

# JBoss Portal 2.2

## Reference Guide

---

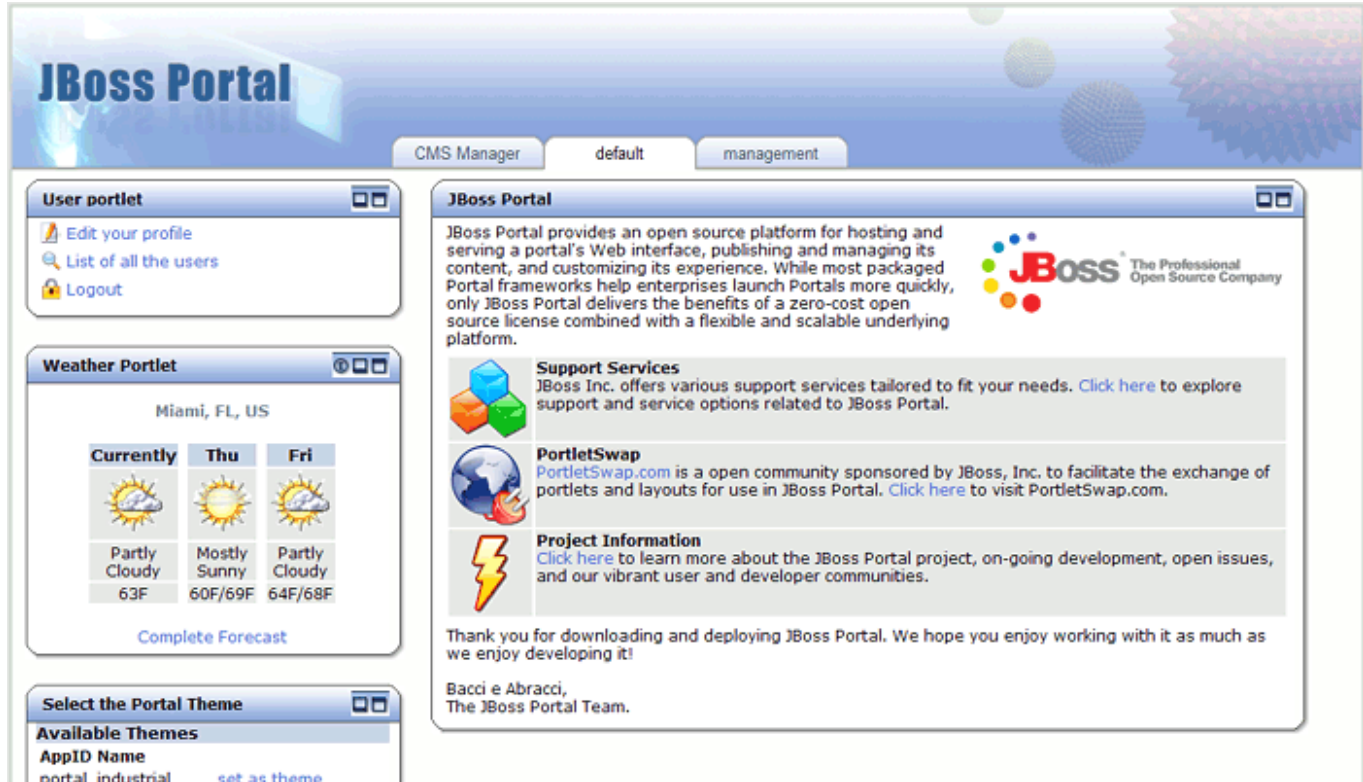
# Table of Contents

JBoss Portal - Overview .....	v
Feature List .....	vii
Target Audience .....	x
Acknowledgements .....	xi
1. Upgrading 2.0 - 2.2 .....	1
1.1. Deployment Descriptors .....	1
1.1.1. Example - Assigning a Portlet on a Page .....	1
1.2. Content Management System .....	2
1.2.1. Migrating Content .....	3
1.3. Forums Migration .....	4
1.3.1. Forums DB schema issues .....	4
1.3.2. Portal 2.0.0 to 2.0.1 Forums migration .....	4
1.3.3. Necessary steps to migrate Forums .....	5
2. JSR168 portlets .....	6
2.1. Introduction .....	6
2.2. The basics .....	6
2.2.1. Portal .....	6
2.2.2. Page composition .....	6
2.2.3. Rendering modes .....	7
3. XML descriptors .....	8
3.1. Introduction .....	8
3.2. Defining a new portlet instance .....	8
3.3. Defining a new portal page. ....	10
3.4. Defining a new portal instance .....	12
3.5. Defining Portlet Instance Preferences .....	13
4. Portal urls .....	15
4.1. Introduction .....	15
4.2. Accessing a portal .....	15
4.3. Accessing a page .....	15
4.4. Accessing CMS Content .....	16
5. Security .....	17
5.1. Portal Security Configuration .....	17
5.1.1. org.jboss.portal.security.config.SecurityConstraint .....	17
5.1.2. org.jboss.portal.security.config.PortalPolicyConfig .....	18
5.1.3. org.jboss.portal.security.config.PortalPolicyConfigService .....	18
5.1.4. org.jboss.portal.security.config.ConfigListener .....	18
5.2. Portal Security Enforcement .....	18
5.2.1. org.jboss.portal.security.PortalPermission .....	18
5.2.2. org.jboss.portal.security.PortalPolicy .....	19
5.2.3. org.jboss.portal.security.PortalSubject .....	19
5.2.4. Portal Policy Enforcement .....	19
5.3. Default Security Implementation .....	20
5.3.1. What is JACC .....	20
5.3.2. JACC Policy and Policy Configuration .....	20

5.3.3. Portal Policy and Policy Service .....	21
5.3.4. Who is asking? .....	22
5.3.4.1. The Jacc Portal Subject .....	22
5.4. Write your own Security Implementation .....	22
5.4.1. Overwriting the JACC policy .....	22
5.4.1.1. Write the Policy class .....	22
5.4.1.2. Configure the Policy to be used by the Portal .....	23
5.4.2. A Security implementation without JACC .....	24
5.4.2.1. The ConfigListener .....	24
5.4.2.2. Policy Enforcement .....	25
6. InterPortlet Communication (IPC) .....	26
6.1. Introduction .....	26
6.2. Configuration .....	26
6.2.1. Service Descriptor .....	26
6.2.2. Portal Descriptor .....	27
6.2.3. The Listener .....	27
6.3. Communicating with another Portlet .....	28
6.4. Communicating with another Page .....	29
7. Layouts and Themes .....	30
7.1. Overview .....	30
7.2. Layouts .....	31
7.2.1. How to define a Layout .....	31
7.2.2. How to use a Layout .....	32
7.2.2.1. Declarative use .....	32
7.2.2.2. Programatic use .....	33
7.2.3. Where to place the Descriptor files .....	33
7.2.4. How to connect a Layout to a Layout Strategy .....	33
7.2.5. Layout JSP-tags .....	33
7.3. Layout Strategy .....	35
7.3.1. What is a Layout Strategy .....	35
7.3.2. How can I use a Layout Strategy .....	35
7.3.2.1. Define a Strategy .....	35
7.3.2.2. Specify the Strategy to use .....	35
7.3.3. Linking the Strategy and the Layout .....	37
7.4. RenderSets .....	37
7.4.1. What is a RenderSet .....	37
7.4.2. How is a RenderSet defined .....	38
7.4.3. How to specify what RenderSet to use .....	38
7.5. Themes .....	40
7.5.1. What is a Theme .....	40
7.5.2. How to define a Theme .....	40
7.5.3. How to use a Theme .....	42
7.5.4. How to write your own Theme .....	43
7.6. Other Theme Functionalities and Features .....	43
7.6.1. Content Rewriting and Header Content Injection .....	43
7.6.2. Declarative CSS Style injection .....	44
7.6.3. Disabling Portlet Decoration .....	44
7.7. Theme Style Guide (based on the Industrial theme) .....	45
7.7.1. Overview .....	45

7.7.2. Main Screen Shot .....	45
7.7.3. List of CSS Selectors .....	46

# JBoss Portal - Overview



Many IT organizations look to achieve a competitive advantage for the enterprise by improving business productivity and reducing costs. Today's top enterprises are realizing this goal by deploying enterprise portals within their IT infrastructure. Enterprise portals simplify access to information by providing a single source of interaction with corporate information. Although today's packaged portal frameworks help enterprises launch portals more quickly, only JBoss Portal can deliver the benefits of a zero-cost open source license combined with a flexible and scalable underlying platform.

JBoss Portal provides an open source and standards-based environment for hosting and serving a portal's Web interface, publishing and managing its content, and customizing its experience. It is entirely standards-based and supports the JSR-168 portlet specification, which allows you to easily plug-in standards-compliant portlets to meet your specific portal needs. JBoss Portal is available through the business-friendly LGPL [<http://www.jboss.com/company/aboutopensource>] open source license and is supported by JBoss Inc. Professional Support and Consulting [<http://www.jboss.com/services/index>]. JBoss support services are available to assist you in designing, developing, deploying, and ultimately managing your portal environment. JBoss Portal is currently developed by JBoss, Inc. developers, Novell developers, and community contributors.

The JBoss Portal framework and architecture includes the portal container and supports a wide range of features including standard portlets, single sign-on, clustering and internationalization. Portal themes and layouts are configurable. Fine-grained security administration down to portlet permissions rounds out the security model. JBoss Portal includes a rich content management system and message board support.

## JBoss Portal Resources:

1. JBoss Portal Home Page [<http://www.jboss.org/products/jbossportal>]

2. Forums:    User    [<http://www.jboss.org/index.html?module=bb&op=viewforum&f=215>]    |    Developer  
[<http://www.jboss.org/index.html?module=bb&op=viewforum&f=205>]
3.    Wiki [<http://www.jboss.com/wiki/Wiki.jsp?page=JBossPortal>]
4.    PortletSwap.com portlet exchange [<http://www.portletswap.com>]
5.    Our    Roadmap  
[<http://jira.jboss.com/jira/browse/JBPORTAL?report=com.atlassian.jira.plugin.system.project:roadmap-panel>]

The JBoss Portal team encourages you to use this guide to install and configure JBoss Portal. If you encounter any configuration issues or simply want to take part in our community, we would love to hear from you in our forums.

---

# Feature List

The following list details features found in this document's related release. For a technical view of our features, view the Project Roadmap and Task List [<http://jira.jboss.com/jira/browse/JPORAL>] .

## Technology and Architecture

- **JEMS:** Leverages the power of JBoss Enterprise Middleware Services : JBoss Application Server, JBoss Cache, JGroups, and Hibernate.
- **DB Agnostic:** Will work with any RDBMS supported by Hibernate
- **SSO/LDAP:** Leverages Tomcat and JBoss single sign on (SSO) solutions.
- **JAAS Authentication:** Custom authentication via JAAS login modules.
- **Cacheing:** Utilizes render-view caching for improved performance.
- **Clusterable:** Cluster support allows for portal state to be clustered for all portal instances.
- **Hot-Deployment:** Leverages JBoss dynamic auto deployment features.
- **SAR Installer:** Browser-based installer makes installation and initial configuration a breeze.

## Supported Standards

- **Portlet Specification and API 1.0 (JSR-168)**
- **Content Repository for Java Technology API (JSR-170)**
- **Java Server Faces 1.2 (JSR-252)**
- **Java Management Extension (JMX) 1.2**
- **Full J2EE 1.4 compliance when used with JBoss AS**

## Portal and Portal Container

- **Multiple Portal Instances:** Ability to have multiple Portal instances running inside of one Portal container.
- **IPC™** Inter-Portlet Communication API enables portlets to create links to other objects such as a page, portal or window .
- **Dynamicity™** The ability for administrators and users to create and destroy objects such as portlets, pages, portals, themes, and layouts at runtime.
- **Internationalization:** Ability to use internationalization resource files for every portlet.
- **Pluggable services:** Authentication performed by the servlet container and JAAS make it possible to swap the authentication scheme.

- **Page-based Architecture:** Allows for the grouping/division of portlets on a per-page basis.
- **Existing Framework support:** Portlets utilizing Struts, Spring MVC, Sun JSF-RI, AJAX, or MyFaces are supported.

### Themes and Layouts

- **Easily swappable themes/layouts:** New themes and layouts containing images can be deployed in WAR archives.
- **Flexible API:** Theme and Layout API are designed to separate the business layer from the presentation layer.
- **Per-page layout strategy:** Different layouts can be assigned to different pages.

### User and Group Functionality

- **User registration/validation:** Configurable registration parameters allow for user email validation before activation.
- **User login:** Makes use of servlet container authentication.
- **Create/Edit Users:** Ability for administrators to create/edit user profiles.
- **Create/Edit Roles:** Ability for administrators create/edit roles.
- **Role Assignment:** Ability for administrators to assign users to roles.

### Permissions Management

- **Extendable permissions API:** Allows custom portlets permissions based on role definition.
- **Administrative interface:** Allows for permissions assignments to roles at any time for any deployed portlet, page, or portal instance.

### Content Management System

- **JCR-compliant:** The CMS is powered by Apache Jackrabbit, an open source implementation of the Java Content Repository API.
- **DB or Filesystem store support:** Configurable content store to either a filesystem or RDBMS.
- **External Blob Support:** Configurable content store allowing large blobs to reside on filesystem and content node references/properties to reside in RDBMS.
- **Versioning support:** All content edited/created is autoversioned with a history of edits that can be viewed at any time.
- **Content Serving Search-engine-friendly URLs:** <http://yourdomain/portal/content/index.html> (Does not apply to portlet actions.)



- **No long portal URLs:** Serve binaries with simple urls. (<http://domain/files/products.pdf>)
- **Multiple HTML Portlet instance support:** Allows for extra instances of static content from the CMS to be served under separate windows.
- **Directory Support:** create, move, delete, copy, and upload entire directory trees.
- **File Functions:** create, move, copy, upload, and delete files.
- **Embedded directory-browser:** When copying, moving, deleting, or creating files, administrators can simply navigate the directory tree to find the collection they want to perform the action on.
- **Ease-of-use architecture:** All actions to be performed on files and folder are one mouse-click away.
- **Full-featured HTML editor:** HTML Editor contains WYSIWYG mode, preview functionality, and HTML source editing mode. HTML commands support tables, fonts, zooming, image and url linking, flash movie support, bulleted and numbered list, and dozens more.
- **Editor style-sheet support:** WYSIWYG editor displays current Portal style-sheet, for easy choosing of classes.
- **Internationalization Support:** Content can be attributed to a specific locale and then served to the user based on his/her browser settings.

## Message Boards

- **Instant reply:** Instant reply feature, makes for one-click replies to posts.
- **Post quoting:** Quote an existing topic and poster within a reply.
- **Flood control:** Prevents abuse of multiple posts withing a set configurable time-frame.
- **Category creation:** Create a category that contains forums within it.
- **Forum creation:** Create a forum and assign it to a specific category.
- **Forum modification:** Edit, move, delete forums.
- **Forum and category reordering:** Reorder categories and forums in the order you would like them to appear on the page.

---

# Target Audience

Portlet developers or those wishing to implement/extend the JBoss Portal framework.

---

# Acknowledgements

We would like to thank **all** the developers that participate in the JBoss Portal project effort.

Specifically,

1. Thomas Heute, for his help on the first-ever version of JBoss Portal and the corresponding documentation. ;-)
2. Remy for his help with Tomcat configuration.
3. Mark Fernandes and Paul Tamaro from Novell, for their hard work in supplying the portal project with usable and attractive themes and layouts.
4. Kev "kevs3d" Roast for supplying us with two working portlets that integrate existing frameworks in to the portal: Sun JSF-RI and Spring MVC Portlet.
5. Swarn "sdhaliwal" Dhaliwal for supplying us with the Struts-Bridge, that will allow for existing struts applications to work with the Portal.

Contributions of any kind are always welcome, you can contribute by providing ideas, filling bug reports, producing some code, designing a theme, writing some documentation, etc... To report a bug please use our Jira system [<http://jira.jboss.com>] .

## Upgrading 2.0 - 2.2

Roy Russo <roy at jboss dot org>

Boleslaw Dawidowicz <boleslaw.dawidowicz@jboss.com>

This chapter addresses migration issues from version 2.0 to 2.2 of JBoss Portal.

### 1.1. Deployment Descriptors

From version 2.0 to 2.2, the JBoss Portal deployment descriptors have changed when defining pages, portlets, and portal instances.

#### 1.1.1. Example - Assigning a Portlet on a Page

To describe the changes made to the deployment descriptors, we have made available an example that you can download [here](http://labs.jboss.com/file-access/default/members/portletswap/downloads/portlets/samples/HelloWorldPortlet.zip): HelloWorld Portlet [http://labs.jboss.com/file-access/default/members/portletswap/downloads/portlets/samples/HelloWorldPortlet.zip]. After this helloworldportlet.ear is deployed, you should be able to access the new portal page by pointing your browser to <http://localhost:8080/portal/portal/default/HelloWorld>.

