action_ bkg.jpg); background-repeat: repeat-x; font-family:Verdana, Arial, Helvetica, sans-serif; min-width:980px; }`
#ali-banner	The main content in the portal banner. This area usually contains company branding, which can include an image referenced in the background-image: url parameter.	`#ali-banner { width:100%; height:80px; background-color: #1D54A6; background-image: url(../img/banner_bkg.jpg); background-repeat: repeat-x; font-family:Verdana, Arial, Helvetica, sans-serif; min-width:980px; }`
#ali-bannerWelcome	The welcome text displayed in the portal banner.	`#ali-bannerWelcome { float:left; background:none; color:#B2D8FF; font-size:.8em; letter-spacing:1px; padding:6px 4px 0px 12px; }`
#ali-bannerNav	The navigation section of the portal banner.	`#ali-bannerNav { float:right; background:none; color:#A6CFF6; font-size:.8em; letter-spacing:1px; padding:6px 12px 4px 4px; text-align:right; }`
#ali-bannerLogo	The logo in the portal banner.	`#ali-bannerLogo { clear:left; float:left; padding:10px 10px 0px 14px; }`

Element	Page Component	Example
#ali-footer	The portal page footer.	`#ali-footer { clear:both; width:100%; height:22px; background-image:url(../img/footer_bkg.gif); background-repeat:repeat-x; color:#FFFFFF; font-family:Arial, Helvetica, sans-serif; font-size:.7em; letter-spacing:1px; text-align:center; margin-top:48px; padding:4px 0 0 0; min-width:980px; }`

4.2 Adaptive Styles Navigation Elements

Navigation elements control style and layout for the navigation section of the portal page, including the menus and breadcrumbs. This is not a complete list; for all available elements, see the mainstyle.css file.

Note: Parameters marked "!important" are related to drop-down lists within navigation elements. These paramaters can be customized, but must not be removed; eliminating them will cause the drop-downs either to not work or to distort.

Element	Navigation Component	Example
#ali-mainNav	The main navigation section of the portal page.	`#ali-mainNav { clear:left; float:left; width:100%; height:30px; background-color:#3068CF; background-image:url(../img/main_nav_tab.gif); background-repeat:repeat-x; border-bottom:solid 1px #5083CB; letter-spacing: 1px; min-width:980px; }`

Element	Navigation Component	Example
#ali-nav	All lists within navigation sections in the portal page.	`#ali-nav, #ali-nav ul { padding: 0; margin: 0; list-style: none; line-height: 1; }`
a.ali-navmenu	The portal navigation menu.	`a.ali-navmenu { color:#385ABD !important; width:170px !important; background-color:#F1F5F9 !important; border-bottom:solid 1px #146BC5; font-size:1.15em; }`
a.ali-nav-actions	A separate style for items in the portal navigation menu that are actions, as opposed to links to pages/sections on the portal. Sets a different background color for these items in the menu to offer a clear distinction between menu navigation links and action links.	`a.ali-nav-actions { color:#2B49AC !important; width:170px !important; background-color:#C9D4E9 !important; border-bottom:solid 1px #146BC5; font-size:1.15em; }`
#ali-secondNavBar	The second-level navigation bar is used on the User Profile page and community pages. A second bar appears below the main navigation bar in a different color and is mainly used to list pages within a community.	`#ali-secondNavBar { clear:both; float:left; width:100%; padding:0; margin:0 0 -21px 0; background-image:url(../img/nav_2nd_pages.gif); background-repeat:repeat-x; letter-spacing:0; font:bold .725em Helvetica; min-width:980px; }`

Element	Navigation Component	Example
#ali-secondPages	The list of secondary pages in a community or other sections of the portal that use a second level of navigation.	`#ali-secondPages { float:left; color:#51617a; background-color:none; width:80%; }`
#ali-secondSub	The drop-down menu for pages within a community (sub-communities).	`#ali-secondSub { float:right; padding:0; margin:0; background-image:url(../img/nav_2nd_sub.gif); background-repeat:repeat-x; border-bottom:solid 1px #82A8F3; border-left:solid 1px #82A8F3; }`
#ali-secondNav	The list of links in the sub-community drop-down menu.	`#ali-secondNav, #ali-secondNav ul { padding: 0; margin: 0; list-style: none; }`
a.ali-secondMenu	The link color and background color for the sub-community drop-down menu.	`a.ali-secondMenu { color:#4467CB; font:bold 8pt Arial, Helvetica, sans-serif; width:161px; background-color:#F1F6FF; border-bottom:solid 1px #83A1D8; }`
#ali-community-name	The look and placement of the community (or home page) name for the second navigation bar.	`#ali-community-name { position:relative; left:-40px; background-image:url(../img/nav_2nd_home.gif); background-repeat:repeat-x; border-right:solid 1px #82A8F3; letter-spacing:1px; color:#5374A1; padding:8px 12px 6px 12px; margin-right:-3px; }`

Element	Navigation Component	Example
#ali-breadcrumb	The breadcrumb list displayed at the top of the portal page content section.	`#ali-breadcrumb { float:left; margin:4px 0 0 12px; padding:0; color:#888888; padding:0; font-family:Helvetica, Arial, sans-serif; font-size:.7em; letter-spacing:1px; }`

4.3 Adaptive Styles Search Elements

Search elements control style and layout for the search components on the portal page, including search forms and the search browse page. This is not a complete list; for all available elements, see the mainstyle.css file.

Element	Search Component	Example
#ali-bannerSearch	The basic (banner) search form in the portal banner.	`#ali-bannerSearch { clear:right; float:right; position:relative; width:400px; padding: 18px 24px 0px 24px; }`
#ali-searchAdvanced	The div below the banner search box that contains the advanced search text link.	`#ali-searchAdvanced { clear:right; float:right; width:320px; margin:0 0 0 0; padding:0 90px 0 0; }`
input.ali-searchBox	The search input box in the search form. Applies to the input box itself only when it appears in the banner.	`input.ali-searchBox { color:#999999; border:outset 1px; padding: 1px; }`

Element	Search Component	Example
input.ali-searchButton	The search button in the search form. Applies to the search button in the banner only.	`input.ali-searchButton { background-image:url(../img/button_search_gradient.gif); background-repeat:repeat-x; border:outset 0px; padding:2px 6px; margin-left:4px; color:#1A48A4; font-size:.8em; }`
input[type="button"]:hover	The search button on mouse-over. Applies to the search button in the banner.	`input[type="button"]:hover { color:#FF6000; }`
#ali-search-modifier-container	The search form on the search results/browse page.	`#ali-search-modifier-container { clear:both; float:left; width:99%; margin:6px 0 0 0; padding:0; min-width:980px; }`
#ali-search-results	The search results section on the search browse page.	`#ali-search-results { clear:both; float:left; width:78%; min-width:625px; margin:12px 0 0 32px; padding:0; }`

4.4 Adaptive Styles Editing Elements

Editing elements control style and layout for editing components on the portal page, including the page editing elements near the navigation breadcrumb and the flyout page and portlet editor elements. This is not a complete list; for all available elements, see the mainstyle.css file.

Element	Editing Component	Example
#ali-pageEdit	The portal actions such as Edit Page, Create Page, etc. Appears opposite the breadcrumb at the top or the portal page.	`#ali-pageEdit { float:right; padding: 0px 0px 12px 0px; font-family:Helvetica, Arial, sans-serif; color:#96b7ED; text-align:right; }`
#ali-edit-container	The div that contains the flyout page editor.	`#ali-edit-container { clear:left; float:left; width:97%; margin:0px 12px 12px 12px; min-width:950px; }`
#ali-edit-toolbar	The bar at the top of the flyout page editor and contains the "Edit Page" text and the "X" (close editor) button for the editor.	`#ali-edit-toolbar { float:left; width:100%; height:21px; margin-top:6px; background-color:#6B91C0; background-image:url(../img/edit_title_topbar.gif); background-repeat:repeat-x; color:#FFFFFF; font-family:Verdana, Arial, Helvetica, sans-serif; font-size:11px; font-weight:bold; letter-spacing:1px; }`

Element	Editing Component	Example

#ali-edit-cornerleft and #ali-edit-cornerright	The top right and top left rounded corners for the flyout page editor.	`#ali-edit-cornerleft {`
		`clear:left;`
		`float:left;`
		`width:8px;`
		`height:21px;`
		
		`background-image:url(../img/edit_corner_topleft.gif);`
		
		`background-repeat:no-repeat;`
		`}`
		
		`#ali-edit-cornerright {`
		`float:right;`
		`position:relative;`
		`right:-2px;`
		`width:8px;`
		`height:21px;`
		
		`background-image:url(../img/edit_corner_topright.gif);`
		
		`background-repeat:no-repeat;`
		`margin:0;`
		`padding:0;`
		`}`

#ali-edit-content	The div containing the main section of the flyout editor below the toolbar and above the rounded corners at the bottom.	`#ali-edit-content {`
		`width:100%;`
		`border-left:solid 1px #6B91C0;`
		`border-right:solid 1px #6B91C0;`
		
		`background-color:#ECEFF4;`
		`color:#6B91C0;`
		`}`

Element	Editing Component	Example
#ali-edit-tabs-container	Can be used to set tabbed navigation within the flyout page editor. (Not used in the default implementation of the flyout page editor.)	`#ali-edit-tabs-container { clear:left; float:left; width:100%; margin:0; padding:0; height:30px; background-image:url(../img/edit_tab_gradient.gif); background-repeat:repeat-x; background-color:#C8DCFF; border-bottom:solid 1px #7497C4; border-right:solid 1px #7497C4; border-left:solid 1px #7497C4; }`
.ali-edit-tabs	Can be used to set the style of tabbed navigation within the flyout page editor. (Not used in the default implementation of the flyout page editor.)	`.ali-edit-tabs { float:left; margin:6px 0 0 0; padding:0; background-color:#CFd3E7; border-left:solid 1px #7497C4; border-top:solid 1px #7497C4; border-right:solid 1px #7497C4; }`
#ali-edit-footer	Contains the elements for the bottom of the flyout page editor, including the bottom outline and the left and right rounded corners.	`#ali-edit-footer { width:100%; clear:left; float:left; background-image:url(../img/edit_bot.gif); background-repeat:repeat-x; height:8px; }`

Element	Editing Component	Example
#ali-edit-botleft and #ali-edit-botright	The divs that contain the rounded corners for the bottom right and bottom left corners of the flyout page editor. The images for these corners are specified in the style as the background image.	`#ali-edit-botleft { clear:left; float:left; width:8px; height:8px; position:relative; left:-1px; background-image:url(../img/edit_corner_botleft.gif); background-repeat:no-repeat; }`
#ali-edit-table	Flyout portlet editor table.	`#ali-edit-table { font-size:1em; color:#000000; margin:0px; }`
#ali-edit-portlets	Flyout portlet editor tab.	`#ali-edit-portlets { float:left; width:100%; border-left:solid 1px #6B91C0; border-right:solid 1px #6B91C0; background-color:#ECEFF4; color:#000000; font-size:11px; }`
#ali-edit-portlets-search	Flyout portlet editor search form.	`#ali-edit-portlets-search { float:left; padding:0 18px 0 14px; border-right:solid 1px #D5D6DA; line-height:31px; }`
#ali-edit-breadcrumb	Flyout portlet editor breadcrumbs.	`#ali-edit-breadcrumb { float:left; color:#2B4A7B; padding: 2px 24px 12px 2px; font-family:Helvetica, Arial, sans-serif; font-size:11px; letter-spacing:1px; }`

Element	Editing Component	Example
#ali-edit-main-col1	Flyout portlet editor folder display.	`#ali-edit-main-col1 { float:left; width:212px; margin:0; padding:0; }`

4.5 Adaptive Styles Directory Elements

Directory elements control style and layout for components in the Directory. This is not a complete list; for all available elements, see the mainstyle.css file.

Element	Directory Component	Example
#ali-kd-title	The title displayed on the Directory page.	`#ali-kd-title { float:left; height:22px; padding:4px 12px 0 12px; border-right:solid 1px #9BBEEE; color:#7197c6; font-size:.8em; font-weight:bold; letter-spacing:2px; }`

Element	Directory Component	Example
#ali-kd-main*	These elements control the main page of the Directory	`#ali-kd-main-bar { clear:both; float:left; width:100%; min-width:980px; margin:0; padding:0; border-top:solid 1px #DBD9D9; border-bottom:solid 1px #C9CED9; background-color:#CFDFFF; background-repeat:repeat-x; height:7px; } #ali-kd-main-col1 { float:left; width:212px; margin:0 60px 48px 36px; } .ali-kd-main-header { width:100%; margin-top:24px; padding:2px 4px; background-color:#E5E9F6; border-bottom:solid 1px #C6CAD4; }`
#ali-kd-breadcrumb	The breadcrumb displayed on the Directory page.	`#ali-kd-breadcrumb { float:left; height:22px; padding:5px 0 0 0; margin-left:-28px; font-size:.75em; font-weight:bold; color:#2B4A7B; }`

Element	Directory Component	Example
#ali-kd-sorting-bar	The div in the Directory and search results pages that contains the pull-down menu for sorting results or listings by "items per page", "item type", etc.	`#ali-kd-sorting-bar { clear:both; float:left; width:100%; min-width:1000px; margin:0; padding:0; height:31px; border-top:solid 1px #D5D4D4; border-bottom:solid 1px #A8B8D9; background-image:url(../img/kd_sort_bkg.gif); background-repeat:repeat-x; font-family:Verdana, Arial, Helvetica, sans-serif; font-size:.8em; color:#000000; font-weight:normal; }`
#ali-kd-documents	The div for listings of items in Directory folders.	`#ali-kd-documents { clear:both; float:left; width:64%; min-width:625px; min-height:500px; margin:0 0 0 32px; padding:6px 64px 48px 0; background-image:url(../img/kd_subfolders_bkg.gif); background-repeat:repeat-y; background-position:right; }`
.ali-kd-doc-office.ali-kd-doc-web.ali-kd-doc-text	The icon and text for specific document types.	`.ali-kd-doc-office { clear:both; color:#000000; font-size:.8em; padding:0 24px 24px 34px; background-image:url(../img/icon_officedoc_24px.gif); background-repeat:no-repeat; }`

Element	Directory Component	Example
#ali-kd-pagenav	Directory navigation section.	`#ali-kd-pagenav { clear:both; float:right; line-height:2em; color:#A1B2C4; font-size:.7em; padding-right:24px; margin-bottom:12px; }`
#ali-kd-side	Directory subfolders and related links sections.	`#ali-kd-side { float:left; right:18px; width:25%; margin:0 0 0 -12px; padding:20px 0 48px 0; min-width:250px; letter-spacing:1px; font-family:Helvetica, sans-serif; font-size:.8em; font-weight:bold; color:#7197C6; }`
#ali-kd-subfolder li	Directory subfolder links.	`#ali-kd-subfolder li { padding:2px 0 2px 24px; list-style:none; background-image:url(../img/icon_folder_16px.gif); background-repeat:no-repeat; background-position:0 50%; }`
.ali-kd-related a	Directory related links.	`.ali-kd-related a { font-size:90%; font-family:Arial, Helvetica, sans-serif; font-weight:normal; color:#3761B7; text-decoration:none; }`

4.6 Adaptive Styles Portlet Elements

These elements control style and layout for portlet elements. This is not a complete list; for all available elements, see the mainstyle.css file. For portlet flyout editor elements, see Section 4.4, "Adaptive Styles Editing Elements".

Element	Portlet Component	Example
.ali-portlet-container	Contains the nested elements of the portlet toolbar, controls rounded corners and content.	`.ali-portlet-container { min-width:250px; margin:4px 0px 6px 0px; }`
.ali-portlet-cornerleft and .ali-portlet-cornerright	Divs that contain the rounded corners for the top right and top left corners of the portlet. The images for these corners are specified in the style as the background image.	`.ali-portlet-cornerright { float:right; width:8px; height:21px; position:relative; right:-2px; background-image:url(../img/portlet_corner_topright.gif); background-repeat:no-repeat; margin:0; padding:0; }`
.ali-portlet-toolbar	The top section of the portlet, where the title of the portlet sits along with action buttons such as minimize, edit and refresh.	`.ali-portlet-toolbar { width:100%; height:21px; background-color:#5C91D8; background-image:url(../img/portlet_title_bar.gif); background-repeat:repeat-x; color:#FFFFFF; font-family:Verdana, Arial, Helvetica, sans-serif; font-size:1.1em; font-weight:bold; letter-spacing:1px; }`
.ali-portlet-controlone and .ali-portlet-controltwo	Used to position the action buttons in the portlet toolbar for actions such as minimize, edit and refresh.	`.ali-portlet-controltwo { float:right; width:13px; margin-bottom:-13px; padding:0px 0px 0px 6px; border: solid 1px #FF0000; }`

Element	Portlet Component	Example
.ali-portlet-content	The div that contains the main content of the portlet. Sets the left and right outlines of the portlet.	`.ali-portlet-content { clear:left; width:100%; border-left:solid 1px #6B91C0; border-right:solid 1px #6B91C0; color:#6B91C0; }`
.ali-portlet-footer	Contains the elements for the bottom of the portlet, including the bottom outline and the left and right rounded corners.	`.ali-portlet-footer { width:100%; background-image:url(../img/portlet_bot.gif); background-repeat:repeat-x; height:8px; }`
.ali-portlet-botleft and.ali-portlet-botright	Divs that contain the rounded corners for the bottom right and bottom left corners of the portlet. The images for these corners are specified in the style as the background image.	`.ali-portlet-botright { float:right; width:8px; height:8px; position:relative; right:-2px; background-image:url(../img/portlet_corner_botright.gif); background-repeat:no-repeat; }`

4.7 Adaptive Styles User Elements

User elements control style and layout for user-related components, including user profile, user activity stream, user friends and user information components. This is not a complete list; for all available elements, see the mainstyle.css file.

Element	User Component	Example
#ali-user-navbar	The user profile navigation menu in portal navigation.	`#ali-user-navbar { clear:both; width:100%; padding:0 0 2px 0; margin:0; height:27px; background-image:url(../img/nav_2nd_pages.gif); background-repeat:repeat-x; letter-spacing:1px; font:bold .725em Helvetica; line-height:24px; min-width:980px; }`
.ali-user-activity*	These elements control the display of user activity stream components. The User Activities portlet usually appears on the user profile page.	`.ali-user-activity-pulldown { clear:both; float:right; width:99%; padding:3px 16px 0 0; color:#000000; text-align:right; font-size:.75em; } .ali-user-activity-stream { margin: 8px 0 0 0; padding:0 0 0 4px; background-color:#EFF2FA; border-bottom:solid 1px #DBDEE4; font-size:.75em; font-weight:bold; letter-spacing:1px; }`
.ali-user-friends*	These elements control the display of the user friends components. The User's Friends list portlet usually appears on the User Profile page. .	`.ali-user-pulldown { clear:both; float:right; width:99%; padding:3px 16px 0 0; color:#000000; font-size:.75em; text-align:right; } .ali-friends-info-title { padding-right:6px; text-align:right; color:#6E7686; font-size:.75em; font-weight:bold; letter-spacing:1px; }`

Element	User Component	Example

#ali-user-geninfo*	These elements control the display of the user general information components, the main information on the User Profile page.	`#ali-user-geninfo-edit { float:right; width:100px; margin:-4px; padding:6px 12px 6px 12px;`
		
		`background-color:#EFF3FF; border-left:solid 1px #C4C8DB; text-align:center; }`
		
		`.ali-user-geninfo-title { padding-right:6px; text-align:right; color:#6E7686; font-size:.65em; font-weight:bold; letter-spacing:0; }`

#ali-user-search	The search section in the user general info component.	`#ali-user-search { float:right; padding:3px 24px; margin:0; height:22px; width:310px;`
		
		`background-image:url(../img/nav_2nd_sub.gif); background-repeat:repeat-x; border-left:solid 1px #6f90cf; }`

4.8 Using Adaptive Styles to Customize Portlet Style and Layout

Adaptive styles allow you to customize specific portlets using the unique ID assigned to each portlet, or use CSS classes to modify the design of a group of portlets. You can also set constraints for portlets, including limiting a specific portlet to a three-column layout or preventing users from collapsing portlets. For an introduction to portlet style elements, see . Section 4.6, "Adaptive Styles Portlet Elements"

4.8.1 Syntax

To apply a CSS tag to a specific portlet, use the portlet ID. the example below increases the space around the portlet title for the portlet with ID 43. (You can also define basic styles for a specific portlet within the portlet code.)

```
#pt-portlet-43 .ali-portlet-title
{
        padding:8px 0 0 0;
}
```

You can also apply styles to groups of portlets, including those on a specific page or in a specific community. To apply styles to a portlet on a specific page or community, use the page or community ID. The example below makes the same modification as above for all the portlet on the page will ID 100.

```
#pt-page-100 .ali-portlet-title
{
        padding:8px 0 0 0;
}
```

4.8.2 Style Customizations

The mainstyle.css file allows you to make a wide range of style changes to portlets. For example, you can change the color scheme of portlets as shown in the example below.

```
#pt-portlet-43 .ali-portlet-content
{
        clear:left;
        width:100%;
        border-left:solid 1px #6B91C0;
        border-right:solid 1px #6B91C0;
        color:#6B91C0;
}
```

4.8.3 Constraints

The mainstyle.css file allows you to set constraints for portlets. For example, you can set the width of a portlet for a specific page or set of pages. You can define portlet settings by page, layout/column, or community. The example below limits the portlet with ID 43 to a width of 250 pixels on the page with ID 100.

```
#pt-page-100 #pt-portlet-43
{
        width: 250px;
}
```

4.9 Using Adaptive Styles to Customize Page Layout

Adaptive styles allow you to modify page layout and design using the portal CSS template file. You can also use CSS to hide specific functionality exposed in the portal page. This page provides basic syntax rules and customization examples.

4.9.1 Syntax

To apply styles to a specific page, use the page ID. The example below sets the background color for the page with ID 100.

```
#pt-page-100
{
        background-color: red;
}
```

You can change style settings for a specific user or type of user (administrator or guest). The example below displays a special header image on all browse-mode pages

for guests. To modify a style for a specific user, replace "guest" with the name of the appropriate portal User object (e.g., .ptPageUser-mycompany domain ad\Joe Smith).

```
.ptPageUser-guest #pt-header
{
        background-image: url(/imageserver/plumtree/portal/private/img/example_
guest.gif);
}
```

4.9.2 Style and Branding Customizations

The mainstyle.css file allows you to make a wide range of style changes to the portal page, including adding custom branding and color schemes. For example, you can add a custom image to the portal banner as shown in the example below.

```
#ali-banner    {
        width:100%;
        height:80px;
        background-color: #1D54A6;
        background-image: url(../img/MyCompany_bkg.jpg);
        background-repeat: repeat-x;
        font-family:Verdana, Arial, Helvetica, sans-serif;
        min-width:980px;
}
```

You can also modify the background color for a single page or a specific community. The example below sets the background color for the community with ID 200.

```
.ptCommunity-200
{
        background-color: #AAA;
}
```

4.9.3 Page Element Customizations

The mainstyle.css file allows you to modify the style of form elements in the portal page, including text boxes and buttons. For example, the code below expands the size of the banner search box.

```
#ali-bannerSearch {
        clear:right;
        float:right;
        position:relative;
        width:600px;
        padding: 18px 24px 0px 24px;
}
```

4.10 Implementing Localized Stylesheets for Adaptive Page Layouts

To provide language-specific stylesheets for internationalized portal implementations, create a localized version of the portal stylesheet and map the language to the each style sheet in the CustomStyles.xml file.

The CustomStyles.xml file is located in the %PT_HOME%\settings\portal\ folder on the portal server. This file also contains default mappings to legacy stylesheets to support any products that do not use Adaptive Page Layouts.

Note: The language-specific stylesheet mappings in CustomStyles.xml only apply to pages that use Adaptive Page Layouts. For details on localizing stylesheet for legacy layouts, see Chapter 5.3, "Adding New Language Style Sheets".

1. Create a localized version of the mainstyle.css file for each language. For example, mystyle-ar.css for Arabic.
2. Modify CustomStyles.xml to specify stylesheets for each supported language. For example, to use mystyle-ar.css for Arabic, add the following mapping to CustomStyles.xml:

```
<StyleSettings>
  <cssMapping>
    <language>ar</language>
    <styles>mystyle-ar.css</styles>
  </cssMapping>
</StyleSettings>
```

3. Include the `<pt://styles>` adaptive tag in the head element of any page that should use a localized stylesheet. The head element must also include the `<pt.standard.stylesheets>` tag to reference the legacy stylesheet, which contains the legacy portlet styles required by any preexisting portlets and by the admin UI. For details on these tags, see the *Oracle WebCenter Interaction Web Service Development Guide*..

```
<head>
<pt.standard.stylesheets/>
<link href="pt://styles" type="text/css" rel="styleSheet"></link>
... </head>
```

Note: The `<pt://styles>` tag can only be used to implement localized stylesheets in pages that use Adaptive Page Layouts.

5

Customizing Portal Layout Using CSS - Legacy User Interface

Oracle WebCenter Interaction provides a CSS template file that contains a wide range of CSS classes and IDs to facilitate customization. The CSS template file is located in the PT_HOME\ptimages\tools\cssmill\templates directory in the portal Image Service.

The structure of the portal page is designed to support customizations on global, per user, per community, per product, per page, and per portlet levels. The portal supports 18 different color schemes and 8 languages out of the box. The CSS Style Sheet Mill allows you to implement custom color schemes and add new language style sheets easily without modifying portal source code.

Note: This approach is provided to support the legacy user interface; the recommended way to customize the portal page is using Adaptive Layouts and Adaptive Styles. For details, see Chapter 3, "Using Adaptive Page Layouts"and Chapter 4, "Using Adaptive Styles (CSS Customization)".

5.1 Customizing Portal Page Layout and Design

All major and minor page elements are assigned either a CSS ID or class, or both. Uniquely identifiable objects (such as a specific page) are given unique ids. Identifiable classes of objects (such as pages in a specific community) are given classes. Each major region of the page is treated as a named box. For an introduction to the portal page, see Chapter 2, "Portal Page Layout".

These changes to the portal UI make it possible to modify page layout and design using the portal CSS template file. You can also use CSS to hide specific functionality exposed in the portal page. This section provides basic syntax rules and customization examples.

5.1.1 Syntax Guidelines

The portal CSS template file follows standard CSS syntax rules. For details on CSS, see http://www.w3.org/Style/CSS/. Below are some basic rules to keep in mind when modifying page styles.

To apply styles to a specific page, use the page ID. The example below sets the background color for the page with ID 1.

```
#pt-page-1
{
background-color: red;
}
```

You can change style settings for a specific user or type of user (administrator or guest). The example below displays a special header image on all browse-mode pages for guests. To modify a style for a specific user, replace "guest" with the name of the appropriate portal user object (for example, `.ptPageUser-mycompany domain ad\Joe Smith`).

```
.ptPageUser-guest #pt-header
{
background-image: url(/imageserver/plumtree/portal/private/img/example_guest.gif);
}
```

You can also change styles for specific communities. The example below sets the background color for the community with ID 200.

```
.ptCommunity-200
{
background-color: #AAA;
}
```

5.1.2 Customizing Layout

The portal CSS template file allows you to make a wide range of changes to the layout of the portal page. Below are some examples of layout customizations.

- **Modify page width**: Specify whether a page spans the whole window or a portion of the window. This provides support for specific audiences such as those on smaller monitors. The example below limits the page to 800 x 200 pixels.

  ```
  .portalContent
  {
  width: 800px;
  height: 200px;
  overflow: auto;
  }
  ```

- **Change navigation tab location**: Modify the location of the portal navigation tabs. You can apply changes to the entire portal, or to specific pages or groups of pages. The example below sets the tabs to appear in the center of the page header.

  ```
  #pt-user-nav
  {
  display: inline;
  margin-left: 15px;
  margin-right: 15px;
  }
  ```

5.1.3 Customizing Style

The portal CSS template file allows you to make a wide range of style changes to the portal page. Below are some examples of style customizations.

- **Customize portal banners and footers**: Change the look and feel for portal banners and footers. You can apply changes to the entire portal, or to specific pages or groups of pages. For example, the code below changes the footer height.

  ```
  #pt-footer { height: 36px; }
  ```

  The code below hides the footer on the page with ID 1.

  ```
  #pt-page-1 #pt-footer { display: none; }
  ```

- **Change the background color for a specific page or community**: Modify the background color for a single page or a specific community. The example below sets the background color for the community with ID 200. To change the color

scheme for the entire portal, modify the style sheet as explained in the next section, Section 5.4, "Deploying Portal Style Sheet Customizations (CSS Mill)".

```
.ptCommunity-200
{
background-color: #AAA;
}
```

- **Change the background for a specific user**: Modify the background of the portal for a specific user or type of user (administrator or guest). The example below displays a background image on all browse-mode pages for administrators.

```
.ptPageUser-administrator
{
background-image: url(/imageserver/plumtree/portal/private/img/example_
administrator.gif);
}
```

- **Customize portal navigation tabs**: Define the dimension of portal tabs.

```
#pt-user-nav
{
 width: 25px;
 }
```

- **Customize form elements**: As with any CSS implementation, you can use the portal CSS template file to control text box sizing, button colors and fonts, and more.
- **Reference images**: Reference images through CSS, including banner and branding images, and background images applied to page components. The example below displays a special header image on all browse-mode pages for guests.

```
.ptPageUser-guest #pt-header
{
background-image: url(/imageserver/plumtree/portal/private/img/example_
guest.gif);
}
```

5.1.4 Setting Constraints

The portal CSS template file allows you to remove specific functionality from the page for a group of users or for a specific page or community.

Note: Using CSS to hide functionality is not a secure means of preventing user-server interaction. All examples are for demonstration purposes only and are not meant to imply a complete solution to any overall security scheme.

- **Disable specific functionality**: Turn off controls for a specific group of users or for a specific page or community. You can disable navigation, search, and a variety of links, including My Home, My Account, Login/Logout and Help. The example below disables search controls for all guests.

```
.ptPageUser-guest #pt-search-controls
{
display: none;
}
```

5.1.5 Changing the Portal Color Scheme

The portal comes with 18 standard color schemes:

1: Purple/Violet	10: Blue Gray

2: Golden Brown	11: Dark Teal
3: Blue Purple	12: Dark Gray
4: Blue Green	13: Olive
5: Medium Brown (Cinnamon)	14: Standard (Royal Blue)
6: Strawberry	15: Pine Green
7: Purple (Grape)	16: Cranberry
8: Gold	17: Orange/ Rust
9: Dark Brown	18: Teal

To create a custom color scheme, start with an existing properties file.

1. In the portal Image Service, navigate to **PT_HOME\ptimages\tools\cssmill\prop-color**.
2. Make a copy of the **color.18.properties** file and rename it to "**color.19.properties**" (it is a best practice to create a new file so you can preserve all original copies of the properties files).
3. Open the new file in a text editor and modify the properties to create your custom color scheme. Enter the same simplified name for the new color scheme for each of the language style sheets. The example below creates a new color scheme based on the United States Postal Service web site.

```
<!-- color.19.properties -->

colorscheme.name.de=usps
colorscheme.name.en=usps
colorscheme.name.es=usps
colorscheme.name.fr=usps
colorscheme.name.it=usps
colorscheme.name.ja=usps
colorscheme.name.ko=usps
colorscheme.name.pt=usps
colorscheme.name.zh=usps

color.bg.darkest=#CC0000
color.bg.darker=#0066CC
color.bg.medium=#B5C4E1
color.bg.lighter=#99CCFF
color.fg.medium=#003399
color.fg.light=#FFFFFF
color.fg.alert.warning=RED
color.fg.alert.confirm=GREEN
color.link.hover=#E7EFA1
```

   Save the new properties file and close it.
4. Navigate to **PT_HOME\ptimages\tools\cssmill**. Open the file **css-mill-ant-1-6.xml**.
5. Search for the **`make_community_css`** target.
6. Copy the last sequence entry and paste it at the end of the list. Make sure to copy the entire entry: `<make_comm_color_css COLOR="18" CSSPATH="@{CSSPATH}"/>`. Modify the COLOR value by changing it to the number for your custom color scheme. The example below uses "19" for the new color property file created in the prevous section.

```
<!-- make_community_css -->
<target name="make_community_css" depends="make_css_dir">
<make_community_css CSSPATH="css/">
</make_community_css>
</target>
<macrodef name="make_community_css">
<attribute name="CSSPATH" default="css/"/>
<sequential>
<make_comm_color_css COLOR="1" CSSPATH="@{CSSPATH}"/>
<make_comm_color_css COLOR="2" CSSPATH="@{CSSPATH}"/>
...
<make_comm_color_css COLOR="18" CSSPATH="@{CSSPATH}"/>
<make_comm_color_css COLOR="19" CSSPATH="@{CSSPATH}"/>

<make_index FILENAME="community-themes.txt" CSSPATH="css/"
INDEX="css/community-themes.txt"/>
</sequential>
</macrodef>
```

7. Search for the **`make_index`** target.
8. Copy the last sequence entry and paste it at the end of the list. Make sure to copy the entire entry: `<append_index_for_color COLOR="18" INDEX="@{INDEX}"/>`. Modify the COLOR value by changing it to the number for your custom color scheme. The example below uses "19" for the new color property file created in the prevous section.

```
target name="make_index" depends="make_css_dir">
<make_index>
</make_index>
</target>

<macrodef name="make_index">
<attribute name="FILENAME" default="community-themes.txt"/>
<attribute name="CSSPATH" default="css/"/>
<attribute name="INDEX" default="css/community-themes.txt"/>
<sequential>
<echo> Making @{INDEX}</echo>
<tstamp prefix="backup"/>
<touch file="@{INDEX}"/>
<copy filtering="false"
overwrite="yes"
file="@{INDEX}"
tofile="backup/@{FILENAME}${timestamp_appendix}"/>

<delete file="@{INDEX}"/>
<touch file="@{INDEX}"/>

<append_index_for_color COLOR="1" INDEX="@{INDEX}"/>
<append_index_for_color COLOR="2" INDEX="@{INDEX}"/>
...
<append_index_for_color COLOR="18" INDEX="@{INDEX}"/>
<append_index_for_color COLOR="19" INDEX="@{INDEX}"/>
</sequential>
</macrodef>
```

9. Save the css-mill-ant-1-6.xml file and close it.

To deploy your customizations, run the CSS Mill as described in Section 5.4, "Deploying Portal Style Sheet Customizations (CSS Mill)".

5.2 Customizing Portlet Layout and Style

The CSS template also allows you to customize portlet content and design in a variety of ways. You can customize specific portlets using the unique ID assigned to each portlet, or use CSS classes to modify the design of a group of portlets (for example, those in the first column of a two-column page). You can also set constraints for portlets, including limiting a specific portlet to a three-column layout or preventing users from collapsing portlets.

This section provides basic syntax rules and customization examples.

5.2.1 Syntax Guidelines

The portal CSS template file follows standard CSS syntax rules. For details on CSS, see http://www.w3.org/Style/CSS/. Below are some basic rules to keep in mind when modifying portlet styles.

To apply a CSS tag to a specific portlet, use the portlet ID. the example below increases the size of the portlet title for the portlet with ID 6. (You can also define basic styles for a specific portlet within the portlet code.)

```
#pt-portlet-6 .portletTitle
{
height: 26px;
}
```

You can also apply styles to groups of portlets, including those on a specific page or in a specific community. To apply styles to a portlet on a specific page or community, use the page or community ID. The example below removes the portlet preferences link for portlets on the page with ID 100.

```
.portletPrefsButton #pt-page-100
{
display: none;
}
```

5.2.2 Customizing Portlet Style

The portal CSS template file allows you to make a wide range of style changes to portlets. Following are examples of portlet style customizations using the portal CSS template file.

- **Change portlet color schemes**: Change the color scheme of portlets for specific columns.

  ```
  #pt-portlet-100 { background-color: white; }
  #pt-portlet-100 .platPortletHeaderBg { background-color: tan; }
  ```

  You can also define basic styles for portlets within the portlet code (instead of in the CSS template file). Use the `pt:token` adaptive tag to reference the portlet ID and ensure that the style is only applied to the current portlet. This code sets the portlet background to tan. (For details on adaptive tags, see the *Oracle WebCenter Interaction Web Service Development Guide*.

  ```
  <pt:namespace pt:token="$$TOKEN$$"
  xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'/>
  <style>
  #pt-portlet-$$TOKEN$$ { background-color: tan; }
  </style>
  ```

- **Add custom portlet padding**: Control the padding around individual portlets, groups of portlets in a column, and selective portlets. The sample code below adds padding below the portlet with ID 43.

```
#pt-portlet-43 .portletBody
{
padding-bottom: 5px;
}
```

5.2.3 Setting Constraints

The portal CSS template file allows you to set constraints for portlets. Below are some examples.

