

GateIn Reference Guide



by the GateIn community , JBoss by Red Hat , and eXo Platform

edited by Scott Mumford (Red Hat), Thomas Heute (Red Hat), Luc Texier (Red Hat), and Christophe Laprun (Red Hat)

1. Introduction	1
1.1. Related Links	1
2. Configuration	3
2.1. Database Configuration	3
2.1.1. Overview	3
2.1.2. Configuring the database for JCR	3
2.1.3. Configuring the database for the default identity store	4
2.2. E-Mail Service Configuration	5
2.2.1. Overview	5
2.2.2. Configuring the outgoing e-mail account	5
3. Portal Development	7
3.1. Skinning the portal	7
3.1.1. Overview	7
3.1.2. Skin Components	7
3.1.3. Skin Selection	8
3.1.4. Skins in Page Markups	8
3.1.5. The Skin Service	9
3.1.6. The Default Skin	11
3.1.7. Creating New Skins	12
3.1.8. Tips and Tricks	21
3.2. Portal Lifecycle	23
3.2.1. Overview	23
3.2.2. Application Server start and stop	23
3.2.3. The Command Servlet	23
3.3. Default Portal Configuration	26
3.3.1. Overview	26
3.3.2. Configuration	26
3.4. Portal Default Permission Configuration	27
3.4.1. Overview	27
3.4.2. Overwrite Portal Default Permissions	29
3.5. Portal Navigation Configuration	29
3.5.1. Overview	29
3.5.2. Portal Navigation	31
3.5.3. Group Navigation	36
3.5.4. User Navigation	36
3.5.5. Tips	37
3.6. Internationalization Configuration	38
3.6.1. Overview	38
3.6.2. Locales configuration	39
3.6.3. ResourceBundleService	41
3.6.4. Navigation Resource Bundles	42
3.6.5. Portlets	43
3.6.6. Translating the language selection form	44
3.7. RTL (Right To Left) Framework	45

3.7.1. Groovy templates	45
3.7.2. Stylesheet	46
3.7.3. Images	47
3.7.4. Client side JavaScript	48
3.8. XML Resources Bundles	48
3.8.1. Motivation	48
3.8.2. XML format	49
3.8.3. Portal support	50
3.9. JavaScript Inter Application Communication	50
3.9.1. Overview	50
3.9.2. Library	50
3.9.3. Syntax	51
3.9.4. Example of Javascript events usage	52
3.10. Upload Component	54
3.10.1. Upload Service	54
3.11. Deactivation of the Ajax Loading Mask Layer	56
3.11.1. Purpose	56
3.11.2. Synchronous issue	56
3.12. JavaScript Configuration	57
4. Portlet development	61
4.1. Portlet Primer	61
4.1.1. JSR-168 and JSR-286 overview	61
4.1.2. Tutorials	63
5. Gadget development	77
5.1. Gadgets	77
5.1.1. Existing Gadgets	79
5.1.2. Create a new Gadget	79
5.1.3. Remote Gadget	79
5.1.4. Gadget Importing	79
5.1.5. Gadget Web Editing	80
5.1.6. Gadget IDE Editing	80
5.1.7. Dashboard Viewing	81
5.2. Setup a Gadget Server	81
5.2.1. Virtual servers for gadget rendering	81
5.2.2. Configuration	82
6. Authentication and Identity	85
6.1. Predefined User Configuration	85
6.1.1. Overview	85
6.1.2. Plugin for adding users, groups and membership types	85
6.1.3. Membership types	85
6.1.4. Groups	86
6.1.5. Users	87
6.1.6. Plugin for monitoring user creation	88
6.2. Authentication Token Configuration	89