All portal, page, and portlet instance deployment is now handled by one file: \*-object.xml. You no longer need the \*-portal.xml, \*-pages.xml, and \*-instances.xml found in JBoss Portal 2.0. For our example we make available *helloworld-object.xml* located under *helloworldportlet.war/WEB-INF/*, and it looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployments>
<deployment>
<if-exists>overwrite</if-exists>
<parent-ref>default</parent-ref>
<properties/>
<page>
<page-name>Hello World</page-name>
<properties/>
<window>
<window-name>HelloWorldPortletWindow</window-name>
<instance-ref>HelloWorldPortletInstance</instance-ref>
<region>center</region>
<height>0</height>
</window>
</page>
</deployment>
<deployment>
<if-exists>overwrite</if-exists>
```

```
<instance>
  <instance-name>HelloWorldPortletInstance</instance-name>
  <component-ref>helloworld.HelloWorldPortlet</component-ref>
</instance>
</deployment>
</deployments>
```

A deployment file can be composed of a set of `<deployments>`. In our example file, above, we are defining a page, placing the `HelloWorldPortlet` as a window on that page, and creating an instance of that portlet. You can then use the Management Portlet (bundled with JBoss Portal) to modify the instances of this portlet, reposition it, and so on...

- **<if-exists>** Possible values are *overwrite* or *keep* . *Overwrite* will destroy the existing object and create a new one based on the content of the deployment. *Keep* will maintain the existing object deployment or create a new one if it does not yet exist.
- **<parent-ref>** Indicates where the object should be hooked in to the portal tree. See the chapter on Dynamicity for the tree-concept within JBoss Portal.
- **<properties>** Properties definition specific to this page, commonly used to define the specific theme and layout to use. If not defined, the default portal layouts/theme combination will be used.
- **<page>** The start of a page definition.
- **<page-name>** The name of the page.
- **<window>** The start of a window definition.
- **<window-name>** The name of the window.
- **<instance-ref>** The instance reference used by this window. Should correspond with the `<instance-name>` variable.
- **<height>** The vertical position of this window within the region defined in the layout.
- **<instance>** The start of an instance definition. page.
- **<instance-name>** Maps to the above `<instance-ref>` variable.
- **<component-ref>** Takes the name of the application followed by the name of the portlet, as defined in the *portlet.xml*

### Note

For further explanation of the deployment descriptor, please view the XMLDescriptor section in the Reference Guide

## 1.2. Content Management System

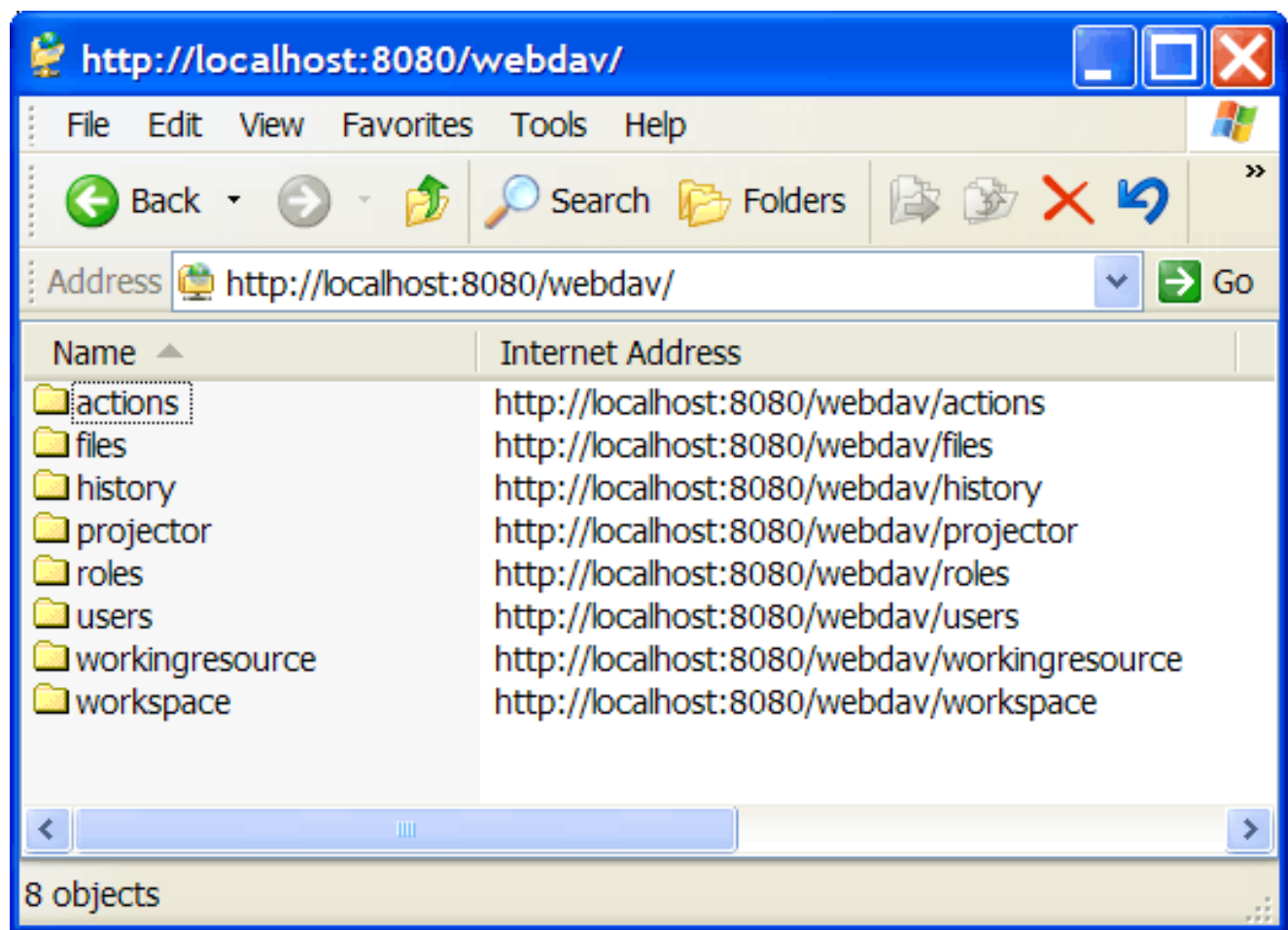
From version 2.0 to 2.2, the JBoss Portal Content Management System changed from using Apache Slide API to the Java Content Repository (JCR), JSR-170 [<http://www.jcp.org/en/jsr/detail?id=170>] using the Apache Jackrabbit [<http://incubator.apache.org/jackrabbit/>] implementation.

### 1.2.1. Migrating Content

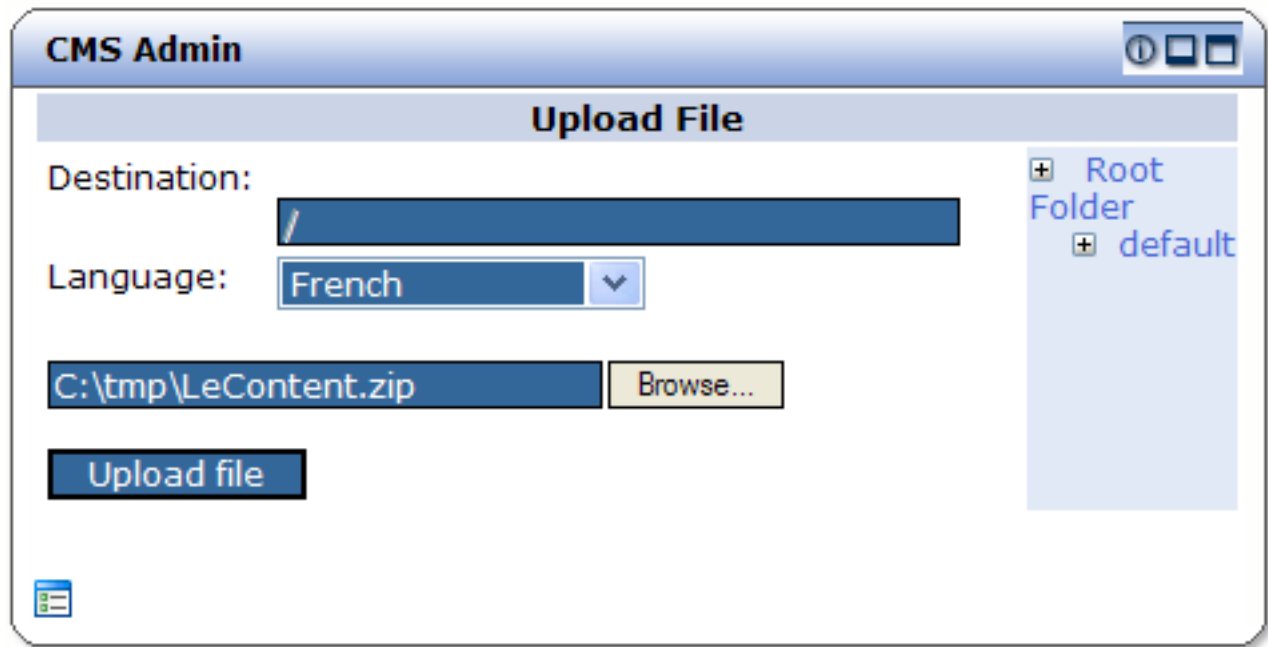
Since the underlying layer of the CMS has changed, it will be necessary for users migrating from 2.0 to move their content, so the following steps describe how to perform this operation.

JBoss Portal v2.0 had native WebDAV support, allowing a user to connect to the content repository via the Operating System, given the proper credentials. You will use this method to extract the content, zip it in an archive, and upload it to the new CMS.

- First, start up your previous installation of JBoss Portal 2.0, and connect to it using MS WebFolders. Using the *Add Network Place* option under *My Network Places*, add a new network place, giving it the path to your webdav repository. By default it is `http://localhost:8080/webdav`. Upon providing the proper credentials, you should see your repository structure.



- Navigate to `http://localhost:8080/webdav/files` and your entire content directory structure with files should be available here. You should be able now to zip these directories and upload them as an archive to the JBoss Portal 2.2 CMS via the CMSAdminPortlet.



## Warning

There are two known issues you need to know about when importing content from the old repository using this method:

- Version information will be lost.
- You must verify that pre-existing links to local resources are correct.

## 1.3. Forums Migration

### 1.3.1. Forums DB schema issues

Database schema differs slightly between portal 2.0.0 and 2.0.1 versions. Some new tables were added for new functionality. There were few columns removed or type changed also.

From 2.0.1 RC2 version portal performs schema update try during startup/deployment. Hibernate SchemaUpdate hbm2ddl tool is able to add new tables or new columns. What it doesn't do is removing unnecessary columns or column sql-type changes.

Besides of that, it is always good to back up your data as this behaviour might depends on different RDBMS versions.

### 1.3.2. Portal 2.0.0 to 2.0.1 Forums migration

In portal 2.0.1 there are some changes in db schema related to Forums Portlet

For eg. columns such as:

- jbp\_forums\_forums --> jbp\_last\_post\_id
- jbp\_forums\_topics --> jbp\_first\_post\_id
- jbp\_forums\_topics --> jbp\_last\_post\_id

are now not used. These are retrieved using Hibernate collections storing capabilities.

Column:

- jbp\_forums\_posts --> jbp\_text

had wrong SQL type. It was 'varchar(255)' in 2.0.0 and it is 'text' in 2.0.1.

### 1.3.3. Nessesary steps to migrate Forums

After upgrading portal to 2.0.1, schema should be updated automaticly and all new nessesary tables/columns created. If this process fail the schema will be dropped/created. Remember to backup your data before doing migration!

After successfull update beware of the fact that you will have:

- a number of unused columns in schema
- texts of messages stored in varchar(255) column - Posts in forums couldn't be longer than 255 chars. In fact longer messages will cause portlet exception...

To deal with second issue we must change jbp\_forums\_posts-->jbp\_text column type. It's very simple to do in MySQL RDBMS:

```
ALTER TABLE jbp_forums_posts CHANGE jbp_text jbp_text text
```

In Postgres it will be:

```
ALTER TABLE portal.jbp_forums_posts ALTER jbp_text TYPE text;
```

This will change column type.

Check in your RDBMS docs if such ALTER TABLE SQL statement works. If not you should probably recreate jbp\_forums\_posts table with proper SELECT/INSERT statement.



## JSR168 portlets

Thomas Heute <theute@jboss.org>

### 2.1. Introduction

The JSR 168 specification aims at defining portlets that can be used by any JSR168 portlet container also called portals. There are different portals out there with commercial and non-commercial licences. In this chapter we will briefly describe such portlets but for more details you should read the specifications available on the web.

As of today, JBoss portal is fully JSR168 1.0 compliant, that means that any JSR168 portlet will behave as it should inside the portal.

### 2.2. The basics

What is really important to know about such portlets is that when a page is displayed it is divided into two distinct parts, an action part on one portlet followed by rendering parts for every portlets displayed on a page. A portal just aggregates all the chunks of HTML rendered by the different portlets of a page.

#### 2.2.1. Portal

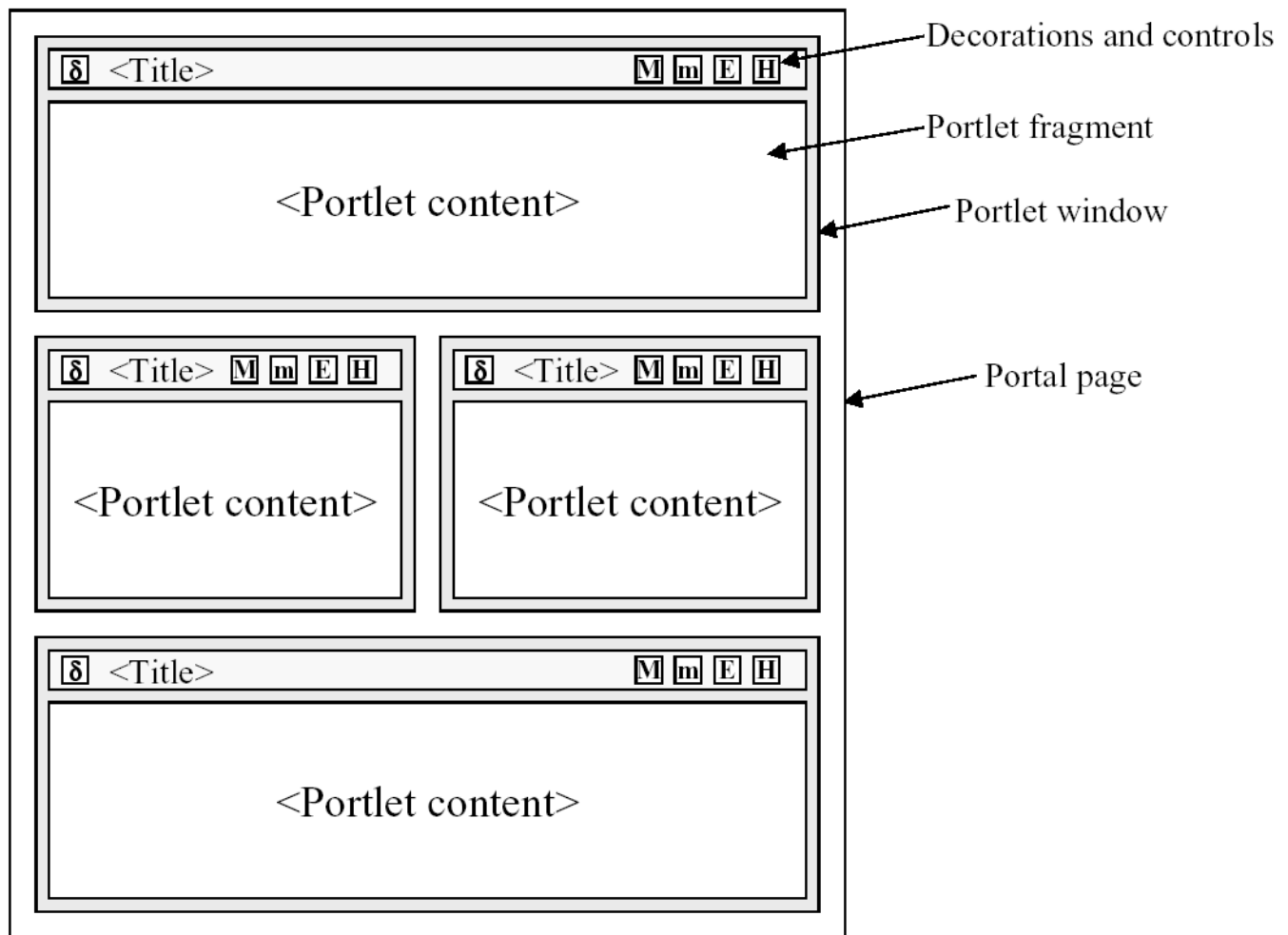
Before we even talk about portlets, let's talk about the container called portal.

A portal is basically a web application in which modules can be easily added or removed. We call those modules 'portlets'. A module can be as complex as a forum, a news management system or as simple as a text or text with images with no possible interaction.

On a single web page different portlets can appear at the same time.

#### 2.2.2. Page composition

A portal can be seen as pages with different areas and inside areas, different windows and each window having one portlet.



### 2.2.3. Rendering modes

A portlet can have different view modes, three modes are defined by the specification but a portal can extend those modes.

## XML descriptors

Roy Russo <roy@jboss.org>

### 3.1. Introduction

To define your portals and page, you will need to create an XML files in order to declare your portlet, portlet instances, windows, pages and then your portals. All portal, page, and portlet instance deployment are handled by one file: \*-object.xml.

### 3.2. Defining a new portlet instance

It may be necessary at times for you to deploy your portlets and not have them assigned to any specific page, so an administrator can then use the management UI to place them where he wishes. This example walks the reader through deploying a portlet instance, but not assigning it to any specific page.

The helloworld-object.xml for a simple HelloWorldPortlet is described below:

```
<?xml version="1.0" encoding="UTF-8"?>
  <deployments>
    <deployment>
      <if-exists>overwrite</if-exists>
      <instance>
        <instance-name>HelloWorldPortletInstance</instance-name>
        <component-ref>helloworld.HelloWorldPortlet</component-ref>
      </instance>
    </deployment>
  </deployments>
```

A deployment file can be composed of a set of <deployments>. In our example, above, we are defining the HelloWorldPortletInstance, and referencing the HelloworldPortlet web application name and the definition in the portlet.xml. You can then use the Management Portlet (bundled with JBoss Portal) to modify the instances of this portlet, reposition it, and so on...




















- **<if-exists>** Possible values are *overwrite* or *keep* . *Overwrite* will destroy the existing object and create a new one based on the content of the deployment. *Keep* will maintain the existing object deployment or create a new one if it does not yet exist.

- **<instance>** The start of an instance definition. page.
- **<instance-name>** Name given to this instance of the portlet.
- **<component-ref>** Takes the name of the application followed by the name of the portlet, as defined in the *portlet.xml*

Once the portlet has been deployed (you can hot deploy it on a live instance of JBoss Portal, as well), you should see it in the Management Portlet under available portlet instances.

## Management Portlet

Manage:  Portal  Instances  Portlet

Id	Portlet	Action
PolicyConfiguratorPortletInstance	PolicyConfiguratorPortlet	
CMSPortletInstance	CMSPortlet	
CMSAdminPortletInstance	CMSAdminPortlet	
ManagementPortletInstance	ManagementPortlet	
NavigationPortletInstance	NavigationPortlet	
UserPortletInstance	UserPortlet	
CatalogPortletInstance	CatalogPortlet	
ThemePortletInstance	ThemeManager	
ThemeSelectorInstance	ThemeSelectorPortlet	
TestPortletInstance	Portlet Test	
CounterPortletInstance	CounterPortlet	
CachedCounterPortletInstance	CachedCounterPortlet	
PortletSessionPortletInstance	PortletSessionPortlet	
CharsetPortletInstance	CharsetPortlet	
ExceptionPortletInstance	ExceptionPortlet	
MissingPortletInstance	Portlet not deployed	
PreferencesPortletInstance	PreferencesPortlet	
PortletAInstance	PortletA	
PortletBInstance	PortletB	
HeaderContentInstance	HeaderContentPortlet	
ContentRewriteInstance	ContentRewritePortlet	
SecuredTestPortletInstance	SecuredTestPortlet	
HintPortletInstance	HintPortlet	
WsrpSelectorTestPortletInstance	WsrpSelectorTestPortlet	
HelloWorldPortletInstance	HelloWorldPortlet	

### 3.3. Defining a new portal page.

To illustrate our example, we have made available a portlet that you can download here: HelloWorld Portlet [<http://labs.jboss.com/file-access/default/members/portletswap/downloads/portlets/samples/HelloWorldPortlet.zip>]

For our example we make available *helloworld-object.xml* located under *helloworldportlet.war/WEB-INF/*, and it looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployments>
  <deployment>
    <if-exists>overwrite</if-exists>
    <parent-ref>default</parent-ref>
    <properties/>
    <page>
      <page-name>Hello World</page-name>
      <properties/>
      <window>
        <window-name>HelloWorldPortletWindow</window-name>
        <instance-ref>HelloWorldPortletInstance</instance-ref>
        <region>center</region>
        <height>0</height>
      </window>
    </page>
  </deployment>
  <deployment>
    <if-exists>overwrite</if-exists>
    <instance>
      <instance-name>HelloWorldPortletInstance</instance-name>
      <component-ref>helloworld.HelloWorldPortlet</component-ref>
    </instance>
  </deployment>
</deployments>
```

A deployment file can be composed of a set of <deployments>. In our example file, above, we are defining a page, placing the HelloWorldPortlet as a window on that page, and creating an instance of that portlet. You can then use the Management Portlet (bundled with JBoss Portal) to modify the instances of this portlet, reposition it, and so on...

- **<if-exists>** Possible values are *overwrite* or *keep*. *Overwrite* will destroy the existing object and create a new one based on the content of the deployment. *Keep* will maintain the existing object deployment or create a new one if it does not yet exist.
- **<parent-ref>** Indicates whether the object should be hooked in to the portal tree.
- **<properties>** Properties definition specific to this page, commonly used to define the specific theme and layout to use. If not defined, the default portal layouts/theme combination will be used.
- **<page>** The start of a page definition.
- **<page-name>** The name of the page.
- **<window>** The start of a window definition.