- **Set the portlet width for specific pages or layouts**: Set the width of a portlet for a specific page or set of pages. You can define portlet settings by page, layout/column, or community. The example below limits the portlet with ID 15 to a width of 250 pixels on the page with ID 1.

  ```
  #pt-page-1 #pt-portlet-15
  {
  width: 250px;
  }
  ```

- **Prevent certain users from collapsing portlets**: Disable the collapse option for a group of users or for portlets on a specific page. You can also prevent users from collapsing a specific portlet by using the portlet ID.

  ```
  .ptPageUser-guest .portletCollapseButton
  {
  display: none;
  }
  ```

  Note: Using CSS to hide functionality is not a secure means of preventing user-server interaction. All examples are for demonstration purposes only and are not meant to imply a complete solution to any overall security scheme.

To deploy your customizations, run the CSS Mill as described in Section 5.4, "Deploying Portal Style Sheet Customizations (CSS Mill)".

5.3 Adding New Language Style Sheets

If you add support for an additional language to the portal, you must add the corresponding style sheets for that language. The portal was designed to make adding languages and generating the language style sheets relatively easy.

Each language file in the **\ptimages\tools\cssmill\prop-text** folder has language-specific values for font style, font size, text style, etc. This design makes it easy to change the default font for each language. For example, if you want the default font for the Japanese user interface to be Tahoma, add Tahoma to the "ja" language file in the prop-text folder.

After adding a language file, you must also edit the **build.xml** file to generate the new language style sheets.

For example, the steps below explain how to add "Dutch" as a portal user interface language.

1. Navigate to the **\ptimages\tools\cssmill\prop-text** folder in the Image Service. Copy one of the existing files to the same folder and rename it using the language conventions in ISO-639-1 and ISO-3166. For example, for Dutch, rename the file to "nl".
2. Open the new file in a text editor and make any necessary modifications for the new language. For example, to add a new default font, you could change the following line: `font.largest=20px`

`verdana,arial,helvetica,"sans-serif"`to:`font.largest=21px Tahoma,"MS PGothic",Verdana,"sans-serif"`Be sure to add the new font for each font attribute in the language file.

3. Navigate to the **\ptimages\tools\cssmill\prop-color** folder in the Image Service. Add the new language's translation for the name of the color in every color properties file. For example, open the color.1.properties file and copy the last `colorscheme.name` entry. Change the name according to the new language ID used in step 1. In this example, you could copy the following line: `colorscheme.name.zh=\\u6DE1\\u7D2B`and change it to:`colorscheme.name.nl=Lavendelblauw`
4. Modify the Ant build script (**build.xml**) to include the new language to the style sheet collection by following the steps below. (This is the only way the script knows to create versions of the new style sheet for each language supported by the portal.)
   a. Navigate to the **\cssmill** directory and open the **build.xml** file in a text editor.
   b. Add an entry for the new language within the **`make_main_css`** target: Copy the last `<antcall target="make_main_language_css">` entry and paste it at the end of the list. Modify the `<param name="LANGUAGE" value="pt"/>` tag by changing the value ("pt") to the language ID used in step 1 ("nl").
   c. Add an entry for the new language within the **`make_comm_color_css`** target: Copy the last `<antcall target="make_comm_lang_color_css">` entry and paste it at the end of the list. Modify the `<param name="LANGUAGE" value="pt"/>` tag by changing the value ("pt") to the language ID used in step 1 ("nl").
   d. Add an entry for the new language within the **`append_index_for_color`** target: Copy the last `<concat destfile="${INDEX}" append="true">mainstyle${COLOR}-pt.css=${colorscheme.name.pt}</concat>` entry and paste it at the end of the list. Change the language id in the new line to the new language id by changing the value ("pt") to the language ID used in step 1 ("nl"). In this example, the new line would look like this:`<concat destfile="${INDEX}" append="true">mainstyle${COLOR}-nl.css=${colorscheme.name.nl}</concat>`
   e. Save the **build.xml** file and close it.
5. Create the new style sheets by running the **make_all** batch file as explained in the next section, Section 5.4, "Deploying Portal Style Sheet Customizations (CSS Mill)".
6. Verify that the new language style sheets were created based on the new language property file. Navigate to the **cssmill\css** directory and confirm that there are 20 new style files with the new language ID used in step 1 ("mainstyle-nl.css"). For further verification, open the **community-themes.txt** file (in the \css directory) and confirm that there is a new entry corresponding to the language ID used in the new language property file.
7. After confirming that your changes are correct, move the new style sheet files from the **\cssmill\css** folder to the **\imageserver\common\public\css** folder used by the portal.
8. Restart the Java Application Server.

5.4 Deploying Portal Style Sheet Customizations (CSS Mill)

After changing the portal style sheet template, you must run the **CSS Style Sheet Mill** to deploy your customizations. The CSS Mill facilitates the management and maintenance of style sheets and allows you to create new style sheets quickly and easily using property files to define key attributes used in the portal's style classes. The portal comes with a set of standard property files, and you can create new files for use in custom style sheets. Although it is possible to edit existing property files, it is recommended practice to make a new property file so you do not lose any information.

The CSS Mill creates all the portal style sheets dynamically using the portal style sheet template file, making them disposable. The entire set is created by running a batch file. This configuration also allows you to update portal style attributes (for example, the background color across all pages) by editing a single root property file; when the batch file is run, the changes are propagated through all instances of the attribute in every style sheet.

5.4.1 CSS Mill Structure

The files used by the CSS Mill are located in the **PT_HOME\ptimages\tools\cssmill** directory in the portal Image Service. This directory includes the following folders:

- **\prop-text** contains text property files; a different file is provided for each language supported by the portal.
- **\prop-color** contains color property files; a different file is provided for each of the 18 standard color combinations available in the portal.
- **\templates** contains the files that define the styles used by the portal. Other products can have their own templates.

Each property name in a property file represents a marker used in a template. The CSS Mill uses the values set in the property files to replace the corresponding markers in the associated style sheet template and create new style sheets for use by the portal. To view where a property name is used within a style sheet, look for the corresponding marker in one of the templates. Markers use the syntax `@MarkerName@`.

The root **\cssmill** folder contains the batch files and the **build.xml** file that provides the necessary Ant scripts to create the style sheets. There are three commonly-used batch files:

- **make_all** creates all portal style sheets by replacing the markers in the templates with the corresponding values from the property files. This script creates a version of each style sheet for each language supported by the portal and places the files in the \css folder, and saves a backup of the previous version in the \backup directory.
- **make_portal_css** creates only the default portal style sheets. The default portal style sheet is the single color style sheet that appears in the default portal.
- **make_community_css** creates only the community style sheets. Community style sheets are the 18x8 style sheets used in header portlets.

5.4.2 Using the CSS Mill

To deploy changes made to the portal style sheet template, you must run the CSS Mill to create new style sheets for the portal. You can use one of the standard color schemes or implement a custom color scheme from a custom properties file. These instructions utilize both the portal server and the portal Image Service.

These instructions use Ant 1.6.x.

1. Open a command prompt on the portal Image Service and change the directory to the CSS Mill root directory (PT_HOME\ptimages\tools\cssmill).
2. Run the following command: `ant make_all`. This command creates new style sheets for each of the properties files.

   Note: If you are implementing a new color scheme, you can use the new style sheets created by the `ant make_all` command or overwrite the default style sheets using `ant make_all -DCOLOR=19` (set the `-DCOLOR` parameter to the number of the properties file that should be used).
3. Open Windows Explorer and navigate to **PT_HOME\ptimages\tools\cssmill\css**. Sort by Date Modified and find the files generated in the previous step.
4. Navigate to **PT_HOME\ptimages\tools\cssmill\css** and copy the stylesheets for the color scheme you want to implement (to continue the example in the previous section, you would copy the **mainstyle19** style sheets). If you did not implement a new color scheme or chose to overwrite the default style sheets, copy the **mainstyle** files instead.
5. Paste the copied files to **PT_HOME\ptimages\imageserver\plumtree\common\public\css**. Select "Yes to all" if asked whether you would like to overwrite the existing files of the same name.
6. On your portal server, open Windows Explorer and navigate to **PT_HOME\settings\portal** and open the **portalconfig.xml** file in a text editor.
7. Find the `<StyleSheetName>` tag (under the `<MyPages>` tag) and change the `value` attribute to `"mainstyle#"` where "#" is the number of the color scheme you want to apply. In the example below, the color scheme is changed to the custom color scheme, #19.

   ```
   <!-- The name for the portal's default stylesheet. -->
   <StyleSheetName value="mainstyle19"></StyleSheetName>
   ```

   Note: If you did not implement a new color scheme or chose to overwrite the default style sheets, the `value` attribute should be `"mainstyle"`.
8. Still on your portal server, reload your portal. (It is not necessary to restart the application server after running the CSS Mill.)

6

Using String Replacement

All strings used in the portal UI are stored in the **PT_HOME\ptportal\6.0\i18n** folder.

Each individual language folder within the i18n directory contains a set of xml files specific to a single language. Folders are named according to the standard ISO 639 language code (i.e., de=German, en=English, es=Spanish, fr=French, it=Italian, ja=Japanese, ko=Korean, nl=Dutch, pt=Portugese, zh=Chinese).

The files in each language folder contain sets of strings for specific sections of the portal UI. The most commonly customized files are listed below:

- ptmsgs_portaladminmsgs.xml: Strings used in the Administration section of the portal.
- ptmsgs_portalbrowsingmsgs.xml: Strings used for most of the messages seen by portal users.
- ptmsgs_portalcommonmsgs.xml: Strings used for common messages repeated throughout the portal.
- ptmsgs_portalinfrastructure.xml: Strings used in the portal's underlying infrastructure components (i.e., the "Finish" and "Cancel" seen on editor pages).

Using these language files, you can customize existing strings or add new strings to the portal UI.

6.1 Customizing Existing Strings in Language Files

The basic procedure for replacing a string in the portal UI is summarized below. See the string replacement examples in the sections that follow for a detailed explanation.

1. Search for the string in the language folder of your choice. To use Windows Explorer's "Containing text" feature, right-click on the language folder and choose Search....
2. Open any files that contain the string in a text editor. (The language files have a UTF-8 byte order mark (BOM) at the beginning of each file to help editors identify the file as UTF-8 character encoding. The BOM for UTF-8 is 0xEF 0xBB 0xBF. Use an editor that is capable of reading and writing UTF-8 files.
3. Replace the string with the message of your choice. Change the text between the `<S> </S>` tags. Some strings are used in more than one place. As noted above, NEVER change the numbers in the `<S>` tags or modify the order of the strings in a language file. Also note that XML tags are case sensitive; be careful not to inadvertently change the case of any tag.
4. After editing an XML language file, view the file in your browser to verify that the XML is well formed.

5. If your portal is load balanced, you must copy the updated language files to all portal servers.
6. Restart your application server and restart the portal. If the portal fails to start up, you might have corrupted the language files. It is a good practice to use Logging Spy to watch the portal load the files to verify that the XML files have been edited correctly.

Note: Making changes to one language folder does not change the same string in any other language folder. To internationalize your string replacements, you must add a translated version of the string to the appropriate file in each language folder.

6.2 Adding Strings to Language Files

Some customizations require additional UI strings. If your portal supports more than one language, adding strings to the portal XML language files allows your new strings to be localized using the portal's multi-language framework.

Note: To add new strings, use a new XML language file or the SampleMsgs.xml file instead of adding strings to any existing ptmsgs*.xml file. Adding strings to ptmsgs*.xml files can result in string number conflicts.

The sample HTML below can be used in a portlet to retrieve the first string from a new XML language file called my_ messsage_file.xml. The portal knows the locale of the current user and retrieves the string from the correct language folder automatically. (The ".xml" extension is not required when specifying the message file name.) For detailed information on adaptive tags, see the*Oracle WebCenter Interaction Web Service Development Guide*.

```
<span xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
<pt:logic.value pt:value="$#1.my_message_file"/>
</span>
```

The **`GetString`** method of the ActivitySpace object can also be used to retrieve strings. The ActivitySpace knows the language of the current user; the `GetString` method automatically retrieves the message from the correct language folder.

The sample code below retrieves the first string from a new XML language file called my_ messsage_file.xml:

```
import com.plumtree.uiinfrastructure.activityspace.*;
...
public String MyNewCode() {
myActivitySpace.GetString(1, "my_message_file");
...
}
```

Note: To add a new XML language file, you must add the file to every language folder, even if you do not provide translated strings for each language. The portal will fail to load if the XML language files are not synchronized for every language.

6.3 Example 1: Hello World Corporation

This example shows how to replace the text displayed at the bottom of all portal pages. As noted earlier, changes to one language folder (in this example, the \en folder) do not change the string for other languages.

1. In your browser window, copy the string you want to search for.
2. Navigate to the **\en** language folder in the \i18n directory.
3. Right-click on the language folder and choose **Search...**.

4. Paste the string into the **Containing text** field and click **Search Now**.
5. Open the **ptmsgs_portalcommonmsgs.xml** file in a text editor.
6. Search for the string within the ptmsgs_portalcommonmsgs.xml file.
7. Replace the string with the string you want displayed on each page (for example, "Hello World Corporation").
8. Save and close the ptmsgs_portalcommonmsgs.xml file.
9. Restart your application server.
10. Reload your portal; the new string should appear in the footer at the bottom of the page.

6.4 Example 2: Custom Login Instructions

This example shows how to replace the login instructions on the main login page.

1. In your browser window, copy the string you want to change, for example "Log in to your personalized portal account".
2. Navigate to the **\en** language folder in the \i18n directory.
3. Right-click on the language folder and select **Search**....
4. Paste the string into the **Containing text** field and click **Search Now**.
5. Open the **ptmsgs_portalcommonmsgs.xml** file in a text editor.
6. Search for the "Log in to your personalized Portal account" string within the ptmsgs_portalcommonmsgs.xml file.
7. Replace the string with the string you want to appear on the login page, for example "Log in to the Hello World portal account".
8. Save and close the ptmsgs_portalcommonmsgs.xml file.
9. Restart your application server.
10. Reload your portal; the new string should appear on the login page.

7

Customizing Experience Definitions

Experience definitions allow you to tailor portal experiences for different groups of users. In a single portal implementation, you can create a distinct user experience for each audience. Using, experience definitions, you can specify which navigation and branding schemes, mandatory links, and default home pages (such as a My Page, a particular community page, or the Directory) to display to each set of users.

Experience definitions work well for organizations that have a variety of audiences or subsidiaries. In a large company, each major department within the organization might need a different view of the portal.

Experience definitions are configured and maintained through portal administration. After creating an experience definition, you must create **experience rules** to assign the experience definition to an audience. For details, see the sections that follow.

For instructions on creating experience definitions and configuring login page options, see the *Administrator Guide for Oracle WebCenter Interaction* and the portal online help.

7.1 Creating Experience Rules

An experience rule contains a list of conditions, all of which must be met for the rule to evaluate to true. When a rule evaluates to true, users are directed to the experience definition specified in the rule. Experience rules are ranked in order of priority; the first rule to evaluate to true is applied. For more detailed information on how experience rules are processed, see Chapter 11.5.3, "Experience Definition Control Flow".

For example, you could create an experience rule based on community memberships. The rule would include a condition of type "community" set to a specific community or communities, for example "Human Resources" and "Personnel." The following **condition types** are available by default:

- **URL**: The URL used to access the portal. You can use an exact URL or use regular expressions with wildcards. For example, if you enter *support* the condition will match any URL containing "support" including http://support.acme.com and https://www.myhome.com/support. The protocols http:// and https:// are ignored in URL matching.
- **IP Address**: The user's IP address. You can use an exact IP address or use regular expressions with wildcards.
- **Group**: The user's group membership.
- **Administrative Folder**: The administrative folder that contains the user object.
- **Community**: The current community (the community being viewed by the user).

You can also create your own condition types, explained in the next section.

An experience rule can have more than one type of condition, and each condition type can have more than one value. A rule will evaluate to true if **all** conditions are met. A condition will be considered met if **any** of the associated values are true. In other words, values within the same condition type are evaluated with an implicit Boolean OR between them, while values of different condition types are evaluated with an implicit Boolean AND between them.

For example, an experience rule with a community condition with values "Human Resources" and "Research" and an URL condition with the value "http://www.plumtree.com" would result in the following expression: (Community = Human Resources OR Personnel) AND (URL = http://www.plumtree.com). Members of either the Human Resources or Personnel community who access the portal using http://www.plumtree.com will be redirected to the experience definition specified in the experience rule. Members of either community that use a different URL will not be redirected. Users who access the portal via http://www.plumtree.com who are not members of either community will not be redirected.

You can create multiple simple rules and combine them to form a complex expression. The portal evaluates rules in the order listed in the Experience Rules Manager and applies the first rule that evaluates to true.

Note: The ranking of experience rules is important. For example, you could create a rule that directs users in the Marketing group to the Marketing experience definition and another rule that directs users in the Research group to the Research experience definition. If some users are in both groups, you must determine which rule should be evaluated first. If you want users who belong to both groups to be directed to the Research experience definition, make sure the Research experience rule is above the Marketing experience rule.

- The **Guest Associations** page in the Experience Rules Manager lists experience rules and the resulting guest user if the rule evaluates to true. The rules listed on this page may be a subset of all rules because the list only includes guest rules that can be evaluated before a user logs in, for example, a URL or IP address rule.
- The **Folder Associations** page shows which administrative folders are associated with which experience definitions. If an experience definition has an associated administrative folder, users created in that folder see the associated experience definition only if no experience definition rule applies to those users. If no experience rule applies to a user, and that user is not in an administrative folder associated with an experience definition, the user sees the default experience definition.

For more information on the Experience Rules Manager, see the portal online help.

7.2 Creating a Custom Condition Type

If one of the standard condition types listed in the previous section does not meet your needs, you can create your own condition type. The portal dynamically discovers and loads all condition types, including custom condition types.

There are two kinds of condition types:

- **Guest Condition Types** can be applied before a user is logged in, using information sent by the browser (or other device).
- **Regular Condition Types** are applied using profile information that is only available after the user has logged in.

The sample code below illustrates how to create a condition type based on the user's browser (Firefox or Internet Explorer). You could use this new condition type to allow only users with Firefox to see the Directory.

The classes referenced below are in the **com.plumtree.portaluiinfrastructure.condition** and **com.plumtree.server.condition** packages. For a full list of interfaces and methods, see the portal API documentation.

7.2.1 Step 1: Create a Class (A*ConditionType)

Extend either the **`ARegularConditionType`** or **`AGuestConditionType`** class (com.plumtree.portaluiinfrastructure.condition), depending on whether you are creating a regular condition type or a guest condition type.

7.2.2 Step 2: Create a Condition Type ID

Use the **`GetTypeID`** (com.plumtree.server.condition) method to retrieve a unique ID for the condition type. All condition types must be uniquely identified, since the ID is used as a key for storing and retrieving information.

Java:

```
public int getTypeID()
{
return ConditionTypeConstants.CONDITIONTYPE_ID_BASE + 1;
}
```

C#:

```
public virtual public override int GetTypeID()
{
return ConditionTypeConstants.CONDITIONTYPE_ID_BASE + 1;
}
```

7.2.3 Step 3: Implement the Compare Method

The **`Compare`** (com.plumtree.server.condition) method evaluates experience rules by comparing the values of condition types with the values for the current user. When the portal encounters a condition, it retrieves the appropriate condition type and calls this method to compare the value of the condition with the current value in the user's environment. The result of the comparison determines whether the condition has been met.

You can add debug messages to be displayed on the MyPage when troubleshooting (see Section 7.2.8, "Debugging" below). Any exceptions caught from this method will be considered as a return value of "false" and will be discarded.

For this example, the `Compare` method compares the browser of the user to the value specified in the condition.

Java:

```
public boolean Compare(XPHashtable htUserEnvironment, IValue conditionValue,
XPStringBuilder sbDebugText) <p class=Numbered style="font-family: Courier;
font-size: 10.0pt; font-weight: normal;">
{
   // Cast the value into a string type.
   String strUserAgent = (String) conditionValue.GetValue();
   BrowserType currentBrowser = (BrowserType) htUserEnvironment.GetElement(new
   Integer(GetTypeID()));
   if (strUserAgent.equals(currentBrowser.GetBrowserName()))
    {
```

```
     if (null != sbDebugText)
      {
        sbDebugText.Append("Condition on User Agent returns true because the User
Agent: ")
        .Append(strUserAgent).Append("matches the one found in the user's
environment: ")
        .Append(currentBrowser.GetBrowserName()).Append("<br>");
          }
      return true;
   } else {
      if(null != sbDebugText)
             {
              sbDebugText.Append("Condition on User Agent returning false because
the User Agent: ")
              .Append(strUserAgent).Append(" does not match the one found in the
user's environment: ")
              .Append(currentBrowser.GetBrowserName()).Append("<br>");
              }
                 return false;
              }
   }
```

C#:

```
public override bool Compare(XPHashtable htUserEnvironment, IValue conditionValue,
XPStringBuilder sbDebugText)
    {
     if (conditionValue.GetType() != ValueTypeEnum.STRING ||
!htUserEnvironment.ContainsKey(GetTypeID()))
      {
        if (null != sbDebugText)
        {
          sbDebugText.Append("Condition on User Agent returning false because
either the condition value is of the wrong type,").Append(" or the User Agent was
not found in the user's environment<br>");
        }
        return false;
      }
      // Cast the value to type: String
      String strUserAgent = (String) conditionValue.GetValue();
      BrowserType currentBrowser = (BrowserType)
htUserEnvironment.GetElement(GetTypeID());
      if (strUserAgent.Equals(currentBrowser.GetBrowserName()))
      {
        if (null != sbDebugText)
        {
          sbDebugText.Append("Condition on User Agent returning true because the
User Agent: ")
.Append(strUserAgent).Append(" matches the one found in the user's environment: ")
.Append(currentBrowser.GetBrowserName())
.Append("<br>");
          }
        return true;
      } else
      {
        if (null != sbDebugText)
        {
          sbDebugText.Append("Condition on User Agent returning false because the
User Agent: ")
.Append(strUserAgent).Append(" does not match the one found in the user's
environment: ")
.Append(currentBrowser.GetBrowserName())
```

```
.Append("<br>");
          }
        return false;
      }
    }
```

7.2.4 Step 4: Retrieve Values

The **`GetCurrentValue`** (com.plumtree.portaluiinfrastructure.condition.A*ConditionType) method retrieves the current value used in the `Compare` method and puts it in a hash table that keeps track of the user's environment.

The **`GetConditionValue`** method retrieves the data and converts it to the expected value type. You can use this method to validate your code, since any value that is not acceptable for the condition will cause an exception to be thrown.

In this example, the method retrieves the user's browser as a string, such as "Firefox" or "MSIE".

Java:

```
public void GetCurrentValue(XPLimitedRequest xpRequest, IPTSession
guestReadOnlySession, XPHashtable htUserEnvironment)
{
htUserEnvironment.PutElement(new Integer(GetTypeID()), new
BrowserType(xpRequest.GetHeader("User-Agent")));
}
public Object GetConditionValue(int nRow, IPTGrowableSortedArrayWrapperRO saData)
{
      Object result = saData.GetItem(nRow, GrowableListModel.EXPLIST_SORTEDARRAY_
PROPID_INPUTTEXT);
          String browser = (String) result;
    if (!browser.equals("MSIE") || !browser.equals("Netscape") ||
!browser.equals("Firefox") || !browser.equals("Mozilla") ||
!browser.equals("Safari"))
           {
            throw new ValidationFailedException();
           }
     return result;
}
```

C#:

```
public override void GetCurrentValue(XPLimitedRequest xpRequest, IPTSession
guestReadOnlySession, XPHashtable htUserEnvironment)
{
htUserEnvironment.PutElement(GetTypeID(), new
BrowserType(xpRequest.GetHeader("User-Agent")));
}
public override Object GetConditionValue(int nRow,
IPTGrowableSortedArrayWrapperRO saData)
    {
      Object result = saData.GetItem(nRow, GrowableListModel.EXPLIST_SORTEDARRAY_
PROPID_INPUTTEXT);
      String browser = (String) result;
      if (!browser.Equals("MSIE") || !browser.Equals("Netscape") ||
!browser.Equals("Firefox") || !browser.Equals("Mozilla") ||
!browser.Equals("Safari"))
      {
        throw new ValidationFailedException();
      }
```

```
      return result;
 }
```

The **`AddItemToMyConditionsList`** (com.plumtree.portaluiinfrastructure.condition.A*ConditionType) method adds values of conditions into a list. These stored values are later used by the `Compare` method.

By default, condition types use a **`GrowableList`** (com.plumtree.uiinfrastructure.expandablelist.GrowableList), but any list structure that extends **`ExpandableList`** (com.plumtree.portaluiinfrastructure.expandablelist) can be used.

This example uses the default `GrowableList`.

Java:

```
//This condition uses a GrowableList. It is called right before the Rules Editor
is opened.
public void AddItemToMyConditionsList(Object objItem, ExpListModel myListModel,
IPTSession ptSession)
 {
 GrowableListModel myGrowableListModel = (GrowableListModel) myListModel;
 myGrowableListModel.AddRowsToList(new String[] {XPConvert.ToString(objItem)});
 }
```

C#:

```
public override void AddItemToMyConditionsList(Object objItem, ExpListModel
myListModel, IPTSession ptSession)
{
      GrowableListModel myGrowableListModel = (GrowableListModel) myListModel;
      myGrowableListModel.AddRowsToList(new
String[]{XPConvert.ToString(objItem)});
}
```

7.2.5 Step 5: Register the Condition Type Class

Add your new condition type to **ConditionTypes.xml** (PT_HOME\settings\portal\dynamicloads\Plugins). The portal uses this file to dynamically discover all condition types.

The first four items listed are the standard condition types installed with the portal. Add your custom condition type to the end of the list. In the example below, the custom condition type created in the previous steps is added as ConditionTypeBrowser.

```
<interface name="com.plumtree.portaluiinfrastructure.condition.AConditionType" />
<interfaceassembly name="portaluiinfrastructure" />
<class
name="com.plumtree.portalpages.condition.conditiontypes.ConditionTypeURLDomain"/>
<class
name="com.plumtree.portalpages.condition.conditiontypes.ConditionTypeClientIPAddre
ss"/>
<class
name="com.plumtree.portalpages.condition.conditiontypes.ConditionTypeCommunityID"/
>
<class
name="com.plumtree.portalpages.condition.conditiontypes.ConditionTypeGroupID"/>
<class
name="com.plumtree.portalpages.condition.conditiontypes.ConditionTypeBrowser"/>
```

7.2.6 Step 6: Deploy Your Custom Code

Once you have coded the custom condition type, you must deploy your custom class for use by the portal. The process is different for Java and .NET.

Java:

1. Place a copy of the new jar file in PT_HOME\ptportal\6.0\lib\java.
2. Add the jar to your portal.war file in PT_HOME\portal\6.0\webapp. Always create a backup of your portal.war file before making any changes.
   a. Unzip the portal.war file.
   b. You will see the following directories: \conf, \META-INF and \WEB-INF. Place a copy of your jar file in \WEB-INF\lib.
   c. Rebuild the portal.war file by zipping up the \conf, \META-INF and \WEB-INF directories.

.NET:

Place the new dll file in the following locations:

- PT_HOME\ptportal\6.0\webapp\portal\web\bin
- PT_HOME\ptportal\6.0\bin\assemblies.

7.2.7 Step 7: Restart the Portal

After you restart the portal, you should see the new condition type in the Experience Rules Manager.

7.2.8 Debugging

You can configure the portal to display debugging messages to troubleshoot problems with your condition types and experience rules. Go to portal administration and click Select Utility | Portal Settings to open the **User Settings Manager**. Under **Debug Mode**, select **Enable Experience Definition Rules Debug Mode** to display experience rules debug messages on My Pages. Enabling this mode adds the option to display debug messages to users' My Account | Display Options | Advanced Settings page.

Part II

Customizing Portal Functionality

Oracle WebCenter Interaction supports customizing and extending all aspects of portal functionality. The most common options are detailed in this section.

This section contains the following chapters:

- Chapter 8, "Customizing Portal Login": The portal login page can be customized for different groups of users. A common customization is to provide different branding on the login page based on the URL used to access the portal. This allows you to provide each group of users with a seamlessly branded portal, including pages viewed as the guest user. This can be implemented easily using Experience Definitions. You can also create a custom login page using Adaptive Layouts; for details, see Chapter 3, "Using Adaptive Page Layouts".
- Chapter 9, "Customizing Portal Navigation": Navigation is a key element of the portal page, providing links to available portal pages and resources, including My Pages, communities, the Directory and Administration. Experience definitions allow you to add custom links to the navigation pane that point to community pages, documents, and web pages without writing any code. Adaptive Layouts allow you to define the navigation section of the page using tags.
- Chapter 10, "Customizing Portal Search": Oracle WebCenter Interaction search indexes and searches all the documents, information, applications, communities, discussions, web sites and other content accessible through the portal. You can customize how search is implemented in the portal, and extend search to include enterprise content.
- The most common way to add functionality to a page is to implement custom portlets. Basic portlets allow you to display custom HTML and content from other applications. You can also use portlets to access portal components, and build portlets that are updated dynamically based on user action and other events. For details on creating portlets, see the *Oracle WebCenter Interaction Web Service Development Guide*.

8

Customizing Portal Login

The login process is a key part of every user's portal experience. The login page is a standard portal page, so there are many tools that allow you to customize the look and feel or functionality of the login experience.

8.1 Customizing the Look and Feel of the Login Page

There are many ways to customize the look and feel of the login page.

- Change the header, footer, top bar and navigation by modifying the default experience definition. For details, see Chapter 7, "Customizing Experience Definitions".
- Create a custom login page using Adaptive Layouts. For details, see Chapter 3, "Using Adaptive Page Layouts".
- Change the text displayed on the page by modifying the corresponding string in the language file(s). For instructions, see Chapter 6, "Using String Replacement".

8.2 Modifying Login Functionality

There are also several options for customizing and extending the functionality of the login page.

- Add custom authentication options to the login page using remote Identity Services. Authentication Sources and Profile Sources allow you to use remote services to import user credentials and information. For details, see the *Oracle WebCenter Interaction Web Service Development Guide*.
- Provide specific users and groups with a customized login experience using experience definitions. For details, see Chapter 7, "Customizing Experience Definitions".
- Adaptive page layouts allow you to customize the portal user interface using adaptive tags in standard XHTML. For details, see Chapter 3, "Using Adaptive Page Layouts".
- Modify portal login functionality using the ILoginActions Programmable Event Interface (PEI). This interface includes methods for before/after login, failed login, and logout. The HelloWorld Login and Login Usage Agreement examples in this section provide sample code and detailed instructions. For details, see Chapter 12, "Using PEIs".
- If none of the above options provides the customization your require, you can change basic login form components through the portalconfig.xml file (PT_HOME\settings\portal). The following settings appear in the Authentication

section of portalconfig.xml. For more information on portalconfig.xml, see the *Administrator Guide for Oracle WebCenter Interaction*.

– The **`AllowDefaultLoginPageAuthSource`** option specifies how the authentication source dropdown appears.

Mode	Display
0 (default)	Dsplays the authentication source dropdown in no special order
1	Hides the dropdown and automatically uses the default prefix for users. Displays a link for users to display the authentication source dropdown to select a non-standard authentication source.
2	Displays the dropdown, but pre-selects the default authentication source.
3	Same as Mode 1 but does not provide a link to display the dropdown.

Note: For modes 1-3, you must set `DefaultAuthSourcePrefix` to the prefix of the default authentication source.

– The **`AuthSourcePrefix[i]`** tags allow you to order the authentication source dropdown in any way you want. Entries in the list should follow the following syntax: `<AuthSourcePrefix[i] value="Auth Source Prefix"></AuthSourcePrefix[i]>` where [i] is replaced with the position in which the item should appear (starting with 1).

To include the auth source in the list, make a new entry with the value you want to appear in the list. This authentication source is used for users created in the user database in the portal. For example, to include this authentication source as the 3rd item in the list, use the following syntax: `<AuthSourcePrefix3 value="My custom auth source"></AuthSourcePrefix3>`

This list will be read in ascending order starting with 1 until there is no next sequential number. The authentication sources with associated prefixes are displayed first, followed by any authentication sources not included in the ordered list.

– **`AllowAutoConnect`** allows you to turn the Remember My Password option on (1) or off (0).

– R**`ememberPassword`** allows you to set how long the portal remembers users' passwords. The value must be formatted in minutes. The default is one week.

9

Customizing Portal Navigation

Customizing portal navigation allows you to change the look and feel of the entire portal browsing experience. Portal navigation includes links to available portal pages and resources, including My Pages, Communities, the Knowledge Directory and Administration. Navigation schemes can also include links to external resources. A navigation scheme defines:

- The location of the navigation pane on the portal page
- The look and feel of the navigation pane, including layout, color scheme and the type of navigation links (i.e., pure HTML links, combobox menus or dropdown menus).

Portal administrators select the navigation scheme for each experience definition in the Experience Rules Manager. The scheme selected is implemented throughout the entire experience definition. For details on experience definitions, see Chapter 7, "Customizing Experience Definitions".

In Oracle WebCenter Interaction, there are three ways to customize portal navigation:

- Built-in navigation options provide a wide range of possibilities. Many customizations can be implemented without writing any code. For information on the portal's built in navigation options, see Section 9.1, "Built-In Navigation Options" .
- Adaptive Tags allow you to build and customize navigation in portlets using only HTML and provided HTML tags representing navigation elements and data. You can even build your own HTML tags to create very advanced and highly customized navigations. For details on using adaptive tags, see Chapter 3, "Using Adaptive Page Layouts" and the *Oracle WebCenter Interaction Web Service Development Guide*.
- Custom navigation schemes are an advanced customization that requires custom code. Custom navigation schemes are explained in Section 9.2, "Creating a Custom Navigation Scheme".

9.1 Built-In Navigation Options

Oracle WebCenter Interaction provides a wide range of possibilities for customizing navigation without writing code.

- Navigation Pane Locations
- Built-in Display Options

9.1.1 Navigation Pane Locations

There are seven possible locations for the navigation pane. For an introduction to the portal page, see Chapter 2, "Portal Page Layout".

- Top Bar
- Above Header
- Below Header
- Above Body
- Below Body
- Above Footer
- Below Footer
- Right
- Left

The portal provides eight built-in Navigation Schemes:

- **Horizontal Dropdown Navigation** (default) is implemented in the Above Header navigation pane using JavaScript-driven dynamic menus.
- **Horizontal Combobox Dropdown Navigation** is implemented in the Above Header navigation pane using combobox menus.
- **Tabbed Section Left Vertical Navigation** combines links in the Left navigation pane and tabs in the Above Header navigation pane.
- **Left Vertical Navigation** displays HTML links in the Left navigation pane.
- **Low Bandwidth and Accessibility Navigation** is similar to left vertical navigation but conforms to 508 accessibility standards.
- **Mandatory Links Only** shows mandatory links if present (including Administration), but no Communities, MyPages, or directory links. This scheme is not intended for use in a deployed portal.
- **Portlet-Ready Navigation**disables all navigation panes and leaves the header and footer enabled. (The header should contain all navigation since there is no other way to navigate.) This option is intended for use with Adaptive Tags.
- **No Navigation** shows no links at all except for Administration (users can log in and out). This scheme is not intended for use in a deployed portal.

The location of the navigation pane can be combined with a different look and feel to provide a completely unique portal experience.

Even if one of the standard navigation options does not meet your needs, always use one of the built-in schemes as a starting place for navigation design.

9.1.2 Built-in Display Options

Some portal page components can disabled or modified without creating a custom navigation scheme:

- **Header and Footer:** The header and footer are configured in the experience definition editor. You can also choose to disable the header and footer.
- **Color Scheme**: The color scheme for navigation schemes is configured in the experience definition editor. For details, see the portal online help. For details on creating custom color schemes, see Chapter 4, "Using Adaptive Styles (CSS

Customization)". If you are not using Adaptive Page Layouts, see Chapter 5, "Customizing Portal Layout Using CSS - Legacy User Interface".

- **My Pages, Communities and Knowledge Directory**: The links to portal areas displayed in a navigation scheme can be disabled for the associated experience definition, on the main page of the Experience Rules Manager. For details, see the portal online help.
- **Portlet Preference links**: The icons displayed in each portlet that link to the associated User Preferences page can be removed using a setting in the portalconfig.xml file. For details, see Section 9.1.3, "Customizing Built-In Display Options (portalconfig.xml)"below.
- **Navigation Pane layout**: You can customize the spacing and width of the navigation panes by modifying the settings in the portalconfig.xml file. For details, see Section 9.1.3, "Customizing Built-In Display Options (portalconfig.xml)"below.

You can also choose to disable the standard navigation panes and provide customized navigation in another component using portlets with adaptive tags. For details, see the *Oracle WebCenter Interaction Web Service Development Guide*.