6.2.1. What is Token Service?	89
6.2.2. Implementing the Token Service API	89
6.2.3. Configuring token services	90
6.3. PicketLink IDM integration	91
6.3.1. Configuration files	91
6.4. Organization API	97
6.5. Accessing User Profile	99
6.6. SSO - Single Sign On	100
6.6.1. Overview	100
6.6.2. CAS - Central Authentication Service	100
6.6.3. JOSSO	108
6.6.4. OpenSSO - The Open Web SSO project	112
6.6.5. SPNEGO	119
7. Web Services for Remote Portlets (WSRP)	125
7.1. Introduction	125
7.2. Level of support in Gateln 3.1	125
7.3. Deploying Gateln's WSRP services	126
7.3.1. Considerations to use WSRP when running Gateln on a non-default port or hostname	127
7.3.2. Considerations to use WSRP with SSL	127
7.4. Making a portlet remotable	127
7.5. Consuming Gateln's WSRP portlets from a remote Consumer	129
7.6. Consuming remote WSRP portlets in Gateln	129
7.6.1. Overview	129
7.6.2. Configuring a remote producer walk-through	130
7.6.3. Configuring access to remote producers via XML	135
7.6.4. Examples	137
7.7. Consumers maintenance	138
7.7.1. Modifying a currently held registration	138
7.7.2. Consumer operations	142
7.7.3. Erasing local registration data	143
7.8. Configuring Gateln's WSRP Producer	144
7.8.1. Overview	144
7.8.2. Default configuration	144
7.8.3. Registration configuration	144
7.8.4. WSRP validation mode	147
8. Advanced Development	149
8.1. Foundations	149
8.1.1. Gateln Kernel	149
8.1.2. Configuring services	150
8.1.3. Configuration syntax	150
8.1.4. InitParams configuration object	154
8.1.5. Configuring a portal container	157
8.1.6. Gateln Extension Mechanism, and Portal Extensions	160

8.1.7. Running Multiple Portals	161
---------------------------------------	-----

Introduction

GateIn 3.1 is the merge of two mature Java projects; JBoss Portal and eXo Portal. This new community project takes the best of both offerings and incorporates them into a single portal framework. The aim is to provide an intuitive user-friendly portal, and a framework to address the needs of today's Web 2.0 applications.



This book provides a deep-dive information about installation and configuration of the services provided by GateIn.

1.1. Related Links

- GateIn homepage: www.gatein.org [http://www.gatein.org]
- GateIn videos: vimeo.com/channels/gatein [http://vimeo.com/channels/gatein]
- GateIn documentation: www.jboss.org/gatein/documentation.html [http://www.jboss.org/gatein/documentation.html]
- GateIn downloads: www.jboss.org/gatein/downloads.html [http://www.jboss.org/gatein/downloads.html]

Configuration

2.1. Database Configuration

2.1.1. Overview

GateIn 3.1 has two different database dependencies. One is the identity service configuration, which depends on Hibernate. The other is Java content repository (JCR) service, which depends on JDBC API, and can integrate with any existing datasource implementation.

When you change the database configuration for the first time, GateIn will automatically generate the proper schema (assuming that the database user has the appropriate permissions).

GateIn 3.1 assumes the default encoding for your database is `latin1`. You may need to change this parameter for your database in order for GateIn 3.1 to work properly.

2.1.2. Configuring the database for JCR

To configure the database used by JCR you will need to edit the file:

```
$JBOSS_HOME/server/default/conf/gatein/configuration.properties
```

For Tomcat, the file is located at

```
$TOMCAT_HOME/gatein/conf/configuration.properties
```

And edit the values of driver, url, username and password with the values for your JDBC connection (please, refer to your database JDBC driver documentation).

```
gatein.jcr.datasource.driver=org.hsqldb.jdbcDriver
gatein.jcr.datasource.url=jdbc:hsqldb:file:${gatein.db.data.dir}/data/jdbcjcr_${name}
gatein.jcr.datasource.username=sa
gatein.jcr.datasource.password=
```

By default, the name of the database is "jdbcjcr_\${name}" - \${name} should be a part of the database name, as it is dynamically replaced by the name of the portal container extension (for instance, gatein-sample-portal.ear defines "sample-portal" as container name and the default portal defines "portal" as container name).