- **<window-name>** The name of the window.
- **<instance-ref>** The instance reference used by this window. Should correspond with the **<instance-name>** variable.
- **<height>** The vertical position of this window within the region defined in the layout.
- **<instance>** The start of an instance definition. page.
- **<instance-name>** Maps to the above **<instance-ref>** variable.
- **<component-ref>** Takes the name of the application followed by the name of the portlet, as defined in the *portlet.xml*

### 3.4. Defining a new portal instance

To illustrate our example, we have made available a portlet that you can download here: HelloPortal [<http://labs.jboss.com/file-access/default/members/portletswap/downloads/portlets/samples/HelloPortal.zip>] .

For our example we make available *helloworld-object.xml* located under *helloworldportlet.war/WEB-INF/* , and it looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployments>
  <deployment>
    <parent-ref/>
    <if-exists>overwrite</if-exists>
    <portal>
      <portal-name>HelloPortal</portal-name>
      <properties>
        <!-- Set the layout for the default portal -->
        <!-- see also portal-layouts.xml -->
        <property>
          <name>layout.id</name>
          <value>generic</value>
        </property>
        <!-- Set the theme for the default portal -->
        <!-- see also portal-themes.xml -->
        <property>
          <name>theme.id</name>
          <value>Nphalanx</value>
        </property>
        <!-- set the default render set name (used by the render tag in layouts) -->
        <!-- see also portal-renderSet.xml -->
        <property>
          <name>theme.renderSetId</name>
          <value>divRenderer</value>
        </property>
        <!-- set the default strategy name (used by the strategy interceptor) -->
        <!-- see also portal-strategies.xml -->
        <property>
          <name>layout.strategyId</name>
          <value>maximizedRegion</value>
        </property>
      </properties>
      <supported-modes>
        <mode>view</mode>
```

```

        <mode>edit</mode>
        <mode>help</mode>
    </supported-modes>
    <supported-window-states>
        <window-state>normal</window-state>
        <window-state>minimized</window-state>
        <window-state>maximized</window-state>
    </supported-window-states>
    <page>
        <page-name>default</page-name>
        <properties/>
        <window>
            <window-name>HelloWorldPortletWindow</window-name>
            <instance-ref>HelloWorldPortletInstance</instance-ref>
            <region>center</region>
            <height>0</height>
        </window>
    </page>
</portal>
</deployment>
<deployment>
    <if-exists>overwrite</if-exists>
    <parent-ref>HelloPortal</parent-ref>
    <page>
        <page-name>foobar</page-name>
        <window>
            <window-name>HelloWorldPortletWindow</window-name>
            <instance-ref>HelloWorldPortletInstance</instance-ref>
            <region>center</region>
            <height>0</height>
        </window>
    </page>
</deployment>
<deployment>
    <if-exists>overwrite</if-exists>
    <instance>
        <instance-name>HelloWorldPortletInstance</instance-name>
        <component-ref>helloworld.HelloWorldPortlet</component-ref>
    </instance>
</deployment>
</deployments>

```

This example, when deployed, will register a new portal instance named `HelloPortal` with two pages in it. The portal instance can be accessed by navigating to: `http://localhost:8080/portal/portal/HelloPortal` for the default page, and `http://localhost:8080/portal/portal/HelloPortal/foobar` , for the second page created.

### Note

You must define a page named `default` for any new portal instance to be accessible via a web browser.

## 3.5. Defining Portlet Instance Preferences

The portlet specification allows a portlet instance to override the preferences that were set in the `portlet.xml`. Using our `HelloWorld` example, we can demonstrate how this is done in JBoss Portal.

In our sample `helloworld-object.xml` for a simple `HelloWorldPortlet` we will add preference attributes as such:

```


```



```
<?xml version="1.0" encoding="UTF-8"?>
<deployments>
  <deployment>
    <if-exists>overwrite</if-exists>
    <instance>
      <instance-name>HelloWorldPortletInstance</instance-name>
      <component-ref>helloworld.HelloWorldPortlet</component-ref>
      <preferences>
        <preference>
          <name>foo</name>
          <value>bar</value>
          <read-only>false</read-only>
        </preference>
      </preferences>
    </instance>
  </deployment>
</deployments>
```

In the example above, we are overriding the portlet.xml preference named *foo* and assigning it a value, *bar* . From within our portlet now, we will access the preferences as the Portlet API allows:

```
String somePref = request.getPreferences().getValue("foo", "somedefaultvalue");
```

Likewise, you can specify in the \*-object.xml descriptor, an array of preferences, as the specification allows:

```
<preference>
  <name>foo</name>
  <value>hi</value>
  <value>hello</value>
  <value>yo!</value>
  <read-only>true</read-only>
</preference>
...
```

---

# 4

## Portal urls

**Julien Viet** <julien@jboss.org>

**Thomas Heute** <theute@jboss.org>

**Roy Russo** <roy@jboss.org>

### 4.1. Introduction

Most of the time portals use very complicated urls, however it is possible to setup entry points in the portal that follow simple patterns.

Each portal container can contain multiple portals and within a given portal, windows are organized in pages, a page simply being a collection of windows associated to a name.

Before reading this chapter you must know how to define a page and a portal, you can refer to the chapter about XML descriptors to have a better understanding of those notions.

### 4.2. Accessing a portal

Each portal container can contains multiple portals, also there is one special portal which is the default portal, i.e the one used when no portal is specified in particular.

- `"/`, will point to the default page of the default portal.
- `"/portal/portalname/"` will point to the default page of the portal `portalname`

### 4.3. Accessing a page

It is possible to have multiple pages per portal. As for portal there is a default page for a given portal. Once the portal has been selected, then a page must be used and all the windows present in that page will be rendered. The page selection mechanism is the following.

- `"/portal/default/pageName"` will render the `pageName` page.

## 4.4. Accessing CMS Content

The CMSPortlet delivers content transparently, without modifying the url displayed. However, if you wish to deliver binary content (gif, jpeg, pdf, zip, etc...), it is desirable to display this content outside of the confines of the portal.

- `"/content/default/images/jboss_logo.gif"` will display the `jboss_logo.gif` outside of the portal. This is accomplished as the portal interprets any path beginning with `/content` as a request for CMS content. As long as the mime-type is not `text/html` or `text/text`, it will be rendered independant of the portal.

## Security

**Thomas Heute** <theute@jboss.org>

**Martin Holzner** <mholzner@novell.com>

New in version 2.2, the portal features role based access control for all portal resources. The definition of security constraints is strait forward and simple. Each portal resource can define zero, one, or more security constraints, each binding a role to the allowed actions for the role and resource. For more details on how to secure portal resources take a look at security section in the user guide.

### 5.1. Portal Security Configuration

Portal security is governed by a set of portal policy configurations. Policy configurations can be created for different permission types, and accessed via the policy configuration service. The policy configuration maintains a set of security constraints, keyed by permission type and uri of the resource.

#### 5.1.1. org.jboss.portal.security.config.SecurityConstraint

```
// create a new constraint
SecurityConstraint constraint = new SecurityConstraint("view,read","Admin");

// get the allowed actions as a set of strings
Set allowedActions = constraint.getActions();

// get the role this constraint is assigned to
String allowedRole = constraint.getRole();
```

Security constraints are the state holder of what role is allowed to execute what actions on a given resource. They are managed via a policy configuration. One policy configuration can be defined for each permission type.

Permissions are currently separated into 3 valid types:

- component
- instance
- portalobject

### 5.1.2. org.jboss.portal.security.config.PortalPolicyConfig

The policy configuration maintains the constraint information for one permission type.

```
PortalPolicyConfig config = null;
//.... get the config from the config service (see below)
String uri = portalResourceHandle;
Set constraints = config.getConstraints(uri);
```

### 5.1.3. org.jboss.portal.security.config.PortalPolicyConfigService

The policy configuration service is the root access point for configuration information. All active policy configurations are registered with this service, and can be accessed from here via their permission type.

```
import org.jboss.portal.security.config.PortalPolicyConfigService;
import org.jboss.portal.security.config.PortalPolicyConfig;
import org.jboss.portal.core.security.PermissionTypes;

PortalPolicyConfigService configService = null;
// configService = ... lookup the configService from the mbean server

// get the policy config for all portlet instances
PortalPolicyConfig instanceConfig = configService.getConfig(PermissionTypes.INSTANCE);
PortalPolicyConfig portalObjectConfig = configService.getConfig(PermissionTypes.PORTAL);
```

### 5.1.4. org.jboss.portal.security.config.ConfigListener

The policy configuration listener registers with the policy configuration, and listens to added, removed, and modified events. Each change of the security constraints maintained in a policy configuration is propagated to this listener. The listener builds the bridge to any security enforcement infrastructure.

The default implementation of the enforcement infrastructure is based in the Java Authorization Contract for Containers (JACC). It is implemented as a policy configuration listener that reacts to configuration change events, and populates the JACC policy with permissions that are created based on the security constraints propagated by the change events.

## 5.2. Portal Security Enforcement

### 5.2.1. org.jboss.portal.security.PortalPermission

The defined security constraints are enforced via portal permissions. The default security implementation that comes with the portal contains a policy configuration listener that converts the constraint information into portal permissions, and populates a JACC (Java Authorization Contract for Containers) based security policy with them.

The configured constraints are enforced via an interceptor in the server invocation stack. For each request to the

portal it creates a new portal permission to check access rights. The created permission contains the requested resource and action. The permission is passed to the portal policy, together with the portal subject to check if the permission is implied in the policy. Access will be granted if one of the security constraints in the policy configurations implies the provided check permission.

A factory is provided to create portal permissions. The `PortalPermissionFactory` is an interface that is implemented as an mbean, and can be injected wherever needed.

```
import org.jboss.portal.security.PortalPermission;
import org.jboss.portal.security.PortalPermissionFactory;
import org.jboss.portal.core.security.PermissionTypes;

PortalPermissionFactory portalPermissionFactory = ...;
PortalPermission componentPerm = portalPermissionFactory.createPermission(PermissionT
```

### 5.2.2. org.jboss.portal.security.PortalPolicy

The portal policy is the interface that is being used to check if access to a requested resource is granted. It is made available as a thread local property throughout the lifetime of a portal request. The implementation of the policy is injected via a server invocation aspect (`PolicyAssociationInterceptor`). This interceptor gets the policy via the implementation of the `PortalPolicyService`. Here is the code to get to the portal policy:

```
import org.jboss.portal.security.PolicyAssociation;
import org.jboss.portal.security.PortalPolicy;

PortalPolicy policy = PolicyAssociation.getPolicy();
```

### 5.2.3. org.jboss.portal.security.PortalSubject

The portal subject represents the current user. Similar to the portal policy, it is made available for the lifetime of the portal request as a thread local property, which is injected by a server invocation aspect. Here is the code to get to the portal policy:

```
import org.jboss.portal.security.SubjectAssociation;
import org.jboss.portal.security.PortalSubject;

PortalSubject subject = SubjectAssociation.getSubject();
```

### 5.2.4. Portal Policy Enforcement

All the described pieces above taken together make up the security enforcement infrastructure. The remaining call to determine whether access is granted or not is:

```
if (policy.implies(subject, permission)){
    // ....do something....
}
```

## 5.3. Default Security Implementation

The default implementation of the security APIs is based on the Java Authorization Contract for Containers (JACC).

### 5.3.1. What is JACC

JACC is part of the J2EE specification, and defines a way for the servlet and ejb containers to secure web resources and ejbs with a role based model.

### 5.3.2. JACC Policy and Policy Configuration

#### 1. `javax.security.jacc.PolicyConfigurationFactory`

The policy configuration factory is an abstract singleton that serves as the access point for policy configurations. It features a static accessor to get the factory implementation.

```
public static PolicyConfigurationFactory getPolicyConfigurationFactory()  
    throws ClassNotFoundException, PolicyContextException  
{  
    ....  
}
```

The policy configuration factory to load can be specified in a VM system property (-D) "javax.security.jacc.PolicyConfigurationFactory.provider". The default implementation that will be loaded if nothing else is defined, is `org.jboss.security.jacc.JBossPolicyConfigurationFactory`.

#### 2. `javax.security.jacc.PolicyConfiguration`

The policy configuration interface is the access point for all configured permissions of one context id. It allows to manipulate the content of the security policy (see below) which is the ultimate policy decision point. In the standard J2EE model, a context id is equivalent to the deployed context: the war or ejb archive. In the case of JBoss portal, this model was extended to allow a more flexible definition of the context id. The default implementation of the portal's `JBossSecurityProvider` (see below) creates a separate context id for each portal resource. This allows the portal to reload the content of a specific portal resources without having to refresh any of the other resource's security attributes.

#### 3. `javax.security.jacc.Policy`

The jacc policy is the ultimate policy decision point. It makes the decision whether a permission is implied or not. To make this decision it can do whatever is needed. The default implementation (`org.jboss.security.jacc.DelegatingPolicy`) delegates the decision to the policy context, which in turn delegates it to the `java.security.Permissions` that make up all configured permissions for a given role in the requested policy context.

#### 4. org.jboss.portal.security.jacc.SecurityProvider

The portal security provider interface wraps the functionality in the jacc PolicyConfigurationFactory. It allows access to all available policy configurations. It also offers a method to check whether a particular policy context is in service or not. The jacc policy singleton is also available via this interface.

```
// check if the provided policy context id is available
boolean inService(String policyContextID) throws PolicyContextException;
// get the policy configuration for the provided policy context id
PolicyConfiguration getPolicyConfiguration(String contextID, boolean remove)
// get the jacc policy singleton
Policy getPolicy();
```

#### 5. org.jboss.portal.security.jacc.JBossSecurityProvider

The jboss security provider extends the portal security provider, and adds access to the server configuration. This additional access allows a provider implementation to consider server configuration properties when making a decision about any of the policy context calls.

```
// get the portal server configuration
ServerConfig getServerConfig();
```

### 5.3.3. Portal Policy and Policy Service

#### 1. org.jboss.portal.security.PortalPolicyService

The portal policy service is the root service from where one can get access to the portal policy. The default implementation of this service interface (org.jboss.portal.security.impl.jacc.PortalPolicyServiceImpl) is an mbean. It uses the implementation of the JBossSecurityProvider interface to get to the jacc policy singleton. The security provider interface is a wrapper of the jacc PolicyConfigurationFactory. Its implementation delegates all calls to the standard jacc PolicyConfigurationFactory.

The PortalPolicyService exposes only one method:

```
PortalPolicy getPolicy();
```

#### 2. org.jboss.portal.security.PortalPolicy

The portal policy has only one method:

```
boolean implies(PortalSubject subject, PortalPermission permission);
```

The default implementation of the portal policy, an inner class of the policy service implementation,



(org.jboss.portal.security.impl.jacc.PortalPolicyServiceImpl\$PortalPolicyImpl), uses the security provider to gain access to the portal policy implementation. It uses the JaccHelper class in the same package to determine the java.security.ProtectionDomain and the policy context id before calling the jacc policy to do the actual permission check.

### 5.3.4. Who is asking?

The remaining piece in the puzzle is the identity. How does the portal determine what user and ultimately what roles to check the access for?

#### 5.3.4.1. The Jacc Portal Subject

As described above, the portal injects the org.jboss.portal.security.PortalSubject as a thread local property that is available throughout the portal request. The default implementation of this portal subject (org.jboss.portal.security.impl.jacc.JaccPortalSubject) wraps the jacc way of getting to the authenticated java.security.Subject, and reading the role memberships off of it. In other words: the mechanism of getting the subject and the subjects roles is identical to the way the application server is handling this security aspect.

## 5.4. Write your own Security Implementation

As you can probably see above, there are many ways to plug into this infrastructure and adapt it to your needs. You will encounter the least amount of work if you plugin at the JACC layer. All you need to do is provide your own implementation of the JACC policy. All the rest of the infrastructure can stay as is. If you don't want to stay with JACC as your security provider, it will require you to do some more work, depending on how far you want to get away from JACC. To implement a security layer without any JACC artifacts, you'll need to provide the policy configuration lister, the policy enforcement interceptor and the policy and subject interceptors. Let's look at this in more detail.

### 5.4.1. Overwriting the JACC policy

#### 5.4.1.1. Write the Policy class

As already mentioned above, the easiest way to get your own behavior for the portal authorization, you can provide your own implementation of the JACC policy. To do so, you'll need to create a class that extends java.security.Policy, and overwrites the implies method. Be aware that the portal goes beyond the JACC specification in that it allows new permission classes, other than the 5 predefined classes from the specification. All permissions used by the portal extend the abstract org.jboss.portal.security.PortalPermission class, which in turn extends java.security.Permission. The currently (as of 2.2) implemented permission classes are ComponentPermission, InstancePermission, and PortalObjectPermission. Your policy implementation can distinguish those permissions, and handle them differently if it so desires.

For an example on how to implement your own, take a look at the portal's default policy implementation. The class is org.jboss.portal.security.impl.jacc.PortalJaccPolicy. This policy is basically a copy of the application servers DelegatingPolicy class. The idea behind it is that the portal policy is stacked on top of the J2SE policy, configured in the VM. The portal policy will handle authorization checks for all the allowed JACC permissions (for ejb and servlet/jsp calls), in addition to the basic J2SE permission types. It allows to configure external permissions that will be treated just like the basic JACC permissions. If the incoming permission to check is an instance of any of

the five JACC permission classes, or one of the configured external permission classes, the policy will handle the implies check, otherwise it will delegate to the J2SE policy that was in place in the VM, before the portal policy was created. Remember that the java security model allows only one policy to be active in the VM at any given time. By stacking the policies there is a way to have several policies active with only one policy configured at the VM level.

It is up to your implementation if you want to copy this behavior of configurable external permissions and stacked policies or not. You could implement a flat policy that treats all permissions the same, but be aware that since your policy will take the place of the only policy in the system, that all policy decisions will be coming into your policy. So all security checks by the J2SE SecurityManager will have to be handled by your policy as well.

One strange anomaly that is currently built into the portal is the fact that your policy needs to adhere to a naming convention. There needs to be a method that is used by the policy service to pick up the policy instance. Your policy needs to provide a method with the following signature: (Note: the method signature is not enforced by any interface or abstract class, since the origin of this requirement is a workaround for an underlying problem with the policy configuration and redeployments of the portal sar):

```
public Policy getPolicyProxy()
```

You can return 'this', or any decorator class of your policy that would allow you to restrict the externally available functionality of the policy.

#### 5.4.1.2. Configure the Policy to be used by the Portal

The policy implementation must be an mbean. It is injected into the policy service mbean to make it available to other interested parties. Take a look at the jboss-service.xml in the jboss-portal.sar (in the meta-inf folder). The mbean definition for the "portal:service=SecurityProvider" injects the "jboss.security:service=JaccPolicyProvider" via the PolicyName attribute. In other words: the service that is configured as "jboss.security:service=JaccPolicyProvider" needs to expose a method as described above, that returns an implementation of the java.security.Policy interface (public Policy getPolicyProxy(){...}). This method will be called by the SecurityProvider service at startup to read the configured policy from the mbean server. Here is the section defining the jacc policy from the jboss-service.xml:

```
<mbean
  code="org.jboss.portal.security.impl.jacc.PortalJaccPolicy"
  name="jboss.security:service=JaccPolicyProvider"
  xmban-dd="org/jboss/portal/security/impl/jacc/PortalJaccPolicy.xml">
  <attribute name="ExternalPermissionTypes">
    org.jboss.portal.core.security.PortalObjectPermission,
    org.jboss.portal.core.security.InstancePermission,
    org.jboss.portal.core.security.ComponentPermission
  </attribute>
</mbean>
```

And here is the section defining the jacc security provider service, and injecting the policy via the policy proxy attribute:

```
<mbean
```

```

code="org.jboss.portal.security.impl.jacc.JBossSecurityProviderImpl"
name="portal:service=SecurityProvider"
xmbean-dd="">
  <xmbean>
    ...more descriptor info here...
  </xmbean>
  <attribute name="PolicyName">jboss.security:service=JaccPolicyProvider</attribute>
  <attribute name="PolicyAttributeName">PolicyProxy</attribute>
</mbean>

```

Note: If there is no policy mbean active for the defined PolicyName and PolicyAttributeName in the SecurityProvider mbean, the policy currently set in the VM will be used instead. The policy will be determined via:

```

import java.security.Policy;
Policy policy = Policy.getPolicy();

```

If your policy extends PortalJaccPolicy a new instance will be created everytime the portal starts. This instance will not be set as the VM wide policy, leaving the original policy in place. In other words: Policy.getPolicy() will not return your policy. To get to your policy, you'll have to lookup the portal security provider, and get it from there.