9.1.3 Customizing Built-In Display Options (portalconfig.xml)

You can customize basic display options for any navigation scheme by modifying the settings in the **portalconfig.xml** file (PT_HOME\settings\portal) in the Navigation section. The settings in this file include toggling the Edit button in portlet banners and changing the spacing or width of the navigation panes. You can also add settings to this file for your custom navigation schemes.

Note: You must restart the portal after making changes to the portalconfig.xml file.

Some settings can be different for each navigation scheme. Navigation schemes are referenced by the NavID defined within the associated INavType class (for details on the INavType class, see the next section). Standard navigation schemes use negative numbers to avoid collisions with custom schemes. The NavIDs for the standard schemes are listed in the table below.

Navigation Scheme	NavID
Horizontal Dropdown (legacy)	-1
Horizontal Combobox Dropdown	-2
Tabbed Section Left Vertical	-3
Left Vertical	-4
Mandatory Links Only	-5
No Navigation	-6
Horizontal Dropdown JavaScript	-7
508 Navigation	-8
Portlet-Ready Navigation	-9

The Horizontal Dropdown JavaScript option (-7) is provided to support legacy 5.0 navigation and requires additional settings. For information, see Horizontal Dropdown Navigation Settings later in this section.

9.1.3.1 *Edit Portlet Preferences Icon*

By default, each portlet with an associated User Preferences page includes an icon that links to the preferences page for that portlet. To remove the Edit Portlets Preferences icon from all portlets on a portal page, use the **intMyPortletPrefButtonInPortletHeader** setting. If this value is set to 1, the icon is displayed; if it is 0, the icon is not shown.

9.1.3.2 *Table Spacing*

Table spacing refers to the space between the different navigation panes (not between portlets). The default table spacing provides a 10-pixel gap between panes. This setting can be different for each navigation scheme. To set the table spacing, use the **intPlumtreeDPTableSpacing** setting. You must append the NavID of the navigation scheme you are modifying. For example, the code below sets the cell spacing for the Tabbed Section Left Vertical scheme (-3) to 15 pixels.

```
<intPlumtreeDPTableSpacing-3 value="15" /> <!-- table spacing -->
```

9.1.3.3 *Navigation Pane Width*

The Left and Right navigation panes have a fixed width. If only one pane is specified, the width is 200 pixels. If both panes are specified, the width of each is 100 pixels. This layout works well for most applications. To override the default width settings, there are four separate settings that allow you to set different widths depending on whether or not both panes are specified: **intPlumtreeDPLeftWidth** and **intPlumtreeDPRightWidth** are used if both panes are specified; **intPlumtreeDPLeftWidthAlone** and **intPlumtreeDPRightWidthAlone** are used if only one pane is specified.

To change these values, uncomment the settings you want to change and enter the appropriate pixel values. As with table spacing, these settings are set separately for each navigation scheme. You must append the NavID of the navigation scheme you are modifying. The example below sets the column width for the Left Vertical scheme (-4).

```
<intPlumtreeDPLeftWidth-4 value="100" />
<!-- left view width when a right view is present -->
 <intPlumtreeDPRightWidth-4 value="100" />
<!\-- right view width when a left view is present -->
 <intPlumtreeDPLeftWidthAlone-4 value="200" />
<!-- left view width when there is no right view -->
 <intPlumtreeDPRightWidthAlone-4 value="200" />
<!-- right view width when there is no left view -->
```

9.1.3.4 *Horizontal Dropdown Navigation Settings*

The rest of the navigation settings in the portalconfig.xml file are specific to the Horizontal Dropdown Navigation Scheme. Standard navigation works dynamically and does not use these settings.

Note: These settings only apply to the Horizontal Dropdown Navigation Scheme (NavID -7).

The **intISCDropDownMenuTruncationWidth** setting limits the number of characters shown in a dropdown menu.

The **intISCDropDownMenuWidth** setting limits the pixel width of a dropdown menu.

The **intHorizontalNavTabTruncationWidth** setting limits the number of characters in the tabs used to access navigation menus.

The **intHorizontalNavTabWidth** setting limits the pixel width for the tabs used to access navigation menus.

9.2 Creating a Custom Navigation Scheme

If the built-in navigation schemes and display options listed in the previous section do not meet your needs, you can create a custom navigation scheme. The easiest way to customize navigation is to use Adaptive Layouts; for details, see Chapter 3, "Using Adaptive Page Layouts". For more advanced customizations, the Adaptive Navigation Framework allows you to create a new look and feel, disable portal components, and add functionality.

- **Top Bar, Header and Footer**: The top bar (including the login and search functionality), the header, and/or the footer can be disabled using the IsFeatureEnabled method in the `INavTypes` interface.
- **JavaScript**: You can use custom JavaScript in your Navigation Schemes through the JavaScriptIncludes method of the`INavTypes` interface.
- **Look and Feel**: You can change the display of navigation components through the `IView` interface.

The Adaptive Navigation Framework is designed for interface-based development. To create a custom navigation scheme, you must implement two interfaces: HelloWorldNavType (INavTypes) and HelloWorldNavView (IView) (com.plumtree.portaluiinfrastructure.navtypes). The methods in these classes are detailed in the example that follows. (For more information, see the Pluggable Navigation API documentation. For links to all portal API documentation, see Appendix C, "Portal API Documentation".) This section also provides instructions on Generating Navigation Links.

Note: Never change existing source code. The best way to modify an existing navigation scheme is to extend it and override the methods that you want to change. This way you can reuse the original code for the parts of the scheme that will stay the same. To facilitate upgrades, write a new class that corresponds to the navigation scheme you want to modify, and make it available through Dynamic Discovery (explained in the next section). Dynamic Discovery handles multiple versions of the same navigation scheme by giving precedence to the last version loaded.

When you upgrade to a new release of the portal, it is very important to check all customized files to see if the original versions have been modified in the upgrade. This way you can migrate any new features and bug fixes into your modified version.

9.2.1 Example: Hello World Navigation Scheme

The example customization below removes the My Pages and Communities tabs in the left navigation pane and replaces them with a "Hello World" string.

Note: This example uses sample code from the sample UI projects package. For details, see Appendix B, "Installing UI Customization Sample Projects". If you were creating a new navigation scheme, you would create your own custom project and custom navigation class with a unique ID (for example, a CustomNav project and a CustomNav class in com.yourcompany.navigation), and compile the new class into a new JAR/DLL file with an intuitive name (for example, CustomNav).

9.2.1.1 HelloWorldNavType (INavTypes)

The **`HelloWorldNavType`** class implements `INavTypes` and defines a new navigation scheme to be used by the portal by giving it a name, assigning an ID, and

defining the View modules for the sections of the UI that display navigation. (All customizable classes follow the naming convention of ending with the name of the interface that they implement.)

1. Open the **HelloWorldNavType** file (.java or .cs) in the sampleplugnav project in the src/com/plumtree/sampleui/navigation directory.
2. The **`GetID()`** method provides a unique ID for the navigation scheme. This ID is used to order the list of navigation schemes available for experience definitions. The standard navigation schemes in the portal use negative numbers for NavIDs (-1 through -7) to avoid collisions with custom schemes. Be sure to pick a unique ID so that you do not cause conflicts with existing custom navigation schemes. The Hello World example uses 200. You can also use the base ID available from the `NavTypeConstants` class. Set the ID for your custom schemes equal to NavTypeConstants.NAV_TYPE_ID_BASE + N, where N is an integer greater than or equal to zero, as shown in the code below.

   Java:

   ```
   public int GetID()
   {
      return NavTypeConstants.NAV_TYPE_ID_BASE + 1;
   }
   ```

   C#:

   ```
   public virtual int GetID()
   {
      return NavTypeConstants.NAV_TYPE_ID_BASE + 1;
   }
   ```

3. The **`GetName()`** method returns the name for the navigation scheme. This name is displayed in the Experience Rules Manager on the Choose Navigation Scheme page. The `_strLangID` argument contains the two letter language code of the current language (i.e., en for English, jp for Japanese).

   Java:

   ```
   public String GetName(String _strLangID)
   {
      return "Hello World" ;
   }
   ```

   C#:

   ```
   public virtual String GetName(String _strLangID)
   {
      return "Hello World";
   }
   ```

4. The **`GetScope()`** method returns the description for the navigation scheme. This description can be any kind of text, but it must be stored in a String variable. The `_strLangID` argument contains the two letter language code of the current language (i.e. en for English, jp for Japanese).

   Java:

   ```
   public String GetScope(String _strLangID)
   {
      return "Hello World Navigation" ;
   }
   ```

   C#:

   ```
   public virtual String GetScope(String _strLangID)
   {
      return "Hello World Navigation";
   }
   ```

5. The **`GetNavAreaView()`** method defines the View for each section of the page, and returns the name of the View class that builds the navigation scheme (i.e., the name returned by View.GetName). In this example, the HelloWorldNavView (IView) is displayed to the left of the body, and nothing is displayed in the other sections.

   Java:

   ```
   public String GetNavAreaView(NavAreaEnum area)
   {
      if (area.equals(NavAreaEnum.LEFTOFBODY))
      {
        return HelloWorldNavView.STR_MVC_CLASS_NAME;
      }

      return null;
   }
   ```

   C#:

   ```
   public virtual String GetNavAreaView(NavAreaEnum area)
   {
      if (area.Equals(NavAreaEnum.LEFTOFBODY))
     {
        return HelloWorldNavView.STR_MVC_CLASS_NAME;
     }
      return null;
   }
   ```

   You can also reuse Views from other navigation schemes in your custom scheme. For example, the code below reuses the Community Section navigation (below the header) from the Horizontal Dropdown navigation scheme.

   ```
   ...
   else if (area.equals(NavAreaEnum.BELOWBANNER))
   {
   return NavigationCommSectionDropDownView.STR_MVC_CLASS_NAME;
   }
   ...
   ```

6. The **`IsFeatureEnabled()`** method returns a boolean to tell the portal whether or not a specific navigation feature is enabled. Using this method you can disable the Top Bar (search field and login buttons), header, or footer. In this example, all three features are enabled. This method is the only way to disable these components. If you try to turn off a header or footer in a navigation scheme by not assigning a portlet, the portal will show the default header or footer. Using the `IsFeatureEnabled` method, you can ensure that no header or footer is shown, even if one is assigned.

   This method is called repeatedly for each component: TOPBAR, HEADER, and FOOTER. You must return True or False for each item.

   Java:

   ```
   public boolean IsFeatureEnabled(NavFeatureEnum feature)
   {
      return true;
   }
   ```

   C#:

   ```
   public virtual bool IsFeatureEnabled(NavFeatureEnum feature)
   {
      return true;
   }
   ```

7. The **`JavaScriptIncludes()`** method allows you to include any custom JavaScript needed for menus or dropdowns. The method returns a collection of HTMLScript elements that either contain JavaScript or include .js files. To use JavaScript in menus or dropdowns, simply wrap the JavaScript in an HTMLScript element and return an HTMLScriptCollection that contains all the required HTMLScript elements for the navigation scheme. This example returns null because it does not use custom JavaScript.

   Note: JavaScript that is specific to only one View can be included in the `DisplayJavascript` method of your View instead of in `JavaScriptIncludes`. Do not call **DisplayJavascript** methods within `JavaScriptIncludes`; these methods are called within PlumtreeDP.

   Java:

```
public HTMLScriptCollection JavaScriptIncludes(AActivitySpace owner)
{
   return null;
}
```

   C#:

```
public virtual HTMLScriptCollection JavaScriptIncludes(AActivitySpace owner)
{
   return null;
}
```

9.2.1.2 HelloWorldNavView (IView)

To create a working navigation scheme, you must build a valid View module class that displays all the necessary portal sections (MyPages, Communities, Directories, and Mandatory Communities) and components (drop-downs or submenus). Because a navigation scheme View is used throughout an entire Experience Definition, it extends from the top level View class within the UI Architecture, the `IView` interface. The `GetNavAreaView` method in `INavTypes` tells the portal which View class to use for each section of the portal page. For more information on Views, see Chapter 13, "Using View Replacement".

As noted above, the **`HelloWorldNavView`** class example implements the `IView` interface and displays the HTML output for the navigation scheme.

1. Open the **HelloWorldNavView** file (.java or .cs) in the **sampleplugnav** project in the src/com/plumtree/sampleui/navigation directory. As with all MVC modules, this example creates a public variable at the top of the View class that sets the name of the class.

```
public static final String STR_MVC_CLASS_NAME = "HelloWorldNavView";
```

2. The **`Create()`** method is used to get a new instance of the View when it is needed. It is very important to update this method if you are copying a file; otherwise your custom class will return an instance of the original class, and your custom View will not appear in the portal.

   Java:

```
public Object Create()
{
   return new HelloWorldNavView();
}
```

   C#:

```
public virtual Object Create()
{
   return new HelloWorldNavView();
```

```
}
```

3. The **`GetName()`** method returns the name of the View, which is used to store and retrieve it in the portal and in the ActivitySpace. When overriding an existing View, this method must return the same value as the View to be overridden.

   Java:

   ```
   public String GetName()
   {
      return STR_MVC_CLASS_NAME;
   }
   ```

   C#:

   ```
   public virtual String GetName()
   {
      return STR_MVC_CLASS_NAME;
   }
   ```

4. The **`Init()`** method provides the View with access to the model and parent Activity Space.

   Java:

   ```
   public void Init(IModelRO model, AActivitySpace parent)
   {
      m_model = (NavigationModel) model;
      m_asOwner = parent;
   }
   ```

   C#:

   ```
   public virtual void Init(IModelRO model, AActivitySpace parent)
   {
     m_model = (NavigationModel) model;
     m_asOwner = parent;
   }
   ```

5. The **`DisplayJavascript()`** method is used to add JavaScript to the page. Since this example does not use JavaScript, the code below returns null.

   Note: The `DisplayJavaScript` method is called for each View in your navigation scheme by PlumtreeDP and its JavaScript is displayed in the Head of page. If you use JavaScript that is common to more than one View in a navigation scheme, it should be included in the `JavaScriptIncludes` method of your `NavType` class instead of in `DisplayJavaScript`. You could get JavaScript errors if the same code is included more than once. (This example does not use JavaScript.) For details on JavaScript navigation, see .Section 9.2.3, "Using Advanced JavaScript Navigation Elements (JSPortalmenus)"

   Java:

   ```
   public HTMLScript DisplayJavascript()
   {
      return null;
   }
   ```

   C#:

   ```
   public virtual HTMLScript DisplayJavascript()
   {
      return null;
   }
   ```

6. The **`Display()`** method is the primary method in the View class and creates the HTML for display to the user. This example outputs a table containing the string HELLO WORLD by creating an HTMLElementCollection, adding a table to it, adding a row to it, adding a cell to the row, and printing the string in the cell.)

Java:

```
public HTMLElement Display()
{
   HTMLElementCollection result = new HTMLElementCollection();

   HTMLTable myTable = new HTMLTable();
   result.AddInnerHTMLElement(myTable);
   HTMLTableRow myRow = new HTMLTableRow();
   myTable.AddInnerHTMLElement(myRow);
   HTMLTableCell myCell = new HTMLTableCell();
   myRow.AddInnerHTMLElement(myCell);
   myCell.AddInnerHTMLString( "HELLO WORLD" );
   return result;
}
```

C#:

```
public virtual HTMLElement Display()
{
  HTMLElementCollection result = new HTMLElementCollection();

  HTMLTable myTable = new HTMLTable();
  result.AddInnerHTMLElement(myTable);
  HTMLTableRow myRow = new HTMLTableRow();
  myTable.AddInnerHTMLElement(myRow);
  HTMLTableCell myCell = new HTMLTableCell();
  myRow.AddInnerHTMLElement(myCell);
  myCell.AddInnerHTMLString("HELLO WORLD");
   return result;
}
```

7. In working navigation schemes, the `Display` method only defines the overall layout and structure of the navigation scheme. Internal methods are used to build each section and process the data required to create each menu item. These **`Write*Section`** methods control how each menu and submenu item is built for the corresponding section of the navigation pane. This includes the text, ASURL, and icon required for each item. All `Write*Section` functions defined within the scope of a View class should use **`IPluggableNavModelRO`** to gather the correct enumerators for a given menu item, then use **`ICPLIstEntryIterator`** and **`CListURLMediator`** to process each individual item and create an HTMLElement to be displayed within the portal. For details, see the next section.

9.2.2 Generating Navigation Links

Navigation schemes or Views within the same scheme can display links in different ways. One View might use HTML anchors, while another could use a JavaScript array to populate DHTML menus. The Navigation Framework uses a set of classes called **mediators** to store and format navigation links. Mediator classes ensure consistency in both appearance and functionality. You can also include links to resources on remote servers; for details, see Creating Gatewayed URLs at the end of this section. (If you are adding navigation links to a portlet, use adaptive tags; for details, see the *Oracle WebCenter Interaction Web Service Development Guide*.)

9.2.2.1 URL Mediators

The `NavigationModel` is the source of data for all navigation Views. The `ASCompoundList` class is a linked list that provides links to the appropriate Views. Individual links are stored as `ListEntry` objects, a container with member variables for all possible elements of a link, including page ID, community ID, ActivitySpace, page name and other control parameters. The `ASCompoundList` is returned as an

ICPListEntryIterator, an interface that allows you to iterate over list entries. Mediator classes are wrappers for `ICPListEntryIterators` that convert each link into the appropriate form. In addition, mediators allow you to set a maximum display string width before truncation, and control link images.

There are two types of mediators:

- **TemplateMediators** use a template to represent a link. A template is a comma-separated String with the type of URL and its parameters. For example, "C,200,-201" where C stands for a Community link template, 200 for the community ID., and 201 for the page ID. The purpose of using a template is to reduce the size of the data sent to the client. The common use for this mediator is for `ListView` menus, in which the onclick event calls a function that constructs the full URL and then redirects the browser. Note: These classes require that the user's browser support JavaScript.
- **FullMediators** generate a full href URL link instead of just a URL template. The resulting link is 508 and low-bandwidth compatible.

There are five `URLMediator` classes included in the Navigation Framework library:

- `CListURLFullMediator` creates ASURLs with target URLs in href, display string and image (if assigned).
- `CListURLFullLinkMediator` is an extension of `CListURLFullMediator` that generates HTMLAnchors using target URLs as the onclick action.
- `CListURLTemplateMediator` takes values from `ASCompoundList` and returns URL templates. This mediator is used by Horizontal Dropdown Navigation.
- `CListURLTemplateLinkMediator` is an extension of `CListURLTemplateMediator` that generates HTMLAnchors using URL templates as the onclick action.
- `CListURLMediator` is not truly a mediator class, but delegates link generation requests to the appropriate link medator. If the user is accessing the 508 or low-bandwidth portal, the class delegates to `CListURLFullMediator`. Otherwise, it delegates to `CListURLTemplateLinkMediator`.

As noted above, all mediators implement the **ICPListEntryIterator** interface. To get the data from a mediator, simply iterate over the items in the list. The code snippet below retrieves the links available on a MyPage and uses `CListURLMediator` to create HTMLElements.

```
// ICPListEntryIterator contains a listing of all actions and links
// for the referenced navigation section for the current user.
// IPluggableNavModelRO Model provides access to all information about
// users navigation. (The m_model variable is defined higher in the
// inheritance structure to avoid allocating another object.)
// GetCategoryLinks retrieves the list of available actions and links
// The boolean parameter determines which links are returned
ICPListEntryIterator iterActions = m_
model.GetCategoryLinks(NavCategoryType.MYPAGE,true);
if (SectionVisible(NavVisibility.VISIBILITY_MYPAGESACTIONS_SECTION))
// Before HTMLElements can be created, the ICPListEntryIterator must
// be transformed into a CListURLMediator
{
  //create a mediator based on the ICPListEntryIterator of URLs
  CListURLMediator mediator = new CListURLMediator(m_asOwner, iterActions);
  mediator.SetImageSize(25, 25);
  mediator.SetLabelMaxLength(30);
```

```
  int i = 0;
  // Take each entry from the Mediator, cast it to an
  // HTMLAnchor, and pass it to AddActionListRow
  while (mediator.Next())
  {
    AddActionListRow(table,(HTMLAnchor)mediator.GetEntry(),((i == 0) ?
    GetActionCollapseURL(NavVisibility.VISIBILITY_MYPAGESACTIONS_SECTION): null));
    i++;
    }
}
```

You can add links to the set provided by `IPluggableNavModelRO` by casting the `ICPListEntryIterator` to an `ASCompoundList`, which provides methods to add links to the list. Refer to the API documentation for `ASCompoundList` for more detailed instructions. You can also add links to content hosted on a remote server to your navigation panes, as detailed in the next section.

9.2.2.2 Creating Gatewayed URLs

In some cases, you might want to include content hosted on a remote server in your navigation scheme. To include links to remote content, the content must be part of a remote portlet. For example, a bug-tracking portlet might include a link to a page that shows the total number of open bugs, and you want to include the same link in a custom navigation scheme. You cannot point to the page directly, since the remote server might not be available to the outside users; you must gateway the link to make it available to all users.

To generate the correct gateway URL, call the following method (and pass PT_CLASSIDS.PT_GADGET_ID as the i`ClassID` argument).

```
portaluiinfrastructure.statichelpers.GatewayHelpers.ConstructPrefPageLink(AActivit
ySpace asOwner, int iPortletID, int iCommunityID, int iPageID, int iPrefType, int
iClassID)
```

The **`iPrefType`** argument determines whether the page is displayed in a popup window or in the main browser window. To display the page in a popup, pass `GatewayHelpers.POPUPFLAG`. To display the page in the main browser window, pass `GatewayHelpers.NOFLAGS`. You must create a window for the popup. The result of this call is a string that you can use as a prefix to generate the gateway URL. For example, to generate a link to the "http://myserver/portlet134/page.asp" that is part of a portlet with ID 134, the call would look something like the following:

```
String strGWURL = GatewayHelpers.ConstructPrefPageLink(m_asOwner, 134,
iCurrentCommunityID, iCurrentPageID, GatewayHelpers.NOFLAGS, PT_CLASSIDS.PT_
GADGET_ID) +
GatewayHelpers.CreateGatewayFriendlyURL(http://myserver/gadget134/page.asp);
```

In addition to the settings in the methods above, you can configure gateway pages to be hosted or non-hosted.

- **Non-hosted** pages just show the content that the remote server sent back.
- **Hosted** pages include the portal banner, footer and navigation links, so that the remote page looks like it came from the portal.

This configuration is defined by the portlet code that contains the link.

Once you have written the code for your navigation scheme, you must deploy it for use by the portal, as described in the next section

9.2.3 Using Advanced JavaScript Navigation Elements (JSPortalmenus)

To implement advanced JavaScript functionality in your navigation elements, you can use the `JSPortalmenus` framework. This framework provides native JavaScript

objects to create menu tabs and dropdown menus. The framework uses common jscomponent functionality in jsutil, reducing the size of JavaScript files downloaded by the browser. The total download size of the JSPortalmenus JavaScript and CSS files is approximately 85 KB.

The major feature of the `JSPortalmenus` framework is that form elements do not burn through the dropdown menus in Internet Explorer browsers, a typical problem with most other dropdown menu frameworks. This behavior is not an issue in Netscape 7.x and Firefox browsers. The JSPortalmenus framework also supports older Netscape browsers, (4.x and 6.x) but burn-through of form elements does happen in these browsers.

The portal uses`JSPortalmenus`in main portal navigation and in portlet title bars.

To create a single tab with a dropdown menu, follow the steps below:

1. Include JavaScript files on the page. `JSPortalmenus` files must be included through the `jsincluder` component.
   - On the server side, as in a navigation scheme, use `ConfigHelper.GetJSIncluderJSComponentInclude` with `ConfigHelper.JSCOMPONENT_JSPORTALMENU` as the component name.
   - In a portlet, use the `jsincluder` adaptive tag: `<pt:ui.include pt:name="jsportalmenus"/>`

Both methods will generate JavaScript to include the `JSPortalmenus`. Manually adding the generated JavaScript is not supported since it might change in future releases.

1. Define an HTML Container element with an ID where the menu should be displayed. For example: `<table><tr><td ID="menuCell1"></td></tr></table>`
2. Create a menu object, add entries to the menu, add the menu to a menu tab object, and associate the HTML block ID with the menu tab object.
   a. Create a menu object: `var menu = new PTPMMenuContainer();`
   b. Add entries to the menu (container `menuItems` is an array):

```
for (var j = 0; j < somearray.length; j++)
{
var strTitle = ...;
var strImgSrc = ...;
var strImgWidth = ...;
var strURL = ...;
menu.menuItems[j] = new PTPMSimpleMenuItem();
menu.menuItems[j].text = strTitle;
menu.menuItems[j].image = new PTPMImage();
menu.menuItems[j].image.imgSrc = strImgSrc;
menu.menuItems[j].image.imgWidth = strImgWidth;
menu.menuItems[j].action = new PTPMJavaScriptAction();
menu.menuItems[j].action.js = 'window.location = "'+strURL+'"';
}
```

   c. Set tab content (HTML is allowed): `var buttonText = "My Menu";`
   d. Add the menu to a menu tab object (static call). Assign tab content and provide the ID of the HTML element where the tab HTML should be inserted.

```
PTPMSelectMenu.init(menu, strDivID, buttonText);
```

   Note: This function should only be called *after* the HTML container has been rendered on the page.

9.3 Deploying a Custom Navigation Scheme

After you create a custom project, you must deploy it to the portal using **Dynamic Discovery**. For detailed information and instructions, see Chapter 18, "Deploying Custom Code Using Dynamic Discovery". For navigation schemes, only the **Jar or DLL-Based Dynamic Discovery** section is necessary. Always confirm that your code was deployed correctly, explained in Viewing Your Customizations in the Portal at the end of this section.

9.3.1 Example: Hello World Navigation Scheme

The example below deploys the Hello World sample code from the previous section. These instructions use Visual Studio in .NET and Ant scripts in Java to deploy your custom code.

First, add the library containing the new class to the **CustomActivitySpaces.xml** file so it can be deployed by Dynamic Discovery.

1. Navigate to **PT_HOME\settings\portal** and open **CustomActivitySpaces.xml** in a text editor (you might have to make the file writable).

   Note: Do not modify the ActivitySpaces.xml file. The CustomActivitySpaces.xml file is functionally identical to the ActivitySpaces.xml file and allows you to enumerate custom components without modifying the code used by standard portal components. For more detailed information, see Chapter 18, "Deploying Custom Code Using Dynamic Discovery".

2. Find the `<AppLibFiles>` tag and add an entry for your project. In the example below, the project is called "sampleplugnav".

```
<AppLibFiles>
  <libfile name="sampleplugnav"/>
</AppLibFiles>
```

You must also run a clean build in order to deploy the custom code. The process is different based on your portal platform; see the appropriate set of instructions below.

Java:

1. Open a command prompt and change the directory to the \ptwebui directory where you installed the portal source code.
2. Run a clean build using the following Ant script: `ant build`
3. Generate a new WAR file for the application server using the following Ant script: `ant install`

   Note: This target deletes and rebuilds all jar files associated with all the UI source projects (as well as the custom projects in the ptwebui folder).

C#:

1. Build the project in Visual Studio.
2. Visual Studio should copy the sampleplugnav.dll file from SOURCE_HOME\sampleplugnav\dotnet\prod\bin to PORTAL_HOME\webapp\portal\bin for you. If there are problems with Dynamic Discovery on startup, you might need to do this step manually. This is necessary to allow Dynamic Discovery to find the new library.

9.3.2 Viewing Your Customizations in the Portal

Once you have deployed your code, view the changes in the portal to confirm that they were loaded correctly. Use Logging Spy to catch any obvious errors.

1. Open Logging Spy. For details, see the *Administrator Guide for Oracle WebCenter Interaction*.
2. Start the portal. In Logging Spy, you should see your sample navigation load up with ID 200.
3. This example navigation scheme cannot be applied to the administrator's portal view because administrators will not be able to successfully navigate the portal. Instead, you must create an experience definition that uses the new navigation scheme. Log in as the administrator and navigate to portal Administration.
4. Create a new Administrative folder (**Create Object** | **Administrative Folder**) and name it Navigation.
5. Open the Experience Rules Manager (**Select Utility** | **Experience Rules Manager**).
6. Create a new Experience Definition (**Create Object** | **Experience Definition**).
7. On the first page of the Experience Definition Editor, click **Add Folder**.
8. Select the **Navigation** folder created in step 4 and click **OK**.
9. In the left menu of the Experience Definition Editor, click **Edit Navigation Options**.
10. Select the **Hello World** navigation scheme and click **Finish**. When prompted for a name for the new Experience Definition, enter Hello World Pluggable Nav.
11. Return to portal Administration and open the **Navigation** folder created in step 4.
12. Create a new user (**Create Object** | **User**) in the Navigation folder and name the new user Navigator. Click **Finish** to save the user.
13. Log out of the portal and log in as the new **Navigator** user. You should see the new Hello World navigation scheme.

The next step is to debug your code, covered in the next section.

9.4 Debugging and Troubleshooting

This section provides technical tips for common problems and instructions on how to debug your new navigation scheme.

9.4.1 Technical Tips

If your custom navigation scheme does not function correctly, first check the following:

- The **`GetID()`** method in `INavTypes` must return a unique ID for the navigation scheme. If you use the same ID as another navigation scheme, only one of the schemes will be available. The standard portal navigation schemes use negative numbers for NavIDs (-1 through -7) to avoid collisions with custom schemes.
- The **`JavaScriptIncludes ()`** method in `INavTypes` must return the JavaScript for all Views included in the navigation scheme. If it does not, the Views will not work correctly and might produce JavaScript errors.
- The **`Create()`** method in a View must return a new instance of the custom class. If this is not done, when the portal attempts to instantiate a new instance of the

custom View using the `Create()` method it will not work. A common problem is cutting and pasting code from an existing View and then forgetting to update this method. Your customization will be loaded by the portal, but the original View will still be displayed.

If this does not solve the problem, debug your code using the instructions below.

9.4.2 Debugging

These instructions use the Hello World navigation scheme created on the previous pages as an example.

Java

1. In Eclipse, stop the Tomcat debugging session and open **HelloWorldNavView.java**.
2. Add a breakpoint as shown below:

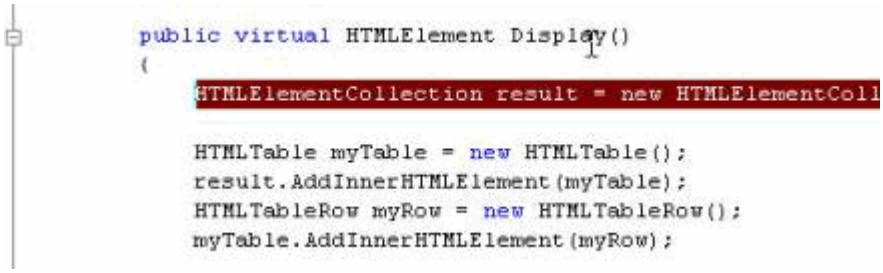
```
public HTMLElement Display()
{
    HTMLElementCollection result = new HTMLElemen

    HTMLTable myTable = new HTMLTable();
    result.AddInnerHTMLElement(myTable);

    HTMLTableRow myRow = new HTMLTableRow();
    myTable.AddInnerHTMLElement(myRow);

    HTMLTableCell myCell = new HTMLTableCell();
    myRow.AddInnerHTMLElement(myCell);

    myCell.AddInnerHTMLString("HELLO WORLD");
```

3. In the Eclipse menu, click **Run** | **Debug…** and select the Tomcat application.
4. Choose the **Classpath** tab, select **Add Projects**, and add the sampleplugnav project.
5. Hit **Debug** (and **Save** to retain your changes).

   In Logging Spy you should see the system load up.
6. Open a browser and navigate to your Java portal. Log in and you should hit the breakpoint.

C#

1. Stop the Visual Studio debugger (and close your browser if it is still open) and open **HelloWorldNavView.cs** in Visual Studio.
2. Add a breakpoint as shown below:



3. Start the Visual Studio debugger (F5 or Start | Debug).
4. Navigate back to your portal and log in; you should hit the breakpoint.

10

Customizing Portal Search

You can customize Oracle WebCenter Interaction Search in a number of ways. This chapter summarizes recommended approaches to common customizations, describes their capabilities and limitations, and points to more complete documentation and sample code.

10.1 Customizing Banner Search and Advanced Search

The banner search box is the text search field that appears at the top of each portal page. By default, banner search will query for the text in the name, description, and full-text content of documents, document folders, communities, portlets, and users. When a user is viewing a community or document folder, banner search also offers an option to restrict the search to the area being viewed.

The advanced search page is used to add search constraints on portal properties. Administrators (not standard users) can search a much wider set of object types from advanced search, including administrative objects.

You can customize the behavior of portal banner search and advanced search in a number of ways. Before modifying portal search, see if one of the following solutions provides the functionality you need.

- **Snapshot Queries** display search results in a portlet and cache results to avoid burdening portal search. For details on Snapshot Queries, see the portal online help.
- **Search Portlets** can provide a customized search form, add constraints to search, and more. The Oracle WebCenter Interaction Development Kit (IDK) remote Search API allows you to run search queries against the Oracle WebCenter Interaction system. For details, see the *Oracle WebCenter Interaction Web Service Development Guide*.
- **Federated Search** uses remote web services to search external repositories from the portal. For details, see the *Oracle WebCenter Interaction Web Service Development Guide*.

If none of these options fulfill your requirements, you can modify the behavior of portal banner search and advanced search. The sections that follow explain the most common customizations:

10.1.1 Customizing the Banner Search Box

You can customize the functionality and appearance of the banner search box in many ways.

10.1.1.1 Search Results Manager

The Search Results Manager allows you to limit banner search to return only documents, and configure banner searches to search properties in addition to the Name, Description, and text content. For details, see the *Administrator Guide for Oracle WebCenter Interaction*.

10.1.1.2 SearchActions Programmable Event Interfaces (PEIs)

SearchActions PEIs can be used to require that all results have a set value for a specific property and configure the portal to record every banner search.

To modify every banner search that users run or perform some action whenever users run a banner search, use the **IBannerSearchActions** PEI. For example, you can add constraints to certain searches, or log some information about each search. For details on implementing PEIs, see Chapter 12, "Using PEIs".

10.1.1.3 Adaptive Page Layouts

Adaptive page layouts allow you to change the look and feel of the portal user interface, including the search box and search results page using adaptive tags in standard XHTML For details, see Chapter 3, "Using Adaptive Page Layouts".

10.1.1.4 View Replacement

To change the look and feel of the banner search box or point banner search at another search results page or at the Federated Search results page, you can modify the associated View, **PlumtreeTopBarView**. PlumtreeTopBarView generates an HTML search form used for banner search, and other parts of the portal banner. For details on Views, see Chapter 13, "Using View Replacement".

- To create a new search box in the portal UI, you can create a portlet or add a search box to the appropriate View. You can use the Oracle WebCenter Interaction Development Kit (IDK) remote Search API or the native PTSearch API to execute the query.
- To use Federated Search in place of banner search, change the form's `in_hi_space` input to point to the NetworkSearch space instead of the SearchResult space, and the `in_hi_control` input to "NetworkSearch" instead of "bannerstart". You'll also need to add an `in_cb_source` input to simulate checking one of the "Search Locations" on the Federated Search page.
- To use an external search page, change the form's target to point to the external page. You will need to rename some other form inputs (such as the query string) to provide the correct names for your target page.

10.1.2 Customizing the Advanced Search Page

You can also customize the advanced search page. The **Search for Text** field on the advanced search page behaves similarly to banner search, with a few minor differences; advanced search queries are never spell-checked and the thesaurus is not used, because this page is meant to be a more precise interface for expert users.

10.1.2.1 SearchActions Programmable Event Interfaces (PEIs)

To modify every advanced search that users run or perform some action whenever users run an advanced search, use the **IAdvancedSearchActions** PEI. For example, you can turn on spelling correction for advanced search queries, add constraints to certain searches, or log some information about each search. For details on implementing PEIs, see Chapter 12, "Using PEIs".

10.1.2.2 View Replacement

To customize the layout of the advanced search page, use one of the approaches below:

- To modify the built-in advanced search page, subclass the Views in the com.plumtree.portalpages.browsing.search.advanced package. These Views inherit from the com.plumtree.portalpages.common.search package, so you might need to copy code from that package as a starting point. Be very careful if you modify the code in common.search; these classes are used elsewhere in the portal. For details on Views, see Chapter 13, "Using View Replacement".
- To replace the built-in advanced search page entirely, create a custom Activity Space with the name "AdvancedSearch" and use the SearchFormFactory class to automate generation of the HTML form. For details on custom Activity Spaces, see Chapter 14, "Creating Custom Activity Spaces".

Removing an object type or folder from the advanced search page does not guarantee that it is unsearchable. A user can still concoct a URL that runs the search, even if the page never submits it. To prevent this, make sure your users do not have rights to see the objects, or use the IAdvancedSearchActions PEI to prevent unallowed queries.

10.1.3 Adding Search Boxes

You can also add search boxes to the portal UI.