In the case of HSQL the databases are created automatically. For any other database you will need to create a database named jdbcjcr_portal (and "jdbcjcr_sample-portal" if you have gatein-sample-

portal.ear in \$JBOSS_HOME/server/default/deploy - note that some databases don't accept '-' in the database name, so you may have to remove \$JBOSS_HOME/server/default/deploy/gatein-sample-portal.ear)

Make sure the user has rights to create tables on jdbcjcr_portal, and to update them as they will be automatically created during the first startup .

Also add your database's JDBC driver into the classpath - you can put it in \$JBOSS_HOME/server/default/lib (or \$TOMCAT_HOME/lib, if you are running on Tomcat)

MySQL example:

Let's configure our JCR to store data in MySQL. Let's pretend we have a user named "gateinuser" with a password "gateinpassword". We would create a database "mygateindb_portal" (remember that _portal is required), and assign our user the rights to create tables.

Then we need to add MySQL's JDBC driver to the classpath, and finally edit gatein.ear/02portal.war/WEB-INF/conf/jcr/jcr-configuration to contain the following:

```
gatein.jcr.datasource.driver=com.mysql.jdbc.Driver
gatein.jcr.datasource.url=jdbc:mysql://localhost:3306/mygateindb${container.name.suffix}
gatein.jcr.datasource.username=gateinuser
gatein.jcr.datasource.password=gateinpassword
```

2.1.3. Configuring the database for the default identity store

By default, users are stored in a database. To change the database in which to store users, you will need to edit the file:

```
$JBOSS_HOME/server/default/conf/gatein/configuration.properties
```

For Tomcat, the file is located at

```
$TOMCAT_HOME/gatein/conf/configuration.properties
```

You will find the same kind of configuration as in jcr-configuration.xml:

```
gatein.idm.datasource.driver=org.hsqldb.jdbcDriver
gatein.idm.datasource.url=jdbc:hsqldb:file:${gatein.db.data.dir}/data/jdbcidm_${name}
gatein.idm.datasource.username=sa
gatein.idm.datasource.password
```

2.2. E-Mail Service Configuration

2.2.1. Overview

GateIn 3.1 includes an e-mail sending service that needs to be configured before it can function properly. This service, for instance, is used to send e-mails to users who forgot their password or username.

2.2.2. Configuring the outgoing e-mail account

The e-mail service can use any SMTP account configured in `$JBOSS_HOME/server/default/conf/gatein/configuration.properties` (or `$TOMCAT_HOME/gatein/conf/configuration.properties` if you are using Tomcat).

The relevant section looks like:

```
# EMail
gatein.email.smtp.username=
gatein.email.smtp.password=
gatein.email.smtp.host=smtp.gmail.com
gatein.email.smtp.port=465
gatein.email.smtp.starttls.enable=true
gatein.email.smtp.auth=true
gatein.email.smtp.socketFactory.port=465
gatein.email.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory
```

It is preconfigured for GMail, so that any GMail account can easily be used (simply use the full GMail address as username, and fill-in the password).

In corporate environments you will want to use your corporate SMTP gateway. When using it over SSL, like in default configuration, you may need to configure a certificate truststore, containing your SMTP server's public certificate. Depending on the key sizes, you may then also need to install Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files for your Java Runtime Environment.

Portal Development

3.1. Skinning the portal

3.1.1. Overview

GateIn 3.1 provides robust skinning support for the entire portal User Interface (UI). This includes support for skinning all of the common portal elements as well as being able to provide custom skins and window decoration for individual portlets. All of this designed with common graphic resource reuse and ease of development in mind.

3.1.2. Skin Components

The complete skinning of a page can be decomposed into three main parts:

Portal Skin

The portal skin contains the css styles for the portal and its various UI components. This should include all the UI components except for the window decorators and portlet specific styles.

Window Styles

The CSS styles associated with the portlet window decorators. The window decorators contain the control buttons and borders surrounding each portlet. Individual portlets can have their own window decorator selected, or be rendered without one.