```

JBossSecurityProvider provider = ....;
Policy policy = provider.getPolicy();

```

You could of course go even further and replace the implementation of the security provider itself. That way, you can decide how to get to the policy in whatever way.

Note that the current implementation is more complex than it should be. This is due to problems in the application server's JACC policy implementation. Once those problems are fixed, the portal's implementation will be simplified. Effectively: the portal should be able to just get the current policy in the VM, (Policy p = Policy.getPolicy();) and use it. The policy would still have to be configured with the external permissions (PortalPermission classes) to treat them like JACC permissions, and not delegate the permission check to the underlying J2SE policy.

## 5.4.2. A Security implementation without JACC

As already mentioned before, implementing an authorization solution for the portal without the use of JACC all together is a bit harder. The default implementation relies havily on many of the JACC concepts. You will have to replace all of them, but it is possible.

### 5.4.2.1. The ConfigListener

To start, you'll need to change the way the security constraint information is converted into permissions. Or perhaps you wouldn't even want to convert them, and use the PortalPolicyConfigService and the PortalPolicyConfig as your one and only store of permission information. If you decide to create your own system of storing and enforcing permissions, you'll have to implement the org.jboss.portal.security.config.ConfigListener interface, and register it with the PortalPolicyConfig(s) you are interested in. Look at the org.jboss.portal.security.impl.JBossPortalPolicyConfigStoreImpl as an example of how this can be done. This is an mbean that injects the ConfigListener via an attribute (depends optional-attribute-name="ConfigListener"). If you take a look at the jboss-service.xml you'll see that this mbean is used three times, and each time a config listener is

injected. The three mbeans represent three different policy configurations, and the same service (portal:policy=JaccPortalPolicyConfigurator) registers with all three of them as a ConfigListener. You would have to replace the implementation of the policy configurator (portal:policy=JaccPortalPolicyConfigurator) and configure the three instances of JBossPortalPolicyConfigStoreImpl to register your config listener instead. Your config listener can do whatever you desire. Just keep in mind that the result needs to be accessible for the policy enforcement.

#### 5.4.2.2. Policy Enforcement

Policy enforcement is initiated via the PolicyEnforcementInterceptor in the server invocation stack. This interceptor uses the results of two other interceptors:

- PolicyAssociationInterceptor
- SubjectAssociationInterceptor

These two interceptors inject the PortalSubject and the PortalPolicy as thread local properties. The policy enforcement interceptor creates a permission based on the requested resource and action, and checks the portal policy to see if the permission implies for the determined portal subject.

You would probably keep the concept of the subject around since this is what the JAAS login module populates with the authentication information from the login process. The portal policy is what is more interesting. Your ConfigListener will populate some store with the constraint information, and that store needs to be made available to the policy enforcement side. The way to do this is via the portal policy from the PolicyAssociationInterceptor. To inject your own implementation of the PortalPolicy interface, you can provide your own version of the PortalPolicyService interface. The default implementation is an mbean that is injected into the PolicyAssociationInterceptor. Write your own version of the PortalPolicyService and configure it in the jboss-service.xml, replacing the default PortalPolicyServiceImpl. Of course you could go even further and replace the PolicyAssociationInterceptor itself no make it independent from the PortalPolicyService. The important thing is that your implementation of the PortalPolicy needs to utilize what your ConfigListener stored before.

For completeness sake, let's take a look at the SubjectAssociationInterceptor as well. It is responsible for injecting the PortalSubject. The PortalSubleject is a flagging interface (has no methods), and the default implementation (JaccPortalSubject) is simply newed up every time the SubjectAssociationInterceptor is invoked. The PortalSubject is later on used by the PortalPolicy to check if a given PortalPermission implies for the given PortalSubject.

# 6

## InterPortlet Communication (IPC)

Roy Russo <roy@jboss.org>

### 6.1. Introduction

The first version of the Portlet Specification (JSR 168), regrettably, did not cover interaction between portlets on a page. The side-effect of diverting the issue to the subsequent release of the specification, has forced portal vendors to each craft their own proprietary API to achieve interportlet communication. This chapter covers the JBoss Portal Interportlet Communication API, and its ability to allow a developer to create links to other portal pages and affect behaviour in other portlets.

#### Note

For the remainder of this chapter, the sample available here [<http://labs.jboss.com/file-access/default/members/portletswap/downloads/portlets/samples/HelloWorldIPC.zip>] , will be used to cover the API.

#### Warning

JBoss Portal IPC enables a one-to-one relationship between portlets! Do not attempt to affect behaviour of more than one portlet per action, as it will not work!

### 6.2. Configuration

#### 6.2.1. Service Descriptor

IPC requires that a listener be available to intercept requests and respond appropriately. To enable the listener, it has to be declared as an mbean in your service descriptor, as in our example under *helloworldipcportlet.sar/META-INF/jboss-service.xml* :

```
<server>
  <mbean code="org.jboss.portal.core.event.PortalEventListenerServiceImpl"
    name="portal:service=ListenerService,type=ipc_listener"
    xmbean-dd=" "
    xmbean-code="org.jboss.portal.common.system.JBossServiceModelMBean">
    <depends optional-attribute-name="Registry" proxy-type="attribute">portal:service=ListenerRegistry</attribute>
    <xmbean/>
    <attribute name="RegistryId">ipc_listener</attribute>
    <attribute name="ListenerClassName">org.jboss.portlet.hello.HelloWorldPortletB$Listener</attribute>
  </mbean>
</server>
```

It is important to note here, that you will most likely not have to change any of these values, except for the following:

- **name:** Both the *mbean type* and the *RegistryId* value must match.
- **ListenerClassName:** Full path to the listening portlet's inner class that acts as the listener.

### 6.2.2. Portal Descriptor

Enabling IPC, also requires that the listener be declared in the \*-object.xml, so the Portal recognizes it. The declaration happens at the page level, as in our example:

```
<deployments>
  <deployment>
    <parent-ref>default</parent-ref>
    <if-exists>overwrite</if-exists>
    <page>
      <page-name>IPC</page-name>
      <listener>ipc_listener</listener>
      <properties/>
    ...
```

#### Note

The *listener* value must match the *name* values in your service descriptor!

### 6.2.3. The Listener

Now that the listener is configured in the service descriptor and the portal can recognize it, we must create the appropriate listener class within our portlet.

In our example, Portlet A will be modifying the behaviour of Portlet B. So then we will declare the listener in Portlet B, the recipient portlet.

```
public static class Listener implements PortalNodeEventListener
{
    public PortalNodeEvent onEvent(PortalNodeEventBubbler bubbler, PortalNodeEvent event)
    {
        PortalNode node = event.getNode();

        // Get node name
        String nodeName = node.getName();

        // See if we need to create a new event or not
        WindowActionEvent newEvent = null;
        if (nodeName.equals("HelloWorldPortletAWindow") && event instanceof WindowActionEvent)
        {
            // Find window B
            WindowActionEvent wae = (WindowActionEvent) event;
            PortalNode windowB = node.resolve("../HelloWorldPortletBWindow");
```

```

        if(windowB != null)
        {
            // We can redirect and modify the view/mode as well!
            newEvent = new WindowActionEvent(windowB);
            newEvent.setMode(wae.getMode());
            //newEvent.setWindowState(WindowState.MAXIMIZED);
            newEvent.setParameters(wae.getParameters());
        }
    }

    if(newEvent != null)
    {
        // If we have a new event redirect to it
        return newEvent;
    }
    else
    {
        // Otherwise bubble up
        return bubbler.dispatch(event);
    }
}
}

```

It is important to note here some of the important items in this listener class. Logic used to determine if the requesting node was Portlet A.:

```
nodeName.equals("HelloWorldPortletAWindow")
```

Get the current window object so we can dispatch the event to it:

```
PortalNode windowB = node.resolve("../HelloWorldPortletBWindow");
```

Set the original parameter from Portlet A, so Portlet B can access them in its `processAction()`:

```
newEvent.setParameters(wae.getParameters());
```

Modify Portlet B windowmode and/or windowstate (ie, Maximize and place in Edit Mode):

```

newEvent.setMode(wae.getMode()); // wae.setMode(Mode.EDIT);
newEvent.setWindowState(wae.getWindowState()); // WindowState.MAXIMIZED

```

## 6.3. Communicating with another Portlet

Creating a link from Portlet A to Portlet B occurs in the same way you would normally create a `PortletURL`:

```

html.append("<form action=\"\" + resp.createActionURL() + \"\" method=\"post\">");
html.append("<input type=\"text\" name=\"sometext\"><br/>");
html.append("<select name=\"color\">");
html.append("<option>blue</option>");
html.append("<option>red</option>");
html.append("<option>black</option>");
html.append("</select>");
html.append("<input type=\"submit\"/>");

```

```
html.append("</form>");
```

As you can see from the code above, we are creating a simple HTML form with an ActionURL from within Portlet A. It will target itself, but be intercepted by the listener, created in Portlet B. This form passes some text and a color value from input fields.

Now, in Portlet B, our listener innerclass will trigger the `processAction()` in Portlet B, reading in the parameters from the Portlet A form:

```
public void processAction(JBossActionRequest request, JBossActionResponse response) throws PortletException {
    String color = request.getParameter("color");
    String text = request.getParameter("sometext");
    if(color != null && text != null)
    {
        response.setRenderParameter("color", color);
        response.setRenderParameter("sometext", text);
    }
}
```

Setting the render parameters, it will now pass them on to the Portlet B, `doView()`;

## 6.4. Communicating with another Page

As seen in the included NavigationPortlet (tabs found at the top of the default layout), we are able to create links to Portal Pages. Included in the download sample [http://labs.jboss.com/file-access/default/members/portletswap/downloads/portlets/samples/HelloWorldIPC.zip] , you can see the basics of creating a link to a Portal Page. I have added comments in-line to explain the code sample below:

```
// Get the ParentNode. Since we are inside a Window, the Parent is the Page
PortalNode thisNode = req.getPortalNode().getParent();

// Get the Node in the Portal hierarchy tree known as "../default"
PortalNode linkToNode = thisNode.resolve("../default");

// Create a RenderURL to the "../default" Page Node
PortalNodeURL pageURL = resp.createRenderURL(linkToNode);

// Output the Node's name and URL for users
html.append("Page: " + linkToNode.getName() + " -> ");
html.append("<a href=\"" + pageURL.toString() + "\">" + linkToNode.getName() + "</a>");
```



## Layouts and Themes

**Martin Holzner** <mholzner@novell.com>

**Mark Fernandes** <mfernandes@novell.com>

### 7.1. Overview

Portals usually render the markup fragments of several portlets, and aggregate these fragments into one page that ultimately gets sent back as response. Each portlet on that page will be decorated by the portal to limit the real estate the portlet has on the page, but also to allow the portal to inject extra functionality on a per portlet basis. Classic examples of this injection are the maximize, minimize and mode change links that will appear in the portlet window, together with the title.

Layouts and themes allow to manipulate the look and feel of the portal. Layouts are responsible to render markup that will wrap the markup fragments produced by the individual portlets. Themes, on the other hand, are responsible to style and enhance this markup.

In JBoss Portal, layouts are implemented as a JSP or a Servlet. Themes are implemented using CSS Style sheets, java script and images. The binding element between layouts and themes are the class and id attributes of the rendered markup.

JBoss Portal has the concept of regions on a page. When a page is defined, and portlet windows are assigned to the page, the region, and order inside the region, has to be specified as well. For portal layouts this has significant meaning. It defines the top most markup container that can wrap portlet content (other than the static markup in the JSP itself). In other words: from a layout perspective all portlets of a page are assigned to one or more regions. Each region can contain one or more portlets. To render the page content to return from a portal request, the portal has to render the layout JSP, and for each region, all the portlets in the region.

Since the markup around each region, and around each portlet inside that region, is effectively the same for all the pages of a portal, it makes sense to encapsulate it in its own entity.

To implement this encapsulation there are several ways:

- JSPs that get included from the layout JSP for each region/portlet
- a taglib that allows to place region, window, and decoration tags into the layout JSP
- a taglib that uses a pluggable API to delegate the markup generation to a set of classes

In JBoss Portal you can currently see two out of these approaches, namely the first and the last. Examples for the first can be found in the portal-core.war, implemented by the nodesk and phalanx layouts. Examples for the third

approach can be found in the same war, implemented by the industrial and Nphalanx layout. What encapsulates the markup generation for each region, window, and portlet decoration in this last approach is what's called the `RenderSet`.

The `RenderSet` consist of four interfaces that correspond with the four markup containers that wrap the markup fragments of one of more portlets:

- `Region`
- `Window`
- `Decoration`
- `Portlet Content`

While we want to leave it open to you to decide which way to implement your layouts and themes, we strongly believe that the last approach is superior, and allows for far more flexibility, and clearer separation of duties between portal developers and web designers.

Portal layouts also have the concept of a layout strategy. The layout strategy is a pluggable API, and lets the layout developer have a last say about the content to be rendered. The strategy is called right after the portal has determined what needs to be rendered as part of the current request. So the strategy is invoked right between the point where the portal knows what needs to be done, and before the actual work is initiated. The strategy gets all the details about what is going to happen, and it can take measures to influence those details.

Some simple examples of those measures are:

- ommit one of the portlets from being rendered
- change the portlet mode or window state of a portlet before it gets rendered
- change the layout to be used for this request
- ...and many more

The last topic to introduce in this overview is the one of portal themes. A theme is a collection of web design artifacts. It defines a set of css, java script and image files that together decide about the look and feel of the portal page. The theme can take a wide spectrum of control over the look and feel. It can limit itself to decide fonts and colors, or it can take over a lot more and decide the placement (location) of portlets and much more.

## 7.2. Layouts

### 7.2.1. How to define a Layout

Layouts are used by the portal to produce the actual markup of a portal response. After all the portlets on a page have been rendered and have produced their markup fragments, the layout is responsible for aggregating all these pieces, mix them with some ingredients from the portal itself, and at the end write the response back to the requesting client.

Layouts can be either a JSP or a Servlet. The portal determines the layout to use via the configured properties of the portal, or the requested page. Both, portal and pages, can define the layout to use in order to render their content. In case both define a layout, the layout defined for the page will overwrite the one defined for the portal.

A Layout is defined in the layout descriptor named `portal-layouts.xml`. This descriptor must be part of the portal application, and is picked up by the layout deployer. If the layout deployer detects such a descriptor in a web application, it will parse the content and register the layouts with the layout service of the portal. Here is an example of such a descriptor file:

```
<layouts>
  <layout>
    <name>phalanx</name>
    <uri>/phalanx/index.jsp</uri>
  </layout>
  <layout>
    <name>industrial</name>
    <uri>/industrial/index.jsp</uri>
    <uri state="maximized">/industrial/maximized.jsp</uri>
  </layout>
</layouts>
```

## 7.2.2. How to use a Layout

### 7.2.2.1. Declarative use

Portals and pages can be configured to use a particular layout. The connection to the desired layout is made in the portal descriptor (`YourNameHere-object.xml`). Here is an example of such a portal descriptor:

```
<portal>
  <portal-name>default</portal-name>
  <properties>
    <!-- Set the layout for the default portal -->
    <!-- see also portal-layouts.xml -->
    <property>
      <name>layout.id</name>
      <value>phalanx</value>
    </property>
  </properties>
  <pages>
    <page>
      <page-name>theme test</page-name>
      <properties>
        <!-- set a difference layout for this page -->
        <property>
          <name>layout.id</name>
          <value>industrial</value>
        </property>
      </properties>
    </page>
  </pages>
</portal>
```

The name specified for the layout to use has to match one of the names defined in the `portal-layouts.xml` descriptor of one of the deployed applications.

As you can see, the portal or page property points to the layout to use via the name of the layout. The name has

been given to the layout in the layout descriptor. It is in that layout descriptor where the name gets linked to the physical resource (the JSP or Servlet) that will actually render the layout.

#### 7.2.2.2. Programatic use

To access a layout from code, you need to get a reference to the `LayoutService` interface. The layout service is an mbean that allows access to the `PortalLayout` interface for each layout that was defined in a portal layout descriptor. As a layout developer you should never have to deal with the layout service directly. Your layout hooks are the portal and page properties to configure the layout, and the layout strategy, where you can change the layout to use for the current request, before the actual render process begins.

### 7.2.3. Where to place the Descriptor files

Both descriptors, the portal and the theme descriptor, are located in the `WEB-INF/` folder of the deployed portal application. Note that this is not limited to the `portal-core.war`, but can be added to any WAR that you deploy to the same server. The Portal runtime will detect the deployed application and introspect the `WEB-INF` folder for known descriptors like the two metioned here. If present, the appropriate meta data is formed and added to the portal runtime. From that time on the resources in that application are available to be used by the portal. This is an elegant way to dynamically add new layouts or themes to the portal without having to bring down , or even rebuild the core portal itself.

### 7.2.4. How to connect a Layout to a Layout Strategy

As you might have noticed already, a layout definition consists of a name and one or more uri elements. We have already seen the function of the name element. Now let's take a closer look at the uri element. In the example above, the phalanx layout defined one uri element only, the industrial layout defines two. What you can see in the industrial layout is the option of defining different uri's for different states. In this example , we configured the layout to use a different JSP if the layout state is maximized. If no such separation is made in the layout descriptor, then the portal will always use the same JSP for this layout. Note that the 'state' attribute value works together with the state that was set by the layout strategy. Please refere to the section about the layout strategy for more details.

### 7.2.5. Layout JSP-tags

The portal comes with a set of JSP tags that allow the layout developer faster development.

There are currently two taglibs, containing tags for different approaches to layouts:

- `portal-layout.tld`
- `theme-basic-lib.tld`

The `theme-basic-lib.tld` contains a list of tags that allow a JSP writer to access the state of the rendered page content. It is built on the assumption that regions, portlet windows and portlet decoration is managed inside the JSP.

The `portal-layout.tld` contains tags that work under the assumption that the `RenderSet` will take care of how regions, portlet windows and the portlet decoration will be rendered. The advantage of this approach is that the resulting JSP is much simpler and easier to read and maintain.

Here is an example layout JSP that uses tags from the latter:

```
<%@ taglib uri="/WEB-INF/theme/portal-layout.tld" prefix="p" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1"
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>JBoss Portal: 2.2 early (Industrial)</title>
<meta http-equiv="Content-Type" content="text/html;" />
<p:theme themeName='phalanx' />
<p:headerContent />
</head>
<body id="body">
<div id="portal-container">
<div id="sizer">
<div id="expander">
<div id="logoName"></div>
<table border="0" cellpadding="0" cellspacing="0" id="header-container">
<tr>
<td align="center" valign="top" id="header"><div id="spacer"></div></td>
</tr>
</table>
<div id="content-container">
<p:region regionName='This-Is-The-Page-Region-To-Query-The-Page'
regionID='This-Is-The-Tag-ID-Attribute-To-Match-The-CSS-Selector' />
<p:region regionName='left' regionID='regionA' />
<p:region regionName='center' regionID='regionB' />
<hr class="cleaner" />
<div id="footer-container" class="portal-copyright">Powered by
<a class="portal-copyright" href="http://www.jboss.com/products/jbossportal">JBoss Portal</a>
Theme by <a class="portal-copyright" href="http://www.novell.com">Novell</a>
</div>
</div>
</div>
</div>
</div>
</body>
</html>
```

## 1. The theme tag

The theme tag looks for the determined theme of the current request (see Portal Themes for more details). If no theme was determined, this tag allows an optional attribute 'themeName' that can be used to specify a default theme to use as a last resort. Based on the determined theme name, the ThemeService is called to lookup the theme with this name and to get the resources associated with this theme. The resulting style and link elements are injected, making sure that war context URLs are resolved appropriately.

## 2. The headerContent tag

This tag allows portlets to inject content into the header. More details about this function are mentioned in the 'Other Theme Functions' section of this document.

## 3. The region tag

The region tag renders all the portlets in the specified region of the current page, using the determined RenderSet to

produce the markup that surrounds the individual portlet markup fragments. The `regionName` attribute functions as a query param into the current page. It determines from what page region the portlets will be rendered in this tag. The `regionID` attribute is what the `RenderSet` can use to generate a css selector for this particular region. In case of the `divRenderer`, a `DIV` tag with an `id` attribute corresponding to the provided value will be rendered for this region. This `id` in turn can be picked up by the CSS to style the region.

## 7.3. Layout Strategy

### 7.3.1. What is a Layout Strategy

The layout strategy is a pluggable API that allows the layout developer to influence the content of the page that is about to be rendered. Based on the current request URL, the portal determined the portal and page that needs to be rendered. The page contains a list of portlets, and those portlets are in a particular navigational state. The navigational state consists of the portlet mode and the window state of the portlet. This information, together with the determined layout, the region and order assignments of each portlet, the allowed window states and portlet modes for both, the portal and the individual portlets, is passed to the layout strategy before the actual rendering is invoked. The layout strategy can check what is about to be rendered, and take action in a limited way to influence the content that is about to be rendered.

### 7.3.2. How can I use a Layout Strategy

#### 7.3.2.1. Define a Strategy

A layout strategy is defined in the strategy descriptor. The descriptor is named `portal-strategies.xml`, and is located in the `WEB-INF/layout` folder of any web application deployed to the server. Here is an example of such a descriptor:

```
<portal-strategies>
  <set name="default">
    <strategy content-type="text/html">
      <implementation>org.jboss.portal.theme.impl.strategy.DefaultStrategyImpl</implementation>
    </strategy>
  </set>
  <set name="maximizedRegion">
    <strategy content-type="text/html">
      <implementation>org.jboss.portal.theme.impl.strategy.MaximizingStrategyImpl</implementation>
    </strategy>
  </set>
</portal-strategies>
```

Layout strategies are defined as sets. A set consists of one or more strategy definitions, separated by content type they are assigned for. The idea behind this is to allow the layout developer to apply different strategies based on requested content type. Each set has a name that is unique in the application context it is deployed in. The strategy can be referred to by this name. As a result of that it is considered a named layout strategy in contrast to an anonymous strategy as described below.

#### 7.3.2.2. Specify the Strategy to use

The strategy that will be used for a portal request is defined as a property of the current layout, the requested portal, or the requested page. If the layout defines a strategy to use it will overwrite all other assignments. If there is no particular strategy defined for the layout, then the page property will overwrite the portal property. If no strategy can be determined, then a last attempt will be made to use the strategy with the name 'default'. If no strategy can be determined at all, the request will proceed normally without invoking any strategy. Here is an example layout descriptor that defines a strategy for the layouts defined:

```
<layouts>
  <strategy content-type="text/html">
    <implementation>com.novell.portal.strategy.MaximizingStrategy</implementation>
  </strategy>

  <layout>
    <name>generic</name>
    <uri>/generic/index.jsp</uri>
    <uri state="maximized">/generic/maximized.jsp</uri>
  </layout>
</layouts>
```

In this case the strategy is anonymous and directly assigned to the generic layout. The strategy cannot be discovered independently from the generic layout. Here is an example portal descriptor that points to a strategy for the portal, and for a particular page:

```
<portal>
  <portal-name>default</portal-name>
  <properties>
    <property>
      <name>layout.strategyId</name>
      <value>default</value>
    </property>
  </properties>
  <pages>
    <default-page>theme test</default-page>
    <page>
      <page-name>theme test</page-name>
      <properties>
        <!-- set a difference layout strategy for this page -->
        <property>
          <name>layout.strategyId</name>
          <value>maximizedRegion</value>
        </property>
      </properties>
      <window>
        <window-name>CatalogPortletWindow</window-name>
        <instance-ref>CatalogPortletInstance</instance-ref>
        <region>left</region>
        <height>0</height>
      </window>
    </page>
  </pages>
</portal>
```

As you can see, analogous to how layouts are referred to, the strategy name is the linking element between the portal descriptor and the layout strategy descriptor. The content type is determined at runtime, and serves as a secondary query parameter to get the correct strategy for this content type out of the set that matches the name provided in the portal descriptor.

### 7.3.3. Linking the Strategy and the Layout

As mentioned above, the layout descriptor can link a strategy directly to the layout. This will overwrite all other defined strategies for the portal or the page, for any page that uses this layout.

The layout strategy can set a state to return to the portal as a result of the strategy evaluation. This state will be matched with the state attribute of the uri element of the layout. If there is a match, then the uri that matches this state will be used as the layout for the current request. So, if the strategy sets a state of 'maximized', the portal will try to use the layout resource that is pointed to for that particular state in the currently selected layout. As you might remember from the previous layout section, a layout can point to another JSP or Servlet based on the state attribute of the uri element, like so:

```
<layouts>
<layout>
<name>industrial</name>
<uri>/industrial/index.jsp</uri>
<uri state="maximized">/industrial/maximized.jsp</uri>
</layout>
</layouts>
```

In this case all requests that don't return a state 'maximized' from the evaluation of the strategy will use the /industrial/index.jsp as the layout. However, if the evaluation of the strategy returns a state of 'maximized' then the request will use /industrial/maximized.jsp as the layout.

## 7.4. RenderSets

### 7.4.1. What is a RenderSet

A RenderSet can be used to produce the markup containers around portlets and portlet regions. The markup for each region, and each portlet window in a region is identical. Further more, it is most likely identical across several layouts. The way portlets are arranged and decorated will most likely not change across layouts. What will change is the look and feel of the decoration, the images, fonts, and colors used to render each portlet window on the page. This is clearly a task for the web designer, and hence should be realized via the portal theme. The layout only needs to provide enough information to the theme so that it can do its job. The RenderSet is exactly that link between the layout and the theme that takes the information available in the portal and renders markup containing the current state of the page and each portlet on it. It makes sure that the markup around each region and portlet contains the selectors that the theme css needs to style the page content appropriately.

A RenderSet consists of the implementations of four interfaces. Each of those interfaces corresponds to a markup container on the page.

Here are the four markup containers and their interface representation:

- Region - RegionRenderer
- Window - WindowRenderer
- Decoration - DecorationRenderer



- Portlet Content - PortletRenderer

All the renderer interfaces are specified in the `org.jboss.portal.theme.render` package.

The four markup containers are hierarchical. The region contains one or more windows. A window contains the portlet decoration and the portlet content.

The region is responsible for arranging the positioning and order of each portlet window. Should they be arranged in a row or a column? If there are more than one portlet window in a region, in what order should they appear?

The window is responsible for placing the window decoration, including the portlet title, over the portlet content, or under, or next to it.

The decoration is responsible for inserting the correct markup with the links to the portlet modes and window states currently available for each portlet.

The portlet content is responsible for inserting the actually rendered markup fragment that was produced by the portlet itself.

## 7.4.2. How is a RenderSet defined

Similar to layouts, render sets must be defined in a RenderSet descriptor. The RenderSet descriptor is located in the WEB-INF/layout folder of a web application, and is named `portal-renderSet.xml`. Here is an example descriptor:

```
<?xml version="1.0" encoding="UTF-8"?>
<portal-renderSet>
  <renderSet name="divRenderer">
    <set content-type="text/html">
      <region-renderer>org.jboss.portal.theme.impl.render.DivRegionRenderer</region-renderer>
      <window-renderer>org.jboss.portal.theme.impl.render.DivWindowRenderer</window-renderer>
      <portlet-renderer>org.jboss.portal.theme.impl.render.DivPortletRenderer</portlet-renderer>
      <decoration-renderer>org.jboss.portal.theme.impl.render.DivDecorationRenderer</decoration-renderer>
    </set>
  </renderSet>
  <renderSet name="emptyRenderer">
    <set content-type="text/html">
      <region-renderer>org.jboss.portal.theme.impl.render.EmptyRegionRenderer</region-renderer>
      <window-renderer>org.jboss.portal.theme.impl.render.EmptyWindowRenderer</window-renderer>
      <portlet-renderer>org.jboss.portal.theme.impl.render.EmptyPortletRenderer</portlet-renderer>
      <decoration-renderer>org.jboss.portal.theme.impl.render.EmptyDecorationRenderer</decoration-renderer>
    </set>
  </renderSet>
</portal-renderSet>
```

## 7.4.3. How to specify what RenderSet to use

Analogous to how a strategy is specified, the RenderSet can be specified as a portal or page property, or a particular layout can specify an anonymous RenderSet to use. Here is an example of a portal descriptor:

```
<?xml version="1.0" encoding="UTF-8"?>
<portal>
  <portal-name>default</portal-name>
```

```

<properties>
<!-- use the divRenderer for this portal -->
<property>
<name>theme.renderSetId</name>
<value>divRenderer</value>
</property>
</properties>
<pages>
<default-page>default</default-page>
<page>
<page-name>default</page-name>
<properties>
<!-- overwrite the portal's renderset for this page -->
<property>
<name>theme.renderSetId</name>
<value>emptyRenderer</value>
</property>
</properties>
<window>
<window-name>TestPortletWindow</window-name>
<instance-ref>TestPortletInstance</instance-ref>
<region>center</region>
<height>0</height>
</window>
</page>
</pages>
</portal>

```

Here is an example of a layout descriptor with an anonymous RenderSet:

```

<?xml version="1.0" encoding="UTF-8"?>
<layouts>
<renderSet>
<set content-type="text/html">
<region-renderer>org.foo.theme.render.MyRegionRenderer</region-renderer>
<window-renderer>org.foo.theme.render.MyWindowRenderer</window-renderer>
<portlet-renderer>org.foo.theme.render.MyPortletRenderer</portlet-renderer>
<decoration-renderer>org.foo.theme.render.MyDecorationRenderer</decoration-renderer>
</set>
</renderSet>
<layout>
<name>generic</name>
<uri>/generic/index.jsp</uri>
<uri state="maximized">/generic/maximized.jsp</uri>
</layout>
</layouts>

```

Again, analogous to layout strategies, the anonymous RenderSet overwrites the one specified for the page, and that overwrites the one specified for the portal. In other words: all pages that use the layout that defines an anonymous RenderSet will use that RenderSet, and ignore what is defined as RenderSet for the portal or the page.

In addition to specifying the renderSet for a portal or a page, each individual portlet window can define what renderSet to use for the one of the three aspects of a window, the window renderer, the decoration renderer, and the portlet renderer. This feature allow you to use the the window renderer implementation from one renderSet, and the decoration renderer from another. Here is an example for a window that uses the implementations of the emptyRenderer renderSet for all three aspects:

```

<window>
  <window-name>NavigationPortletWindow</window-name>
  <instance-ref>NavigationPortletInstance</instance-ref>
  <region>navigation</region>
  <height>0</height>
  <!-- overwrite portal and page properties set for the renderSet for this window -->
  <properties>
    <!-- use the window renderer from the emptyRenderer renderSet -->
    <property>
      <name>theme.windowRendererId</name>
      <value>emptyRenderer</value>
    </property>
    <!-- use the decoration renderer from the emptyRenderer renderSet -->
    <property>
      <name>theme.decorationRendererId</name>
      <value>emptyRenderer</value>
    </property>
    <!-- use the portlet renderer from the emptyRenderer renderSet -->
    <property>
      <name>theme.portletRendererId</name>
      <value>emptyRenderer</value>
    </property>
  </properties>
</window>

```

## 7.5. Themes

### 7.5.1. What is a Theme

A portal theme is a collection of CSS styles, JavaScript files, and images, that all work together to style and enhance the rendered markup of the portal page. The theme works together with the layout and the RenderSet in producing the content and final look and feel of the portal response. Through clean separation of markup and styles a much more flexible and powerful approach to theming portals is possible. While this approach is not enforced, it is strongly encouraged. If you follow the definitions of the ThemeStyleGuide (see later), it is not necessary to change the layout or the strategy, or the RenderSet to achieve very different look and feels for the portal. All you need to change is the theme. Since the theme has no binary dependencies, it is very simple to swap it, or change individual items of it. No compile or redeploy is necessary. Themes can be added or removed while the portal is active. Themes can be deployed in separate web applications furthering even more the flexibility of this approach. Web developers don't have to work with JSPs. They can stay in their favorite design tool and simply work against the exploded war content that is deployed into the portal. The results can be validated live in the portal.

### 7.5.2. How to define a Theme

Themes can be added as part of any web application that is deployed to the portal server. All that is needed is a theme descriptor file that is part of the deployed archive. This descriptor indicates to the portal what themes and theme resources are becoming available to the portal. The theme deployer scans the descriptor and adds the theme(s) to the ThemeService, which in turn makes the themes available for consumption by the portal. Here is an example of a theme descriptor:

```

<themes>
  <theme>

```

```

<name>nodesk</name>
<link href="/nodesk/css/portal_style.css" rel="stylesheet" type="text/css" />
<link rel="shortcut icon" href="/images/favicon.ico" />
</theme>
<theme>
<name>phalanx</name>
<link href="/phalanx/css/portal_style.css" rel="stylesheet" type="text/css" />
<link rel="shortcut icon" href="/images/favicon.ico" />
</theme>

<theme>
<name>industrial-CSSSelect</name>
<link rel="stylesheet" id="main_css" href="/industrial/portal_style.css" type="text/css" />
<link rel="shortcut icon" href="/industrial/images/favicon.ico" />

<script language="JavaScript" type="text/javascript">
// MAF - script to switch current tab and css in layout...
function switchCss(currentTab,colNum) {
var obj = currentTab;
var objParent = obj.parentNode;

if (document.getElementById("current") != null) {
var o = document.getElementById("current");
o.setAttribute("id","");
o.className = 'hoverOff';
objParent.setAttribute("id","current");
}

var css = document.getElementById("main_css");
source = css.href;
if (colNum == "3Col") {
if (source.indexOf("portal_style.css" != -1)) {
source = source.replace("portal_style.css","portal_style_3Col.css");
}
if (source.indexOf("portal_style_1Col.css" != -1)) {
source = source.replace("portal_style_1Col.css","portal_style_3Col.css");
}
}
if (colNum == "2Col") {
if (source.indexOf("portal_style_3Col.css" != -1)) {
source = source.replace("portal_style_3Col.css","portal_style.css");
}
if (source.indexOf("portal_style_1Col.css" != -1)) {
source = source.replace("portal_style_1Col.css","portal_style.css");
}
}
if (colNum == "1Col") {
if (source.indexOf("portal_style_3Col.css" != -1)) {
source = source.replace("portal_style_3Col.css","portal_style_1Col.css");
}
if (source.indexOf("portal_style.css" != -1)) {
source = source.replace("portal_style.css","portal_style_1Col.css");
}
}

css.href = source;
}
</script>
</theme>
</themes>

```

Themes are defined in the portal-themes.xml theme descriptor, which is located in the WEB-INF/ folder of the web application.

### 7.5.3. How to use a Theme

Again, analogous to the way it is done for layouts, themes are specified in the portal descriptor as a portal or page property. The page property overwrites the portal property. In addition to these two options, themes can also be specified as part of the theme JSP tag , that is placed on the layout JSP. Here is an example portal descriptor that specifies the phalanx theme as the theme for the entire portal, and the industrial theme for the theme test page:

```
<portal>
<portal-name>default</portal-name>
<properties>
<!-- Set the theme for the default portal -->
<property>
<name>layout.id</name>
<value>phalanx</value>
</property>
</properties>
<pages>
<page>
<page-name>theme test</page-name>
<properties>
<!-- set a difference layout for this page -->
<property>
<name>layout.id</name>
<value>industrial</value>
</property>
</properties>
<window>
<window-name>CatalogPortletWindow</window-name>
<instance-ref>CatalogPortletInstance</instance-ref>
<region>left</region>
<height>0</height>
</window>
</page>
</pages>
</portal>
```

And here is an example of a layout JSP that defines a default theme to use if no other theme was defined for the portal or page:

```
<%@ taglib uri="/WEB-INF/theme/portal-layout.tld" prefix="p" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1"
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title><%= "JBoss Portal :: 2.2 early (Industrial)" %></title>
<meta http-equiv="Content-Type" content="text/html;" />
<p:theme themeName='industrial' />
<p:headerContent />
</head>
<body id="body">
<div id="portal-container">
<div id="sizer">
<div id="expander">
<div id="logoName"></div>
<table border="0" cellpadding="0" cellspacing="0" id="header-container">
<tr>
<td align="center" valign="top" id="header"><div id="spacer"></div></td>
</tr>
</table>
```

```

<div id="content-container">
  <p:region regionName='This-Is-The-Page-Region-To-Query-The-Page'
    regionID='This-Is-The-Tag-ID-Attribute-To-Match-The-CSS-Selector' />
  <p:region regionName='left' regionID='regionA' />
  <p:region regionName='center' regionID='regionB' />
  <hr class="cleaner" />
  <div id="footer-container" class="portal-copyright">Powered by
  <a class="portal-copyright" href="http://www.jboss.com/products/jbossportal">JBoss Portal<
  Theme by <a class="portal-copyright" href="http://www.novell.com">Novell</a>
</div>
</div>
</div>
</div>
</div>
</body>
</html>

```

For the function of the individual tags in this example, please refer to the layout section of this document.

### 7.5.4. How to write your own Theme

Ask your favorite web designer and/or consult the ThemeStyleGuide in this document ;)

## 7.6. Other Theme Functionalities and Features

This section contains all the functionalities that don't fit with any of the other topics. Bits and pieces of useful functions that are related to the theme and layout functionality.

### 7.6.1. Content Rewriting and Header Content Injection

Portlets can have their content rewritten by the portal. This is useful if you want to uniquely namespace markup (JavaScript functions for example) in the scope of a page. The rewrite functionality can be applied to the portlet content (the markup fragment) and to content a portlet wants to inject into the header. The rewrite is implemented as specified in the WSRP (OASIS: Web Services for Remote Portlets; producer write). As a result of this, the token to use for rewrite is the WSRP specified "wsrp\_rewrite\_". If the portlet sets the following response property

```
res.setProperty("WSRP_REWRITE", "true");
```

all occurrences of the wsrp\_rewrite\_ token in the portlet fragment will be replaced with a unique token (the window id). If the portlet also specifies content to be injected into the header of the page, that content is also subject to this rewrite.