- **Portlets** can be used to provide search boxes that query the portal or external repositories. For details on portlets and the Oracle WebCenter Interaction Development Kit (IDK) remote search API, see the *Oracle WebCenter Interaction Web Service Development Guide*.
- **Custom Views** can include additional search boxes. Use the SearchFormFactory class to automate generation of the HTML form. For more information on Views, see Chapter 13, "Using View Replacement".

10.1.3.1 Adding Pathways Search

Oracle Pathways is delivered with a set of Adaptive Tags to add Oracle Pathways UI elements to the portal UI, allowing users to access Oracle Pathways search from a portal page.

The following tags provide access to Oracle Pathways UI elements. For details on basic tag syntax and usage, see the *Oracle WebCenter Interaction Web Service Development Guide*.

Tag	Function
`pt:pathways.pathwayssearchform`	Adds an OraclePathways search box.
`pt:pathways.pathwayssearchbutton`	Adds an Oracle Pathways search button.
`pt:pathways.pathwayshome pt:id="menutabs"`	Returns the URL to the Oracle Pathways home page. This tag is a data tag and must be used in conjunction with a display tag. For details on data tags, see the *Oracle WebCenter Interaction Web Service Development Guide*.

The `pt:text` parameter defines the text to be displayed in the UI element. If you do not include the `pt:text` parameter, no text will be displayed.

The example below adds an Oracle Pathways search box and button to the top bar in the portal banner, a tab to the main portal menu, and a link to the portal Directory tab. This example also includes tags from the UI Elements (pt:ui) and Navigation (pt:plugnav) Adaptive Tag libraries.

Note: This code is taken from the header example included with the Oracle Pathways installation (<PT_HOME>\ptimages\pathways\private\navtags\header.html). To install these samples in your portal environment, import the pathways_navigation_samples.pte file.

```
<span xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>

<!-- Topbar -->
<table cellpadding="0" cellspacing="0" width="100%" border="0" class="banTopbarBg"
id="pt-topbar">
<tr>
            <td align="left" valign="middle" nowrap="nowrap">
                        <pt:ptui.myhome pt:usespan="true"/>

                        <span class="banGreetingText banText" id="pt-user-nav">
                                    <pt:ptui.welcome pt:usespan="true" />
                                    <span class="spacer"
style="padding-left:8px;"></span>
                                    <pt:ptui.myaccount pt:usespan="true" />
                                    <span class="spacer"
style="padding-left:8px;"></span>
                              <pt:ptui.login pt:usespan="true"/>
                        </span>
            </td>

            <td align="right" valign="middle" nowrap="nowrap">
            <pt:ptui.rulesdebug/>
            <pt:ptui.help/>

            <pt:ptui.searchform pt:usespan="true">
                        <pt:ptui.basicsearchbutton/>
                        <pt:ptui.advancedsearchbutton/>
                        <pt:ptui.federatedsearchbutton/>
            </pt:ptui.searchform>

            <!-- Add the Pathways Banner Search Elements -->

            <pt:pathways.pathwayssearchform pt:usespan="true" pt:text="Pathways:">
                        <pt:pathways.pathwayssearchbutton/>
            </pt:pathways.pathwayssearchform>

            </td>
</tr>
</table>

<!-- Topbar section end -->

<!-- Dropdown menus section begin -->
<pt:ptdata.communityactionsdata pt:id="commmenu" />
<pt:ptdata.mycommunitiesdata pt:id="commmenu" />
<pt:ptdata.mandatorylinksdata pt:id="mandlinks" />
<pt:ptdata.mandtabcommsdata pt:id="menutabs" />
<pt:ptdata.administrationdata pt:id="menutabs" />

<!-- Add Pathways Home link as a menu tab. -->
```

```
<pt:pathways.pathwayshome pt:id="menutabs" pt:text="Pathways Home"/>

<pt:ptdata.mypageactionsdata pt:id="mypagemenu" />
<pt:ptdata.mypagesdata pt:id="mypagemenu" />
<pt:ptdata.currcommunitypagesdata pt:id="commpages" />
<pt:ptdata.currsubcommunitiesdata pt:id="subcomms" />
<pt:ptdata.currrelatedcommunitiesdata pt:id="relcomms" />
<pt:ptdata.directorybrowsedata pt:id="directorymenu" />
<pt:ptdata.directoryeditdata pt:id="directorymenu" />

<!-- Add Pathways Home link to the directory menu. -->
<pt:pathways.pathwayshome pt:id="directorymenu" pt:text="Pathways Home"/>

<pt:ptdata.mandatorylinknamedata pt:key="mandlinksname" />

<pt:plugnav.ddmenurowcontainer pt:menuvar="midrowmenu" pt:hideemptymenus="true" >
            <pt:plugnav.ddmenutab pt:containervar="midrowmenu"
pt:datavar="mypagemenu"             pt:text="$#1840.ptmsgs_portalbrowsingmsgs" />
            <pt:plugnav.ddmenutab pt:containervar="midrowmenu"
pt:datavar="commmenu"                 pt:text="$#1841.ptmsgs_portalbrowsingmsgs"
/>
            <pt:plugnav.ddmenutab pt:containervar="midrowmenu"
pt:datavar="directorymenu"             pt:text="$#1842.ptmsgs_portalbrowsingmsgs"
/>
            <pt:plugnav.ddmenutab pt:containervar="midrowmenu"
pt:datavar="mandlinks"                    pt:text="$mandlinksname" />
            <pt:plugnav.ddmenusimpletabs pt:containervar="midrowmenu"
pt:datavar="menutabs" />
</pt:plugnav.ddmenurowcontainer>

<!-- Dropdown menus section end -->

...

</span>
```

10.2 Customizing the Search Results Page

The search results page is the portal page that users see after submitting any banner or advanced search. (This page is not used for Federated Search.)

You can customize the appearance of the search results page. There are some limitations because the portal uses this page in a number of ways; the Add Portlets and Join Communities pages are both customized versions of the search results page. The sections that follow summarize the most common customizations.

10.2.1 Using Search Results Portlets

To display the results of a common query for a specific audience, you can use a Snapshot Query and Content Snapshot Portlet without writing any code. For details, see the portal online help. To create a custom results page that is hosted remotely, write a remote portlet that calls the Oracle WebCenter Interaction Development Kit (IDK) remote search API. Your "results page" is simply a community page with this portlet on it. For details, see the *Oracle WebCenter Interaction Web Service Development Guide*.

10.2.2 Using Adaptive Page Layouts

Adaptive page layouts allow you to change the look and feel of the search results page using adaptive tags in standard XHTML For details, see Chapter 3, "Using Adaptive Page Layouts".

10.2.3 Using View Replacement

If the Search Results Adaptive Layout does not provide the customization you need, you can customize the corresponding View classes. You can make minor modifications to the standard Views, or use them as a base to write your own Activity Space that replaces the main portal search results page.

The HTML for the portal's search results page is generated by View classes in the following packages:

- **com.plumtree.portaluiinfrastructure.search** (in portaluiinfrastructure.jar)
- **com.plumtree.portalpages.browsing.results.search** (in portalpages.jar)

The View classes use SearchResultModel, in com.plumtree.portaluiinfrastructure.search, as their corresponding model. SearchResultModel wraps PTSearch. Do not modify SearchResultModel, because the portal reuses it in many ways.

The search results View classes call read-only methods on SearchResultModel to get information about the search results. The most important View classes are GroupedResultsView and GroupedResultsViewHelper. For more information on Views, see Chapter 13, "Using View Replacement".

- **GroupedResultsView** loops over the search results and calls methods in GroupedResultsViewHelper.
- **GroupedResultsViewHelper** generates the HTML for each result. This View can be replaced at runtime by any other class that implements the same interface, to support results tables with a different look and feel.

Four other Views also appear on the search results page.

- **SearchSummaryView** inspects the original request and summarizes it (for example , "Showing 10 results of 52, and dogg was corrected to dog").
- **FollowupSearchView** lets the user run another search from the results page, either within the same results with the same settings, or a new search of the whole portal.
- **OrganizationView** and its helper **ReorganizationViewHelper** let the user re-sort by last-modified date, folder, object type, or other properties, and show drill down links into these categories.
- **PaginationView** provides links to more search results.

There are corresponding Control classes for each of the links in these Views (the UI framework invokes the Control's execute method when a user clicks a link). These Controls include PaginationControl, ReorganizationControl, SearchWithinResultsControl, and BreadcrumbControl. Most of these Controls have a static makeURL() method that you call from the View code to make a new link. In most cases, you only need to copy and customize Views and DisplayPages. The Models and Controls for this page are designed to be reusable (they are used several places in the portal).

To replace the built-in results page entirely, call your new ActivitySpace **SearchResult**. You can also use your customized page as the target for a custom search form; set the `in_hi_space` input to the name of the new Activity Space.

Note: Test your changes thoroughly. Test both banner and advanced search. Make sure banner search works from every type of portal page (My Page, community, and Directory). Confirm that it works with every search result type

10.2.4 Adding Properties to the Sort By Menu

By default, the **Sort By** drop-down list on the search results page lets users organize results in four ways:

- **Relevance (Rank)**: Results are presented in decreasing order of relevance. To improve relevance ranking on the search results page, see Section 10.2.6, "Improving Relevance Ranking".
- **Last Modified Date**: Results are re-ranked according to the date and time each object was last modified, with the most recently modified items ranked first.
- **Folder**: Results are grouped according to the Knowledge Directory or Administration folders in which the results are stored. Clicking one of the folder links restricts the results to those that are stored in that folder.
- **Object Type**: Results are grouped according to object type. Clicking one of the object type links restricts the results to those that are of a particular object type.

Adding new properties to the "Sort by" menu allows users to group results according to the property value. To display columns for additional properties on the search results page, you must modify the associated View as explained above.

Note: If the properties are not included in the portal search query, you must add the properties to the search request by writing a Portal Event Interface (PEI). See the next section for more information.

To add search properties to the "Sort by" menu, edit portalconfig.xml as explained below.

1. Determine the object ID of the property you want to add. Find the object in the directory and open it; the object ID is displayed in the associated editor. You can also determine the ID from the object link. The "in_hi_ObjectId" URL argument contains the integer that represents the object ID.
2. Open the **portalconfig.xml** file in a text editor (PT_HOME\settings\portal). **Note**: Always make a backup copy before editing; formatting errors in this file can disable the portal completely.
3. Find the `<component name="portal:Search">` section within portalconfig.xml. You will add two tags within this tag:
   - `<setting name="CategoryName_1">` : This tag determines the name of the option as it will be displayed in the Sort by menu. It is not required to be the same as the name of the property. This string will be the same in all locales (the related code is not currently internationalized).
   - `<setting name="CategoryField_1">:` The integer object ID as determined in step 1 above, preceded by "PT" (all caps, no spaces). For the example above, object ID 200, the value would be PT200.

   For example:

```
<component name="portal:Search"
type="http://www.plumtree.com/config/component/types/portal/search">
```

```
   <!-- The default number of search results to show per search results page.
-->
   <setting name="NumSearchResultsPerPage">
      <value xsi:type="xsd:string"/>
   </setting>
   <setting name="CategoryName_1">
      <value xsi:type="DocumentTitle"/>
   </setting>
   <setting name="CategoryField_1">
      <value xsi:type="PT105"/>
   </setting>
...
</component>
```

4. You can add multiple categorization options by adding successive tags (CategoryName_2, CategoryField_2, CategoryName_3, CategoryField_3, etc.), as long as the sequence numbers are consecutive. If you ever delete an option, you must edit the tags to keep the numbers consecutive. Do not forget the trailing "/" before the closing ">".
5. Save and close portalconfig.xml, stop and restart the portal. Run a search to see your new "Sort by" option.

10.2.5 Adding Search Categorization Properties

To query for additional properties using portal search, add new properties to the portal and associate them with the appropriate objects and documents. Once the properties are defined, you must make sure they are returned by the search query.

When choosing new properties to use for search categorization, there are two issues to consider:

1. Will the property be defined for a large percentage of search results? If 90% of results do not have the property defined, then most results will fall into the "uncategorized" group, and the new categorization will not be useful. Make sure that at least half of all documents and objects have values for the property before adding it as a categorization option.
2. Will the property values make good category titles? For categorization to work well, each value should be a single word or a short noun phrase, for example, "New England", "Midwest", "Product Management", "Food and Drug Administration". The values should not be full sentences or long lists of keywords, for example, "This content service crawls the New York Times finance section." It will look odd if a full sentence is returned as a category title.

Once you have chosen properties to use for categorization, the next steps are to ensure that the properties are defined in the portal and values for those properties are being assigned to documents and objects.

10.2.5.1 Defining Properties

To create a new property, navigate to Administration and select **Create Object | Property**. Make sure the following options are checked:

- This property is supported for use with documents
- This property is visible in the UI
- Searchable

Consider making the property mandatory, since categorization will be of maximum value if every item has a value for the property. Name the property object appropriately and click **Finish**.

10.2.5.2 Assigning Property Values

After you create a new property, you must make sure that values are defined for the property. This procedure differs for documents (cards) and administrative objects. The process is summarized below; for more information, see the portal online help.

For documents, the most efficient way to assign property values is through the content crawler that imports the documents. First, create a Content Type object, add the associated property, and set the "mapped attribute" appropriately. For example, for HTML documents the `<META>` tag might contain the value. If the value should be the same for all documents the content crawler imports, set the "default" when adding the property to the Content Type. When you create content crawlers, associate your new Content Type with the documents to be crawled. You can also manually import documents and edit their properties.

Property values for administrative objects are generally set manually. First, go to Administration and select **Utilities** | G**lobal Object Properties Map**, and associate the property with each object type for which it should be defined. After the property is associated with an object, you can edit the property value in each object's editor. If the property is mandatory, a value will be required when creating or editing an object.

Customizing query behavior for banner search and advanced search is covered in the previous section.

10.2.6 Improving Relevance Ranking

You can improve the relevance ranking of search results in a variety of ways.

10.2.6.1 Best Bets (Banner Search)

To hard-wire the top few results for specific search queries, use the Best Bets feature (banner search only). Results may be documents, document folders, communities, portlets, and users. For more information and instructions, see the portal online help.

10.2.6.2 Search Field Weightings (Banner Search)

To change the weight of different search fields, use the Search Results Manager utility (Banner Fields Alias page). You can also list additional properties to be searched (for example, a "keywords" or "author" field).

Perform these adjustments with care and test them carefully on all common searches. These settings affect all portal banner searches, and most adjustments will make some rankings better and others worse.

10.2.6.3 Search Thesaurus

To replace search terms or define synonyms, load a thesaurus list into portal search and enable it. Thesaurus expansion replaces a term or phrase in a user's search with a set of custom, related terms before the search is performed. This feature allows you to improve search quality by handling unique, obscure, or industry-specific terminology. Fore more information, see the *Administrator Guide for Oracle WebCenter Interaction*.

10.3 Using Federated Search

Federated Search provides access to external repositories without adding documents to the portal Knowledge Directory. Federated Search is especially useful for content that is updated frequently or is only accessed by a small number of portal users. For details, see the *Oracle WebCenter Interaction Web Service Development Guide*.

Part III

Advanced UI Customization

The basic customizations in the previous sections require little or no custom code. If these options do not provide a solution, you can replace portal components with custom versions. The advanced customizations below require Java or C# coding. For an introduction to the inner workings of the portal UI, see Chapter 11, "Portal UI Architecture"

This section contains the following chapters:

- Chapter 12, "Using PEIs": Portal Event Interfaces (PEIs) are used to execute custom actions in many places throughout the portal. For example, you can modify search queries before they are processed, or perform validation when users attempt to create new portal objects.
- Chapter 13, "Using View Replacement": You can completely customize the display of portal components by creating a custom version of the associated View class(es).
- Chapter 14, "Creating Custom Activity Spaces": Activity Spaces group task-specific actions into logical sets to provide portal developers with base functionality, and combine related pages to create cohesive Model-View-Control (MVC) objects. Everything in the portal is an Activity Space: a MyPage, an administrative editor, even the Directory tree. A custom Activity Space allows you to add new pages to your portal.
- Oracle WebCenter Interaction includes a collection of useful tools and components to support UI customization. For details, see Chapter 15, "Accessing Portal Objects", Chapter 16, "Adding Custom Images", and Chapter 17, "Using VarPacks (Variable Packages)"

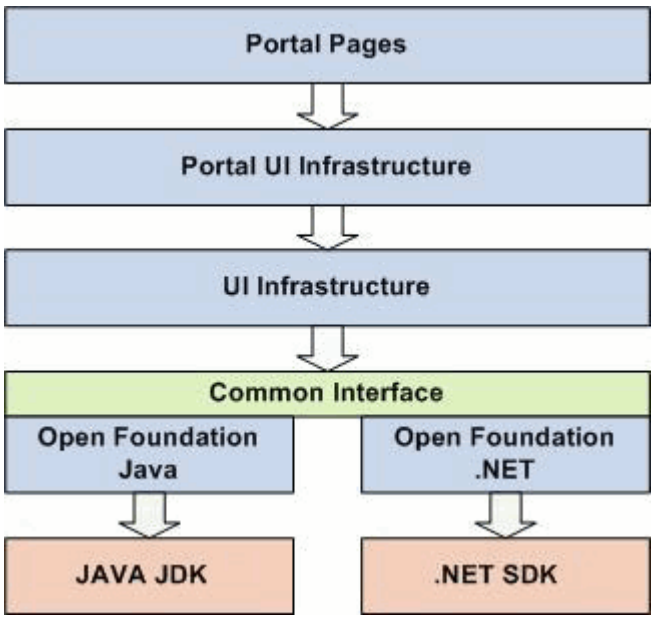
11

Portal UI Architecture

If you are implementing advanced UI customizations, this chapter provides detailed portal architecture information.

11.1 Portal UI Layers

The figure below shows the major portal UI projects and how they depend on each other. (For an introduction to the portal page design, see Chapter 2, "Portal Page Layout".)



11.1.1 Portal UI Infrastructure

Portal UI Infrastructure is another level of infrastructure and frameworks that mirrors UI Infrastructure and contains the implementation that depends on the portal server API. For example, the implementation of the tree in the UI Infrastructure project is a generic tree of items, whereas the version of the tree in Portal UI Infrastructure is a tree of PTObjects (i.e., Users, Groups, Content Crawlers, etc.).

11.1.2 Portal Pages

Portal Pages contains the source code for most of the portal's end-user pages. This project is divided into three main categories: admin pages (pages under

Administration), browsing pages (end-user pages that do not appear under Administration) and common pages (pages common to both Administration and end-user browsing). Within these three categories, packages are separated into feature groups, e.g., browsing.login, admin.editors.group , common.search.

There are many reasons the portal UI is built using a layered approach:

- **Unified Code Base**: The entire portal UI is written in Java and automatically converted to C# using a proprietary tool built specifically for the portal. Both versions of the UI undergo equal testing. Their level of quality is equivalent and they offer the exact same set of features.
- **MVC Architecture**: All UI source code is implemented following the Model-View-Control (MVC) design pattern. Separating presentation from logic makes front-end customization simpler, and facilitates upgrades when a new version of the portal becomes available. For more details on the MVC design pattern, see the next section.
- **Object-Oriented Code**: All UI source code is written in object-oriented and compiled code (Java or C#). As a consequence, there is no JSP or ASP. Object-oriented code has a number of advantages; the most important for the portal are upgrading, refactoring and maintenance.
- **Strong Infrastructure and Frameworks**: Several UI projects are specifically dedicated to infrastructure, frameworks and reusable components. Examples include the Activity Space framework (Chapter 14, "Creating Custom Activity Spaces"), Programmable Event Interfaces (Chapter 12, "Using PEIs"), and Dynamic Discovery (cChapter 18, "Deploying Custom Code Using Dynamic Discovery").These projects are extensively tested and should be leveraged as much as possible when customizing the UI.

11.2 MVC Architecture

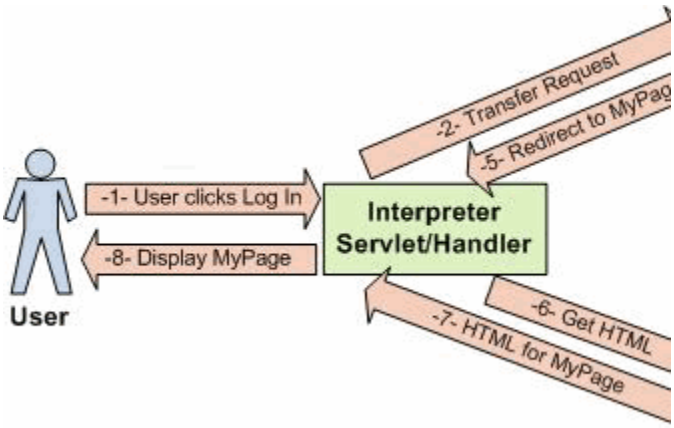
The architecture of the portal UI is based on the **Model-View-Control (MVC)** design pattern. The MVC paradigm allows you to separate the code that handles business logic from the code that controls presentation and event handling. Each page in the portal is made up of a combination of at least one Model and View, and one or more Controls.

- **Model** classes store the data for a page or page section. A single page might use one or more Model classes, depending on how much of the page data can be shared by other types of pages. A Model defines how data is accessed and set for a given page, including any functions necessary for security or data validation and modification. Models encapsulate calls to the portal server API and also store UI-specific data. Data that is globally accessed by the UI is available from the Activity Space object (discussed in the next section). All other data should be stored in a Model.
- **View** classes contain HTMLElements and HTMLConstructs that describe how the data from the Model should be displayed to the user. In the portal UI design, **DisplayPage** objects are used to aggregate View objects to encapsulate all the information needed to render a particular page. Some Views are common throughout the portal and some are specific to certain pages. For example, the banner that makes up the majority of the portal is a common View that defines the color scheme and where the search section will be displayed. In contrast, the View used to create and modify data within a User Profile is specific to the User Profile function and is seen only on that page.

- **Control** are actions or sets of actions that are executed when a specific event is triggered. Multiple Controls can be defined within a page, each with its own functional specification. For example, one Control might produce a popup window that allows the user to browse for a specific object and places the selection within the View, and another could save the new data to the Model.

11.2.1 Example: Login MVC Pattern

The diagram below shows a simplified version of how the classes used for the Login page interact with each other using the Model-View-Control (MVC) pattern.



1. The Control either redirects to another page or returns to the same page with new data since the Model has been updated. In this case, the Control redirects to the MyPage.
2. The Interpreter gets the HTML for the page requested by the Control from the appropriate View, in this case the **MyPage View**.
3. The View returns the HTML for the page.
4. The Interpreter sends the HTML for the page back to the browser.

Note that the redirect from the Login page to the MyPage occurred without returning to the client. These **Server Redirects** are handled by the Interpreter. The portal favors server redirects over regular redirects (i.e., Response.redirect) because they avoid unnecessary roundtrips to the client. Regular redirects are used in a few cases; for example, some Security Modes require a redirect to switch between HTTP and HTTPS pages.

11.3 Activity Spaces

An **Activity Space** groups multiple task-specific actions into a logical set and provides the programmer with base functionality. An Activity Space takes several related pages and turns them into one group of Model-View-Control objects. Most of the classes in the UI projects are part of an Activity Space, for example:

- Login page (plumtree.portalpages.browsing.login.LoginAS)
- My Pages (plumtree.portalpages.browsing.myportal.mypages.MyPageAS)
- Select Portlet page (plumtree.portalpages.browsing.objectselection.portlets.PortletSelectionEditorAS)
- Knowledge Directory (plumtree.portalpages.browsing.directory.DirAS)

- Search (plumtree.portalpages.browsing.search.basic.BasicSearchAS or plumtree.portalpages.browsing.search.advanced.AdvancedSearchAS)
- Group editor (plumtree.portalpages.admin.editors.group.GroupEditorAS)
- Tree Control (plumtree.portaluiinfrastructure.tree.TreeAS)

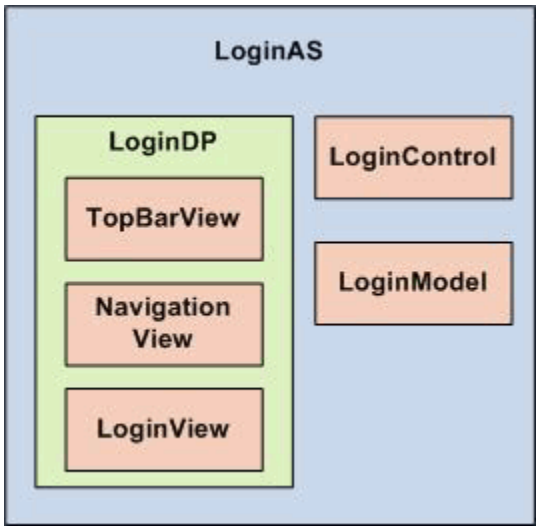
The Activity Space is the base unit; the Interpreter dispatches incoming requests to the correct Activity Space. Redirects can be done from one Activity Space to another. For example, when the client requests the Login page, the Interpreter redirects the request to LoginAS. As shown in the example in the previous section, LoginAS redirects to MyPageAS.

Each Activity Space contains at least one Model, View and Control. Most Activity Spaces contain at least one **Display Page**. Display Pages correspond to actual portal pages; there is a one-to-one mapping between portal pages and DP classes. Each portal page is broken into areas and one View is implemented for each area. The Display Page puts the Views together to render the page.

To browse to a specific Activity Space, you must add the name of the Activity Space to the URL. The name of the Activity Space corresponds to the value of the STR_MVC_CLASS_NAME constant in the AS class. For details, see the example that follows.

11.3.1 Example: Login Activity Space

The diagram below shows a simplified version of the members of the Login Activity Space. This Activity Space contains one Control, one Model, and one Display Page that contains three Views.



All classes in the UI projects follow the naming conventions shown in the diagram.

Object Type	Name Suffix
Activity Space	AS
Display Page	DP
View	View
Control	Control
Model	Model

The classes for the Login Activity Space (LoginAS) are located in the UI projects (com.plumtree.portalpages.browsing.login).

Note: NavigationView and TopBarView are located under \common because they are used in all portal pages.

As noted above, you can browse to an Activity Space by adding the name of the Activity Space to the URL. The name of the Activity Space corresponds to the value of the STR_MVC_CLASS_NAME constant in the AS class. For example, the value of STR_MVC_CLASS_NAME in LoginAS is "Login" so the URL to access the login page would be: **http://localhost/portal/server.pt?space=Login**.

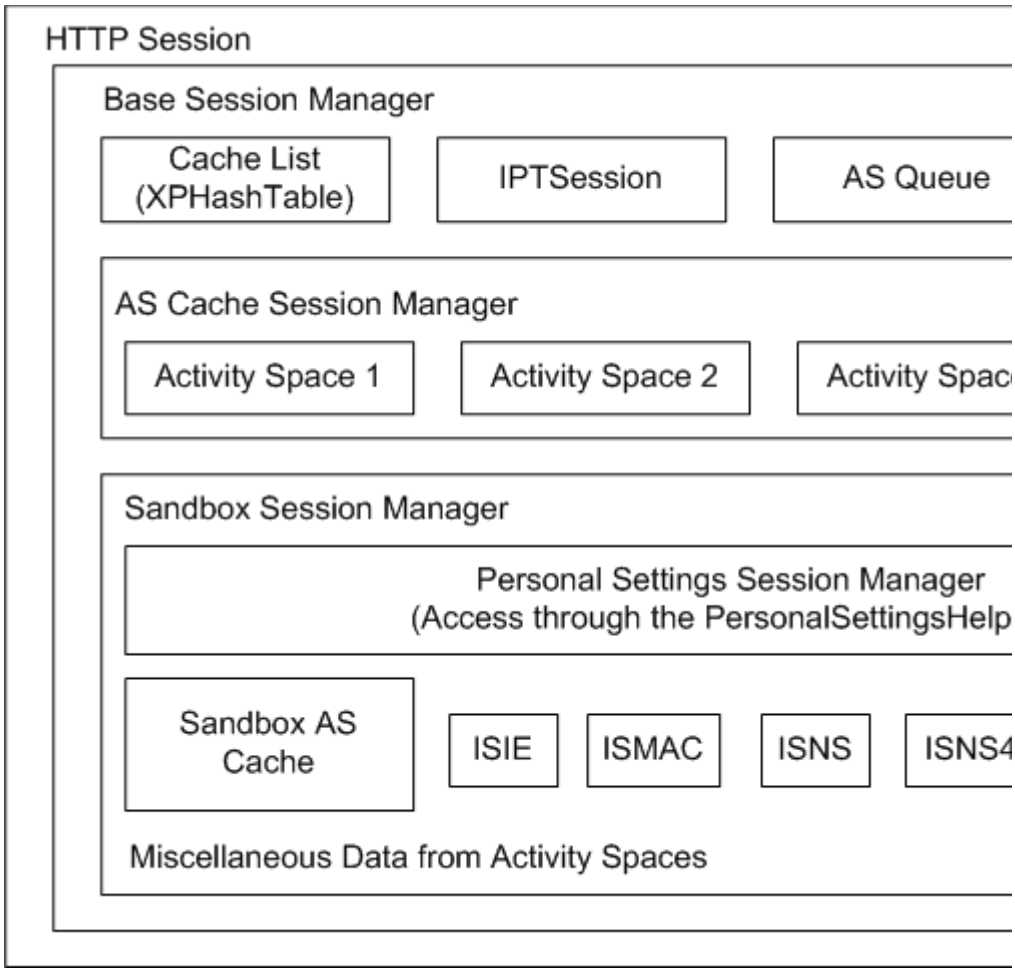
For more information and detailed instructions on Activity Space customization, see Chapter 14, "Creating Custom Activity Spaces".

11.4 Session Management

Access to the HTTP session in Java and the HTTP session state object in .NET is controlled through the **HTTPMemoryManagement** module. This wraps the underlying session object into a Session Manager, which allows you to manage the session hierarchically.

The session contains several infrastructure-level data structures, such as the Activity Space cache. It also provides a Sandbox Session Manager, where UI developers can store data on the session. In an MVC architecture, most data storage should be done on the Model, but there are situations where it is necessary to store data directly in the Sandbox Session Manager to share data between ActivitySpaces. Using the Model for data storage is preferable because the Model is cleaned up when the ActivitySpace is removed from the cache; the Sandbox is only cleaned up when the user logs out. The Sandbox contains booleans that describe the user's browser (i.e., IE, Netscape, Macintosh, etc.).

The diagram below shows what is stored on the session and how it is organized.



11.5 Request Control Flow

As explained in the previous sections, the portal UI is accessed through the MVC-based Activity Space Framework. Instead of navigating to UI pages via file links, each user request is handled by an Interpreter that determines the page (i.e., Activity Space) to be displayed.

This section summarizes what happens when a request is made by filling in the gaps between the user request and page display. These pages take a look inside the Interpreter to see some of the common checks that are processed for each request, explore the relationship between the Interpreter and Activity Spaces, and gain a better understanding of how control flows from one to another.

11.5.1 Interpreter Control Flow

When a request is made to the portal, the **Interpreter** controls the UI page to be displayed to the user. The Interpreter is simply a class in the UIInfrastructure library that serves as an entry point for HTTP requests; it "interprets" the query string to determine the contents of the request and decide where to send the user. Since the Interpreter is involved with every request, you should have an understanding of some of the logic it performs.

When an HTTP request is made to the portal entry point (e.g., server.pt), the **`HandleRequest`** method in the Interpreter is called. This method receives the

necessary information required for processing: request, response, application, and session. Several checks and functions are performed, summarized below in the order they occur:

1. **Check for successful startup:** If the portal fails to start up, the Interpreter cannot perform a redirect, because it has no Activity Space. Instead, it writes the following error message directly on the response: "The server has experienced an error on startup. This problem must be fixed before using the system." When starting your portal, open Logging Spy and check for a successful startup message for the UI Infrastructure component. If there is a problem, you should see a startup failure message in Logging Spy.

   In most cases, you will also see an error page in your browser, though some portal errors are drastic enough such that you will not reach the portal's error page. For example, if the path you supplied for the portal in your config file is incorrect, your libraries (including UIInfrastructure) will not load; the code will not reach the Interpreter and you will likely see an error message from your web server.

2. **Check for gatewayed request:** Gatewayed requests are handled by the **`HandleGatewayRequest`** method in the **`GatewayHandlers`** class. Any minor differences in processing are noted in the explanations that follow.

3. **Check security mode:** The Interpreter confirms that the request matches the current security mode of the portal:

   - In mode 0, the portal is either secure or insecure, depending on how the user accesses the portal.
   - In mode 1, only Activity Spaces specified in the config file SecureActivitySpaces are secure.
   - In mode 2, the entire portal is secure and all requests must be made via HTTPS.

4. To perform this check, `HandleRequest` calls the helper method **`CheckHTTPSecurityAndRedirect`**. This method checks whether the incoming request matches the security mode, and redirects as necessary. If the portal is expecting an SSL request (modes 1 or 2) but a non-SSL request comes in, the Interpreter creates a 302 redirect to the same page using a secure URL. This redirect is constructed using the secure URL mapping entered in the x_config.xml file:

   ```
   <!-- URLMapping - Entry 0 -->
   <URLFromRequest0 value="*"></URLFromRequest0>
   <ApplicationURL0 value="http://myserver/portal/server.pt"></ApplicationURL0>
   <SecureApplicationURL0
   value="https://myserver/portal/server.pt"></SecureApplicationURL0>
   ```

5. **Load query string settings:** At this point, the Interpreter accepts the request and attempts to interpret (parse) its content by calling the helper method **`LoadQSSettings`**. This method begins by extracting any expected query string arguments such as "space", "in_hi_space", "control", "in_hi_control", etc. These arguments are stored so they can be passed back for processing. Before returning to `HandleRequest`, the method performs two more functions. First, the arguments are checked for unsafe characters. If the arguments have suspicious characters (indicative of cross-site hack attempts), an error will be thrown and shown in Logging Spy.

6. Finally, the method checks the **Accepts** and **User-Agent** headers to determine if the request is coming from a known device, such as a PDA (the list of known devices is specified in the config file devices.xml). If a match is found, the appropriate markup format will be used; otherwise, standard HTML will be used.

7. **Increment performance counters:** The portal keeps track of several performance benchmarks, including total number of hits. The Interpreter is used to track this statistic because it is involved with every request. For each request made, this counter is incremented. (Gatewayed requests are recorded in a separate counter.)
8. **Session locking and request management:** After all the information is retrieved from the request, the Interpreter gets to work. To ensure that it works on one request at a time per session, a queue is used to store requests. If the queue has reached its limit, new requests are denied and the portal reports an error message.
9. **Determine browser settings:** After establishing a good request, the Interpreter retrieves and stores the user's browser settings, including the type of browser (IE or Netscape), and the version if available. These settings can then be used to make minor modifications so the UI displays correctly for the user.
10. **Log in:** Once the request is prepared, the Interpreter processes it through the portal. The first step is to make sure the user has the correct access. The Interpreter checks if the "Remember my Password" cookie has been set and is valid. If the cookie does not exist or is invalid, and the current session is empty or has timed out, the Interpreter redirects the user to the login page. Until a valid user logs in, the portal defaults the current user to Guest, and access is limited. The login process also involves experience rules evaluation; for details, see Section 11.5.3, "Experience Definition Control Flow". If the login check succeeds, the Interpreter continues, passing the control flow to the desired Activity Space, covered in the next section.

11.5.2 Activity Space Control Flow

After completing the basic checks listed in the previous section, the Interpreter turns the control flow over to the requested Activity Space. The major steps in this process are summarized below. The entire Activity Space control flow is illustrated in the diagram at the bottom of this section. Page requests also involve experience rules evaluation; for details, see Section 11.5.3, "Experience Definition Control Flow".

1. **Locate Activity Space:** Before an Activity Space can be run, it must first be loaded and initialized. Since multiple users are constantly accessing different portal pages, the Interpreter first looks in the cache to see if the Activity Space already exists. This action is shown in Logging Spy.
2. **Invoke OnPageStart PEI:** Before the Activity Space is processed, the **OnPageStart** PEI is invoked. This PEI allows you to customize the behavior of the portal before any page is loaded. To learn more about PEIs and supported events, see Chapter 12, "Using PEIs".
3. **Determine Control:** The first step in processing the Activity Space is to determine which Control should be used. If no Control is specified, then the Activity Space's default Control is used. If there is no default, the Control is left empty. The arguments for the Control are then retrieved. (For security purposes and SSO, the login page Control requires further processing.)
4. **Pass Control to Activity Space:** If the Control is valid, the **CheckActionSecurityAndExecute** method is called. This method is implemented for each Control, usually by the developer of the Activity Space, or by a development framework (e.g., the Editor framework for Activity Spaces). When the method is called, control flow essentially passes to the Activity Space. Any code implemented for the Activity Space is run according to the logic in the Control. When the method finishes, it returns a Redirect object, and control is passed back to the Interpreter. An example of this method, from PortalPages, is

shown below. In this implementation, the Control calls the Model to perform some action, and then returns a NULL for the Redirect object.

```
publicclass OpenSubFolderControl implements IControl

public RedirectCheckActionSecurityAndExecute(XPHashtablearguments)
{
     String[] sValues = (String[])arguments.GetElement(m_SubFolderID);

     if (sValues ==null)
     {
       log.Error("could not find folder ID in query string.");
       return null;
     }

     m_asModel.OpenSubFolder(XPConvert.ToInteger(sValues[0]));

     returnnull;
}
```

5. **Redirect Control:** When control returns to the Interpreter, it checks the Redirect object returned by the Activity Space:
   - If the Redirect object is NULL, the Interpreter processes the Display Page set by the Activity Space (or by the Control itself).
   - If the Redirect is not NULL, the Interpreter processes the Redirect in a loop until all internal Redirects are handled, at which point the final Display Page is processed. Note: The Gateway does not support internal Redirects. Gatewayed requests are executed through a true HTTP 302 redirect. The only time a redirect occurs from a gatewayed request is if the user does not have sufficient privileges, at which point the user is returned to the login page.
   - If the Redirect is an HTTP Redirect, the final Display Page response is set and the Interpreter starts from the beginning.