Portlet Skins

The portlet skins effect how portlets are rendered on the page. There are two main ways this can be affected:

Portlet Specification CSS Classes

The portlet specification defines a set of css classes that should be available to portlets. GateIn 3.1 provides these classes as part of the portal skin. This allows each portal skin to define its own look and feel for these default values.

Portlet Skins

GateIn 3.1 provides a means for portlet css files to be loaded based on the current portal skin. This allows a portlet to provide different css styles to better match the current portal look and feel. Portlet skins provide a much more customizable css experience than just using the portlet specification css classes.



Note

The window decorators and the default portlet specification css classes should be considered separate types of skinning components, but they need to be included as part of the overall portal skin. The portal skin must include these component's css classes or they will not be displayed correctly.

A portlet skin doesn't need to be included as part of the portal skin and can be included within the portlets web application.

3.1.3. Skin Selection

3.1.3.1. Skin Selection Through the User Interface

There are a few means in which a skin can be selected to be displayed to the user. The easiest way to change the skin is select it through the user interface. An admin can change the default skin for the portal, or a logged in user can select which skin they would prefer to be displayed.

Please see the User Guide for information on how to change the skin using the user interface.

3.1.3.2. Setting the Default Skin within the Configuration Files

The default skin can also be configured through the portal configuration files if using the admin user interface is not desired. This will allow for the portal to have the new default skin ready when GateIn 3.1 is first started.

The default skin of the portal is called `Default`. To change this value add a `skin` tag in the `02portal.war/WEB-INF/conf/portal/portal/classic/portal.xml` configuration file.

To change the skin to `MySkin` you would make the following changes:

```
<portal-config>
  <portal-name>classic</portal-name>
  <locale>en</locale>
  <access-permissions>Everyone</access-permissions>
  <edit-permission>*:/platform/administrators</edit-permission>
  <skin>MySkin</skin>
  ...
</portal-config>
```

3.1.4. Skins in Page Markups

A GateIn 3.1 skin contains css styles for the portal's components but also shares components that may be reused in portlets. When GateIn 3.1 generates a portal page markup, it inserts stylesheet links in the page's `head` tag.

There are two main types of css links that will appear in the `head` tag: a link to the portal skin css file and a link to the portlet skin css files.

Portal Skin

The portal skin will appear as a single link to a css file. This link will contain contents from all the portal skin classes merged into one file. This allow for the portal skin to be transfered more quickly as a single file instead of many multiple smaller files. Included with every page render.

Portlet Skin

Each portlet on a page may contribute its own style. The link to the portlet skin will only appear on the page if that portlet is loaded on the current page. A page may contain many portlet skin css links or none.

In the code fragment below you can see the two types of links:

```
<head>
...
<!-- The portal skin -->
<link id="CoreSkin" rel="stylesheet" type="text/css" href="/eXoResources/skin/Stylesheet.css"
/>

<!-- The portlet skins -->
<link id="web_FooterPortlet" rel="stylesheet" type="text/css" href= "/web/skin/portal/webui/
component/UIFooterPortlet/DefaultStylesheet.css" />
<link id="web_NavigationPortlet" rel="stylesheet" type="text/css" href= "/web/skin/portal/webui/
component/UINavigationPortlet/DefaultStylesheet.css" />
<link id="web_HomePagePortlet" rel="stylesheet" type="text/css" href= "/portal/templates/skin/
webui/component/UIHomePagePortlet/DefaultStylesheet.css" />
<link id="web_BannerPortlet" rel="stylesheet" type="text/css" href= "/web/skin/portal/webui/
component/UIBannerPortlet/DefaultStylesheet.css" />
...
</head>
```



Note

Window styles and the portlet specification CSS classes are included within the portal skin.

3.1.5. The Skin Service

The skin service is a GateIn 3.1 service which manages the various types of skins. It is responsible for discovering and deploying the skins into the portal.