```

res.setProperty("HEADER_CONTENT", "
<script>function wsrp_rewrite_OnFocus(){alert('hello button');}</script>
");

```

Note that in order for the header content injection to work, the layout needs to make use of the headerContent JSP tag, like:

```

<%@ taglib uri="/WEB-INF/theme/portal-layout.tld" prefix="p" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
<html xmlns="http://www.w3.org/1999/xhtml">

```

```

<head>
<title><JBoss Portal 2.2 early</title>
<meta http-equiv="Content-Type" content="text/html;" />

<p:headerContent />

</head>
<body id="body">
<p>...</p>
</body>
</html>

```

## 7.6.2. Declarative CSS Style injection

If a portlet needs a CSS style sheet to be injected via a link tag in the page header, it can do so by providing the context relative URI to the file in the jboss-portlet.xml descriptor, like:

```

<portlet-app>
<portlet>
<portlet-name>HeaderContentPortlet</portlet-name>
<header-content>
<link rel="stylesheet" type="text/css" href="/portlet-styles/HeaderContent.css" title="" media="screen" />
</header-content>
</portlet>
</portlet-app>

```

This functionality, just like the previously described header content injection, requires the layout JSP to add the "headerContent" JSP tag (see example above). One thing to note here is the order of the tags. If the headerContent tag is placed after the theme tag, it will allow portlet injected CSS files to overwrite the theme's behaviour, making this feature even more powerful!

## 7.6.3. Disabling Portlet Decoration

One possible use of window properties is demonstrated in the divRenderer RenderSet implementation. If a window definition (in the portal descriptor) contains a property like:

```

<window>
  <window-name>HintPortletWindow</window-name>
  <instance-ref>HintPortletInstance</instance-ref>
  <region>center</region>
  <height>0</height>
  <properties>
    <!-- turn the decoration off for this portlet (i.e. no title and mode/state links) -->
    <property>
      <name>theme.decorationRendererId</name>
      <value>emptyRenderer</value>
    </property>
  </properties>
</window>

```

the DivWindowRenderer will use the decoration renderer from the emptyRenderer RenderSet to render the decoration.

tion for this window (not delegate to the `DivDecorationRenderer`). As a result, the portlet window will be part of the rendered page, but it will not have a title, nor will it have any links to change the portlet mode or window state.

## 7.7. Theme Style Guide (based on the Industrial theme)

### 7.7.1. Overview

This document outlines the different selectors used to handle the layout and look/feel of the Industrial theme included in the JBoss portal.

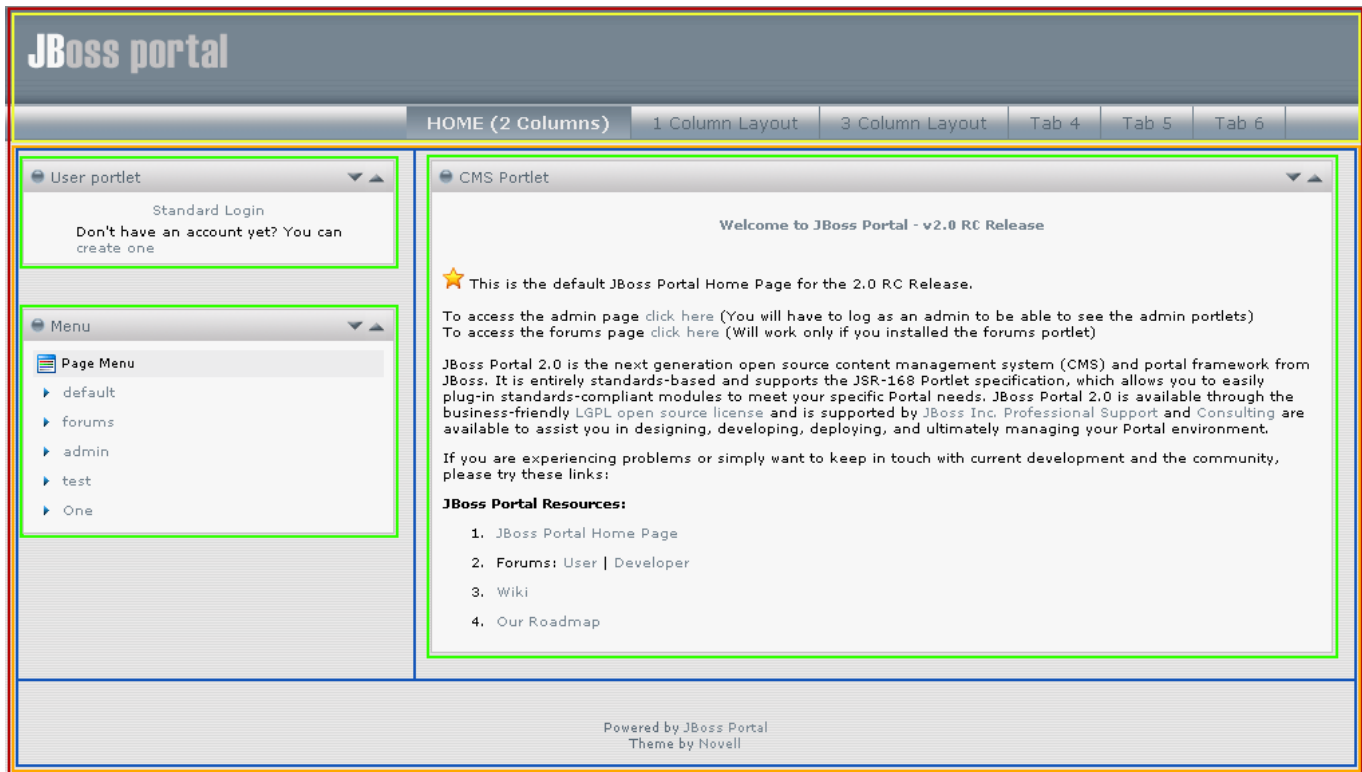
A couple of things to know about the theming approach discussed below:

- Main premise behind this approach was to provide a clean separation between the business and presentation layer of the portal. As we go through each selector and explain the relation to the visual presentation on the page, this will become more apparent.
- The flexibility of the selectors used in the theme stylesheet allow a designer to very easily customize the visual aspects of the portal, thereby taking the responsibility off of the developers hands through allowing the designer to quickly achieve the desired effect w/out the need to dive down into code and/or having to deploy changes to the portal. This saves time and allows both developers and designers to focus on what they do best.
- This theme incorporates a liquid layout approach which allows elements on a page to expand/contract based on screen resolution and provides a consistent look across varying display settings. However, the stylesheet is adaptable to facilitate a fixed layout and/or combination approach where elements are pixel based and completely independent of viewport.
- The pieces that make up the portal theme consist of at least one stylesheet and any associated images. Having a consolidated set of files to control the portal look and feel allows administrators to effortlessly swap themes on the fly. In addition, this clean separation of the pieces that make up a specific theme will enable sharing and collaboration of different themes by those looking to get involved or contribute to the open source initiative.

### 7.7.2. Main Screen Shot

Screen shot using color outline of main ID selectors used to control presentation and layout:





- Red Border - portal-container
- Yellow Border - header-container
- Orange Border - content-container
- Blue Border - regionA/regionB
- Green Border - portlet-container

### 7.7.3. List of CSS Selectors

The following is a list of the selectors used in the theme stylesheet, including a brief explanation of how each selector is used in the portal:

- Portal Body Selector

```
#body {
  background-image: url(images/portal_background.gif);
  margin: 0px;
  padding: 0px;
}
```

Usage: This selector controls the background of the page, and can be modified to set a base font-family, layout margin, etc. that will be inherited by all child elements that do not have their own individual style applied. By default, the selector pulls an image background for the page.

- Portal Header Selectors

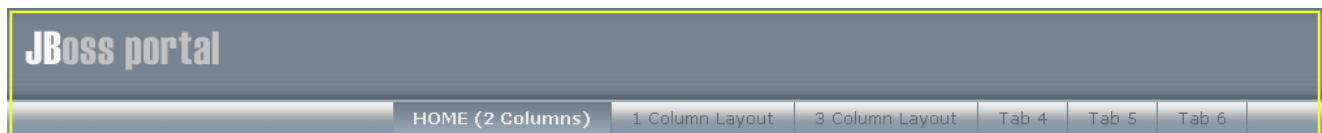
```
#spacer {
width: 1024px;
line-height: 0px;
font-size: 0px;
height: 0px;
}
```

Usage: Spacer div used to keep header at certain width regardless of display size. This is done to avoid overlapping of tab navigation in header. To account for different display sizes, this selector can be modified to force a horizontal scroll in the browser which eliminates any issue with overlapping elements in the header.

```
#header-container {
background-image: url(images/portal_background.gif);
background-repeat: repeat-y;
height: 100%;
min-width: 1000px;
width: 100%;
/* test to reposition header on page
position: absolute;
bottom: 5px;*/
}
```

Usage: Wrapper selector used to control the position of the header on the page (see yellow border in screen shot). This selector is applied as an ID on the table used to structure the header. You can adjust the attributes to reposition the header location on the page and/or create margin space on the top, right, bottom and left sides of the header.

Screenshot:



```
#header {
background-image: url(images/header.gif);
background-repeat: repeat-x;
height: 100px;
padding: 0px;
/*margin: 0 25% 0 25%;*/
}
```

Usage: This selector applies the header background image in the portal. It can be adjusted to accommodate a header background of a certain width/height or, as it currently does, repeat the header graphic so that it tiles across the header portion of the page.

```
#logoName {
background-image: url(images/JBossLogo.gif);
background-repeat: no-repeat;
width: 187px;
height: 35px;
position: absolute;
```

```
left: 15px;
top: 16px;
z-index: 2;
}
```

Usage: Logo selector which is used to brand the header with a specific, customized logo. The style is applied as an ID on an absolutely positioned DIV element which enables it to be moved to any location on the page, and allows it to be adjusted to accommodate a logo of any set width/height.

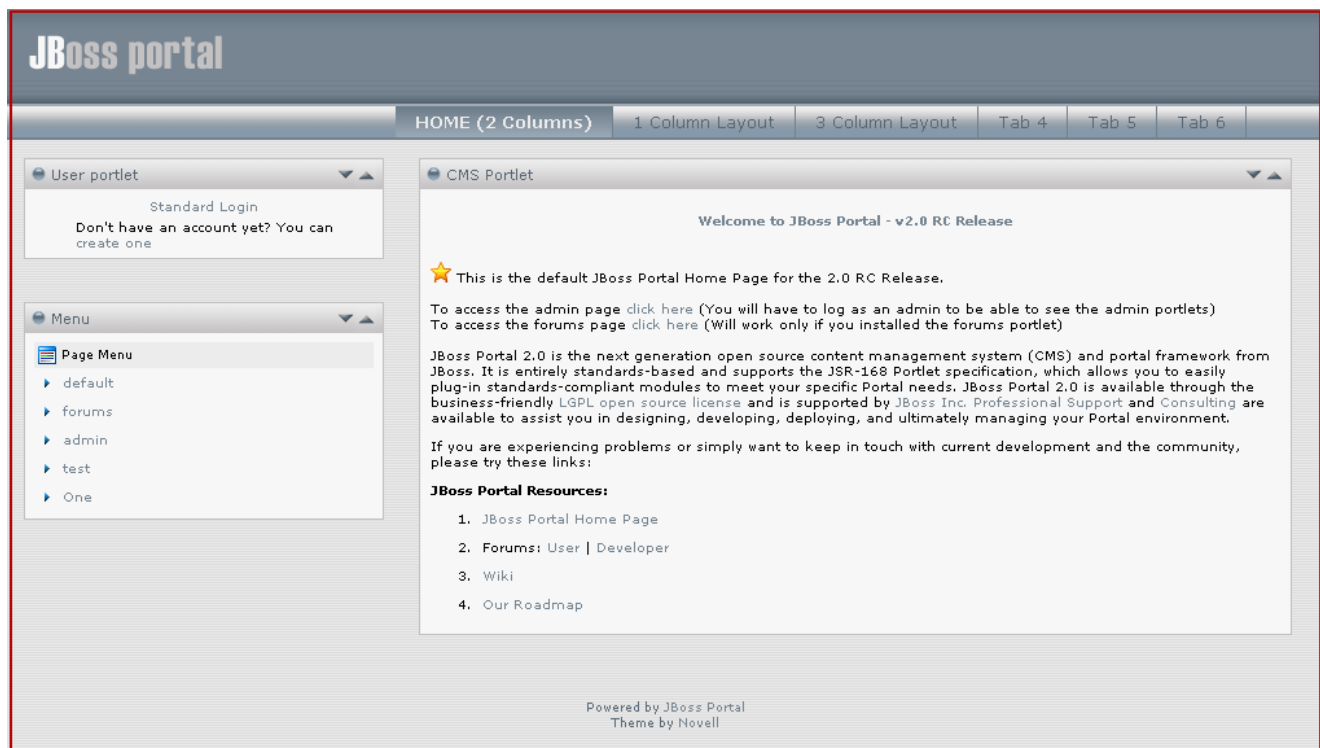
- Portal Layout Region Selectors

```
#portal-container {
/*width: 100%;*/

/*IE specific approach to preserve min-width for portlet regions */
padding: 0 350px 0 350px;
}
```

Usage: Wrapper for entire portal which starts/ends after/before the BODY tag (see red border in screen shot). The padding attribute for this selector is used to preserve a minimum width setting for the portlet regions (discussed below). Similar to body selector, this style can be modified to create margin or padding space on the top, right, bottom and left sections of the page. It provides the design capability to accommodate most layouts (e.g. a centered look such as the phalanx theme where there is some spacing around the content of the portal, or a full width look as illustrated in the Industrial theme).

Screenshot:



```
/* min width for IE */
#expander {
```

```
margin: 0 -350px 0 -350px;
position: relative;
}

/* min width for IE */
#sizer {
width: 100%;
}

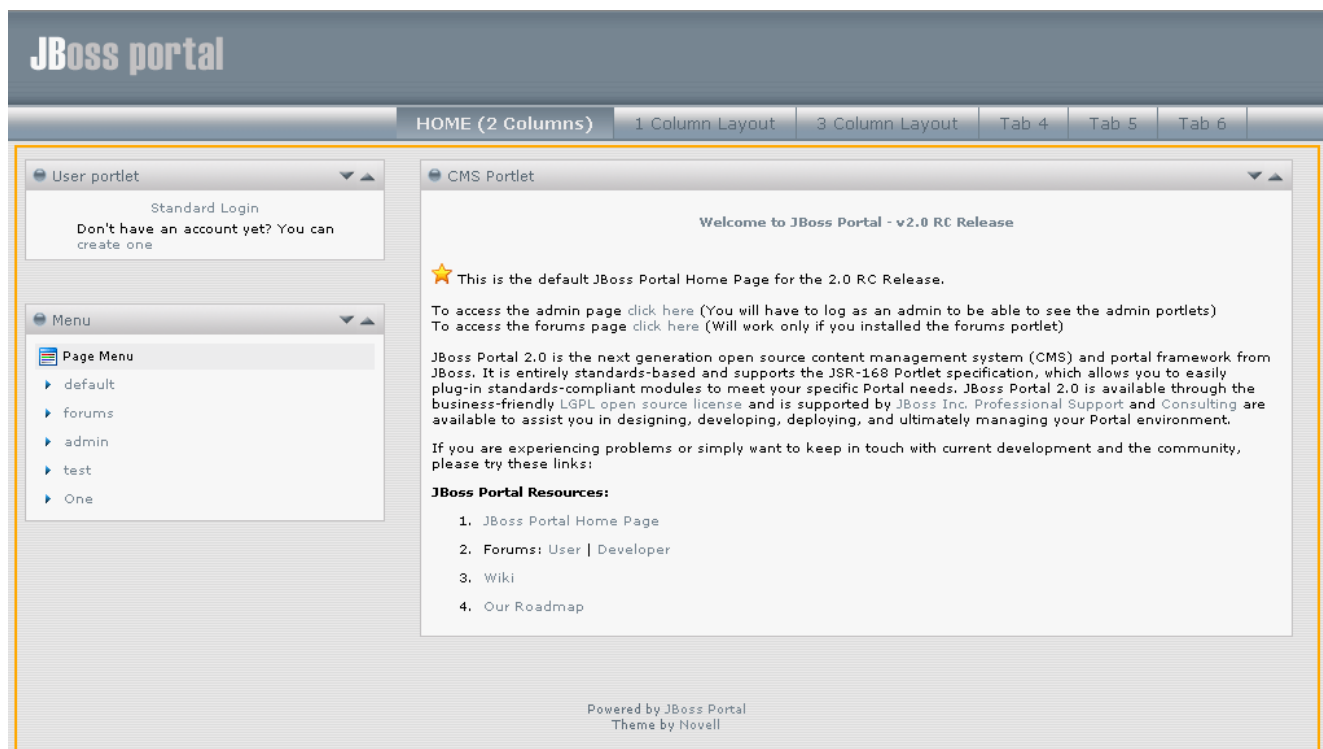
/* IE min width */
* html #portal-container,
* html #sizer,
* html #expander { height: 0; }
```

Usage: These selectors are used in conjunction with the above, `portal-container`, selector to preserve a minimum width setting for the portlet regions. This was implemented to maintain a consistent look across different browsers.

```
/*table that contains all regions. does not include header*/
#content-container {
height: 100%;
text-align:left;
max-width: 1600px;
min-width: 800px;
}
```

Usage: Wrapper that contains all regions in portal with the exception of the header (see orange border in screen shot). Its attributes can be adjusted to create margin space on page, as well as control positioning of the area of the page below the header.

Screenshot:



```
#regionA {
/* test to swap columns with regionB...
float: right; */

width: 30%;
float: left;
margin: 0px;
padding: 0px;
min-width: 250px;
}
```

Usage: First portlet region located within the content-container (see blue border in screen shot). This selector controls the width of the region as well as its location on the page. Designers can very easily reposition this region in the portal (e.g. swap left regionA with right regionB, etc.) by adjusting the attributes of this selector.

```
#regionB {
/*test to swap columns with regionA...
margin: 0 30% 0 0; */

/* two column layout*/
margin: 0 0 0 30%;
padding: 0;
width: 69%;

/* test to add 3rd region in layout...
width: 40%;
float: left;*/
}
```

Usage: Second portlet region located within the content-container (see blue border in screen shot). Similar to regionA, this selector controls the width of the region as well as its location on the page.