   With this mechanism, an Activity Space can use multiple controls, perform multiple actions, and repost as needed. The excerpt from PortalPages shown below demonstrates this logic. After the page is deleted, a redirect is created that sets a Control to direct the browser to the user's home page.

```
publicclass DeletePageControlimplements IControl

public RedirectCheckActionSecurityAndExecute(XPHashtablearguments)
{\
   String[] sValues = (String[])arguments.GetElement(STR_PAGE_ID);

   if(sValues == null)
   {
     log.Error("could not find page ID in query string.");
     return null;
   }

   int nDeletePageID = XPConvert.ToInteger(sValues[0]);

   m_asModel.DeletePage(nDeletePageID);

   IPTSession objSession = (IPTSession)m_asOwner.GetUserSession();
   int nUserID = objSession.GetSessionInfo().GetCurrentUserID();
   int nHomePageID = -1 * nUserID;

   Redirect redirect = new Redirect();
```

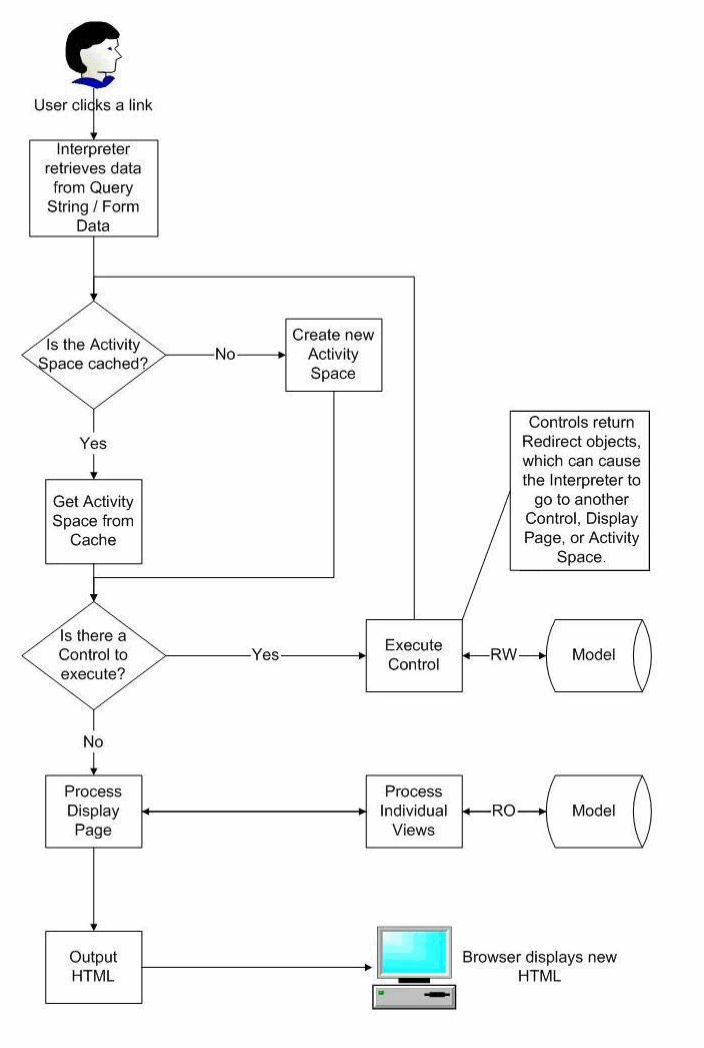
```
        redirect.SetLinkCreateNewSpace(m_asOwner.GetName(), m_asOwner);
        redirect.SetControl(SetPageControl.STR_MVC_CLASS_NAME);
        redirect.AddControlArgument(SetPageControl.STR_PAGE_ID, nHomePageID);

        return redirect;
    }
```

In the Logging Spy trace, the user begins on the MyPage Activity Space. When the user tries to delete a MyPage, control is delegated to `DeletePageControl`. The `RedirectCheckActionSecurityAndExecute` method is invoked, and as shown in the code above, a call is made to the Model to delete the page. This action is shown in Logging Spy ("... removed 1 pages ...").

`DeletePageControl` then creates a new Redirect by sending control to `SetPageControl` with the argument to go to the user's home page. Logging Spy shows that the Interpreter handles this redirect. Since `SetPageControl` does not return any further redirects, processing is complete, and the Display Page is shown.

6. **Render Display Page:** The control flow ends with the rendering of the Display Page. At this point the UI has completed loading, and the Interpreter waits for the user to make another request, restarting the cycle.

The diagram below illustrates the Activity Space control flow:



As noted in the previous section, gatewayed requests are treated differently by the Interpreter.

If the page contains portlets with tags, the request will also be handled by the Transformer; for details see Section 11.5.4, "Adaptive Tag Control Flow".

11.5.3 Experience Definition Control Flow

Experience definitions let you tailor portal experiences for different groups of users. In a single portal implementation, you can create a distinct user experience for each audience. Experience definitions let you specify which navigation and branding schemes, mandatory links, and default home pages to display to each set of users.

In every request cycle, experience rules are evaluated a maximum of two times. These two phases may or may not resolve to the same experience definition.

11.5.3.1 Login (Guest User) Evaluation

The first experience rules evaluation phase takes place when a user accesses the portal and has not yet been authenticated. This evaluation determines which Guest User object to log in. Since the current user has not been authenticated, the session needs a Guest User object to browse the portal.

The login experience rules evaluation returns the experience definition for the first rule that evaluates to true, and uses the associated Guest User object. (Each experience definition has an associated Guest User object and default login page, either the standard login page or the Guest User's My Page. For details on the Experience Definition Login Settings page, see the portal online help.)

Note: At this time, only experience rules relevant to unauthenticated users can be evaluated. Since no user is logged into the portal, and the destination page has not been determined, rules with conditions based on user properties or destination page are meaningless. Only rules with globally determined conditions like the time of day, browser type, request URL, etc., can be evaluated. If the evaluation of all relevant rules return false, the user is logged in as the Default Guest User object.

11.5.3.2 Page Request Evaluation

After a user is logged in, experience rules are evaluated to determine which experience definition object to use in displaying the requested page. This evaluation occurs after all the Control actions have executed, and all redirects have been followed. For more information on redirects, see Section 11.2, "MVC Architecture".

At this point in the request cycle, the destination page has been determined and the user is logged into the portal. The experience definition is determined as follows:

1. All experience rules are evaluated and the first rule that has all conditions met returns an experience definition.
2. If none of the experience rules evaluate to true, the experience definition is determined by folder association. (Each experience definition can be associated with a folder that contains User objects.)
3. If no experience definition is associated with the user's folder, the default experience definition is used.

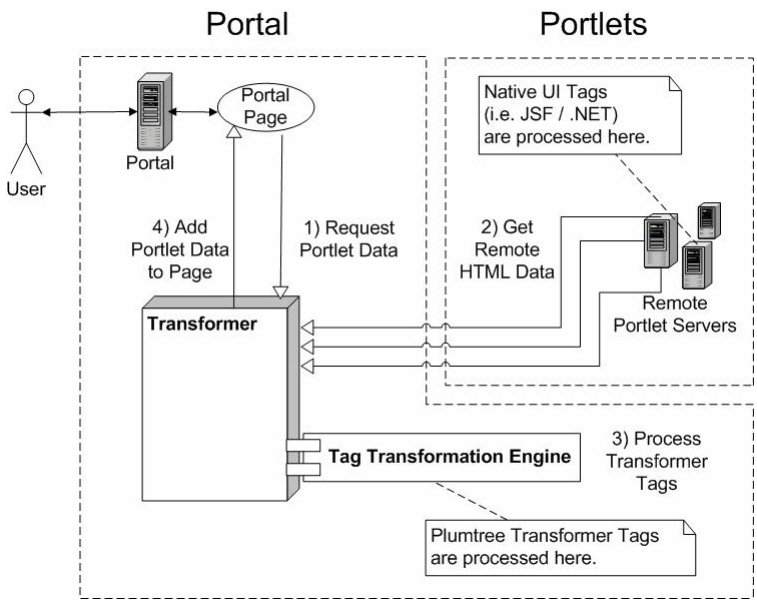
The requested page is displayed using the stylesheet, header navigation, etc., for the returned experience definition.

Note: Users are no longer tied to a single experience definition; therefore a user could view a page with one experience definition, click on a link and view the next page with a different experience definition.

11.5.4 Adaptive Tag Control Flow

Adaptive Tags allow web designers to utilize portal data directly in their HTML. The portal ships with a set of Adaptive Tag libraries, and you can create custom tags if additional functionality is needed.

The figure below shows the control flow of a typical portal request that makes use of Adaptive Tags.



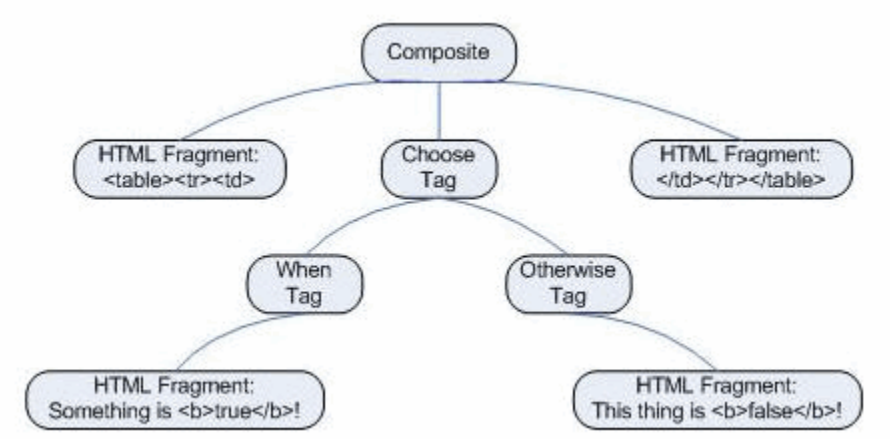
1. First, the portal page requests portlet data from the Transformer.
2. The Transformer retrieves the requested portlets from the remote portlet servers. Native UI tags, such as JSP Tags or .NET web controls, are processed on the remote server before the HTML is returned to the Transformer.
3. The Transformer converts the HTML into markup data including both HTML and Adaptive Tags. This markup data is passed to the Tag Transformation Engine, which processes the tags and converts them into standard HTML.
4. Finally, the HTML is returned to the portal page where it is displayed to the end user.

It is important to note that native UI tags are processed first on the remote portlet server, and Adaptive Tags are processed later in the Tag Transformation Engine, described below.

11.5.4.1 Tag Transformation Engine

The **Tag Transformation Engine** converts markup data from the Transformer into a tree of HTML and Adaptive Tags. The Engine moves through the tree and outputs HTML and processes the tags. When a tag is processed, it can cause all of its child nodes to be processed, or it can skip that entire section of the tree.

This figure below shows an example of a tree.



In this example, when the Choose tag is executed, it determines whether or not the current user matches the conditions in the choose clause. If it does, the When tag will display the HTML inside the tag. If not, the Otherwise tag will display its HTML. For details on these tags, see the *Oracle WebCenter Interaction Web Service Development Guide*.

12

Using PEIs

The **Programmable Event Interface (PEI)** framework defines a set of user actions that fire programmatic events that can be used to execute custom code without editing the portal source code. This approach allows you to upgrade the portal as service packs become available without losing your customizations.

A PEI is a Java or C# interface that defines event methods. For example, the ILoginActions PEI defines methods that are called when a user logs into the portal (e.g., On BeforeLogin, OnAfterLogin). The PEI framework provides interfaces only. To customize the portal using a PEI, you must create a class that implements the PEI and make the class available to the portal using Dynamic Discovery. This chapter provides a full ist of available PEIs and detailed instructions on implementation and deployment.

Note: The events available via PEIs are fired only in response to a user's direct action on the portal. If an action occurs as a result of some automated process or a direct call to the portal server, PEI methods are not called. For a thorough explanation of how PEIs are executed by the portal, see Section 12.5, "Lifecycle of a PEI" at the end of this chapter.

12.1 Step 1: Choosing a PEI

There is a wide range of PEIs available, each with a specific purpose. The table below summarizes the available PEIs and associated methods. For detailed information on available methods for each PEI, see the API documentation. (For links to all portal API documentation, see Appendix C, "Portal API Documentation".)

PEI Name and Description	Available Methods
ILoginActions and **ILoginActions2** (.uiinfrastructure.pei) allow you to create functionality within the scope of the login process. The login process is the most commonly customized functionality in the portal. ILoginActions2 extends ILoginActions and supports returning redirects for failed logins using SSO. For details, see the API documentation.	▪ OnBeforeLogin ▪ OnAfterLogin ▪ OnFailedLogin ▪ OnBeforeLogout
IOpenerActions (.uiinfrastructure.pei) allows you to implement custom functionality when the Common Opener is used to open an object or redirect to an Activity Space.	▪ OnBeforeOpen
IPageActions (.uiinfrastructure.pei) allows you to add code to every page processed by the portal. This PEI should be used sparingly.	▪ OnPageStart ▪ OnPageFinish

PEI Name and Description	Available Methods
IDisplayJavaScript (.uiinfrastructure.pei) allows you to add Javascript to every banner and editor page. This PEI should be used sparingly.	■ DisplayJavaScript
INewEditObjectActions (.portaluiinfrastructure.pei) allows you to implement custom functionality to be processed during the most common of all administrative actions: creating or editing a new object within the portal (the items that can be created from the Create Object menu in portal administration). For additional functionality, use IObjectActions (described below)	■ OnCreateObject ■ OnEditObject ■ OnBeforeStoreObject ■ OnAfterStoreObject
IDirectoryActions (.portalpages.pei) allows you to implement functionality in response to directory actions. For example, executing extra code when a user opens a folder or document within the portal Knowledge Directory.	■ OnOpenFolder ■ OnBeforeCreateDirectoryFolder ■ OnAfterCreateDirectoryFolder ■ OnBeforeDeleteDirectoryFolder ■ OAfterDeleteDirectoryFolder ■ OnBeforeDeleteDocument ■ OnBeforeCreateABOJob ■ OnAfterCreateABOJob ■ OnClickThroughToDoc
IUserProfileActions (.portalpages.pei) allows you to execute functionality when a user attempts to modify User Profile information.	■ OnBeforeChangeUserProfile ■ OnBeforeStoreUserProfile
IPasswordActions (.portalpages.pei) allows you to enforce restrictions on the password or verify the text entered by the user.	■ OnBeforeChangePassword
ICreateAccountActions (.portalpages.pei) allows you to execute functionality when a new user attempts to create an account, either through the Create Account button on the login page or in response to an invitation.	■ OnBeforeCreateAccount ■ OnAcceptInvite
IMyPortalPageActions (.portalpages.pei) allows you to perform validation before allowing users to add or remove portal pages.	■ OnBeforeAddMyPortalPage ■ OnAfterAddMyPortalPage ■ OnBeforeRemoveMyPortalPage ■ OnAfterRemoveMyPortalPage ■ OnBeforeEditMyPortalPage ■ OnAfterEditMyPortalPage
ICommunityActions (.portalpages.pei) allows you to add functionality dynamically when a user directly joins a Community or unsubscribes from a Community.	■ OnAfterUserJoinsCommunity ■ OnAfterUserQuitsCommunity
IAdvancedSearchActions and **IBannerSearchActions** (.portalpages.pei) allow you to make modifications to the query being processed.	■ CustomizeQueryOnBeforeSearch ■ GetCustomActionsOnBeforeSearch
INetworkSearchActions (.portalpages.pei) allows you to make changes to network searches after they have been submitted by the user.	■ OnBeforeNetworkSearchProcess

PEI Name and Description	Available Methods
ISearchSettingActions (.portalpages.pei) allows you to track creation and deletion of saved searches (Snapshot Queries), and control naming and encoding for new saved searches.	■ OnBeforeSaveSearch ■ OnAfterSaveSearch ■ OnBeforeRemoveSavedSearch ■ OnAfterModifyOrRemoveSavedSearch
IObjectActions (.portalpages.pei) allows you to add functionality to almost any event that occurs during portal administration, including Delete, Move, Copy, and Object Migration. Each method on this PEI is executed when the corresponding event is processed within portal Administration and determines if the process should continue or if modifications are required. Note: Use the *ObjectActions PEI sparingly; these functions are loaded and processed each time the corresponding event is called.	■ OnBeforeDeleteObject ■ OnBeforeMoveObject ■ OnBeforeMigrateObject ■ OnBeforeCopyObject ■ OnBeforeCreateABOJob ■ OnAfterCreateABOJob ■ OnBeforeCreateAdminFolder ■ OnBeforeCopyAdminFolder ■ OnBeforeDeleteAdminFolder

12.2 Implementing a PEI in a Custom Class

To customize portal functionality using a PEI, you must create a class that implements the PEI.

Below are some important objects used by multiple PEIs:

- The **`IPTSession`** object is always available to PEIs. If the object is not passed to the PEI in the argument, it can be retrieved from the `ActivitySpace` object (`((IPTSession)myActivitySpace.GetUserSession()`). The `IPTSession` object is the portal session for the current user. This object lets you perform any action that the current user has the rights to execute on the portal server. For example, if the current user has the necessary rights, you can create new portal objects, query existing objects, browse the Directory or query MyPages and communities.
- The **`ActivitySpace`** object is passed to many PEI functions. This object provides access to the user session and any user specific information, as well as other Activity Space-specific information including page name, Control name and server name. The `ActivitySpace` object is the container for all the Model, View, and Control classes that comprise a particular piece of functionality. Any objects required by a piece of functionality can usually be retrieved from the parent **ActivitySpace** object.
- The **`ApplicationData`** object is is passed to PEIs that are not sent the **ActivitySpace** object. The object provides access to application-specific data from the Activity Space, including the web application and the web session. The `ApplicationData` object also lets you perform some actions on the Request object. For example, you can get and set cookies, get the requested URL or set values on the HTTP header.

For more detailed information on objects and methods, see the API documentation.

To create a custom PEI, follow the steps below.

1. Create your own custom project and custom PEI class (for example, a CustomLoginPEI project and a CustomLoginPEI class in com.yourcompany.pei.login).

2. Edit the new class in your custom project.
3. Compile the new class into a new JAR/DLL file with an intuitive name. Note: You must re-compile PEIs against each new release of the .NET portal even if you have not made any modifications.

This section provides three examples of implementing PEIs: Example 1: Hello World Login PEI, Example 2: Login Usage Agreement, and Example 3: Banner Search Customization.

12.2.1 Example 1: Hello World Login PEI

The sample customization below adds a message ("HELLO WORLD ") to be displayed in Logging Spy after a user logs in.

The name of the file is important. All classes that implement PEIs should use a file name the references the name of the interface that they implement. The `ILoginActions` interface defines a few simple methods: `OnBeforeLogin`, `OnAfterLogin`, `OnFailedLogin`, and `OnBeforeLogout`. Use your IDE to navigate to the interface definition or view the API documentation for PEI classes.

1. This example uses the sample code from the sample UI projects package. For details, see Appendix B, "Installing UI Customization Sample Projects". Install the files and add the **sampleloginpei** project to your IDE.
2. Open the **HelloWorldLoginActions** file (.java or .cs). This file implements the **ILoginActions** PEI.
3. The **`OnAfterLogin()`** method allows for some functionality to occur once the user has successfully logged in and then possibly do a redirect to someplace other than the MyPage. As noted above, this code tells Logging Spy to to print out a HELLO WORLD string after a user logs in. This code uses the Error tracing type; you must confirm that Logging Spy Error tracing is enabled when you deploy the code to view the message.

   Java:

   ```
   public Redirect OnAfterLogin(Object _oUserSession, ApplicationData _appData)
   {
      // Print out "HELLO WORLD" to PTSpy
   log.Error("HELLO WORLD");
      return null;
   }
   ```

   C#:

   ```
   public virtual Redirect OnAfterLogin(Object _oUserSession, ApplicationData _
   appData)
   {
   // Print out "HELLO WORLD" to PTSpy
   log.Error("HELLO WORLD");
   return null;
   }
   ```

4. The remaining methods in `ILoginActions` are unused in this example, so they simply return null. They could be used to perform similar actions before logging in or out, or when a login fails. For a more advanced login customization, see the next example.

.NET only: After the code is complete, you must associate the main portal project with the custom project.

1. In Visual Studio, navigate to the SOURCE_HOME\portal.NET\prod and open portal*.sln.

2. Expand the project, right-click on References, and select Add Reference.
3. On the Projects tab, highlight the sampleloginpei project.
4. Click Select and OK. Click OK to close the References window.
5. Build the portal* solution to ensure that all projects build successfully.
6. Close Visual Studio.

Once you have written the code for your new PEI, you must deploy it for use by the portal, described in the next section, Section 12.3, "Step 3: Deploying a Custom PEI".

12.2.2 Example 2: Login Usage Agreement

This customization redirects guest users to a usage agreement page when they log in. Based on whether they accept or reject the agreement, they are redirected to the appropriate My Page or back to the Login page. The customization includes the following classes:

- The **LoginAgreementActions** class implements the **ILoginActions** PEI and executes custom code when users log in.
- The **GuestLoginAgreementControl** class implements both the **ILoginControl** and **IHTTPControl** interfaces and logs users in as a custom guest user.
- The **MarkAsGuestControl** class sets a variable on the HTTP Session to show that this custom guest user should still be treated as a guest user, basically invalidating their session.
- The **LoginAgreementRepostControl** class handles the users choice to accept the agreement or to reject it.

The sections below summarize the functionality of each class. To view all the code for this customization, see the sampleagreementlogin project.

12.2.2.1 LoginAgreementActions

The **LoginAgreementActions** class implements **ILoginActions** and executes custom code when users log in to the portal.

The **`OnAfterLogin()`** method gets the PTSession and checks if the login is for an actual user, then checks if the user has already accepted the agreement. If the user has accepted the agreement, the login proceeds to the portal normally; if the user has not accepted the agreement, the login page redirects to **GuestLoginAgreementControl**.

```
public Redirect OnAfterLogin(Object _oUserSession, ApplicationData _appData) {
   IPTSession ptSession = (IPTSession) _oUserSession;
if (ptSession.GetSessionInfo().GetCurrentUserID() != PT_INTRINSICS.PT_USER_GUEST)
{
IPTSessionInfo sessionInfo = ptSession.GetSessionInfo();
Object[][] result = sessionInfo.LookupPreference(AGREED, 0);
if (result[1][0] == null || !((String) result[1][0]).equals(TRUE)))
{
  Redirect guestRedirect = new Redirect();
  guestRedirect.SetLinkCreateNewSpace(LoginAgreementAS.STR_MVC_CLASS_NAME, null);
  guestRedirect.SetControl(GuestLoginAgreementControl.STR_MVC_CLASS_NAME);
  return guestRedirect;
}
else
{
  return null;
}
```

```
}
Return null;
}
```

12.2.2.2 GuestLoginAgreementControl

The **GuestLoginAgreementControl** class is responsible for logging users in and out. When users are first redirected to GuestLoginAgreementControl they are logged out and logged back in as a guest. After they have seen the agreements page, it redirects them back to GuestLoginAgreementControl, which logs users in through their account or redirects them back to the login page.

The GuestLoginAgreementControl class implements both the **ILoginControl** and **IHTTPControl** interfaces. The **`SetHttpItems()`** method accesses the HTTP request and PageData objects, used later in the **`AttemptLogin`** method. It is very important to clear these member variables (set them equal to null) after they have been used to make sure they are not leaked.

```
public void SetHTTPItems(IXPRequest _request, IWebData _pageData)
{
// In a given request, this method is called first.
m_xpRequest = _request;
m_WebData = _pageData;
}
```

The **`CheckActionSecurityAndExecute()`** method is used to return a Redirect object. This Redirect is followed after the login from the Control is processed. The Redirect object returned depends on the user's current status.

- When users are directed to this control the first time, they are redirected to **MarkAsGuestControl**, which marks them as a guest and invalidates the session. This is to prevent users from viewing parts of the portal to which they do not have access. The users **PTSession** is cached for later use.
- After users have viewed the agreements page and are redirected back to this control (the user session retrieved from the persistent sub-session is not null), the CheckActionSecurityAndExecute method checks whether they accepted the agreement or not. If they accepted the agreement, a redirect to their My Pages is returned. If they did not accept the agreement, they are redirected back to **MarkAsGuestControl**, which invalidates the session and redirects to the Login page.

```
public Redirect CheckActionSecurityAndExecute(XPHashtable _arguments)
{
Redirect rReturn = new Redirect();
rReturn.SetLinkCreateNewSpace(LoginAgreementAS.STR_MVC_CLASS_NAME, m_asOwner);
rReturn.SetControl(MarkAsGuestControl.STR_MVC_CLASS_NAME);
ISessionManager perSession = m_asOwner.GetPersistentSubSession();
IPTSession userSession = (IPTSession) perSession.GetAttribute(SESSION);
if (userSession != null)
{
IPTSessionInfo sessionInfo = userSession.GetSessionInfo();
Object[][] result = sessionInfo.LookupPreference(AGREED, 0);
String strResult = (String) result[1][0];
if (strResult != null && strResult.equals(TRUE))
{
      m_userSession = userSession;
      m_bAcceptance = true;
      Redirect myRedirect = new Redirect();
      myRedirect.SetLinkCreateNewSpace(MyPageAS.STR_MVC_CLASS_NAME, m_asOwner);
      myRedirect.SetControl(MyPageAS.STR_MVC_CLASS_NAME);
      return myRedirect;
```

```
}
else
{
      Redirect markGuest = new Redirect();
     markGuest.SetLinkCreateNewSpace(LoginAgreementAS.STR_MVC_CLASS_NAME, null);
      markGuest.SetControl(MarkAsGuestControl.STR_MVC_CLASS_NAME);
      perSession.RemoveAttribute(SESSION);
      return markGuest;
}
}
IPTSession ptsession = (IPTSession) m_asOwner.GetUserSession();
perSession.SetAttribute(SESSION, ptsession);
return (Redirect) rReturn;
}
```

The **`DoGetSession()`** method tells the Interpreter whether or not to log in a new user during this request. This method creates a default guest user PTSession to be used in `GetSession()` if the user has not accepted the usage agreement. If a user has already accepted the usage agreement, this method does nothing.

```
public boolean DoGetSession()
{
try
{
String strCustomGuestName = "guest";
String strCustomGuestPassword = "";
if (!m_bAcceptance)
{
     // Connect as the guest user
     m_userSession = PortalObjectsFactory.CreateSession();
     m_userSession.Connect(strCustomGuestName, strCustomGuestPassword, null);
}
}
catch (Exception e)
{
log.Error(e, "Unable to connect as guest.");
m_userSession = null;
// Unable to connect to custom guest user, do not log in
return false;
}
return true;
}
```

The **`GetSession()`** method returns a PTSession for the current user. The LoginHelper **`AttemptLogin`** method attempts to log users out of their account and log in to the default guest user account (using the PTSession created by `DoGetSession()` above) and calls all the appropriate login PEIs. This code nulls out the IHTTPControl member variables to make sure they are not leaked and returns the PTSession created earlier. It also removes the user's PTSession cached in the persistent sub-session if it is no longer needed.

Only after users have accepted the agreement are they allowed to log in through their account. In this case, the PTSession is the user session they originally logged in through, that was stored when they were first directed to this control in **CheckActionSecurityAndExecute**. If the agreement was not accepted, the PTSession returned is a new session for the guest user.

```
public Object GetSession()
{
if (null != m_userSession)
{
LoginResult rReturn = null;
```

```
try
{
     // Login the custom guest user.  This calls the login PEIs.
     rReturn = LoginHelper.INSTANCE.AttemptLogin(m_userSession, m_asOwner, m_
xpRequest, m_WebData);
}
catch (Exception e)
{
     log.Error(e, "AttemptLogin() failed.");
}

          if (!rReturn.m_bSuccess)
{
     log.Error("GuestLoginControl AttemptLogin() as guest failed: " + rReturn.m_
strError);
}

          if (null != rReturn.m_Redirect)
{
     log.Error("GuestLoginControl AttemptLogin() return redirect ignored.");
}
}
IPTSession userSession = m_userSession;
// Null out the IHTTPControl data so we don't retain the memory after the
// request is done (i.e. leak the memory)
m_xpRequest = null;
m_WebData = null;
m_userSession = null;

    // clean up
if (m_bAcceptance)
{
ISessionManager perSession = m_asOwner.GetPersistentSubSession();
perSession.RemoveAttribute(SESSION);
m_bAcceptance = false;
}
return userSession;
}
```

12.2.2.3 MarkAsGuestControl

The **MarkAsGuestControl** class is another essential component in this customization. In order to prevent users from potentially viewing parts of the portal to which they should not have access, it is necessary to invalidate their session when logged in. This control essentially marks users as a guest, making sure they are unable to access any portion of the portal. This control is used when users are first logged out and logged back in as a guest, and also if users reject the agreement and are logged in as a guest and redirected to the login page.

The **`CheckActionSecurityAndExecute()`** method in this control sets a variable on the HTTP Session to show that users should still be treated as a guest user. The method sets an attribute (variable) on the user sub-session (HTTP Session). Setting the `USERSESSIONVALID` attribute to False tells the TopBar to treat users as a guest or non-authenticated user. The method then checks whether the user rejected the agreement or has not yet been prompted with the agreement by checking for a cached PTSession in the persistent sub-session. If the user has not yet been prompted with the agreement (i.e., there is a PTSession in the sub-session), the login cookie is removed to prevent a session timeout and they are redirected to **LoginAgreementRepostControl**. If the user has rejected the agreement, they are redirected to the login page.

```
public Redirect CheckActionSecurityAndExecute(XPHashtable _arguments)
{
m_asOwner.GetSubSession().SetAttribute(Interpreter.USERSESSIONVALID,
Boolean.FALSE);
ISessionManager perSession = m_asOwner.GetPersistentSubSession();
IPTSession userSession = (IPTSession) perSession.GetAttribute(SESSION);
if (userSession != null)
{
LoginHandlers.ClearLoginOccurredCookiePresent(m_asOwner.GetCurrentHTTPResponse());
Redirect rReturn = new Redirect();
rReturn.SetIsHTTPRedirect(true);
rReturn.SetLinkGetSpaceIfCached(LoginAgreementAS.STR_MVC_CLASS_NAME, m_asOwner);
rReturn.SetControl(LoginAgreementRepostControl.STR_MVC_CLASS_NAME);
return rReturn;
}
else
{
Redirect toLogin = new Redirect(); toLogin.SetLinkGetSpaceIfCached(LoginAS.STR_
MVC_CLASS_NAME, m_asOwner); toLogin.SetControl(DefaultLoginControl.STR_MVC_CLASS_
NAME);
toLogin.SetControl(LoginControl.STR_MVC_CLASS_NAME);
return toLogin;
}
}
```

12.2.2.4 LoginAgreementRepostControl

The **LoginAgreementRepostControl** class implements the second half of this customization. GuestLoginAgreementControl handles the necessary details of logging a user in and out of the portal, while LoginAgreementRepostControl handles the corresponding user actions on the agreements page.

The **`CheckActionSecurityAndExecute()`** method in this control first checks if the user has already accepted the agreement. If users accept the agreement by clicking OK, a value is stored in the users preferences so they do not see the agreements page on future logins. Whether they accept the agreement or not, all users are redirected back to GuestLoginAgreementControl, which redirects to the proper Activity Space (the login page or the user's My Pages). The cached PTSession in the persistent sub-session is removed by GuestLoginAgreementControl either in `GetSession()` or `CheckActionSecurityAndExecute()` depending on whether the user has accepted the agreement or not.

```
public Redirect CheckActionSecurityAndExecute(XPHashtable arguments)
{
String[] sPostToSelf = (String[]) arguments.GetElement(RepostControl.HTMLINPUT_
POSTTOSELF);
if(sPostToSelf == null)
{
 return null;
}
else
{
 if(XPConvert.ToInteger(sPostToSelf[0])== POSTTOSELF_ACTION_OK)
 {
      ISessionManager perSession = m_asOwner.GetPersistentSubSession();
      IPTSession newSession = (IPTSession) perSession.GetAttribute(SESSION);
      IPTSessionInfo mySessionInfo = newSession.GetSessionInfo();
      mySessionInfo.AddPreference(AGREED, TRUE, 0);
      Redirect guestRedirect = new Redirect();
      guestRedirect.SetLinkCreateNewSpace(LoginAgreementAS.STR_MVC_CLASS_NAME,
```

```
null);
      return guestRedirect;
 }
 else if(XPConvert.ToInteger(sPostToSelf[0]) == POSTTOSELF_ACTION_CANCEL)
 {
      Redirect guestRedirect = new Redirect();
      guestRedirect.SetLinkCreateNewSpace(LoginAgreementAS.STR_MVC_CLASS_NAME,
null);
      guestRedirect.SetControl(GuestLoginAgreementControl.STR_MVC_CLASS_NAME);
      return guestRedirect;
 }
 else
 {
      log.Debug("invalid POSTTOSELF option.");
      return null;
 }
}
}
```

12.2.3 Example 3: Banner Search Customization

The examples in this section customize portal banner search functionality through the **IBeforeBannerSearchActions** PEI.