```
#regionC {
/* inclusion of 3rd region - comment out for 2 region testing
padding: 0px;
width: 27%;
float: left;*/
display: none;
}
```

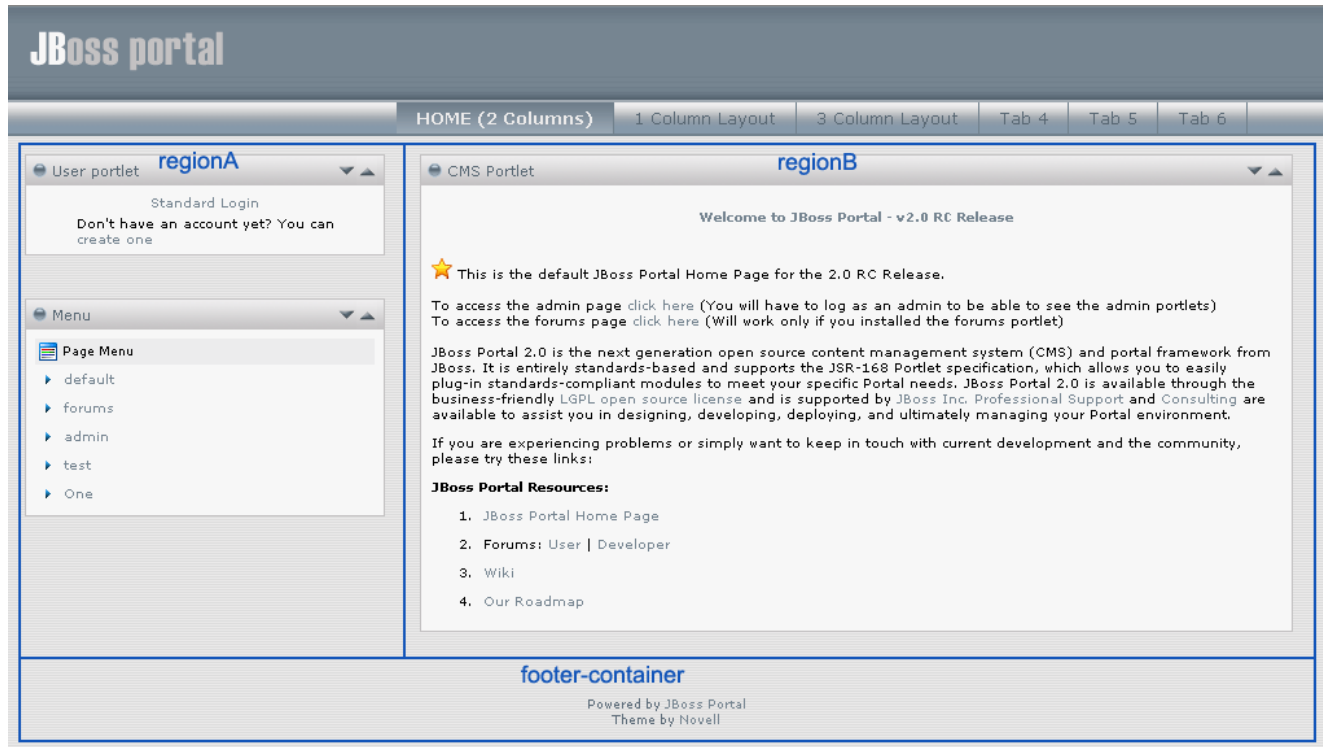
Usage: Third portlet region located within the content-container (please refer to blue border in screen shot representing regionA and regionB for an example). Used for 3 column layout. Similar to regionA and regionB, this selector controls the width of the region as well as its location on the page.

```
/* give a maximized portlet more space */
#regionMaximized {
width: 100%;
float: left;
margin: 0px;
padding: 0px;
min-width: 400px;
}
```

Usage: Portlet region located within the content-container (please refer to blue border in screen shot represent-

ing regionA and regionB for an example). Used for a one column layout to allow one portlet to take over the entire page. Similar to regionA, regionB, and regionB, this selector controls the width of the region as well as its location on the page.

Screenshot:



```
hr.cleaner {
clear:both;
height:1px;
margin: -1px 0 0 0;
padding:0;
border:none;
visibility: hidden;
}
```

Usage: Used to clear floats in regionA, regionB and regionC DIVs so that footer spans bottom of page.

```
#footer-container {
margin: 30px 25% 0 25%;
text-align: center;
}
```

Usage: Footer region located towards the bottom of the content-container (see above screen shot). This region spans the entire width of the page, but can be adjusted (just like regionA, regionB and regionC) to take on a certain position and width/height in the layout.

```
#navigation-container {}
```

Usage: Unused at this time.

```
#sub-navigation-container {}
```

Usage: Unused at this time.

- Tab Navigation Selectors for Header

```
UL#tabsHeader {
margin: 0;
padding-left: 300px;
min-width: 550px;
}
```

Usage: Used to provide position (through padding attribute) of tabbed navigational items in header. A padding-left of 300px gives space for the left hand logo area and can be adjusted as needed to set the desired location for the navigation.

```
UL#tabsHeader li {
list-style: none;
float: left;
margin-left: 0px;
margin-top: 74px;
margin-right: 0px;
line-height: 24px;
padding: 0px;
border-left: 1px solid #72828E;
}
```

Usage: Selector used to style list items as horizontal navigation and to set the spacing and position of each nav item that's available.

```
UL#tabsHeader li:hover {
background-image: url(images/highlightedTab.gif);
background-repeat: repeat-x;
}
```

Usage: Used to provide hover pseudo class on navigation items so that the tab background will change upon mouseover. Note that currently IE only supports the hover pseudo class on links, so this selector will only affect non-IE browsers (e.g. FireFox, etc.).

```
UL#tabsHeader li.hoverOn {
background-image: url(images/highlightedTab.gif);
background-repeat: repeat-x;
}

UL#tabsHeader li.hoverOff {
background-image:none;
}
```

Usage: These two selectors are implemented to account for the fact that IE cannot understand the use of a pseudo class on the LI element. They provide the same mouseover effect as the “UL#tabsHeader li:hover#?” selector when hovering the navigation item in IE, and are used in combination with onmouseover/onmouseout

event handlers in the header navigation:

```
<li onmouseover="this.className='hoverOn'" onmouseout="this.className='hoverOff'">
<a href="#">Tab Nav</a>
</li>
```

```
UL#tabsHeader a {
display: block;
float: left;
padding: 4px 15px 5px 15px;
text-decoration: none;
font: 13px/normal Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;
background: 100% 0 no-repeat;
color: #596874;
}
```

Usage: This selector styles the navigational links, indicating padding surrounding the link as well as font family, color and text-decoration.

```
UL#tabsHeader a:hover {
text-decoration: underline;
}
```

Usage: Used to underline navigational links when hovering with mouse. Unlike the `li:hover` pseudo class, IE does support the hover effect on links, so there is no need for a separate set of selectors to deal with this effect.

```
UL#tabsHeader #current, UL#tabsHeader #current a {
font: 13px/normal Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;
font-weight: 600;
color: #EBEAEA;
background-image: url(images/activeTab.gif);
background-repeat: repeat-x;
border-right: 0px;
border-left: 0px;
}
```

Usage: This selector is set on the current/selected navigation item to style both the background of the tab as well as font properties such as color and weight. Example:

```
<li id="current" onmouseover="this.className='hoverOn'" onmouseout="this.className='hoverOff'">
<a href="#">Tab Nav</a>
</li>
```

```
/* backslash for IE5-Mac */
UL#tabsHeader a {float: none;} /* End Mac Hack */
html>body UL#tabsHeader a {width: auto;} /* fixes IE issues */
```

Usage: Also known in the industry as an example of the “Holly Hack#?”, the above is added to the stylesheet to



handle certain buggy issues with IE. This section of the stylesheet should be left alone as subsequent changes can effect the way things behave in IE.

```
li.currentTabBackground {
background: #fff;
}

li.currentTabBackgroundSubNav {
background: #eeeeef;
}
```

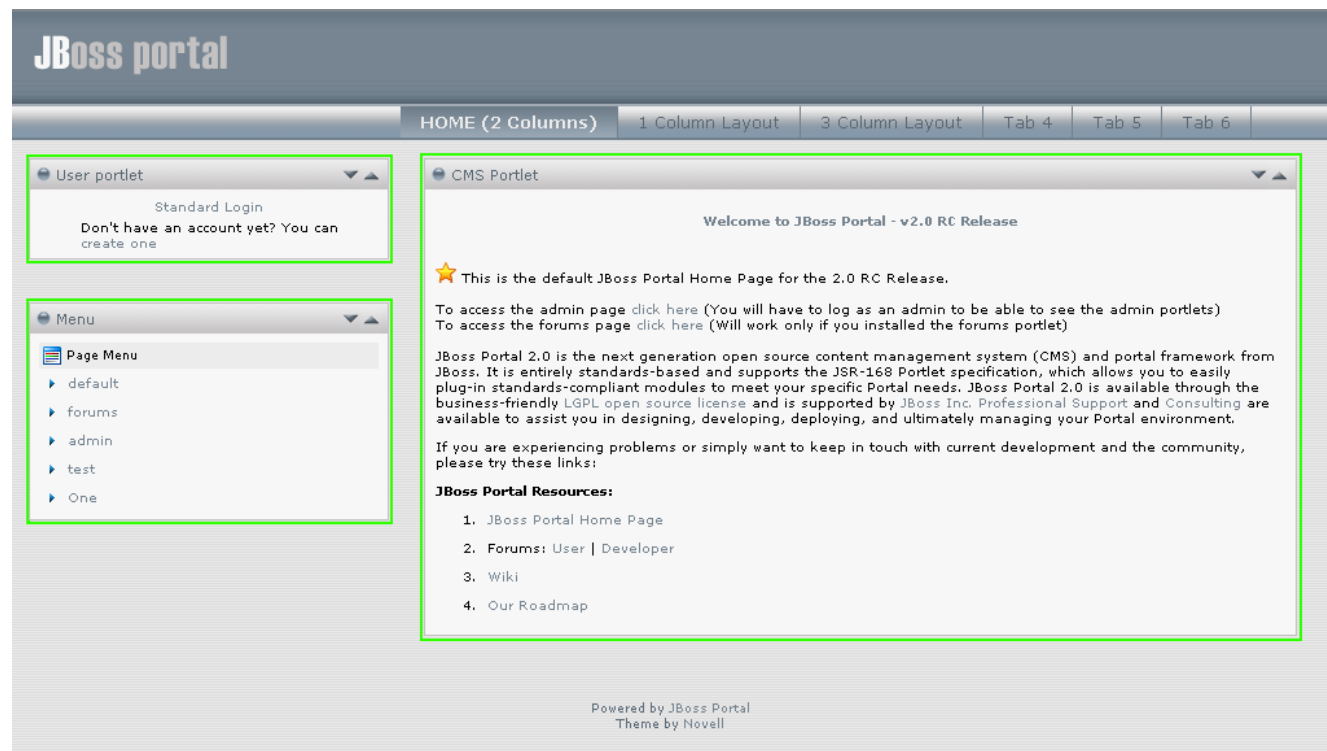
Usage: The above two selectors are not currently in use. Included to account for future changes to the navigation where multiple tiers/levels might be incorporated.

- Portlet Container Window Selectors

```
.portlet-container {
padding: 10px;
}
```

Usage: Wrapper that surrounds the portlet windows (see green border in screen shot). Currently, this selector is used to create space (padding) between the portlets displayed in each particular region.

Screenshot:



```
.portlet-titlebar-title {
font-family: Verdana, Arial, Helvetica, sans-serif;
font-size: 11px;
font-weight: 500;
color: #596874;
}
```

```
white-space: nowrap;
line-height: 100%;
float: left;
text-indent: 15px;
}
```

Usage: Class used to style the title of each portlet window. Attributes of this selector set font properties, indentation and position of title.

```
.portlet-titlebar-decoration {
background-image: url(images/portlet-win-decoration.gif);
background-repeat: no-repeat;
height: 11px;
width: 11px;
float: left;
position: relative;
top: 6px;
}
```

Usage: Used to display top left portlet window decoration (e.g. sphere icon in Industrial theme). Attributes for this selector set position and dimensions of this decoration.

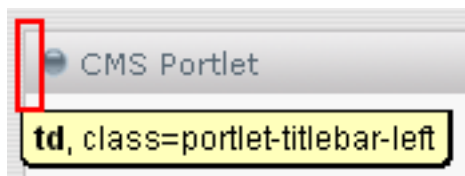
```
.portlet-mode-container {
float: right;
}
```

Usage: Wrapper that contains the portlet window modes that display in the top right section of the portlet windows.

```
.portlet-titlebar-left {
background-image: url(images/portlet-top-left.gif);
background-repeat: no-repeat;
width: 9px;
height: 33px;
background-position: right;
min-width: 9px;
}
```

Usage: Used to style the top left corner of the portlet window. Each portlet window consists of one table that has 3 columns and 3 rows. This selector styles the first column (TD) in the first row (TR).

Screenshot:

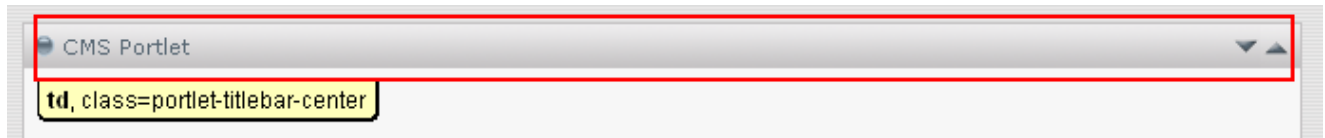


```
.portlet-titlebar-center {
background-image: url(images/portlet-top-middle.gif);
background-repeat: repeat-x;
height: 33px;
}
```

```
}
```

Usage: Used to style the center section of the portlet title bar. Each portlet window consists of one table that has 3 columns and 3 rows. This selector styles the second column (TD) in the first row (TR).

Screenshot:



```
.portlet-titlebar-right {
background-image: url(images/portlet-top-right.gif);
background-repeat: no-repeat;
width: 10px;
height: 33px;
min-width: 10px;
}
```

Usage: Used to style the top right corner of the portlet window. Each portlet window consists of one table that has 3 columns and 3 rows. This selector styles the third column (TD) in the first row (TR).

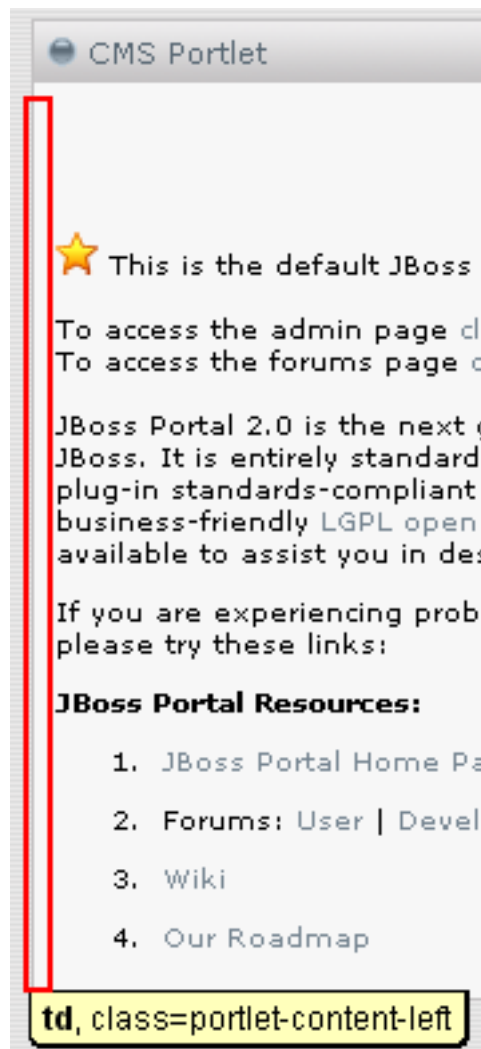
Screenshot:



```
.portlet-content-left {
background-image: url(images/portlet-left-vertical.gif);
height: 100%;
background-repeat: repeat-y;
background-position: right;
width: 9px;
min-width: 9px;
}
```

Usage: Used to style the left hand vertical lines that make up the portlet window. Each portlet window consists of one table that has 3 columns and 3 rows. This selector styles the first column (TD) in the second row (TR).

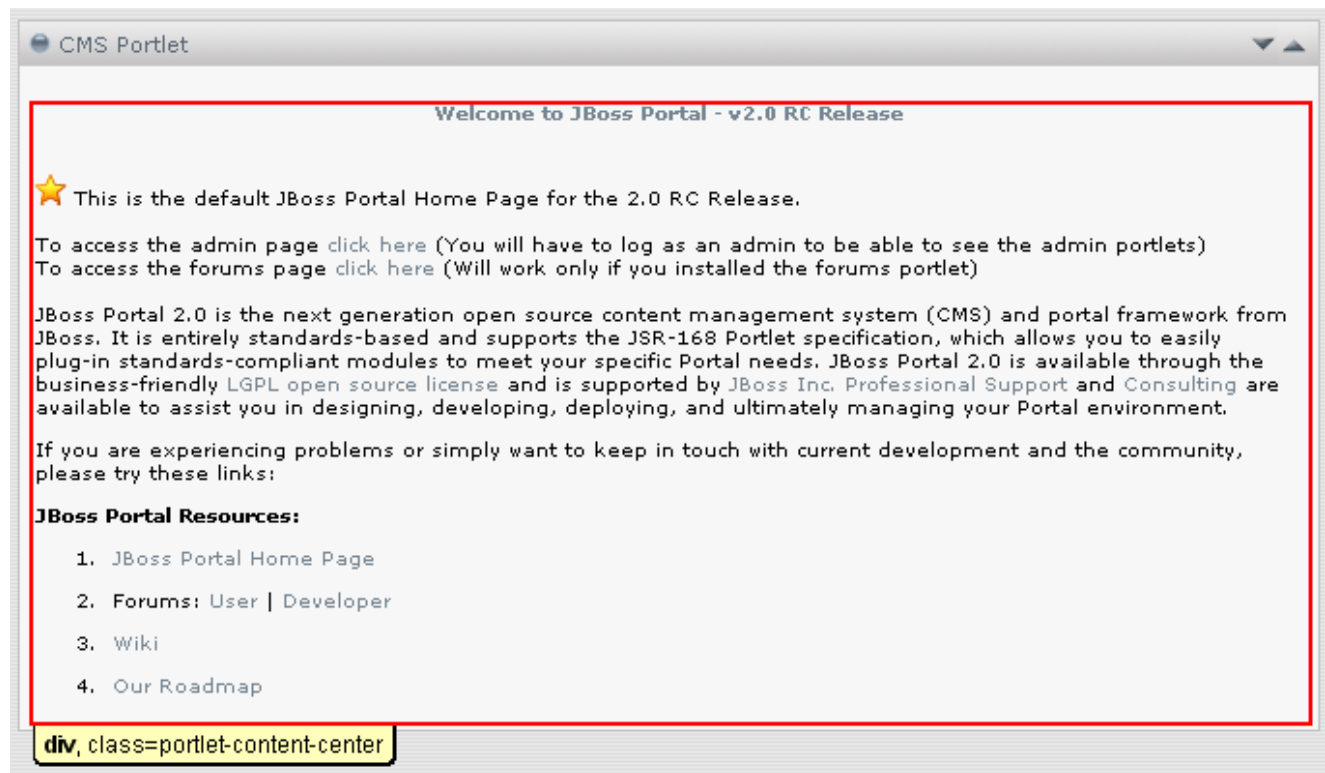
Screenshot:



```
.portlet-content-center {
background-color: #f7f7f7;
background-repeat: repeat;
vertical-align: top;
font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;
font-size: 13px;
}
```

Usage: Used to style the center, content area where the portlet content is injected into the portlet window (see below screen). Attributes for this selector control the positioning of the portlet content as well as the background and font properties. Each portlet window consists of one table that has 3 columns and 3 rows. This selector styles the second column (TD) in the second row (TR).

Screenshot:



```
.portlet-body {  
  background-color: #f7f7f7;  
}
```

Usage: An extra selector for controlling the content section of the portlet windows (see below screen). This was added to better deal with structuring the content that gets inserted/rendered in the portlet windows, specifically if the content is causing display problems in a portlet.

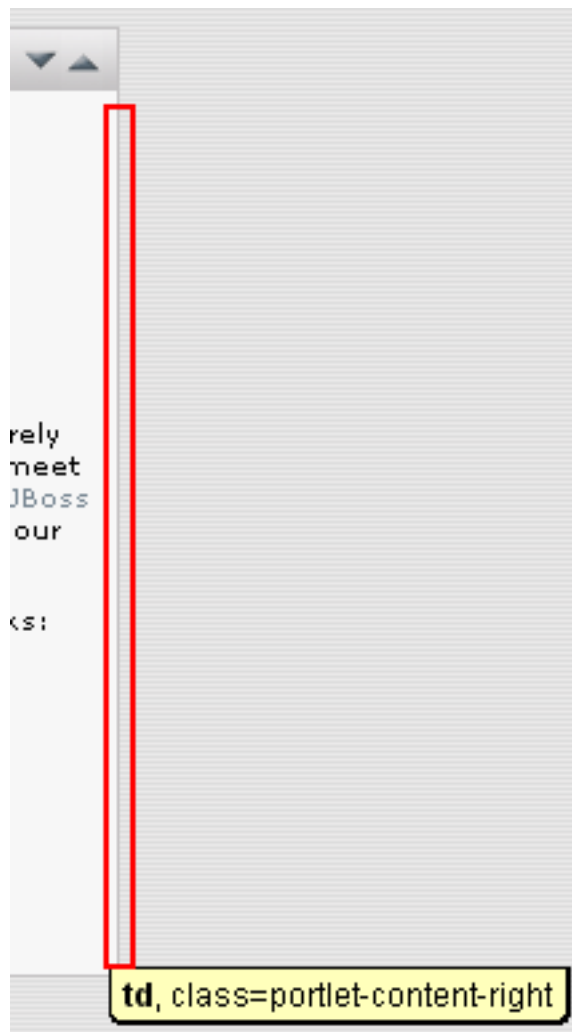
Screenshot:



```
.portlet-content-right {
background-image: url(images/portlet-right-vertical.gif);
height: 100%;
background-repeat: repeat-y;
background-position: left;
width: 10px;
min-width: 10px;
}
```

Usage: Used to style the right hand vertical lines that make up the portlet window. Each portlet window consists of one table that has 3 columns and 3 rows. This selector styles the third column (TD) in the second row (TR).

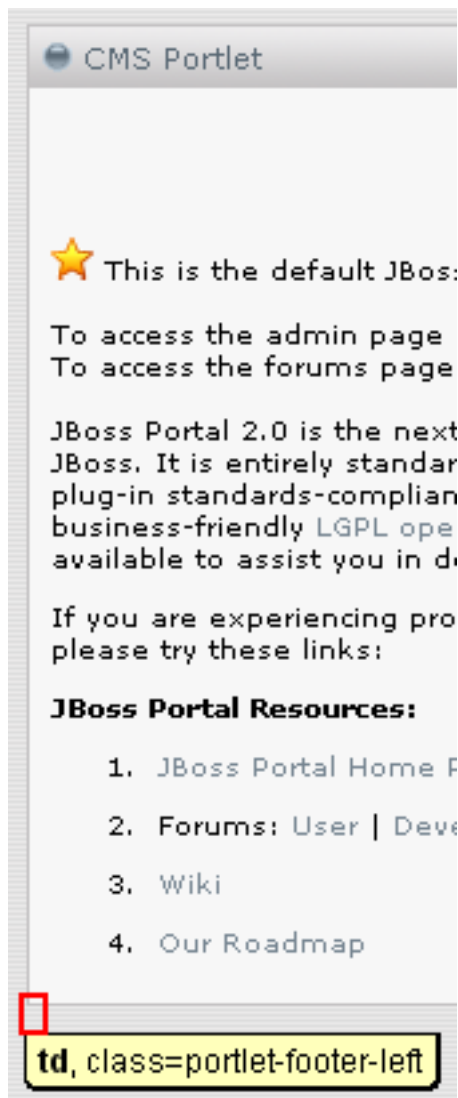
Screenshot:



```
.portlet-footer-left {
background-image: url(images/portlet-bottom-left.gif);
width: 9px;
height: 9px;
background-repeat: no-repeat;
background-position: top right;
min-width: 9px;
}
```

Usage: Used to style the bottom left corner of the portlet window. Each portlet window consists of one table that has 3 columns and 3 rows. This selector styles the first column (TD) in the third row (TR).

Screenshot:



```
.portlet-footer-center {
background-image: url(images/portlet-bottom-middle.gif);
height: 14px;
background-repeat: repeat-x;
}
```

Usage: Used to style the bottom, center of the portlet window (i.e. the bottom horizontal line in the Industrial theme). Each portlet window consists of one table that has 3 columns and 3 rows. This selector styles the second column (TD) in the third row (TR).

Screenshot:



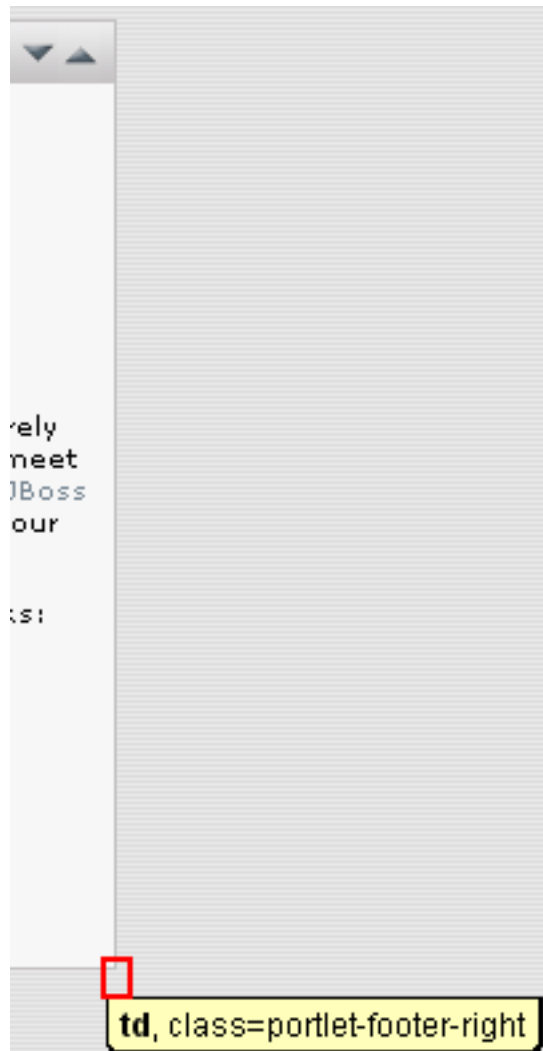
```
.portlet-footer-right {
background-image: url(images/portlet-bottom-right.gif);
width: 10px;
height: 9px;
}
```



```
background-repeat: no-repeat;  
min-width: 10px;  
}
```

Usage: Used to style the bottom right corner of the portlet window. Each portlet window consists of one table that has 3 columns and 3 rows. This selector styles the third column (TD) in the third row (TR).

Screenshot:



- Portlet Window Mode Selectors

```
.portlet-mode-maximized {  
background-image: url(images/maximize.gif);  
width: 16px;  
height: 23px;  
background-repeat: no-repeat;  
float: left;  
display: inline;  
cursor: pointer;  
}
```

Usage: Selector used to display the portlet maximize mode. Attributes for this selector control the display and dimensions of the maximize icon, including the behavior of the mouse pointer when hovering the mode.

```
.portlet-mode-minimized {  
background-image: url(images/minimize.gif);  
width: 16px;  
height: 23px;  
background-repeat: no-repeat;  
float: left;  
display: inline;  
cursor: pointer;  
}
```

Usage: Selector used to display the portlet minimize mode. Attributes for this selector control the display and dimensions of the minimize icon, including the behavior of the mouse pointer when hovering the mode.

```
.portlet-mode-normal {  
background-image: url(images/normal.gif);  
width: 16px;  
height: 23px;  
background-repeat: no-repeat;  
float: left;  
display: inline;  
cursor: pointer;  
}
```

Usage: Selector used to display the portlet normal mode (i.e. the icon that when clicked, restores the portlet to the original, default view). Attributes for this selector control the display and dimensions of the normal icon, including the behavior of the mouse pointer when hovering the mode.

```
.portlet-mode-help {  
background-image: url(images/help.gif);  
width: 16px;  
height: 23px;  
background-repeat: no-repeat;  
float: left;  
display: inline;  
cursor: pointer;  
}
```

Usage: Selector used to display the portlet help mode. Attributes for this selector control the display and dimensions of the help icon, including the behavior of the mouse pointer when hovering the mode.

```
.portlet-mode-edit {  
background-image: url(images/edit.gif);  
width: 16px;  
height: 23px;  
background-repeat: no-repeat;  
float: left;  
display: inline;  
cursor: pointer;  
}
```

Usage: Selector used to display the portlet edit mode. Attributes for this selector control the display and dimensions of the edit icon, including the behavior of the mouse pointer when hovering the mode.

```
.portlet-mode-remove {
background-image: url(images/remove.gif);
width: 16px;
height: 23px;
background-repeat: no-repeat;
float: left;
display: inline;
cursor: pointer;
}
```

Usage: Currently not available. But here is the intended use: Selector used to display the portlet remove mode. Attributes for this selector control the display and dimensions of the remove icon, including the behavior of the mouse pointer when hovering the mode.

```
.portlet-mode-view {
background-image: url(images/view.gif);
width: 16px;
height: 23px;
background-repeat: no-repeat;
float: left;
display: inline;
cursor: pointer;
}
```

Usage: Selector used to display the portlet view mode. Attributes for this selector control the display and dimensions of the view icon, including the behavior of the mouse pointer when hovering the mode.

```
.portlet-mode-reload {
background-image: url(images/reload.gif);
width: 16px;
height: 23px;
background-repeat: no-repeat;
float: left;
display: inline;
cursor: pointer;
}
```

Usage: Currently not available. But here is the intended use: Selector used to display the portlet reload mode. Attributes for this selector control the display and dimensions of the reload icon, including the behavior of the mouse pointer when hovering the mode.

- Copyright Selectors

```
.portal-copyright {
font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;
font-size: 9px;
color: #5E6D7A;
}

a.portal-copyright {
color: #768591;
text-decoration: none;
}

a.portal-copyright:hover {
```

```
color: #96A5B1;
text-decoration: none;
}
```

Usage: The above three selectors are used to style copyright content in the portal. The portal-copyright selector sets the font properties (color, etc.), and the a.portal-copyright/a.portal-copyright: hover selectors style any links that are part of the copyright information.

- Element Selectors

```
a {
color: #768591;
text-decoration: none;
}
a: hover {
color: #96A5B1;
text-decoration: none;
}
```

Usage: The above two selectors style all anchor elements that do not have their own class/selector applied.

```
INPUT {
font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;
font-size: 10px;
}
```

Usage: The above selector styles all INPUT elements that do not have their own class/selector applied.

```
SELECT {
font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;
font-size: 10px;
}
```

Usage: The above selector styles all SELECT elements that do not have their own class/selector applied.

```
FONT {
font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;
font-size: 10px;
color: #768591;
}
```

Usage: The above selector styles all FONT elements that do not have their own class/selector applied.

```
FIELDSET {
background-color: #f7f7f7;
border: 1px solid #BABDB6;
padding: 6px;
}
```

Usage: The above selector styles all FIELDSET elements that do not have their own class/selector applied.

```
LEGEND {  
  background-color: transparent;  
  padding-left: 6px;  
  padding-right: 6px;  
  padding-bottom: 0px;  
  font-size: 14px;  
}
```

Usage: The above selector styles all LEGEND elements that do not have their own class/selector applied.

- Table Selectors

```
.portlet-table-header {}
```

Usage: Not currently in use. Intended for styling tables (specifically, the TH or table header elements) that get rendered within a portlet window.

```
.portlet-table-body {}
```

Usage: Not currently in use. Intended for styling the table body element used to group rows in a table.

```
.portlet-table-alternate {}
```

Usage: Not currently in use. Used to style the background color (and possibly other attributes) for every other row within a table.

```
.portlet-table-selected {}
```

Usage: Not currently in use. Used to style text, color, etc. in a selected cell range.

```
.portlet-table-subheader {}
```

Usage: Not currently in use. Used to style a subheading within a table that gets rendered in a portlet.

```
.portlet-table-footer {}
```

Usage: Not currently in use. Similar to portlet-table-header and portlet-table-body, this selector is used to style the table footer element which is used to group the footer row in a table.

```
.portlet-table-text {}
```

Usage: Text that belongs to the table but does not fall in one of the other categories (e.g. explanatory or help text that is associated with the table). This selector can also be modified to provide styled text that can be used in all tables that are rendered within a portlet.

- FONT Selectors

```
.portlet-font {  
  color:#000;  
  font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;  
  font-size: 10px;  
}
```

Usage: Used to style the font properties on text used in a portlet. Typically this class is used for the display of non-accentuated information.

```
.portlet-font-dim {  
  color:#888385;  
  font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;  
  font-size: 10px;  
}
```

Usage: A lighter version (color-wise) of the portlet-font selector.

- **FORM Selectors**

```
.portlet-form-label {  
  color:#4A4A4A;  
  text-decoration:none;  
  font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;  
  font-size: 9px;  
}
```

Usage: Text used for the descriptive label of an entire form (not the label for each actual form field).

```
.portlet-form-button {  
  font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;  
  font-size: 9px;  
  font-weight: bold;  
  color: #270F07;  
}
```

Usage: Used to style portlet form buttons (e.g. Submit).

```
.portlet-icon-label {}
```

Usage: Not currently in use. Text that appears beside a context dependent action icon.

```
.portlet-dlg-icon-label {}
```

Usage: Not currently in use. Text that appears beside a "standard" icon (e.g Ok, or Cancel).

```
.portlet-form-field-label {  
  font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;  
  font-size: 9px;  
  color: #4A4A4A;  
}
```

Usage: Selector used to style portlet form field labels.

```
.portlet-form-field {  
font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;  
font-size: 9px;  
color: #4A4A4A;  
margin-top: 10px;  
}
```

Usage: Selector used to style portlet form fields (i.e. INPUT controls, SELECT elements, etc.).

- **LINK Selectors**

```
.portal-links:link {  
font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;  
font-size: 9px;  
font-weight: bold;  
color: #242424;  
text-decoration: none;  
}  
  
.portal-links:hover {  
font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;  
font-size: 9px;  
font-weight: bold;  
color: #5699B7;  
text-decoration: none;  
}  
  
.portal-links:active {  
font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;  
font-size: 9px;  
font-weight: bold;  
color: #242424;  
text-decoration: none;  
}  
  
.portal-links:visited {  
font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;  
font-size: 9px;  
font-weight: bold;  
color: #242424;  
text-decoration: none;  
}
```

Usage: The above four selectors are used to style links in the portal. Each pseudo class (i.e. hover, active, etc.) provides a different link style.

- **MESSAGE Selectors**

```
.portlet-msg-status {  
font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;  
font-size: 10px;  
font-style: normal;  
color: #788793;  
}
```

Usage: Selector used to signify the status of a current operation that takes place in the portlet (e.g. “saving results#?”, “step 1 of 4#?”).

```
.portlet-msg-info {  
font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;  
font-size: 9px;  
font-style: italic;  
color: #000;  
}
```

Usage: Selector used to signify general information in a portlet (e.g. help messages).

```
.portlet-msg-error {  
color:red;  
font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;  
font-size: 9px;  
font-weight: bold;  
}
```

Usage: Selector used to signify an error message in the portlet (e.g. form validation error).

```
.portlet-msg-alert {  
font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;  
font-size: 9px;  
font-weight: bold;  
color: #821717;  
}
```

Usage: Selector used to style an alert that is displayed to the user.

```
.portlet-msg-success {  
font-family: Verdana, Arial, Helvetica, Geneva, Swiss, SunSans-Regular;  
font-size: 9px;  
font-weight: bold;  
color: #359630;  
}
```

Usage: Selector used to indicate successful completion of an action in a portlet (e.g. “save successful#?”).

- SECTION Selectors

```
.portlet-section-header {  
font-weight: bold;  
font-family: Verdana, Arial, Helvetica, sans-serif;  
font-size: 13px;  
color: #768591;  
background-color: #f7f7f7;  
}
```

Usage: Table or section header.



```
.portlet-section-body {  
font-family: Verdana, Arial, Helvetica, sans-serif;  
font-size: 10px;  
}
```

Usage: Normal text in a table cell.

```
.portlet-section-alternate {  
background-color: #eeced;e  
font-family: Verdana, Arial, Helvetica, sans-serif;  
font-size: 9px;  
}
```

Usage: Used to style background color and text in every other table row.

```
.portlet-section-selected {  
background-color: #89AEC6;  
font-family: Verdana, Arial, Helvetica, sans-serif;  
font-size: 9px;  
}
```

Usage: Used to style background and font properties in a selected cell range.

```
.portlet-section-subheader {  
font-weight: bold;  
font-size: 10px;  
font-family: Verdana, Arial, Helvetica, sans-serif;  
color: #000;  
}
```

Usage: Used to style a subheading within a table/section that gets rendered in a portlet.

```
.portlet-section-footer {  
font-family: Verdana, Arial, Helvetica, sans-serif;  
background-color: #f7f7f7;  
font-size: 8px;  
}
```

Usage: Used to style footer area of a section/table that gets rendered in a portlet.

```
.portlet-section-text {}
```

Usage: Not currently used. Text that belongs to a section but does not fall in one of the other categories. This selector can also be modified to provide styled text that can be used in all sections that are rendered within a portlet.

- MENU Selectors

```
.portlet-menu {}
```

Usage: Not currently used. General menu settings such as background color, margins, etc.

```
.portlet-menu-item {  
  color: #242424;  
  text-decoration: none;  
  font-family: Verdana, Arial, Helvetica, sans-serif;  
  font-size: 9px;  
}
```

Usage: Not currently used. Normal, unselected menu item.

```
.portlet-menu-item:hover {  
  color: #5699B7;  
  text-decoration: none;  
  font-family: Verdana, Arial, Helvetica, sans-serif;  
  font-size: 9px;  
}
```

Usage: Not currently used. Used to style hover effect on a normal, unselected menu item.

```
.portlet-menu-item-selected {}
```

Usage: Not currently used. Applies to selected menu items.

```
.portlet-menu-item-selected:hover {  
  
}
```

Usage: Not currently used. Selector styles the hover effect on a selected menu item.

```
.portlet-menu-cascade-item {}
```

Usage: Not currently used. Normal, unselected menu item that has sub-menus.

```
.portlet-menu-cascade-item-selected {}
```

Usage: Not currently used. Selected sub-menu item.

```
.portlet-menu-description {}
```

Usage: Not currently used. Descriptive text for the menu (e.g. in a help context below the menu).

```
.portlet-menu-caption {}
```

Usage: Not currently used. Selector used to style menu captions.

- WSRP Selectors

```
.portlet-horizontal-separator {}
```

Usage: Not currently used. A separator bar similar to a horizontal rule, but with styling matching the page.

```
.portlet-nestedTitle-bar {}
```

Usage: Not currently used. Allows portlets to mimic the title bar when nesting something.

```
.portlet-nestedTitle {}
```

Usage: Not currently used. Allows portlets to match the textual character of the title on the title bar.

```
.portlet-tab {}
```

Usage: Not currently used. Support portlets having tabs in the same style as the page or other portlets.

```
.portlet-tab-active {}
```

Usage: Not currently used. Highlight the tab currently being shown.

```
.portlet-tab-selected {}
```

Usage: Not currently used. Highlight the selected tab (not yet active).

```
.portlet-tab-disabled {}
```

Usage: Not currently used. A tab which can not be currently activated.

```
.portlet-tab-area {}
```

Usage: Not currently used. Top level style for the content of a tab.