- Adding Strings to Search Queries
- Adding Properties to Search Fields
- Adding Constraints to Properties
- Restricting Banner Search

12.2.3.1 Adding Strings to Search Queries

This code adds the string "oracle" to every banner search query.

```
package com.samplecompany.portalpages.pei;
import com.plumtree.server.*;
import com.plumtree.uiinfrastructure.activityspace.*;
import com.plumtree.portaluiinfrastructure.search.*;
import com.plumtree.portalpages.pei.*;
public class SampleBannerSearchPEI1 implements IBannerSearchActions
{
public void CustomizeQueryOnBeforeSearch(AActivitySpace _asCurrentSpace,
IPTSession _ptUserSession, QueryArguments _qaQueryInfo)
{
// Require items to contain the string "oracle" in
// addition to the user's query

_qaQueryInfo.userQuery = "(" + _qaQueryInfo.userQuery + ") and oracle";
}
public SearchSettingCollection GetCustomSettingsOnBeforeSearch(AActivitySpace _
asCurrentSpace, IPTSession _ptUserSession)
{
return null;
}
}
```

12.2.3.2 Adding Properties to Search Fields

This example adds an author property to the set of fields to be searched.

```
package com.samplecompany.portalpages.pei;
import com.plumtree.server.*;
import com.plumtree.uiinfrastructure.activityspace.*;
import com.plumtree.portaluiinfrastructure.search.*;
import com.plumtree.portalpages.pei.*;
public class SampleBannerSearchPEI2 implements IBannerSearchActions
{
public void CustomizeQueryOnBeforeSearch(AActivitySpace _asCurrentSpace,
IPTSession _ptUserSession, QueryArguments _qaQueryInfo)
{
// Add the "author" property (property 103) to the set
// of fields to be searched
_qaQueryInfo.basicFields = "PT1[0.5],PT2[0.2],PT50[0.2],PT103[0.1]";
}
public SearchSettingCollection GetCustomSettingsOnBeforeSearch(AActivitySpace _
asCurrentSpace, IPTSession _ptUserSession)
{
return null;
}
}
```

12.2.3.3 Adding Constraints to Properties

This example adds a constraint that the author property must contain "oracle" (in addition to the user's query, which can match on Name, Description, or Content).

```
package com.samplecompany.portalpages.pei;

import com.plumtree.server.*;
import com.plumtree.uiinfrastructure.activityspace.*;
import com.plumtree.portaluiinfrastructure.search.*;
import com.plumtree.portalpages.pei.*;

public class SampleBannerSearchPEI3 implements IBannerSearchActions

{

public void CustomizeQueryOnBeforeSearch(AActivitySpace _asCurrentSpace,
IPTSession _ptUserSession, QueryArguments _qaQueryInfo)

{

// Add a constraint that the "author" property contain
// "oracle"in addition to the user's query (on the name,
// description, and content fields)

// Create an advanced-search filter to replace the simple query
IPTFilter filter = PortalObjectsFactory.CreateSearchFilter();

// Set the user's query as the search string
filter.SetSearchString(_qaQueryInfo.userQuery);

// Want to AND the user's query with the property constraint
filter.SetOperator(PT_BOOLOPS.PT_BOOLOP_AND);

// Create the property part of the filter
IPTPropertyFilterClauses clause =
(IPTPropertyFilterClauses)
filter.GetNewFilterItem(PT_FILTER_ITEM_TYPES.PT_FILTER_ITEM_CLAUSES);
clause.SetOperator(PT_BOOLOPS.PT_BOOLOP_AND);
```

```
// Attach it to the filter
filter.SetPropertyFilter(clause);

// Create the single "author contains oracle" statement
IPTPropertyFilterStatement statement =
(IPTPropertyFilterStatement)  filter.GetNewFilterItem(
PT_FILTER_ITEM_TYPES.PT_FILTER_ITEM_STATEMENT);
statement.SetOperand(103);   // property 103 == author
statement.SetOperator(PT_FILTEROPS.PT_FILTEROP_CONTAINS);
statement.SetValue("oracle");

// Attach statement to clause
clause.AddItem(statement, 0);

// Use the filter in place of the original user query
_qaQueryInfo.advancedFilter = filter;
_qaQueryInfo.userQuery = null;

}

public SearchSettingCollection GetCustomSettingsOnBeforeSearch(AActivitySpace _
asCurrentSpace, IPTSession _ptUserSession)
{
return null;
}
}
```

12.2.3.4 Restricting Banner Search

This example restricts banner search to match only documents and folders by turning off banner search of users, portlets, communities, Collaboration, and Publisher. This code also turns off spell correction.

```
package com.samplecompany.portalpages.pei;
import com.plumtree.server.*;
import com.plumtree.uiinfrastructure.activityspace.*;
import com.plumtree.portaluiinfrastructure.search.*;
import com.plumtree.portalpages.pei.*;

public class SampleBannerSearchPEI4 implements IBannerSearchActions
{
public void CustomizeQueryOnBeforeSearch(AActivitySpace _asCurrentSpace,
IPTSession _ptUserSession, QueryArguments _qaQueryInfo)
{
// Nothing here
}
public SearchSettingCollection GetCustomSettingsOnBeforeSearch(AActivitySpace _
asCurrentSpace, IPTSession _ptUserSession)
{
SearchSettingCollection c = new SearchSettingCollection();
// Restrict to cards and folders only, no other objects
c.add(PT_SEARCH_SETTING.PT_SEARCHSETTING_OBJTYPES,
new int[] { PT_CLASSIDS.PT_CATALOGCARD_ID, PT_CLASSIDS.PT_CATALOGFOLDER_ID });

// Restrict to portal items only, no Collab or Content
c.add(PT_SEARCH_SETTING.PT_SEARCHSETTING_APPS, PT_SEARCH_APPS.PT_SEARCH_APPS_
PORTAL);
// Turn off spell correction
c.add(PT_SEARCH_SETTING.PT_SEARCHSETTING_SPELLCHECK, false);
return c;
}
```

```
}
```

Once you have written the code for your new PEI, you must deploy it for use by the portal, described in the next section.

12.3 Step 3: Deploying a Custom PEI

After you create a custom project as described in the previous section, you must deploy it to the portal using **Dynamic Discovery**. For detailed information and instructions, see Chapter 18, "Deploying Custom Code Using Dynamic Discovery". To deploy a PEI, use Interface-Based Dynamic Discovery.

The example below deploys the Hello World Login PEI sample code from the previous section. Always confirm that your code was deployed correctly, explained in Section 12.3.2, "Viewing Your Customization in the Portal" at the bottom of this section.

12.3.1 Example: Deploying the Hello World Login PEI

These instructions use Visual Studio in .NET and Ant scripts in Java to deploy your custom code.

First, add the library containing the new HelloWorldLoginAction class to the **LoginActions.xml** file so it can be deployed by Dynamic Discovery.

1. Navigate to **PT_HOME\settings\portal\dynamicloads\PEIs** and open **LoginActions.xml** in a text editor.
2. Add the name of the new PEI to the existing XML as shown below. Make sure that the **HelloWorldLoginAction** is listed *after* the **PTLoginActions** class and the spelling and capitalization is exactly the same as the full class name.

```
<root>
<interface name="com.plumtree.uiinfrastructure.pei.ILoginActions"/>
<interfaceassembly name="uiinfrastructure"/>
<class name="com.plumtree.portalpages.pei.PTLoginActions"/>
<class name="com.plumtree.sampleui.pei.HelloWorldLoginActions"/>
</root>
```

You must also run a clean build in order to deploy the custom code.

Java:

1. Open a command prompt and change the directory to the \ptwebui directory where you installed the portal source code
2. Run a clean build using the following Ant script: `ant build`.
3. Generate a new WAR file for the application server using the following Ant script: `ant install`.

   Note: This target deletes and rebuilds all jar files associated with all the UI source projects (as well as the custom projects in the ptwebui folder).

C#:

1. Build the project in Visual Studio.
2. Visual Studio should copy the sampleview.dll file from SOURCE_HOME\sampleview\dotnet\prod\bin to PORTAL_HOME\webapp\portal\bin for you. If there are problems with Dynamic Discovery on startup, you might need to do this step manually. This is necessary to allow Dynamic Discovery to find the new library.

12.3.2 Viewing Your Customization in the Portal

Once you have deployed your code, view the changes in the portal to confirm that they were loaded correctly.

1. Open Logging Spy. For details, see the *Administrator Guide for Oracle WebCenter Interaction*.
2. Click the **Set Filters** button to open the **Filter Settings** dialog. Make sure the **Error** checkbox is selected. (You will not be able to see the customization run if this logging level is not enabled.)
3. Start the portal and view Logging Spy. During startup, you should see a message about the loading of LoginActions classes; two ILoginAction classes should be loaded. If no ILoginActions classes were loaded, you might have misspelled or mis-capitalized one of the names.
4. Open a new browser window and navigate to the portal. Do not log in. You should see the "HELLO WORLD" string in Logging Spy. This is because when you first hit the portal, you are logged in as the guest user automatically to display the login page.
5. Login as the administrator. You should see the "HELLO WORLD" string again in Logging Spy because you have explicitly logged in as a user.

The next step is to debug your code.

12.4 Step 4: Debugging and Troubleshooting

This section provides technical tips for common problems and instructions on how to debug your new PEI.

12.4.1 Technical Tips

If your custom PEI does not function, first make sure the full class name of the **HelloWorldLoginAction** PEI is listed in LoginActions.xml *exactly* as it is spelled in the code. It will not load if spelled or capitalized incorrectly. This may not produce any errors during startup. One way to check that the ILoginActions loaded correctly is to make sure that Logging Spy says the correct number was loaded (2). This is explained in Section 12.3.2, "Viewing Your Customization in the Portal".

If this does not solve the problem, debug your code using the instructions below.

12.4.2 Debugging

These instructions use the Hello World Login PEI class created in the previous sections as an example.

Java

1. In Eclipse, stop the Tomcat debugging session and open **HelloWorldLoginActions.java**.
2. Add a breakpoint at the `log.Error` line .
3. In the Eclipse menu, click **Run** | **Debug…** and select the Tomcat application.
4. Choose the **Classpath** tab, select **Add Projects**, and add the sampleloginpei project.
5. Hit **Debug** (and **Save** to retain your changes).

6. Navigate to your portal and view the login page. You should hit this breakpoint, since you are automatically logged in as the guest user when you first view the portal in a new browser.

C#

1. Stop the Visual Studio debugger (and close your browser if it is still open) and open **HelloWorldLoginActions.cs** in Visual Studio.
2. Add a breakpoint at the `log.Error` line.
3. Start the Visual Studio debugger (F5 or Start | Debug).
4. Navigate to your portal and view the login page. You should hit this breakpoint, since you are automatically logged in as the guest user when you first view the portal in a new browser.

12.5 Lifecycle of a PEI

This section traces the lifecycle of a PEI and provide a comprehensive view of what happens to it, from the mechanism that loads the PEI to the actual code invoked when a PEI event occurs.

12.5.1 Step 1: Loading the PEI

All PEIs are loaded at runtime when the portal first starts up. Because PEIs are loaded using Dynamic Discovery, they can be plugged into the portal without modifying existing UI code. This section examines some of the code that makes up the PEI infrastructure. This code is from the UIInfrastructure project, and the source for this project is not distributed with the portal; it is included here solely for the purpose of understanding PEIs.

When the portal first starts up, it is initialized using an Init method in the `AppWarmUp` class from the com.plumtree.uiinfrastructure package:

```
public class AppWarmUp
public static final void Init(String strVarPackXMLFile, String strApplicationName,
String _strPlatform)
```

The **`Init`** method initializes most aspects of the portal, including the loading of PEIs. The following code from the `Init` method shows exactly how it happens. First, it establishes the path from which the portal will get the PEI loading information, which is in the \dynamicloads\PEIs folder inside the portal configuration folder (PT_HOME\settings\portal\dynamicloads\PEIs). This directory contains the files that allow you to add a new PEI into the system. The code then makes a call to the `LoadSettings` helper method with this path.

```
String strPortalDynamicLoadFolder = strPortalConfFolder + strFileSeparator +
"dynamicloads" + strFileSeparator;
// CODE TRUNCATED HERE FOR BREVITY
try
{
   // Load Dynamic Loads
   if(PTDebug.IsInfoTracingEnabled(Component.UI_Infrastructure))
   {
     PTDebug.Trace(Component.UI_Infrastructure, TraceType.Info, "Loading the
dynamic loads.");
   }
      LoadSettings(application, strPortalDynamicLoadFolder, strLibHomePath);
}
```

The `LoadSettings` helper method takes the path to the \dynamicloads\PEIs folder, and attempts to discover the corresponding PEI for each XML file in the folder (the looping of files is omitted in the code below). As the information from each XML file is loaded and the PEI instances are created, they are stored in the portal application; the `strCacheString` reference in the code below is then used to retrieve the instances (explained in the next section).

The `GetCachingManager.SetEntry` (and analogous `GetCachingManager.GetEntry`) methods are frequently used to put custom objects on the portal application. These objects can then be retrieved in other custom code. These methods are also used in other dynamically discovered objects, such as custom navigation schemes.

```
private static final void LoadSettings(IApplication app, String strSettingsFolder,
String strLibHomePath)
strCacheString = strFileName.substring(0, nEndIndex);
// CODE TRUNCATED HERE FOR BREVITY
try
{
   settings = XPDynamicDiscovery.GetInstancesFromXML(strSettingsFolder +
strFileName, strLibHomePath);
   if (null != settings)
   {
     if (PTDebug.IsInfoTracingEnabled(Component.UI_Infrastructure))
     {
       PTDebug.Trace(Component.UI_Infrastructure, TraceType.Info,
         " Found: " + settings.length + " instances of interface" +
         " described in " + strCacheString);
     }
   }
}
// CODE TRUNCATED HERE FOR BREVITY
app.GetCachingManager().SetEntry(strCacheString, settings);
```

Each XML file in the \dynamicloads\PEIs directory is processed at startup. If Logging Spy is running, you can view a report of how many interfaces were discovered for each file. This is a good way to validate that your PEI was added correctly. For example, if you want to add a LoginActions PEI, first start the portal without changing the LoginActions.xml file. Write down the number of instances for LoginActions that Logging Spy reports. Add your PEI to LoginActions.xml, restart the portal, and compare the number of instances to the number you wrote down. It should have incremented by the number of interfaces you added.

12.5.1.1 Memory Debug Page

The Memory Debug page is another useful tool for gathering information about the portal. This page provides a summary of your memory internals and lists the objects residing in the HTTPSession, the Activity Space Cache, the Portal Application, etc. As explained above, the PEI instances created from each XML file are stored in the Portal Application cache. The Memory Debug page displays an object representation of an array for each of the XML files (UserProfileActions is highlighted in the image below).

To access the Memory Debug page, log in to the portal as a user with Administrative rights, and navigate to the page by appending the following string to the portal URL: ?space=MemoryDebug.

Once the PEIs are loaded, the portal must execute them in response to the appropriate actions, as explained next.

12.5.2 Step 2: Executing the PEI

To understand what happens when a PEI event is invoked, you must look at the supporting PEI framework code in the portal. The code that supports each PEI differs slightly, but the concept is the same.

This example looks at the code that supports the OnAfterLogin PEI, located in the LoginHelper class in the com.plumtree.uiinfrastructure.login package. The **`DoTasksAfterLogin`** method is called by the portal after a successful login attempt:

```
public final class LoginHelper

private Redirect DoTasksAfterLogin(Object userSession, IXPRequest request,
IWebData webData, IApplication application, ISessionManager sessionManager)
```

One of the first actions of `DoTasksAfterLogin` is to retrieve the PEIs loaded in the portal application cache. As explained in the previous section, PEI instances are created from the XML files in the \dynamicloads\PEIs folder and stored on the portal application cache using strCacheString.

The **`strCacheString`** key is the filename of the XML file without the extension. This example deals with a Login PEI, so the `DoTasksAfterLogin` method uses the key "LoginActions" to refer to the PEI instances loaded from processing the LoginActions.XML file.

```
oa = (Object[]) application.GetCachingManager().GetEntry("LoginActions");
```

After the appropriate PEI instances are retrieved from the application, the following loop iterates through them by calling the **`OnAfterLogin`** method on each of the LoginAction PEI instances. As you can see from the logic and the comment in the code, each PEI placed in LoginActions.XML is called in order until one of them returns a valid Redirect.

For example:

- If none of the PEIs return a valid Redirect, all of the PEIs will be called.
- If the first PEI returns a valid Redirect, then it will be the only PEI called. All others will be ignored.
- If the method called on the PEI returns void, another PEI will be called.

Note: This is the general process for PEI framework code, but there are exceptions in which the loop is different. For example, the `DoTaskOnFailedLogin` method that handles `OnFailedLogin` for a Login PEI returns a String instead of a Redirect. In this case, the method calls all instances of the PEIs regardless of the String returned, concatenates them together and returns the final result.

```
for(int x = 0; x < oa.length; x ++)
{
   o = oa[x];
   // CODE TRUNCATED HERE FOR BREVITY
   if (o != null)
   {
      iActions = (ILoginActions) o;
      //   OnAfterLogin returns a Redirect.
      //   If it returns a valid object then this Redirect is stored,
      //   and possibly returned. If there are multiple implementations
      //   of this method, and multiple valid redirects are returned,
      //   then the last one will stick, and the others will be forgotten.
      rTemp = iActions.OnAfterLogin(userSession, myData);
      if (rTemp != null)
      {
        rReturn = rTemp;
```

```
            return rReturn;
          }
        }
      }
```

13

Using View Replacement

The architecture of the portal UI is based on the **Model, View and Control (MVC)** design pattern. Well known among UI developers, MVC enables you to separate the code that handles business logic from the code that controls presentation and event handling. Each page in the portal is made up of a combination of at least one Model and View, and can include one or more Controls.

- A **Model** class stores the data for a page or page section. A single page might use one or more Model classes, depending on how much of the page data can be shared by other types of pages. A Model defines how data is accessed and set for a given page. Models encapsulate calls to the portal server API and also store UI-specific data.
- A **View** class contains HTMLElements and HTMLConstructs that describe how the data from the Model should be displayed to the user. In the Oracle WebCenter Interaction UI design, a DisplayPage object aggregates one or more View objects to encapsulate all the programmatic information needed to render a particular page. Some Views are common throughout the portal and some are specific to certain pages. For example, the banner that makes up the majority of the portal is a common View that defines the color scheme and the location of the search section. In contrast, the View used to create and modify data within a User Profile is specific to the User Profile function and is seen only on that page.
- A **Control** is an action or set of actions that are executed when a specific event is triggered. Multiple Controls can be defined within a page, each with its own functional specification. For example, one Control might produce a popup window that allows the user to browse for a specific object and places the selection within the View, and another could save the new data to the Model.

You can customize the display of portal components by creating a custom version of the associated **View** class(es). The Oracle WebCenter Interaction UI Framework allows you to implement your customizations without modifying the portal code. This approach is safer, more efficient, and facilitates future upgrades.

Note: In most cases, you should modify only the View class of an Activity Space. Modifying Model and Control modules is supported, but consistency problems could occur if you do not test all related modules carefully.

This chapter provides instructions on how to create and deploy a custom View, and includes sample code that shows how to create a new view for the portal login page. The Hello World Login Page example illustrates how you can replace sections of portal code with your own customized code. As long as you maintain the contract dictated by Oracle WebCenter Interaction interfaces, your code will plug in seamlessly.

13.1 Identifying the Activity Space

In order to change the HTML displayed in the portal, you must locate where the HTML is generated in the portal code. Portal code is grouped into Activity Spaces, which contain multiple Views that generate the actual HTML. Therefore, you must first identify the Activity Space responsible for the page you want to modify; then you can find the View that creates the HTML you are interested in, and replace that View with your custom View.

There are several ways to find the code for the component you want to customize. It is usually easiest to find the name of an Activity Space by looking at the associated page URL or Logging Spy, and then browsing the source code to find that Activity Space.

There are two ways to find the name of an Activity Space:

- Open a page in the portal that includes the component. The URL to the page should contain a query string argument that specifies the name of the Activity Space. Look in the query string for **space=** or **in_hi_space=**. The value after the equals sign should be the STR_MVC_CLASS_NAME of the Activity Space. (Note: This approach will not work if you navigated to the page via a form post, as the URL will not show the query string arguments.)
- Turn on Logging Spy with Info and Action tracing enabled. Open a page in the portal that includes the component. You will see several messages in Logging Spy regarding the current Activity Space and Display Page (e.g., "current space is Login", "current control is DefaultLoginControl", and "Displaying page Login"). These messages can help you determine which Activity Space and/or Display Page is generating the HTML you want to customize. The first message ("current space") contains the name of the current Activity Space. The Display Page, noted in the "Displaying page" message, groups together different Views into a single HTML page.

Once you have found the name of the Activity Space, search for it in the portal UI packages (com.plumtree.uiinfrastructure, com.plumtree.portaluiinfrastructure, and com.plumtree.portalpages) and determine which View(s) you want to modify.

- **com.plumtree.portalpages**: Most Views will be located in this package, which contains the UI code for the majority of the portal.
- **com.plumtree.uiinfrastructure**: This package contains generic framework components that are used to build the UI.
- **com.plumtree.portaluiinfrastructure**: This package contains portal-specific framework components.

The portalpages and portaluiinfrastructure packages are divided into admin, browsing, and common sections. The admin section is for the Administrative Site (Editors, etc...) and the browsing section contains end-user facing pages (MyPages, Directory, etc...). The common section contains code that is used for both admin and browsing.

13.1.1 Example: Hello World Login Page

For example, you might locate the View that creates the HTML for the portal login page by following the steps below.

1. Open the portal in your browser and click the "Login" link.
2. Look at the URL in your browser. Right after the question mark, it should say "space=Login" (the ordering of query string arguments is not guaranteed). This tells you that the Login HTML is generated by the Login Activity Space. In

Internet Explorer, if the window you are viewing does not have the Address bar, you can often hit Ctrl-N and open the page in a new browser window that will have the Address bar.

3. Open up your IDE and view the portal source code.
4. The Login page is a page in the portal, so it should be in the portalpages package. The Login page is viewed by end-users, not just administrators, so it is most likely in the browsing package, although parts of it might be in the common package.
5. Browse to com.plumtree.portalpages.browsing.login and look for a file named LoginAS. "AS" stands for Activity Space. There is also a file named LoginView; that is the file you will modify.

You can also search the UI source code using your IDE or Windows Explorer to find the correct files. You can search for either the Activity Space name or a unique string in the HTML source. To find a unique string in the HTML source, open a page in the portal that includes the component. View the HTML source for the page and find a unique (non-generated) tag in the section of the HTML that includes the UI component you want to customize. Search for the tag in the portal UI packages.

13.2 Creating a Custom View

When you modify a piece of the UI, never change existing source code. It is a best practice to start with the original source, making modifications as necessary. The UI source code is shipped with the product. (For links to all portal API documentation, see Appendix C, "Portal API Documentation".)

The best way to modify an existing View is to extend it and override the methods that you want to change. This way you can avoid class cast exceptions if any code references the original class name.

To create a custom View, follow the steps below. For a simplified example, see the Example: Hello World Login Page sample code that follows.

1. Create your own custom project and custom View class (e.g., a CustomLogin project and a CustomLoginView class in com.yourcompany.login).
2. Edit the new class in your custom project. Make sure to follow the **Requirements and Best Practices** below:
   - When replacing a pre-existing View with a custom View (as opposed to creating a brand new View), the STR_MVC_CLASS_NAME returned by the `GetName()` method must not be modified; this ensures that the portal will replace the original View with your custom View. To find the STR_MVC_CLASS_NAME, look in the View class' `GetName()` method (the constant also appears at the top of the file).
   - If you are modifying only one component of the Activity Space, make sure to import the original Activity Space package to guarantee that the other modules will be available.
   - The `Create()` method must return a new instance of the custom class. Otherwise, when the portal attempts to instantiate a new instance of the custom View using the **Create()** method, it will not work. A common problem is cutting and pasting code from an existing View and then forgetting to update this method. Your customization will be loaded by the portal, but the original View will still be displayed.
3. Compile the new class into a new JAR/DLL file with an intuitive name.

IMPORTANT: In most cases, you should modify only the View class of an Activity Space. Modifying Model and Control components is supported, but consistency problems could occur if you do not test all related modules carefully. When you upgrade to a new release of the portal, it is very important to check all customized files to see if the original versions have been modified in the upgrade. This way you can migrate any new features and bug fixes into your modified version. (For information on modifying the functionality of the login page, see Chapter 12, "Using PEIs".)

13.2.1 Example: Hello World Login Page

For example, the sample customization below starts with existing code and adds a line ("Hello World") to be displayed on the login page.

The name of the file is important; it ends with "View." All presentation layer classes that implement the IView interface follow this naming convention. The IView interface defines a few simple methods: Init, Display, and DisplayJavaScript. Use your IDE to navigate to the interface definition or view the API documentation for the IView class.

1. This example uses the sample code from the sample UI projects package. For details, see Appendix B, "Installing UI Customization Sample Projects". Install the files and add the sampleview project to your IDE.
2. Open the **HelloWorldView** file (.java or .cs) in the sampleview project. To create this file, you would make a copy of the **LoginView** file at com.plumtree.portalpages.browsing.login and make the modifications detailed below.
3. The **`Create()`** method gets a new instance of the View when it is needed. It is very important to update this method when copying a file; otherwise your custom class will return an instance of the original class, and your customization will not appear in the portal.

   Java:

   ```
   public Object Create()
   {
      return new HelloWorldView();
   }
   ```

   C#:

   ```
   public virtual Object Create()
   {
        return new HelloWorldView();
   }
   ```

4. The **`GetName()`** method returns the name of the View, which is used to store and retrieve the View class in the portal and in the ActivitySpace. When overriding an existing View, this method must return the same value as the View that will be overridden.

   Java:

   ```
   public String GetName()
   {
      return STR_MVC_CLASS_NAME;
   }
   ```

   C#:

   ```
   public virtual String GetName()
   {
      return STR_MVC_CLASS_NAME;
   }
   ```

5. The **`Init()`** method provides the View with access to the model and parent Activity Space. Copy this code directly from the **`LoginView`** class.

   Java:

   ```
   public void Init(IModelRO model, AActivitySpace parent)
   {
      m_asmLoginModelRO = (ILoginModelRO) model;
      m_asOwner    = parent;
   }
   ```

   C#:

   ```
   public virtual void Init(IModelRO model, AActivitySpace parent)
   {
     m_asmLoginModelRO = (ILoginModelRO) model;
     m_asOwner = parent;
   }
   ```

6. The **`DisplayJavascript()`** method is used to add javascript to the page. This example does not use javascript for this example, so it returns null.

   Java:

   ```
   public HTMLScript DisplayJavascript()
   {
      return null;
   }
   ```

   C#:

   ```
   public virtual HTMLScript DisplayJavascript()
   {
      return null;
   }
   ```

7. The **`Display()`** method creates the HTML for display to the user. Copy the code for this method from the **`LoginView`** class (com.plumtree.portalpages.browsing.login). This code outputs a table containing the HTML for the login form.

   This example adds a row to the table that prints the string "HELLO WORLD." (It adds a cell to the row and prints the string in the cell.) As shown in the code snippet below, the code is added after calling the **`MakeLoginForm()`** helper method. This helper method creates the HTML form that contains the username and password text boxes. (This code is in a separate class so it can be reused in the Login Portlet.)

   Java:

   ```
   LoginHTML.MakeLoginForm(myForm, sFormName, m_asOwner.GetName(),
      m_asOwner.GetSpaceID(), LoginControl.STR_MVC_CLASS_NAME,
      m_asOwner, b508, "" , myCreateLink, bShowAuthsourceDropdown,
   bAllowAuthSourceDropdownDisplay);
   /*
   * Add this custom code to print HELLO WORLD after the Login Form directions.
   */
   HTMLTableRow myRow = new HTMLTableRow();
   myTable.AddInnerHTMLElement(myRow);
   HTMLTableCell myCell = new HTMLTableCell();
   myRow.AddInnerHTMLElement(myCell);
   myCell.AddInnerHTMLEncodedString( "HELLO WORLD" );
   /*
   * End of HELLO WORLD code
   */
   ```

   C#:

```
LoginHTML.MakeLoginForm(myForm, sFormName, m_asOwner.GetName(), m_
asOwner.GetSpaceID(),
  LoginControl.STR_MVC_CLASS_NAME, m_asOwner, b508, "", myCreateLink,
bShowAuthsourceDropdown, bAllowAuthSourceDropdownDisplay);
/*
* Add this custom code to print HELLO WORLD after the Login Form directions.
*/
HTMLTableRow myRow = new HTMLTableRow();
myTable.AddInnerHTMLElement(myRow);
HTMLTableCell myCell = new HTMLTableCell();
myRow.AddInnerHTMLElement(myCell);
myCell.AddInnerHTMLEncodedString("HELLO WORLD");
/*
* End of HELLO WORLD code
*/
```

Once you have written the code for your new View, you must deploy it for use by the portal, described in the next section

13.3 Deploying a Custom View

After you create a custom View as explained in the previous section, you must deploy it to the portal using **Dynamic Discovery**. For detailed information and instructions, see Chapter 18, "Deploying Custom Code Using Dynamic Discovery". To deploy a View replacement, use Jar or DLL-Based Dynamic Discovery.

The example below deploys the Hello World Login Page sample code from the previous section. Once you have deployed you code, confirm that your code was deployed correctly as explained in Section 13.3.2, "Viewing Your Customization in the Portal" at the bottom of this section.

13.3.1 Example: Hello World Login Page

These instructions use Visual Studio in .NET and Ant scripts in Java to deploy your custom code.

First, add the library containing the new HelloWorldView class to the **CustomActivitySpaces.xml** file so it can be deployed by Dynamic Discovery.

1. Navigate to **PT_HOME\settings\portal** and open **CustomActivitySpaces.xml** in a text editor (you might have to make the file writable). **Note**: Do not modify the ActivitySpaces.xml file. The CustomActivitySpaces.xml file is functionally identical to the ActivitySpaces.xml file and allows you to enumerate custom components without modifying the code used by standard portal components.
2. Find the `<AppLibFiles>` tag and add an entry for your project:

   ```
   <AppLibFiles> <libfile name="sampleview"/> </AppLibFiles>
   ```

You must also run a clean build in order to deploy the custom code.

Java:

1. Open a command prompt and change the directory to the \ptwebui directory where you installed the portal source code
2. Run a clean build using the following Ant script: `ant build`
3. Generate a new WAR file for the application server using the following Ant script: `ant install`

   Note: This target deletes and rebuilds all jar files associated with all the UI source projects (as well as the custom projects in the ptwebui folder).

C#:

1. Build the project in Visual Studio.
2. Visual Studio should copy the sampleview.dll file from SOURCE_HOME\sampleview\dotnet\prod\bin to PORTAL_HOME\webapp\portal\bin for you. If there are problems with Dynamic Discovery on startup, you might need to do this step manually. This is necessary to allow Dynamic Discovery to find the new library.

13.3.2 Viewing Your Customization in the Portal

Once you have deployed your code, view the changes in the portal to confirm that they were loaded correctly. Use Logging Spy to catch any obvious errors.

1. Open Logging Spy. For details, see the *Administrator Guide for Oracle WebCenter Interaction*.
2. Start the portal.
3. Open a new browser window and navigate to the portal. You should see your customization on the login page (the "HELLO WORLD" string appears after the login instructions).

The next step is to debug your code, covered in the next section.

13.4 Debugging and Troubleshooting

This section provides technical tips for common problems and instructions on how to debug your new class.

13.4.1 Technical Tips

If your custom View is not displayed correctly, first check the following items:

- The `GetName()` method in your custom class must return the same STR_MVC_CLASS_NAME used by the class that will be overridden; this ensures that the portal will replace the original View with your custom View. To find the STR_MVC_CLASS_NAME, look in the View class' `GetName()` method (the constant also appears at the top of the file).
- The `Create()` method must return a new instance of the custom class. Otherwise, your customization will be loaded by the portal, but the original View will still be displayed.
- If you modified only one component of the Activity Space, make sure you imported the original Activity Space package to provide access to the other modules.

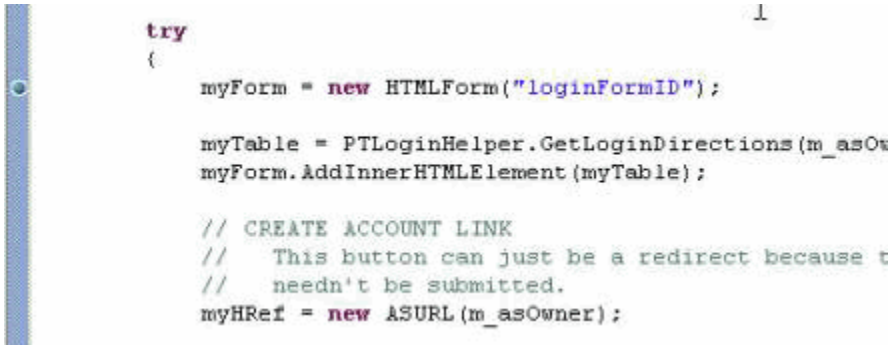
If none of these tips solve the problem, debug your code using the instructions below.

13.4.2 Debugging

These instructions use the Hello World Login Page class created in the previous section as an example.

Java

1. In Eclipse, stop the Tomcat debugging session and open **SampleView.java**.
2. Add a breakpoint as shown below:



3. In the Eclipse menu, click **Run** | **Debug…** and select the Tomcat application.
4. Choose the **Classpath** tab, select **Add Projects**, and add the sampleview project.
5. Hit **Debug** (and **Save** to retain your changes).
6. Open a browser and navigate to your portal. You should hit the breakpoint, since you are debugging the login page.

C#

1. Stop the Visual Studio debugger (and close your browser if it is still open) and open **HelloWorldView.cs** in Visual Studio.
2. Add a breakpoint as shown below:

3. Start the Visual Studio debugger (F5 or Start | Debug).
4. Navigate to your portal and log in again. You should hit this breakpoint, since you are debugging the login page.

14

Creating Custom Activity Spaces

Activity Spaces group task-specific actions into logical sets to provide portal developers with base functionality, and combine related pages to create cohesive Model-View-Control (MVC) objects. Everything in the portal is an Activity Space: a MyPage, an administrative editor, even the Directory tree.

For example, consider the Content Crawler editor used to create a new Content Crawler object in the portal. The entire editor is represented by one Activity Space object that uses several Models, Views/DisplayPages, and Controls. The Content Crawler Editor Activity Space also uses functionality inherited from base classes, including banner display and page-to-page editor navigation.

Every Activity Space within the portal is derived from one of the base Activity Space classes, which include the following:

- The **Editor Activity Space** defines how form-based editors work within the MVC paradigm. This framework is used for activities that require one or more pages of data entry, including creating portal objects and modifying settings that affect the entire portal. As noted above, the Content Crawler Editor inherits functionality from this base class.
- The **AForm Activity Space** defines a basic form submission page that does not require advanced editor functionality. This Activity Space is used in the MyPage, Knowledge Directory, and other portal pages.

Recently used Activity Spaces are cached on the user's HTTP Session. (A caching algorithm helps ensure a scalable HTTP Session size limit.)

A custom Activity Space allows you to add new pages to your portal. As long as you maintain the contract dictated by the portal interfaces, your code will plug in seamlessly. This chapter provides details on Activity Space components as well as step-by-step instructions on creating a custom Activity Space.

Note: To customize existing pages, the recommended approach is to use Adaptive Layouts; for details, see Chapter 3, "Using Adaptive Page Layouts". To change existing UI code or add new components not available via Adaptive Layouts, use View Replacement; for details, see Chapter 13, "Using View Replacement".

14.1 Activity Space Components

Each Activity Space includes the following classes:

- Activity Space
- Display Page
- Model

- View
- Control (optional)

Activity Spaces can include multiple implementations of DisplayPage, Model, View and Control.

14.1.1 Activity Space

The **Activity Space** class contains the Model-View-Control (MVC) framework and is responsible for initializing the Model, View and Control objects. It also stores data that is globally accessed by the UI.

14.1.2 Display Page

A **DisplayPage** class returns the View(s) used for the page. This simple class puts together the HTML and makes it viewable to the user. Each DisplayPage corresponds to an actual portal page; there is a one-to-one mapping between DisplayPages and portal pages.

14.1.3 Model

A **Model** class defines how data is accessed and set for a given page, including any functions necessary for security or data validation and modification. This class also encapsulates calls to the Portal Server API and stores UI-specific data. Control classes use a Model class to perform changes requested by the user. View classes use a Model class to retrieve data for display.

Note: Model classes must implement a **"Read-Only" Model interface** in order to discourage data manipulation in View classes. In most cases, you should add all public accessor methods to this read-only interface.

14.1.4 View

A **View** class contains the HTML for the component it represents and describes how the data from the associated Model should be displayed to the user.

14.1.5 Control

A **Control** class contains an action or set of actions that will be executed when a specific event is triggered. For example, a mouse click could trigger a popup window that displays a View.

14.2 Step 1: Creating an Activity Space

When you modify a piece of the portal UI, you should never change existing source code. You can create new, separate pages by using custom Activity Spaces, or overwrite existing code in a class through View Replacement. The UI source code is shipped with the portal, and you can view or download the API documentation for the portal UI packages.

To create a custom Activity Space, follow the steps below. For a simplified example, see the Example: Sample Activity Space sample code that follows.

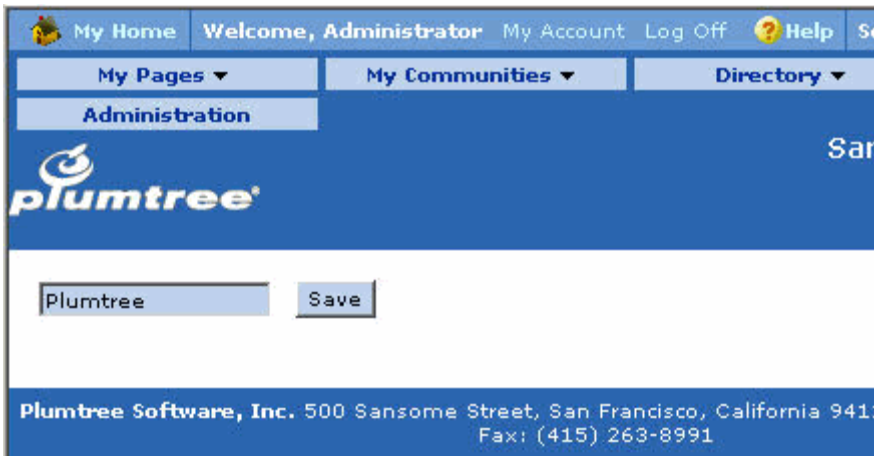
1. Create your own custom project, including the custom classes listed in the previous section.

2. Edit each class in your custom project, following the **Requirements and Best Practices** below:
   - The **`Create()`** method must return a new instance of the custom class (i.e., `CustomActivitySpace()`). Otherwise, when the portal attempts to instantiate a new instance of the custom classes using the Create method, it will not work. A common problem is cutting and pasting code from an existing Model, View or Control class and then forgetting to update this method.
   - The **`GetRepostControlName()`** method inside the `SampleActivitySpaceAS` class must return the name for the Control you want to use.
   - The name returned by the **`GetName()`** method must be unique for each type. For example, you can only have one View named "test" and only one Model named "test".
3. Compile each class into a new jar/dll file with an intuitive name (for example, CustomActivitySpaceAS).

14.2.1 Example: Sample Activity Space

This sample customization demonstrates how to create a sample Activity Space with a simple change text feature to illustrate the use of a Control for repost actions. It contains two modes to display two different looks for the page: display and edit. An Activity Space can have more than one View, but in this example, the code for the two possible Views overlaps, so it is easier to use a variable to switch between the two modes. When the user first accesses the sample Activity Space, the page is displayed in Display Text mode as shown below.

When the user clicks the Change Text button, the page is reloaded and displayed in Edit Text mode as shown below. The user can then type a different message in the text box. When the user clicks Save, the page is reloaded again, and displays the new message in Display Text mode.



This sampleactivityspace project uses the form framework and repost actions. The code does not allow guest users to access the custom Activity Space. This Activity Space has one custom View class, and utilizes a mode mechanism to display two different looks. The DisplayPage class extends PlumtreeDP, so the page also utilizes common Views (i.e., navigation, header and footer).

The name of the files is important; each one must follow the naming convention for the class it inherits or interface(s) it implements. For example, the name of the custom Activity Space class must end in "AS" (i.e., SampleActivitySpaceAS).

This example uses the sample code in the sample UI projects package. For details, see Appendix B, "Installing UI Customization Sample Projects". Install the files and add the sampleactivityspace project to your IDE.

1. Open the **SampleActivitySpaceAS** file (.java or .cs) in the sampleactivityspace project.

   The **`CheckBasicAccess()`** method guarantees that the guest user will not be able to access this Activity Space.

   Java:

```
public boolean CheckBasicAccess(String_strPage,String_strControl,
     boolean bSameSpace)
   {
     super.CheckBasicAccess(_strPage,_strControl,bSameSpace);
     //guest users cannot have access
     return !PlumtreeHelpers.IsGuestSession(this);
   }
```

   C#:

```
public override bool CheckBasicAccess(String _strPage, String _strControl,
bool bSameSpace)
 {
base.CheckBasicAccess(_strPage, _strControl, bSameSpace);
//guest users cannot have access
return !PlumtreeHelpers.IsGuestSession(this);
 }
```

   The **`GetRepostControlName()`** method is required if the Activity Space has a Control. This method will return the name of the repost control class. If this method is not included, the repost control will not work. (If your Activity Space does not have a Control, this method is not needed.)

   Java:

```
public String GetRepostControlName()
   {
return SampleActivitySpaceRepostControl.STR_MVC_CLASS_NAME;
   }
```

C#:

```
public override String GetRepostControlName()
 {
return SampleActivitySpaceRepostControl.STR_MVC_CLASS_NAME;
 }
```

The **`Init()`** method registers the Model, DisplayPage, View, and Control.

Java:

```
public void Init(){

super.Init();

// Model
RegisterModel(SampleActivitySpaceModel.STR_MVC_CLASS_NAME);
IModel myModel=GetModel(SampleActivitySpaceModel.STR_MVC_CLASS_NAME);

// Display Page
SampleActivitySpaceDPmyPage= new SampleActivitySpaceDP();
// use the form framework
myPage.SetAddMainForm(true);
RegisterPage(myPage);

// View
RegisterView(SampleActivitySpaceView.STR_MVC_CLASS_NAME,myModel);

//Control
RegisterControl(SampleActivitySpaceRepostControl.STR_MVC_CLASS_NAME, myModel);
}
```

C#:

```
public override void Init()
{
base.Init();

// Model
RegisterModel(SampleActivitySpaceModel.STR_MVC_CLASS_NAME);
IModel myModel =   GetModel(SampleActivitySpaceModel.STR_MVC_CLASS_NAME);

// Display Page
SampleActivitySpaceDP myPage = new SampleActivitySpaceDP();
// use the form framework
myPage.SetAddMainForm(true);
RegisterPage(myPage);

// View
RegisterView(SampleActivitySpaceView.STR_MVC_CLASS_NAME,   myModel);

//Control
RegisterControl(SampleActivitySpaceRepostControl.STR_MVC_CLASS_NAME, myModel);
}
```

2. Open the **SampleActivitySpaceDP** file (.java or .cs) in the sampleactivityspace project.

   The **`GetTitleForBanner()`** method returns the title to be displayed on the header.

Java:

```
public String GetTitleForBanner()
{
return "Sample Activity Space";
}
```

C#:

```
public override String GetTitleForBanner()
{
return "Sample Activity Space";
}
```

The **`GetSubtitleForBanner()`** method returns the subtitle to be displayed on the header.

Java:

```
public String GetSubtitleForBanner()
{
return "Page 1";
}
```

C#:

```
public override String GetSubtitleForBanner()
{
return "Page 1";
}
```

The **`PageDisplay()`** method gets all the HTML from the View(s) and puts it together to build the page's display.

Java:

```
public HTMLElement PageDisplay()
{
return m_asOwner.GetView(SampleActivitySpaceView.STR_MVC_CLASS_NAME).Display();
}
```

C#:

```
public override HTMLElement PageDisplay()
{
return m_asOwner.GetView(SampleActivitySpaceView.STR_MVC_CLASS_NAME).Display();
}
```

3. Open the **SampleActivitySpaceModel** file (.java or .cs) in the sampleactivityspace project. The class contains a field `m_nMode` to store the page's current mode. The text that is displayed is stored by the `m_strDisplayText` variable.

   The **`SavePage()`** method saves the text entered in the text box while the page is in edit mode.

   Java:

```
public int SavePage(String_sPageName, XPHashtable_htFormData)
{
String[]data;
data=(String[])_htFormData.GetElement(SampleActivitySpaceView.EDIT_TEXT_BOX);
if((data!=null)&&!(0==data.length))
{
       m_strDisplayText=data[0];
}
return RepostControl.PAGE_STATUS_VALID;
}
```

   C#:

```
public virtual int SavePage(String _sPageName, XPHashtable _htFormData)
```

```
{
String[] data;
data = (String[])     _htFormData.GetElement(SampleActivitySpaceView.EDIT_TEXT_
BOX);
if ((data != null) && !(0 == data.Length))
{
     m_strDisplayText = data[0];
}
return RepostControl.PAGE_STATUS_VALID;
}
```

The **`ChangeToEditMode()`** method sets the mode variable to EDIT_TEXT_MODE.

Java:

```
public void ChangeToEditMode()
{
 m_nMode = EDIT_TEXT_MODE;
}
```

C#:

```
public virtual void ChangeToEditMode()
{
 m_nMode = EDIT_TEXT_MODE;
}
```

The **`ChangeToDisplayMode()`** method sets the mode variable to DISPLAY_TEXT_MODE.

Java:

```
public void ChangeToDisplayMode()
{
m_nMode = DISPLAY_TEXT_MODE;
}
```

C#:

```
public virtual void ChangeToDisplayMode()
{
 m_nMode = DISPLAY_TEXT_MODE;
}
```

4. Open the **SampleActivitySpaceRepostControl** file (.java or .cs) in the sampleactivityspace project.

   The **`PerformAction()`** method calls the method in the Model associated to the action performed on the View. For example, a repost action takes the input and saves it under a hidden variable to be passed as an argument to the Control's PerformAction method. In this example, the POSTTOSELF_ACTION_CHANGE_TEXT action includes another method call to the model, which alters the mode variable.

   Java:

```
protected void PerformAction(int _nAction){
super.PerformAction(_nAction);
switch(_nAction){
       case POSTTOSELF_ACTION_CHANGE_TEXT:
         ((SampleActivitySpaceModel)m_model).ChangeToEditMode();
         break;
       case POSTTOSELF_ACTION_DISPLAY_TEXT:
                  ((SampleActivitySpaceModel)m_model).ChangeToDisplayMode();
         break;
     }
   }
```

C#:

```
protected override void PerformAction(int _nAction)
{
  base.PerformAction(_nAction);
  switch (_nAction){
    case POSTTOSELF_ACTION_CHANGE_TEXT:
     ((SampleActivitySpaceModel) m_model).ChangeToEditMode();
      break;
    case POSTTOSELF_ACTION_DISPLAY_TEXT:
     ((SampleActivitySpaceModel) m_model).ChangeToDisplayMode();
      break;
 }
}
```

5. Open the **SampleActivitySpaceView** file (.java or .cs) in the sampleactivityspace project.

   The **`Display()`** method calls the `GetHTMLForDisplayText()` and `GetHTMLForEditText()` helper methods to display the HTML Elements.

   Java:

```
public HTMLElementDisplay(){
int nMode;
HTMLElement myResult = new HTMLElementCollection();
try
{
    // extract the mode from the model
    nMode=((SampleActivitySpaceModel)m_model).GetMode();
       switch(nMode){
         case DISPLAY_TEXT_MODE:
           myResult.AddInnerHTMLElement(GetHTMLForDisplayText());
           break;
         case EDIT_TEXT_MODE:
           myResult.AddInnerHTMLElement(GetHTMLForEditText());
           break;
       }
}
catch(Exception e){
       log.Error(Component.Portal_Admin,
 "Unexpected exception when displaying the sample activity space.");
       }
     }
return myResult;
}
```

   C#:

```
public virtual HTMLElement Display()
{
int nMode;
HTMLElement myResult = new HTMLElementCollection();
try
{
   // extract the mode from the model
   nMode = ((SampleActivitySpaceModel) m_model).GetMode();
    switch (nMode){
        case DISPLAY_TEXT_MODE:
           myResult.AddInnerHTMLElement(GetHTMLForDisplayText());
           break;
        case EDIT_TEXT_MODE:
           myResult.AddInnerHTMLElement(GetHTMLForEditText());
           break;
```

```
   }
}
catch (Exception e)
{
   log.Error(Component.Portal_Admin, "Unexpected exception when displaying the
sample activity space.");
}
return myResult;
}
```

The following portion of the helper method **`GetHTMLForDisplayText()`** displays the page in Display Text mode and simply shows the text and a change button.

Java:

```
private HTMLElement GetHTMLForDisplayText(){
HTMLElement myResult= new HTMLElementCollection();
     HTMLTable myTable;
     HTMLTableRow myRow;
     HTMLTableCell myCell;
     HTMLB myBold;
     HTMLInput myInput;

     myTable = new HTMLTable();
     myResult.AddInnerHTMLElement(myTable);
     myTable.SetBorder("0");
     myTable.SetCellPadding("5");
     myTable.SetCellSpacing("0");
     myRow = new HTMLTableRow();
     myTable.AddInnerHTMLElement(myRow);

     // text that can be altered
     myCell = new HTMLTableCell();
     myCell.SetStyleClass(PTStyleClass.OBJECT_TEXT);
     myRow.AddInnerHTMLElement(myCell);
     myBold = new HTMLB();
     myCell.AddInnerHTMLElement(myBold);
myBold.AddInnerHTMLString(((SampleActivitySpaceModel)m_
model).GetDisplayText());

     // Change button
          myInput= new HTMLInput(HTMLInputTypes.BUTTON,HTMLBUTTON_CHANGE,"");
     myInput.SetValue("Change Text");
     myInput.SetStyleClass(PTStyleClass.FORM_EDITOR_BTN_TEXT);
     myInput.SetOnClick(
       "postToSelf("
        +SampleActivitySpaceRepostControl.POSTTOSELF_ACTION_CHANGE_TEXT
         +");");

     // add some spacing between input box and button
     myCell.AddInnerHTMLString(CommonHTMLStrings.SPACE);
     myCell.AddInnerHTMLString(CommonHTMLStrings.SPACE);
     myCell.AddInnerHTMLString(CommonHTMLStrings.SPACE);

     // add button
     myCell.AddInnerHTMLElement(myInput);
     return myResult;
}
```

C#:

```
private HTMLElement GetHTMLForDisplayText()
```

```
{
  HTMLElement myResult = new HTMLElementCollection();
  HTMLTable myTable;
  HTMLTableRow myRow;
  HTMLTableCell myCell;
  HTMLB myBold;
  HTMLInput myInput;

  myTable = new HTMLTable();
  myResult.AddInnerHTMLElement(myTable);
  myTable.SetBorder("0");
  myTable.SetCellPadding("5");
  myTable.SetCellSpacing("0");
  myRow = new HTMLTableRow();
  myTable.AddInnerHTMLElement(myRow);

  // text that can be altered
  myCell = new HTMLTableCell();
  myCell.SetStyleClass(PTStyleClass.OBJECT_TEXT);
  myRow.AddInnerHTMLElement(myCell);
  myBold = new HTMLB();
  myCell.AddInnerHTMLElement(myBold);
  myBold.AddInnerHTMLString(((SampleActivitySpaceModel)   m_
model).GetDisplayText());

  // Change button
  myInput = new HTMLInput(HTMLInputTypes.BUTTON,   HTMLBUTTON_CHANGE, "");
  myInput.SetValue("Change Text");
  myInput.SetStyleClass(PTStyleClass.FORM_EDITOR_BTN_TEXT);
  myInput.SetOnClick("postToSelf(" +
SampleActivitySpaceRepostControl.POSTTOSELF_ACTION_CHANGE_TEXT +   ");");

  // add some spacing between input box and button
  myCell.AddInnerHTMLString(CommonHTMLStrings.SPACE);
  myCell.AddInnerHTMLString(CommonHTMLStrings.SPACE);
  myCell.AddInnerHTMLString(CommonHTMLStrings.SPACE);

  // add button
  myCell.AddInnerHTMLElement(myInput);
  return myResult;
}
```

The following portion of the helper method **`GetHTMLForEditText()`** displays the page in Edit Text mode, showing a text box holding the previously entered text and a Save button.

Java:

```
private HTMLElement GetHTMLForEditText(){
     HTMLElement myResult = new HTMLElementCollection();

     HTMLTable myTable;
     HTMLTableRow myRow;
     HTMLTableCell myCell;
     HTMLB myBold;
     HTMLInput myInput;

     myTable = new HTMLTable();
     myResult.AddInnerHTMLElement(myTable);
     myTable.SetBorder("0");
     myTable.SetCellPadding("5");
     myTable.SetCellSpacing("0");
```

```
     myRow = new HTMLTableRow();
     myTable.AddInnerHTMLElement(myRow);

     // text box for inputting new text
     myCell = new HTMLTableCell();
     myRow.AddInnerHTMLElement(myCell);
     myInput = new HTMLInput(HTMLInputTypes.TEXT,EDIT_TEXT_BOX,"");
     myInput.SetStyleClass(PTStyleClass.FORM_EDITOR_BTN_TEXT);
          myInput.SetValue(((SampleActivitySpaceModel)m_
model).GetDisplayText());
     myCell.AddInnerHTMLElement(myInput);

     // Save button
     myInput= new HTMLInput(HTMLInputTypes.BUTTON,HTMLBUTTON_SAVE_TEXT,"");
     myInput.SetValue("Save");
     myInput.SetStyleClass(PTStyleClass.FORM_EDITOR_BTN_TEXT);
     myInput.SetOnClick(
       "postToSelf("
        +SampleActivitySpaceRepostControl.POSTTOSELF_ACTION_DISPLAY_TEXT
         +");");

     // add some spacing
     myCell.AddInnerHTMLString(CommonHTMLStrings.SPACE);
     myCell.AddInnerHTMLString(CommonHTMLStrings.SPACE);
     myCell.AddInnerHTMLString(CommonHTMLStrings.SPACE);

     //add button
     myCell.AddInnerHTMLElement(myInput);
     return myResult;

}
```

C#:

```
private HTMLElement GetHTMLForEditText() {

  HTMLElement myResult = new HTMLElementCollection();
  HTMLTable myTable;
  HTMLTableRow myRow;
  HTMLTableCell myCell;
  HTMLB myBold;
  HTMLInput myInput;

  myTable = new HTMLTable();
  myResult.AddInnerHTMLElement(myTable);
  myTable.SetBorder("0");
  myTable.SetCellPadding("5");
  myTable.SetCellSpacing("0");
  myRow = new HTMLTableRow();
  myTable.AddInnerHTMLElement(myRow);

  // text box for inputting new text
  myCell = new HTMLTableCell();
  myRow.AddInnerHTMLElement(myCell);
  myInput = new HTMLInput(HTMLInputTypes.TEXT, EDIT_TEXT_BOX, "");
  myInput.SetStyleClass(PTStyleClass.FORM_EDITOR_BTN_TEXT);
  myInput.SetValue(((SampleActivitySpaceModel)m_model).GetDisplayText());
  myCell.AddInnerHTMLElement(myInput);

  // Save button
  myInput = new   HTMLInput(HTMLInputTypes.BUTTON,HTMLBUTTON_SAVE_TEXT, "");
  myInput.SetValue("Save");
```

```
    myInput.SetStyleClass(PTStyleClass.FORM_EDITOR_BTN_TEXT);
    myInput.SetOnClick("postToSelf(" +
  SampleActivitySpaceRepostControl.POSTTOSELF_ACTION_DISPLAY_TEXT   + ");");

    // add some spacing
    myCell.AddInnerHTMLString(CommonHTMLStrings.SPACE);
    myCell.AddInnerHTMLString(CommonHTMLStrings.SPACE);
    myCell.AddInnerHTMLString(CommonHTMLStrings.SPACE);

    //add button
    myCell.AddInnerHTMLElement(myInput);

    return myResult;
  }
```

Once you have written the code for your new Activity Space, you must deploy it for use by the portal, described next.

14.3 Step 2: Deploying a Custom Project

After you create a custom project as described in the previous section, you must deploy it to the portal using **Dynamic Discovery**. For detailed information and instructions, see Chapter 18, "Deploying Custom Code Using Dynamic Discovery". (To deploy a custom Activity Space, use jar or dll-based Dynamic Discovery.)

The example below deploys the sample Activity Space code detailed in the previous section. Once you have deployed your code, confirm that it was deployed correctly as explained in Section 14.3.2, "Viewing Your Customization in the Portal".

14.3.1 Example: Sample Activity Space

These instructions use Visual Studio in .NET and Ant scripts in Java to deploy code.

Java

1. Open a command prompt and change the directory to SOURCE_HOME\ptshared\5.1.x (the location of the buildui.xml script).
2. Run a clean build using the following Ant calls: `ant build ant install`

The build target compiles the project, and the install target deploys the .war file.

C#

1. Build the project in Visual Studio.
2. Visual Studio should copy the sampleactivityspace.dll file from SOURCE_HOME\sampleactivityspace\dotnet\prod\bin to PORTAL_HOME\webapp\portal\bin for you. If there are problems with Dynamic Discovery on startup, you might need to do this step manually. This is necessary to allow Dynamic Discovery to find the new library.

14.3.2 Viewing Your Customization in the Portal

Once you have deployed your code, view the changes in the portal to confirm that they were loaded correctly. Use Logging Spy to catch any obvious errors.

1. Open Logging Spy. For details, see *Administrator Guide for Oracle WebCenter Interaction*.
2. Start the portal.

3. Open a new browser window and navigate to the portal. Try to access the sample Activity Space by appending **?space=SampleActivitySpace** to the current URL (i.for example, http://localhost:8080/portal/server.pt?space=SampleActivitySpace). It should fail because the the guest user should not be able to access the page.
4. Login as administrator and try to access the Activity Space again. You should be directed to the page in Display Text mode as shown in the first image in Step 1: Creating an Activity Space.
5. Click the **Change Text** button to see the page in Edit Text mode.
6. Type "Hello World" in the text box and click **Save**. The page should reload and the new message should appear in Display Text mode.
7. Change the text several times to make sure it works.

The next step is to debug your code, covered next.

14.4 Step 3: Debugging and Troubleshooting

This section lists technical tips for common problems and provides instructions on how to debug your new classes.

14.4.1 Technical Tips

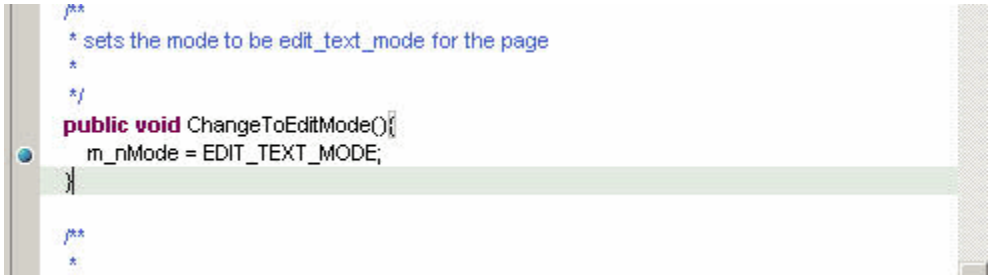
If the page is not displayed in Edit Text mode when the Change Text button is clicked, make sure the **`GetRepostControlName()`** method in the SampleActivitySpaceAS class returns the name for the Control that you want to use. If this doesn't solve the problem, debug your code using the instructions below.

14.4.2 Debugging

These instructions use the sample Activity Space classes created in the previous sections as an example.

Java

1. In Eclipse, stop the Tomcat debugging session and open **SampleActivitySpaceModel.java**.
2. Add a breakpoint as shown below:

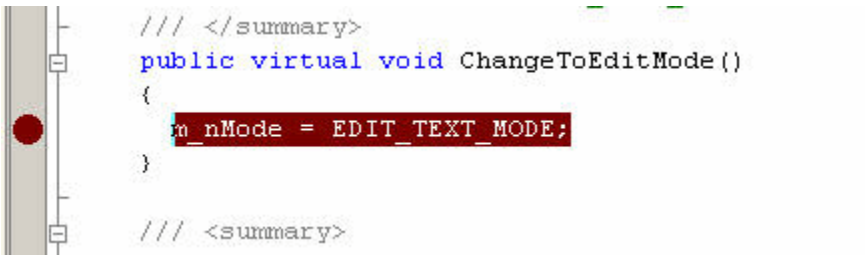


3. In the Eclipse menu, click **Run | Debug...** and select the Tomcat application.
4. Choose the **Classpath** tab, select **Add Projects**, and add the sampleactivityspace project.
5. Hit **Debug** (and **Save** to retain your changes).

6. Open a browser and navigate to your Java portal. You should hit the breakpoint, since you are debugging the login page.

C#

1. Stop the Visual Studio debugger (and close your browser if it is still open) and open **SampleActivitySpaceModel.cs** in Visual Studio.
2. Add a breakpoint as shown below:



3. Start the Visual Studio debugger (F5 or Debug | Start).
4. Navigate to your portal and log in again. You should hit this breakpoint, since you are debugging the login page.

15

Accessing Portal Objects

Many web services, applications and UI customizations require access to portal objects and pages. The portal API provides several options:

- **Adaptive tags** allow you to reference portal objects in portlets and UI components. The **`openerlink`** tag allows you to open any portal object from any gatewayed HTML, such as a portlet. For details on adaptive tags, see *Oracle WebCenter Interaction Web Service Development Guide*.
- The **Common Object Opener** allows you to open any portal object from anywhere within the portal. The `CommonOpener_OpenObject` function is included in every page generated by the portal application, and can be called from any piece of UI or from within a portlet. For details on using the Common Object Opener in UI code, see Section 15.1, "Using the Common Object Opener". Portlets can also call the function remotely through the Oracle WebCenter Interaction Development Kit (IDK). For details, see the *Oracle WebCenter Interaction Web Service Development Guide*.
- The **ASURL** object allows you to create portal-specific URLs to Activity Spaces as Strings or HTMLAnchor objects. For details on using the ASURL object, see Section 15.2, "Using ASURL and Redirect".
- The **Redirect** object can return any URL, and handles the page change to the new URL. The portal uses the Redirect object to redirect control flow from one Activity Space or Control to another. For details on using the Redirect object, see Section 15.2, "Using ASURL and Redirect".

15.1 Using the Common Object Opener

The **Common Object Opener** allows you to open any portal object from anywhere within the portal UI. (To retrieve an URL as a String or HTMLAnchor object, use ASURL) Portlets can also call functions remotely through the Oracle WebCenter Interaction Development Kit (IDK). For details, see the *Oracle WebCenter Interaction Web Service Development Guide*.

The **`PTCommonOpener`** class is included in every page generated by the portal application, and can be called from any piece of UI or from within a portlet. The most commonly used function in this class is **`getOpenerURLOpenObjID`**.

```
function getOpenerURLOpenObjID (_nClassID, _nObjectID, _
strOptQSArgs, _nOpenerMode)
```

This javascript creates a URL that calls the Opener Activity Space and its corresponding OpenObject Control (the only MVC module in the Opener ActivitySpace). The URL includes all necessary parameters for the Control. As noted

above, the Common Opener function is available from any portal page. The `getOpenerURLOpenObjID` function has four required parameters:

- **`nClassID`** refers to the numeric ID for the type of object being opened (e.g., User = 1, Portlet = 43, Content Source = 35). A full list of object types can be found in the PTCLASSDESCRIPTION table in the com.plumtree.portaluiinfrastructure.classtypedescriptors package.
- **`nObjectID`** is the identifier for the specific object being opened. Every object has a unique identifier (ObjectID) stored within the database (for example, for the Users object type: Administrator = 1, Guest = 2, Default Profile = 3). The ObjectID can be retrieved through the com.plumtree.server package and usually through a QueryResult.
- **`strOptQSArgs`** allows you to add arguments to the query string that are not included by default. This argument may be empty.
- **`nOpenerMode`** defines the mode in which the object will be opened, which controls the actions that can be performed on the object (Create = 0, Edit = 1, View = 2, View Meta Data = 3).

As noted above, the `getOpenerURLOpenObjID` function returns a URL. To open the URL you must pass it to another function. PTCommonOpener includes two handy functions for opening URLs: **`openInSameWindow`** and **`openInNewWindow`**.

Call either of these functions in a standard OnClick event. The example below opens the User Editor ActivitySpace as the Administrator user in View mode.

```
<INPUT value="Opener Click" type="Button" name="btnSubmit"
onClick="PTCommonOpener.openInSameWindow(PTCommonOpener.getOpene
rURLOpenObjID(1,1,'null',2));"/>
```

You can execute custom functionality whenever the Common Opener is used to open an object or direct to an Activity Space through the IOpenerActions PEI. For more details on PTCommonOpener and its functions, see the Common Opener API documentation.

15.1.1 Custom Activity Spaces and Non-Portal Pages

As noted earlier, PTCommonOpener functions can be called from within any portal page. You can also use these functions in a custom editor or browsing page by making a call to the Opener ActivitySpace. The Common Opener functionality can be implemented in a variety of ways; the method explained here is the easiest and most commonly used.

Note: To use the following method, first import the package that contains the Opener ActivitySpace (com.plumtree.portaluiinfrastructure.classtypedescriptors.classframework).

Within the **`DisplayJavaScript`** method in the View class of your custom Activity Space, make a call to **`GetOpenerJavascript`** as shown in the code snippet below. This function requires one parameter: the Activity Space that contains the javascript function (inherited from the top level `AActivitySpace` object (m_asOwner)). The HTMLScript element returned contains Common Opener javascript functions.

```
public HTMLScript DisplayJavascript()
{
HTMLScript myScript;
myScript = new HTMLScript("text/javaScript");
myScript = PTOpenerLinks.GetOpenerJavascript(m_asOwner);
return myScript;
```

```
}
```

This code creates a new HTMLScript element, initializes it, and makes a call to `GetOpenerJavascript`.

Other methods for generating javascript code on the server are described in the API documentation for `PTOpenerLinks` in the portaluiinfrastructure package.

15.2 Using ASURL and Redirect

The **ASURL** and **Redirect** objects are used throughout the portal application to access different Activity Spaces and their Controls. Although they provide similar functionality, each serves a specific purpose.

The ASURL class supports automated creation of portal-specific URLs as Strings or HTMLAnchor objects. The Redirect object can return any URL, and handles the page change to the new URL.

15.2.1 ASURL

The URLs created using the ASURL class are always based on the BaseURL obtained from the config.xml file. ASURL methods allow you to append the necessary arguments to the base URL and create a complete ASURL object.

Once the ASURL object has been created, it can return the URL in a variety of ways. You can generate an HTMLAnchor element that contains the full URL, which can be used within an Activity Space. You can also provide the URL as a String to be used in an OnClick event, or as a "document.location" URL for use in client-side javascript.

For example, within the LoginView class that makes up the Login screen of the portal, the CreateAccount button automatically redirects to the CreateAccount Activity Space. The code that creates the ASURL is shown here:

```
myHRef = new ASURL();
myHRef.SetLinkGetSpaceIfCached(CreateAccountAS.STR_MVC_CLASS_NAME,
this.m_asOwner);
myHRef.SetControl(CreateAccountControl.STR_MVC_CLASS_NAME);
myHRef.AddInnerHTMLString(m_asOwner.GetString(637, "ptmsgs_portalbrowsingmsgs"));
myCell.addInnerHTMLElement(myHRef.GetURLAsHTMLElement());
```

The code above begins by creating a new instance of the ASURL object. The object initializes itself by grabbing the BaseURL and using it to start the URL. Method calls on the ASURL object append the query string parameters required by the application to determine the target Activity Space.

15.2.1.1 SetLinkGetSpaceIfCached

The most important parameters that must be added to the URL are the Activity Space name and cache ID. Activity Spaces are cached on the HTTP Session and retrieved for re-use through a cache ID. The name and ID parameters in the ASURL can be populated by passing the Activity Space into the `SetLinkGetSpaceifCached` method. This creates a URL to that particular Activity Space.

If you do not have access to the Activity Space, pass in the Activity Space name. This creates a URL that will search the cache for the named Activity Space, or create a new Activity Space if the referenced one cannot be found. This is very useful for Activity Spaces such as the MyPage Activity Space, which is usually cached on the HTTP Session; you can retrieve a cache ID even if you do not have access to the Activity Space itself.

15.2.1.2 SetLinkCreateNewSpace

If the Activity Space is not cached, you can add it using the `SetLinkCreateNewSpace` method.

15.2.1.3 SetControl

Set the Control name for the URL using the `SetControl` method call.

15.2.1.4 AddInnerHTMLString

Once the two primary parameters are set, the URL can be generated, and you can add HTMLElements or Strings to the anchor as needed. In this example, the message corresponding to ID 637 in the portalbrowsingmsgs language file ("Create an account") is added using the `AddInnerHTMLString` method. For details on language files, see Chapter 6, "Using String Replacement".

15.2.1.5 AddInnerHTMLElement

Once the entire URL is complete, place it into an HTMLAnchor element using the `AddInnerHTMLElement` method, as shown in the code snippet. In this example, the returned HTMLAnchor is used in the View class of the Create Account Editor .

15.2.1.6 GetURLAsString

You can also choose to return the URL as a String using the `GetURLAsString` method.

For a full list of ASURL methods, see the Common Opener API documentation.

15.2.2 Redirect

The **Redirect** object is very similar to ASURL, but can return any URL, and handles the page change to the new URL. (ASURL can only be used for portal URLs and only returns the URL as a String or HTMLAnchor.)

The portal uses the Redirect object to redirect control flow from one Activity Space or Control to another. In many cases, the redirect happens internally, and no HTTP redirect (status code 302) is sent to the browser. One redirect can chain to another. In conditions when you need the browser to change pages, you can force an HTTP redirect using the `SetIsHTTPRedirect` method (covered below).

As with ASURL, a variety of different methods allow you to set query string parameters for portal-specific URLs, as shown in the sample code below:

```
Redirect repostURL = new Redirect();
repostURL.SetLinkCreateNewSpace(FolderEditorAS.STR_MVC_CLASS_NAME, m_asOwner);
repostURL.SetControl(EditorStartControl.STR_MVC_CLASS_NAME);
repostURL.AddControlArgument(EditorStartControl.QS_EDITOR_TYPE,
String.valueOf(EditorStartControl.EDITOR_START_FLAG_EDIT));
repostURL.AddControlArgument(ObjEditorModel.EDITOR_QS_INT_CLASS_ID,
String.valueOf(PT_CLASSIDS.PT_CATALOGFOLDER_ID));
repostURL.AddControlArgument(ObjEditorModel.EDITOR_QS_INT_QS_OBJECT_ID,
String.valueOf(((IDirModelRO) m_dirModel).GetCurrentFolderID()));
SetRedirect(repostURL);
```

The code above begins by creating a new instance of the Redirect object. The method calls made to the object append the query string parameters required by the application to determine the target.

15.2.2.1 SetLinkCreateNewSpace

Set the ActivitySpace name through the `SetLinkCreateNewSpace` method.

15.2.2.2 SetControl

Set the Control name for the URL through the `SetControl` method.

15.2.2.3 AddControlArgument

If the control requires query string parameters, add them through calls to the `AddControlArgument` method.

15.2.2.4 SetRedirect

Once the entire URL has been built, the redirect call is returned through the `SetRedirect` method, inherited from the RepostControl class.

If you need to show an external page in the browser or set a cookie (which requires a full HTTP 302 redirect), you can still use the Redirect object by using `SetIsHTTPRedirect` and `SetLinkToExternalURL`.

15.2.2.5 SetIsHTTPRedirect

Call `SetIsHTTPRedirect` to set the redirect type:

- **True** will force a full HTTP 302 redirect.
- **False** will cause a server-side redirect.

After forcing a 302 redirect, you must handle the redirect back to the portal.

15.2.2.6 SetLinkToExternalURL

To redirect to a page outside the portal, call the `SetLinkToExternalURL` method and pass in the target URL as a String.

Note: To redirect outside the portal, `SetIsHTTPRedirect` must be set to True and the URL must be encoded (you can use one of the standard encoding methods in the HTMLElements package).

16

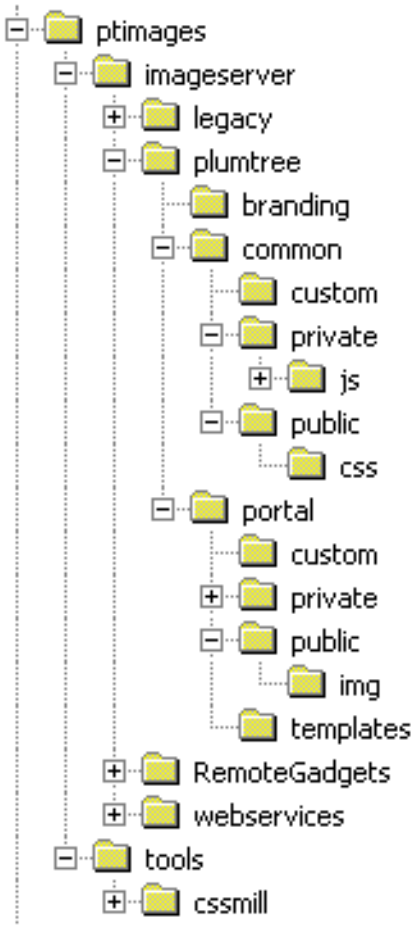
Adding Custom Images

The Oracle WebCenter Interaction Image Service hosts all static web-based components, including style sheets, images, and JavaScript files. Removing these components from the main portal server reduces the amount of processing required to make pages available to the user.

To add custom images to the portal UI, always copy the image file to the specially designated location in the portal Image Service. This ensures that your custom images do not conflict with the many portal images.

16.1 Image Service Structure

The image below shows the standard directory structure of the Image Service.



More directories appear as products and services are added to the portal. Some areas can be customized and might be slightly different in an established implementation. There are two directories directly below the main **\ptimages** directory:

- The **\tools** directory contains tools that help make the Image Service and its components more dynamic.
- The **\imageserver** directory contains all the web-enabled components of the Image Service used by the portal, including images, style sheets, and JavaScript files. (The \legacy, \RemoteGadgets and \webservices folders are included for backward compatibility.)

All Oracle WebCenter Interaction components are found under the **\imageserver\plumtree** folder; this configuration defines the namespace controlled by Oracle WebCenter Interaction. If you make any additions to the \imageserver directory, first create a subfolder with the associated company or product name.

The two main folders within the \plumtree directory are \common and \portal.

- The **\common** directory contains all objects that are used in the portal application and also by associated server applications.
- The **\portal** directory contains any objects that are part of the Image Service that are shipped with the portal. This includes images, style sheets, JavaScript objects, and help files.

Within each of the \common and \portal directories, there is a \public and \private folder. This distinction allows developers to determine whether or not a component is static and backward-compatible.

- Anything within a **\public** directory can be considered available for use. A public component may change slightly throughout the release cycle of the related product, but it will always be there.
- A **\private** directory contains implementation-specific components that should not be accessed by developers; the content within a \private directory can be changed or removed in a future release.
- Any new content added for UI customization should be stored in a **\custom** folder at the same level as the \public and \private directories in the appropriate area to ensure that the files will not be modified by the installer in a future release. Adding custom images is explained next.

16.2 Adding a Custom Image

As explained above, custom images should be stored in the following location in a standard portal installation: \ptimages\imageserver\plumtree\portal\custom\img.

If the image appears in a portlet or other gatewayed page, you can use the //images URI constant in the transformer Adaptive Tag library to access the URL to the Image Service. This is the recommended method. For details on using adaptive tags, see the *Oracle WebCenter Interaction Web Service Development Guide*.

If the image appears in a section of custom UI, you can also access the Image Service custom image folder using the code below.

```
// Get the base Image Service URL for a custom static content item
String strMyImageServerBaseURL = ImgSvrURLHelper.GetInstance().GetCustomURL(m_
asOwner.GetIsSecuredSpace(), "portal");

// Add "img/" to that URL
strMyImageServerBaseURL = XPStringUtility.ForceEndsWith(strMyImageServerBaseURL,
"/") + ImgSvrURLHelper.STR_L4_FILE_TYPE_IMG + "/";

// Create the completed URL to the image
String strImageURL = strMyImageServerBaseURL + "myImageName.gif";

// Assign the properties needed by this image to variables
String strImageAltTag = "Click here to see XYZ!";
String strImageHeight = "25";
String strImageWidth = "25";

// Create a new object of type HTMLImg
HTMLImg myImage = new HTMLImg(strImageURL, strImageAltTag, strImageHeight,
strImageWidth);
```

To make sure that you have generated a good image, you can print the HTML to be generated to Logging Spy using the following code:

```
// Print to PTSpy
String strTest = myImage.GetDisplayString();
PTDebug.Trace(Component.Portal_UI_Infrastructure, TraceType.Error, "***" + strTest
+ "***");
```

After redeploying your code and looking at Logging Spy, you should be able to confirm that the link is valid. It should look something like the following:

```
<img src="http://localhost/imageserver/plumtree/portal/custom/img/myImageName.gif"
alt="Click here to see XYZ!" height="25" width="25"></img>
```

17

Using VarPacks (Variable Packages)

A **Variable Package (VarPack)** is essentially a set of name-value pairs stored on the application. VarPacks are mainly used to store the values from an xml file (e.g., PT_HOME\settings\portal\j_config.xml, n_config.xml, or device.xml) in memory so the data can be used easily without directly accessing the xml file.

Oracle WebCenter Interaction ships with a variety of VarPacks that are used throughout the portal. These are loaded into the portal through the PT_HOME\settings\portal\VarPacks.xml file. For an example, see PTConfigVarPack (com.plumtree.portaluiinfrastructure.application.varpacks).

As with Models, Views, and Controls, VarPacks can be deployed using Dynamic Discovery, which allows the framework to be completely extensible. All you need to do is extend the **XMLBaseVarPack** or **BaseVarPack** class, as shown in this chapter. Custom VarPacks can be used in PEIs or custom Views, among other things, to simplify the details of accessing an xml data file.

This chapter provides examples and instructions on how to use a custom VarPack, and include step-by-step instructions and sample code for creating and implementing a new Hello World VarPack.

17.1 Example VarPack Uses

Two common places to use the VarPack would be a custom View (or Control) or a PEI. For example, a custom VarPack could provide table cell width sizes. The code snippet below shows how to retrieve a value from the VarPack.

Java:

```
HelloWorldVarPack varPack;
varPack = (HelloWorldVarPack)
  m_asOwner.GetVarPack(HelloWorldVarPack.VARPACK_ID);
String strWidth=varPack.GetValueAsString("HelloWorldData","HelloWorldCellWidth");
myCell.SetWidth(strWidth);
```

C#:

```
HelloWorldVarPack varPack;
varPack = (HelloWorldVarPack)
   m_asOwner.GetVarPack(HelloWorldVarPack.VARPACK_ID);
StringstrWidth = varPack.GetValueAsString("HelloWorldData","HelloWorldCellWidth");
myCell.SetWidth(strWidth);
```

A custom VarPack could also be used in an ILoginActions PEI to do custom processing based on the current administrative folder. For example, you could do something like the code shown below.

Java:

```
IPTSessionptSession = (IPTSession) _oUserSession;
IApplicationapp = appData.GetApplication();
XMLBaseVarPack varPack;
varPack = app.GetVarPackManager().GetVariablePackage(HelloWorldVarPack.VARPACK_
ID);
intnFolderID = varPack.GetValueAsInt("HelloWorldData","HelloWorldFolderID");
if (ptSession.GetSessionInfo().GetCurrentUserAdminFolderID() == nFolderID)
{
   // Custom login code.
}
```

C#:

```
IPTSession ptSession = (IPTSession) _oUserSession;
IApplication app = appData.GetApplication();
XMLBaseVarPack varPack;
varPack = app.GetVarPackManager().GetVariablePackage(HelloWorldVarPack.VARPACK_
ID);
int nFolderID = varPack.GetValueAsInt("HelloWorldData", "HelloWorldFolderID");
if (ptSession.GetSessionInfo().GetCurrentUserAdminFolderID() == nFolderID)
{
   // Custom login code.
}
```

17.2 Implementing a VarPack

To create a custom VarPack, create a class that extends `XMLBaseVarPack` or `BaseVarPack`. `XMLBaseVarPack` handles the reading of xml files for you. For more detailed information, see the portal API Documentation.

1. Create a custom project and custom VarPack class (for example, a CustomVarPack project and a CustomVarPack class in com.yourcompany.application.varpacks that extends `XMLBaseVarPack`).
2. Edit the new class in your custom project as needed.
3. Compile the new class into a new JAR/DLL file with an intuitive name. All VarPack file names should follow the standard naming convention and end with "VarPack" (e.g., CustomLoginVarPack).

Note: `XMLBaseVarPack`only reads in xml files that are two levels deep with the data in the value attribute of the lowest tag (`<root><section><sub-section value="data">`). It can be customized to read other formats, as explained below.

17.2.1 Example: Hello World VarPack

In this example, the HelloWorldVarPack class extends `XMLBaseVarPack`and reads in the contents of the helloworld.xml file and stores them on the application. As noted above, the name of the file is important; it ends with "VarPack".

This example uses the sample code from the sample UI projects package. For details, see Appendix B, "Installing UI Customization Sample Projects".

1. Install the files in the SampleProjects-50x.zip package and add the samplevarpack project to your IDE.
2. Open the HelloWorldVarPack file (.java or .cs) in the samplevarpack project in the src/com/plumtree/sampleui/application/varpacks directory (which is under the prod directory in Java). This file extends the `XMLBaseVarPack` class (com.plumtree.uiinfrastructure.application.varpacks).

3. The **`GetVarPackID()`** method provides the name that will be used to store this VarPack on the application. The method returns a public static variable containing the name of the VarPack. The VARPACK_ID variable is is used to get the name of the VarPack so it can be retrieved from the application.

   Java:

   ```
   /** The string ID of this variable package. */
   public static final String VARPACK_ID = "HelloWorldVarPack";
   public String GetVarPackID()
   {
      return VARPACK_ID;
   }
   ```

   C#:

   ```
   /// <summary>
   /// The string ID of this variable package.
   /// </summary>
   public const String VARPACK_ID = "HelloWorldVarPack";
   public override String GetVarPackID()
   {
      return VARPACK_ID;
   }
   ```

4. The **`GetVarPackXMLFileName()`** method provides the name of the xml file to be read and stored in this VarPack.

   Java:

   ```
   public String GetVarPackXMLFileName()
   {
      // The name of the file to read into the VarPack.
      return "helloworld.xml";
   }
   ```

   C#:

   ```
   public override String GetVarPackXMLFileName()
   {
      // The name of the file to read into the VarPack.
      return "helloworld.xml";
   }
   ```

5. As noted above, `XMLBaseVarPack` only supports the specific xml format `<root><section><sub-section value="data">`. To use xml files formatted in other ways, override the **`LoadSettingsIntoXPHashtable()`** method.

Once you have written the code for your new VarPack, you must deploy it for use by the portal, described next.

17.3 Deploying a Custom VarPack

After you create a custom project as explained in the previous section, you must deploy it to the portal using Dynamic Discovery. For detailed information and instructions, see Chapter 18, "Deploying Custom Code Using Dynamic Discovery". To deploy a VarPack, use Jar or DLL-Based Dynamic Discovery.

The example below deploys the Hello World VarPack sample code from the previous section. Once you have deployed your code, confirm that it was deployed correctly as explained in Section 17.3.1, "Viewing Your Customization in the Portal".

These instructions use Visual Studio in .NET and Ant scripts in Java to deploy your custom code.

First, add the library containing the new HelloWorld VarPack class to the CustomVarPacks.xml file so it can be deployed by Dynamic Discovery as explained below.

1. Navigate to PT_HOME\settings\portal and open CustomVarPacks.xml in a text editor.
2. Add the name of your new VarPack (e.g., HelloWorldVarPack) to the existing XML as shown below. Make sure that the spelling and capitalization is exactly the same as the full class name.

```
<root>
  <interface name="IVarPack"/>
  <interfaceassembly name="httpmemorymanagement"/>
  <class name="com.plumtree.sampleui.application.varpacks.HelloWorldVarPack"/>
</root>
```

3. Create the new XML file that will be read by HelloWorldVarPack: open a text editor and create a new file named helloworld.xml and save it in PT_HOME\settings\portal.
4. Add the xml describing the data as shown below and save the file.

```
<?xml version="1.0"?>
<HelloWorldConfig>
  <HelloWorldData>
    <Hello value="World"/>
    <HelloWorld value="Hello World!"/>
    <HelloWorldCellWidth value="50"/>
    <HelloWorldFolderID value="1"/>
  </HelloWorldData>
</HelloWorldConfig>
```

Once you have created the required files, you must run a clean build to deploy the custom code, as explained below.

Java:

1. Open a command prompt and change the directory to the \ptwebui directory where you installed the portal source code.
2. Run a clean build using the following Ant script: `ant build`.
3. Generate a new WAR file for the application server using the following Ant script: `ant install`.

   Note: This target deletes and rebuilds all jar files associated with all the UI source projects (as well as the custom projects in the ptwebui folder).

C#:

1. Build the project in Visual Studio.
2. Visual Studio should copy the samplevarpack.dll file from SOURCE_HOME\samplevarpack\dotnet\prod\bin to PORTAL_HOME\webapp\portal\bin for you. If there are problems with Dynamic Discovery on startup, you might need to do this step manually. This is necessary to allow Dynamic Discovery to find the new library.

17.3.1 Viewing Your Customization in the Portal

Once you have deployed your code, view the changes in the portal to confirm that they were loaded correctly. Use Logging Spy to catch any obvious errors.

1. Open Logging Spy. For details , see the *Administrator Guide for Oracle WebCenter Interaction*.
2. Click the **Set Filters** button to open the **Filter Settings** dialog. Make sure the **Debug** checkbox is selected. (You will not be able to see the customization run if this logging level is not enabled.)
3. Start the portal and view Logging Spy. During startup, Logging Spy should display a message regarding loading Custom VarPacks. Earlier in the startup process, standard VarPacks are loaded from VarPacks.xml. Be careful not to get these two confused. If no Custom VarPacks were loaded, check the spelling and capitalization of HelloWorldVarPack in the CustomVarPacks.xml file.
4. Open a new browser window and navigate to the portal. Log in as Administrator.
5. Append the following argument to the end of the portal URL: ?space=MemoryDebug and browse to your portal.
6. Scroll down to the Variable Packages section; you should see the HelloWorldVarPack listed.
7. Click the View button next to HelloWorldVarPack to view the data stored in the VarPack.

If you were unable to view the HelloWorldVarPack, see the next section for debugging instructions.

17.4 Debugging and Troubleshooting

This page provides technical tips for common problems and instructions on how to debug your new VarPack.

17.4.1 Technical Tips

If your custom VarPack is not loaded correctly, first check the following items:

- Be careful to list the full class name of the HelloWorldVarPack class in CustomVarPacks.xml exactly as it is spelled in the code. It will not load if spelled or capitalized incorrectly. This might not produce errors during startup. One way to check that the VarPack is loaded correctly is to make sure that Logging Spy says the correct number was loaded (i.e., 1).
- Make sure that the name of the xml data file (HelloWorld.xml) is the same as the file name returned by the `GetVarPackXMLFileName()` method, and that the xml data file is located in the PT_HOME\settings\portal directory.
- Check the syntax of the xml data in HelloWorld.xml. If there are syntax errors or the file is formatted incorrectly, it will not load. The XMLBaseVarPack class can only read in xml files that have a single root node with multiple section nodes underneath it. The section nodes contain multiple name/value pair nodes where the name of the node is the name and the value attribute is the value (i.e. `<NodeName value="NodeValue"/>`). If you need to use a different format, you must override the `LoadSettingsIntoXPHashtable()` method.

If none of these tips solve the problem, debug your code using the instructions below.

17.4.2 Debugging

These instructions use the HelloWorldVarPack class created on the previous pages as an example.

Java:

1. In Eclipse, stop the Tomcat debugging session and open HelloWorldVarPack.java.
2. Add a breakpoint at the `return "helloworld.xml"` line.
3. In the Eclipse menu, click Run | Debug… and select the Tomcat application.
4. Choose the Classpath tab, select Add Projects, and add the samplevarpack project.
5. Hit Debug (and Save to retain your changes).
6. Start the portal.
7. You should hit this breakpoint once during startup, when the XMLBaseVarPack class asks the HelloWorldVarPack which xml file to load.

C#:

1. Stop the Visual Studio debugger (and close your browser if it is still open) and open HelloWorldVarPack.cs in Visual Studio.
2. Add a breakpoint as shown below.
3. Start the Visual Studio debugger (F5 or Start | Debug) and wait for the portal to start up.
4. You should hit this breakpoint once during startup, when the XMLBaseVarPack class asks the HelloWorldVarPack which xml file to load.

18

Deploying Custom Code Using Dynamic Discovery

Dynamic Discovery allows you to plug your changes into the UI architecture without modifying any existing UI code. Dynamic Discovery is a framework for detecting extensible UI features. When the portal is loaded on startup, this tool automatically creates instances of objects according to an XML file. Dynamic Discovery can be configured in a few different ways. For example, it can load all instances of one specific interface from one or more libraries, or load instances of specific classes.

The main benefit of Dynamic Discovery is the ability to customize the portal without changing existing code. Using Dynamic Discovery, you can overwrite core portal classes, change the behavior of some features, customize pages, and add features to the portal simply by editing XML files and adding new libraries. Dynamic Discovery was created to make the UI as extensible as possible and to facilitate upgrading the UI after customizations have been made. If you allow your modifications to be dynamically discovered, your code will remain separate from the portal code, making upgrades easier.

18.1 Dynamic Discovery Configuration Files

Dynamic Discovery relies on three configuration files in the <PT_HOME>\settings\portal directory:

- The **ActivitySpaces.xml** file enumerates all the JAR (or DLL) files that Oracle WebCenter Interaction uses to discover classes that implement IModel, IView, and IControl. Additionally, the portal searches these lib files for navigation schemes (INavTypes). Do not modify this file.
- The **CustomActivitySpaces.xml** file is empty at the time a new portal ships. Functionally, this file is identical to ActivitySpaces.xml. It is a mechanism to help you keep custom code separate from platform code. (The distinction is not enforced. It is intended to help you keep custom code separate and easily identifiable.)
- The *.xml files in the **\dynamicloads** subdirectory are also used by Dynamic Discovery. For example, all PEIs are dynamically discovered using a separate *Actions.xml file in the \PEIs subfolder. At startup, the portal loops through all of these files and calls GetInstancesFromXML on each. The returned array of objects is cached on the portal application with the key being the name or the XML file. When an event occurs, all the classes that have implemented the corresponding interface are loaded into the portal application, and each implemented function is processed in the order shown within the XML file.

18.2 Using Dynamic Discovery

The portal application uses an XML file to identify customized UI components during startup and place them in the proper location. There are two mechanisms available:

- **Interface-Based Dynamic Discovery** supports the most commonly customized events, represented by Portal Event Interfaces (PEIs). For detailed information on PEIs, see Chapter 12, "Using PEIs".
- **JAR- or DLL-Based Dynamic Discovery** allows you to make other extensions or changes to the UI (for example, View replacement or a new Navigation Scheme) without changing any existing source code.

A summary of each mechanism is provided below.

18.2.1 Interface-Based Dynamic Discovery

All PEIs are dynamically recognized through interface-based Dynamic Discovery. An XML file is provided for each PEI that contains a listing of each implementation of the specific PEI. Multiple implementations can be placed within the XML file; each implementation is processed in the order that it appears.

1. Find the XML file for the PEI (files for all PEIs can be found in the PT_HOME\settings\portal\dynamicloads\PEIs directory).
2. Open the file in a text editor and add your class name to the list. (For .NET, you must first stop the WWW Service on the portal server.) The example below is from the LoginActions.xml file.

   ```
   <root>
   <interface name="com.plumtree.uiinfrastructure.pei.ILoginActions"/>
   <interfaceassembly name="uiinfrastructure"/>
   <class name="com.plumtree.portalpages.pei.PTLoginActions"/>
   <class name="com.plumtree.sampleui.pei.HelloWorldLoginActions"/>
   </root>
   ```

   Note: The package referenced in the XML file must include the class that implements the corresponding PEI (as described in Using PEIs). The Dynamic Discovery process looks for a class file that matches the name listed and the PEI defined within the XML file. The name is case sensitive.
3. Restart your application server (Java) / the WWW Service on the portal server (.NET).
4. To verify that your UI modifications were loaded correctly, turn on Logging Spy and watch for Dynamic Discovery lines. There will be a separate line corresponding to each file loaded from each XML file.

For detailed instructions on PEI deployment, see Chapter 12, "Using PEIs", Section 12.3, "Step 3: Deploying a Custom PEI".

18.2.2 Jar or DLL-Based Dynamic Discovery

JAR- or DLL-based Dynamic Discovery allows you to make extensions or changes to the UI without changing any existing source code. By using new code instead of modifying existing code, no customizations are overwritten when the portal application is upgraded to a new release. Your code can take precedence over existing UI code, or you can create a completely new section for the UI. These UI modifications are picked up by the Dynamic Discovery package when the portal application is started.

To make an addition to the UI, create a new section using the ActivitySpace/MVC paradigm. Navigation schemes are a simplified example; for detailed instructions on deploying a new navigation scheme, see Chapter 9, "Customizing Portal Navigation", Section 9.3, "Deploying a Custom Navigation Scheme".

To modify a piece of the UI, write a new class that corresponds to the component of the UI that you want to modify. Dynamic Discovery will use the version loaded from CustomActivitySpaces.xml in place of the original version loaded from ActivitySpaces.xml. If there are multiple versions of the same component loaded in CustomActivitySpaces.xml, Dynamic Discovery gives precedence to the last version loaded. For detailed instructions on deploying a View replacement, see Chapter 13, "Using View Replacement", Section 13.3, "Deploying a Custom View".

Once the entire Activity Space has been created and compiled into a JAR/DLL file, follow the steps below to deploy the file using Dynamic Discovery.

1. (.NET only) Stop the WWW Service on the portal server.
2. Deploy the customized JAR/DLL file to the portal.
   - Java: Rebuild the WAR file with the customized JAR in it, and copy the customized JAR file to PORTAL_HOME\ptportal\6.0\lib\java.
   - NET: Copy the customized DLL to PORTAL_HOME\ptportal\6.0\webapp\portal\web\bin (this is in the virtual directory for the portal application).
3. Open the PT_HOME\settings\portal\CustomActivitySpaces.xml file in a text editor and add your customized JAR/DLL file to the list. Dynamic Discovery will use the version loaded from CustomActivitySpaces.xml in place of the original version loaded from ActivitySpaces.xml. If there are multiple versions of the same component loaded in CustomActivitySpaces.xml, Dynamic Discovery gives precedence to the last version loaded. The example below adds a View class called "sampleview".

   ```
   <AppLibFiles>
   <libfile name="sampleview"/>
   </AppLibFiles>
   ```
4. Restart your application server (Java) / the WWW Service on the portal server (.NET).
5. When the portal application starts up, each line is processed and each JAR/DLL file that corresponds to a `<libfile name=""/>` attribute is loaded.

Part IV

Appendices and Additional References

The following appendices and additional references provide useful information related to UI customization.

- Appendix A, "Portal Configuration Files"
- Appendix B, "Installing UI Customization Sample Projects"
- Appendix C, "Portal API Documentation"

A

Portal Configuration Files

The portal configuration files are used to manage settings for the portal and its components. In addition, configuration files are used by the portal to discover and load custom code. Some configuration files are used in customization, but many should not be modified.

Configuration files are installed in <PT_HOME>\settings\ (i.e., C:\Program Files\plumtree\settings in Windows and /opt/plumtree/settings in Linux).

A.1 Common Settings

The serverconfig.xml configuration file in the **<PT_HOME>\settings\common** folder governs the settings that are common to all portal components.

Configuration File	Description
serverconfig.xml	Sets the connection and path information for portal components. ■ Portal host computer from which logs are collected ■ Connection information for portal databases ■ HTTP proxy settings and caching ■ Crawler and gateway transactions ■ Automation service For details about specific settings, see the*Administrator Guide for Oracle WebCenter Interaction*.

A.2 Plug-Ins

The configuration files in **<PT_HOME>\settings\portal\dynamicloads\Plugins** load custom classes used to implement portal customization.

All configuration files under **\dynamicloads** are dynamically discovered. The Dynamic Discovery framework automatically detects extensible UI features based on the objects referenced in XML configuration files. For details on using Dynamic Discovery, see Chapter 18, "Deploying Custom Code Using Dynamic Discovery".

Configuration File	Description
ConditionTypes.xml	Loads all conditions types, including custom condition types. The condition types appear in the Experience Rules Manager. For more information on condition types, see Chapter 7, "Customizing Experience Definitions".

Configuration File	Description
InterpreterFilters.xml	Loads filters. Filters intercept or modify an incoming requests before and after the incoming request is processed by the interpreter.
OpenerPlugins.xml	Loads opener plug-ins. Openers are modules that perform certain actions based on your URL criteria. OpenerPlugins.xml loads URL mapping to an activity space. For more information on the common opener, see Chapter 15, "Accessing Portal Objects". (Deploying opener plug-ins is similar to deploying PEIs; for details, see Chapter 18, "Deploying Custom Code Using Dynamic Discovery".)

A.3 Programmable Event Interfaces

The configuration files in the **<PT_HOME>\settings\portal\dynamicloads\PEI** folder load custom classes related to the Programmable Event Interface (PEI) framework. PEIs define a set of actions that can be used to execute custom code without editing the portal source code. Each PEI has an associated XML file that lists all implementations of the PEI, used by Dynamic Discovery to add the code to the portal at startup. For details on using PEIs, see Chapter 12, "Using PEIs".

Configuration File	Description
AdvancedSearchActions.xml	Loads custom classes that make modifications to advanced search queries.
BannerSearchActions.xml	Loads custom classes that make modifications to banner search queries.
CommunityActions.xml	Loads custom classes that add functionality when a user joins or unsubscribes from a community.
CreateAccountActions.xml	Loads custom classes that execute functionality when a new user attempts to create an account, either through the Create Account button on the login page or in response to an invitation.
DirectoryActions.xml	Loads custom classes that implement functionality in response to Directory actions.
DisplayJavascriptActions.xml	Loads custom classes that add JavaScript to every banner and editor page.
LoginActions.xml	Loads custom classes that create functionality within the scope of the login process.
MyPortalPageActions.xml	Loads custom classes that perform validation before allowing users to add or remove portal pages.
NetworkSearchActions.xml	Loads custom classes that make changes to network searches after they have been submitted by the user.
NewEditObjectActions.xml	Loads custom classes that implement custom functionality to be processed when a user creates or edits a portal object.
ObjectActions.xml	Loads custom classes that add functionality to most administrative tasks, including delete, move, copy, and object migration.
OpenerActions.xml (new 6.0)	Loads custom classes that perform functionality before the Common Opener opens an object or directs the user to an activity space.

Configuration File	Description
PageActions.xml	Loads custom classes that add code to every page processed by the portal.
PasswordActions.xml	Loads custom classes that enforce restrictions on the password or verify the text entered by the user.
SearchSettingActions.xml	Loads custom classes that track creation and deletion of saved searches and control naming and encoding for new saved searches.
UserProfileActions.xml	Loads custom classes that execute functionality when a user attempts to modify User Profile information.

A.4 Utilities

The configuration files in **<PT_HOME>\settings\portal\dynamicloads\Utilities** are related to administrative utilities.

For detailed information on portal utilities, see the portal online help.

Configuration File	Description
DisplayDiagnosticPages.xml	Loads all diagnostic monitoring tools in the System Health Monitor utility of the portal administration. Do not edit this file unless you created your own custom diagnostic monitor.
DisplayPlumtreeUtilities.xml	Along with DisplayServerSettings.xml (described below), this file loads the utilities in the Select Utility drop-down list in portal administration.
DisplayPortalSettings.xml	Loads the settings in My Account.
DisplayServerSettings.xml	Along with DisplayPlumtreeUtilities.xml (described above), this file loads the utilities in the Select Utility drop-down list in portal administration.

A.5 Object Descriptions

The read-only configuration files in **<PT_HOME>\settings\portal\dynamicloads\ObjectDescriptions** describe portal objects and external providers.

Configuration File	Description
ClassTypeDesc.xml	**Do not edit.** Describes Oracle WebCenter Interaction objects. Loads classes that contain information about objects, such as associated icons, related class IDs, localized names, and Activity Space redirection.
ProvInfo.xml	**Do not edit..** Describes processes that Oracle WebCenter Interaction implements for web service providers. Loads classes that contain information about web service providers.

A.6 Miscellaneous

The configuration files in **<PT_HOME>\settings\portal** load custom classes and settings related to the portal. Unlike the configuration files in the **\dynamicloads** folder, which lists fully-qualified class names, the configuration files directly under the

portal folder list JAR or DLL files. The file names of these XML files are hard-coded and are loaded by the portal upon startup.

Configuration File	Description
ActivitySpaces.xml	**Do not edit.** Any custom Activity Spaces should be added to CustomActivitySpaces.xml (described below). The **ActivitySpaces.xml** file enumerates the JAR or DLL files that Oracle WebCenter Interaction uses to discover base classes for the portal UI.
CustomActivitySpaces.xml	This file is empty when the portal ships. Functionally, the file is identical to ActivitySpaces.xml, but includes only custom Activity Spaces. This approach helps keep custom code separate from platform code. For details on creating custom Activity Spaces, see Chapter 14, "Creating Custom Activity Spaces".
Tags.xml (6.0)	***Do not edit***. Any custom tags should be added to CustomTags.xml (described below). The Tags.xml file loads the standard adaptive tags referenced by the included JAR or DLL files. For details on adaptive tags, see *Oracle WebCenter Interaction Web Service Development Guide*.
CustomTags.xml (6.0)	This file is empty when the portal ships. Functionally, this file is identical to Tags.xml, but includes only custom Adaptive Tags as referenced by the included JAR or DLL files. For details on creating custom Adaptive Tags, see *Oracle WebCenter Interaction Web Service Development Guide*.
VarPacks.xml	**Do not edit.** Any custom VarPacks should be added to CustomVarPacks.xml. The VarPacks.xml file loads the standard variable packages included with Oracle WebCenter Interaction.
CustomVarPacks.xml	This file is empty when a new portal ships. Functionally, this file is identical to VarPacks.xml, but includes only custom VarPacks. Variable packages store values from an XML file (such as portalconfig.xml) in memory so the data can be used easily without directly accessing the XML file. For details on custom VarPacks, see Chapter 17, "Using VarPacks (Variable Packages)".

Configuration File	Description
portalconfig.xml	Sets various settings for the portal. ■ Home directory for the portal JAR or DLL files, as well as the base URL of the Image Service and administrative portal ■ Proxy server for the portal, the HTML document type specification, and virtual directory path for the portal and SSO ■ Portal base URL ■ Accessibility and bandwidth settings ■ Personal settings that should be cached on the HTTP session of each user, such as greeting messages, display options, preferred language, refresh rate, and time zone ■ Security, SSL encryption, authentication, and guest access to the portal ■ Language of objects and portal style sheets ■ Crawler radius (number of page links to crawl) ■ Administration status of a computer (browsing or administrative) For details about the settings in this configuration file, see the *Administrator Guide for Oracle WebCenter Interaction*.
portal.xml	This file contains sample settings for Apache Tomcat. Copy this file and replace the portal.xml file in your Tomcat directory.
version.xml	**Do not edit.** The version.xml file is created by the installer to make sure that it does not override the most recent version of the common components.

B

Installing UI Customization Sample Projects

Many advanced UI customization development topics use the sample code in the **SampleProjects-1013x.zip** available on the Oracle Technology Network. For details on advanced UI customization, see Part III, "Advanced UI Customization".

The instructions that follow explain how to install the files and add the projects to your IDE. The default file locations are shown below:

- **SOURCE_HOME:** C:\plumtree_ui_source
- **PORTAL_HOME:** C:\Program Files\plumtree\ptportal\10.1.3

B.1 Sample Projects

The sample code package includes the following projects:

- **sampleactivityspace** illustrates how to create a custom Activity Space.
- **sampleagreementlogin** shows how to block access to the portal based on acceptance of terms and conditions. For details, see the README.txt in the project directory.
- **sampleloginpei** illustrates how to implement a login PEI (Programmable Event Interface).
- **sampleplugnav** shows how to create a custom navigation scheme.
- **samplenavmsg** shows how to use a custom navigation scheme to display messages on every page. For details, see the README.txt in the project directory.
- **samplenavmsgnovarpack** extends one of the portal's navigation schemes. For details, see the README.txt in the project directory.
- **sampleproject** shows how to create a custom project.
- **sampleredirectpei** shows how to use a PEI (Programmable Event Interface) to redirect on login.
- **sampleselectstartpage** shows how add a portal page that allows users to select their home community. For details, see the README.txt in the project directory.
- **samplesubportalguests** shows how to implement multiple branded login pages. **Note**: This customization can now be completed without any code using experience definitions.
- **sampletag** shows how to create a vertical navigation bar with adaptive tags.
- **samplevarpack** illustrates how to implement a variable package (varpack).
- **sampleview** shows how to replace the login view with a custom view.

B.2 Installing the Java Sample Projects

The instructions that follow explain how to install the files and add the projects to your IDE.

B.2.1 Preparing the Sample Code

To prepare the samples for running:

1. Extract **SampleProjects-1013x.zip** to a new SAMPLE_HOME location (for example, C:\Sample Code). The directory structure created will include \ptwebui-samples, _buildcommon, and _local_repository. **Warning**: Do not unzip the samples on top of the UI source code directory (SOURCE_HOME); they will conflict with existing build scripts.
2. Confirm that you are using Ant 1.6.2 and Java JDK 1.4.2 or above. You must configure **ANT_HOME** and **JAVA_HOME** to point to the respective home directories (for example, ANT_HOME=c:\jakarta-ant-1.6.2).
3. Make sure your JDK's bin directory is on your PATH. Test it by trying to run the JDK application "jar" from the command line.
4. Build the samples by navigating to **SAMPLE_HOME** and typing the following: `ant build`This will create a new subdirectory in SAMPLE_HOME called \buildoutput\java-samples that contains the sample JAR files.

B.2.2 Importing to Eclipse

To import the sample code into Eclipse:

1. Extract **SampleProjects-1013x.zip** to SOURCE_HOME\portalui\10.1.3 **Warning**: This will overwrite some of the existing build files with new ones that contain targets for the sample projects. If you have previously downloaded the sample code and modified these files to add your own custom projects, your modifications will be overwritten.
2. Open Eclipse, and click **File** | **Import** | **Existing Project into Workspace**.
3. Click **Next**.
4. Browse to the \java subdirectory of the project you want to import in **SOURCE_HOME** (i.e., \plumtree_ui_source\sampleview\java).
5. Click **OK** and **Finish**.

B.2.3 Deploying the Sample Code JARs

Last, deploy the sample code JARs. Since the sample jar files must be accessible to both your web application server and the portal's Dynamic Discovery loader, you must copy the sample JARs to two locations.

1. Copy the sample JARs from SAMPLE_HOME\buildoutput\java-samples to PORTAL_HOME\lib\java so they can be accessed by the portal's Dynamic Discovery mechanism.
2. Add the JARs to your web application server using the instructions below:
   a. Change directories to PORTAL_HOME\webapp.
   b. Move or delete the .ear file.
   c. Unzip the WAR file using Java's JAR command: `jar -xvf portal.war`

d. Copy the sample JAR files from SAMPLE_HOME\buildoutput\java-samples to PORTAL_HOME\webapp\WEB-INF\lib.

e. Re-zip the WAR file using the JAR command: `jar -cvf portal.war *.*`

3. Add the sample projects to the debug configuration:

   a. In Eclipse, go to **Run - Debug** and open the Tomcat debug configuration.

   b. Go to the classpath tab and select **User Entries**. Click **Add Projects** and add any sample projects you want to debug.

B.3 Installing the .NET Sample Projects

The instructions that follow explain how to install the files and add the projects to your IDE. These instructions are for the Visual Studio IDE.

1. Extract **SampleProjects-1013x.zip** to SOURCE_HOME\portalui\10.1.3.

   Warning: This will overwrite some of the existing files in the _buildcommon folder with new ones that contain targets for the sample projects. If you have modified these files to add your own custom projects, your modifications will be overwritten.

2. Open a command prompt and navigate to directory SOURCE_HOME\portalui\10.1.3\ptwebui-samples. Execute the following command to generate the .csproj files for the sample projects: `ant vstudio`**Note**: To perform this step, tthe LOCAL_REPO environment variable must be set to point to your local repository (usually at PT_HOME/_customer_repository).

3. Navigate to SOURCE_HOME\portalui\10.1.3\ptwebui\portal\dotnet\prod and open the **portal.sln** project in Visual Studio.

4. Right-click on theportalsolution, and click **Add | Existing Project**.

5. Navigate to the \dotnet\prod directory of the project you want to import in SOURCE_HOME and select the .project file.

6. Expand the portal project, right-click**References**, and select**Add Reference**.

7. On the**Projects**tab, highlight the project you want to add.

8. Click**Select**and**OK**.

9. Click**OK**to close the References window.

10. Build the portal solution to ensure that all the projects build successfully.

C

Portal API Documentation

The following sections provide links to portal API documentation.

C.1 Adaptive Tags

The tagdocs provide details on tags used for Adaptive Page Layouts. For details, see Chapter 3, "Using Adaptive Page Layouts". The tagdocs can be viewed in the following location: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/devguide/apidocs/tagdocs/index.html (Download a zip of the package here: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/devguide/apidocs/tagdocs/TagDocs.zip.)

You can also create custom adaptive tags. The API documentation for the Adaptive Tag Framework can be viewed in the following location:

- Java: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/java/tags/index.html(Download a zip of the package here: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/java/tags/TagsJavaDocs.zip.)
- .NET: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/dotnet/tags/index.html(Download a zip of the package here: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/java/tags/TagsNDocs.zip.)

C.2 Portal UI Packages

The following packages provide selected libraries from the portal UI API package for customizing specific functionality.

- **PEIs** (portalpages.pei, portaluiinfrastructure.pei, portaluiinfrastructure.tags.pei and uiinfrastructure.pei): These libraries are focused on the portal's Programmable Event Interfaces API. For details on implementing custom code using PEIs, see Chapter 12, "Using PEIs". The PEI API documentation can be found in the following locations:
  - Java: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/java/pei/index.html(Download a zip of the package here: http://download.oracle.com/docs/cd/E13158_

01/alui/wci/docs103/uiguide/apidocs/java/pei/PEIJavaDocs.zip.)

  - .NET: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/dotnet/pei/index.html(Download a zip of the package here: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/java/pei/PEINDocs.zip.)

- **Pluggable Navigation** (nullnavigation.*, portalnavigation.*, portalpages.common.mediator, portalpages.common.plugnav, portalpages.common.uiparts, portaluiinfrastructure.application.varpacks, portaluiinfrastructure.compoundlist, and portaluiinfrastructure.navtype): These libraries are focused on the Pluggable Navigation API. Note: Most navigation customizations can be implemented without writing custom code against the portal UI. For details, see Chapter 9, "Customizing Portal Navigation". The Pluggable Navigation API documentation can be found in the following locations:

  - Java: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/java/plugnav/index.html(Download a zip of the package here: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/java/plugnav/PlugNavJavaDocs.zip.)

  - .NET: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/dotnet/plugnav/index.html(Download a zip of the package here: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/dotnet/plugnav/PlugNavNDocs.zip.)

- **UI Infrastructure**: The uiinfrastructure library represents the layer of the portal UI that defines the core architectural classes for the portal UI. This layer has no dependencies on the plumtree.server layer. It addresses generic problems like how to store state across pages in a consistent, reusable fashion. The UI Infrastructure API documentation can be found in the following location:

  - Java: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/java/portalui/index.html

  - .NET: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/dotnet/uiinfrastructure/index.html

- **Open Foundation**: This section contains the cross-platform (Java) development library underneath all portal UI layers. The Open Foundation API documentation can be found in the following location:

  - Java: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/java/openfoundation/index.html

C.3 Portal Server API

This package contains libraries that can only be invoked on a machine with core portal server libraries. The Portal Server API documentation can be found in the following locations:

- Java: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/java/portalserver/index.html
- .NET: http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/uiguide/apidocs/dotnet/portalserver/plumtreeserver.xml

A subset of features are available on the remote tier using the Programmable Remote Client (PRC). For details on using the PRC, see the *Oracle WebCenter Interaction Web Service Development Guide*.