3.1.5.1. Skin configuration

GateIn 3.1 automatically discovers web archives that contain a file descriptor for skins (`WEB-INF/gatein-resources.xml`). This file is responsible for specifying the portal, portlet and window decorators to be deployed into the skin service.

The full schema can be found in lib directory:

`exo.portal.component.portal.jar/gatein_resources_1_0.xsd`

Here is an example where we define a skin (MySkin) with its CSS location, and specify a few window decorator skins:

```
<gatein-resources>
  <portal-skin>
    <skin-name>MySkin</skin-name>
    <css-path>/skin/myskin.css</css-path>
    <overwrite>false</overwrite>
  </portal-skin>
</gatein-resources>

<!-- window style -->
<window-style>
  <style-name>MyThemeCategory</style-name>
  <style-theme>
    <theme-name>MyThemeBlue</theme-name>
  </style-theme>
  <style-theme>
    <theme-name>MyThemeRed</theme-name>
  </style-theme>
  ...
</window-style>
```

3.1.5.2. Resource Request Filter

Because of the Right-To-Left support all CSS files need to be retrieved through a Servlet filter and the web application needs to be configured to activate this filter. This is already done for `01eXoResources.war` web application which contains the default skin.

Any new web applications containing skinning css files will need to have the following added to their `web.xml` :

```
<filter>
  <filter-name>ResourceRequestFilter</filter-name>
  <filter-class>org.exoplatform.portal.application.ResourceRequestFilter</filter-class>
</filter>
```



```

</filter>

<filter-mapping>
<filter-name>ResourceRequestFilter</filter-name>
<url-pattern>*.css</url-pattern>
</filter-mapping>

```



Note

The `display-name` element will also need to be specified in the `web.xml` for the skinning service to work properly with the web application.

3.1.6. The Default Skin

The default skin for GateIn 3.1 is located as part of the `01eXoResource.war`. The main files associated with the skin is show below:

WEB-INF/gatein-resources.xml

1

WEB-INF/web.xml

2

skin/Stylesheet.css

- 1 gatein-resources.xml: defines the skin setup to use
- 2 web.xml: contains the resource filer and has the display-name set
- 3 Stylesheet.css: contains the CSS class definitions for this skin.

gatein-resources.xml

For the default portal skin, this file contains definitions for the portal skin, the window decorations that this skin provides and well as defining some javascript resources which are not related to the skin. The default portal skin doesn't directly define portlet skins, these should be provided by the portlets themselves.

web.xml

For the default portal skin, the `web.xml` of the `eXoResources.war` will contains a lot of information which is mostly irrelevant to the portal skinning. The areas of interest in this file is the `resourcerequestfilter` and the fact that the `display-name` is set.

Stylesheet.css

The main portal skin stylesheet. The file is the main entry point to the css class definitions for the skin. Below is shown the contents of this file:

```
@import url(DefaultSkin/portal/webui/component/UIPortalApplicationSkin.css);  
@import url(DefaultSkin/webui/component/Stylesheet.css);  
@import url(PortletThemes/Stylesheet.css);  
@import url(Portlet/Stylesheet.css);
```

- 1 Skin for the main portal page.
- 2 Skins for various portal components.
- 3 Window decoration skins.
- 4 The portlet specification css classes.

Instead of defining all the CSS classes in this one file we are instead importing other css stylesheet files, some of which may also import other CSS stylesheets. The css classes are split up between multiple files to make it easier for new skins to reuse parts of the default skin.

To reuse a CSS stylesheet from the default portal skin you would need to reference the default skin from `eXoResources`. For example, to include the window decorators from the default skin within a new portal skin you would need to use this import:

```
@import url(/eXoResources/skin/Portlet/Stylesheet.css);
```



Note

When the portal skin is added to the page, it merge all the css stylesheets into a single file.

3.1.7. Creating New Skins

3.1.7.1. Creating a New Portal Skin

A new portal will need to be added to the portal through the skin service. As such the web application which contains the skin will need to be properly configured for the skin service to discover them. This means properly configuring the `ResourceRequestFilter` and `gatein-resources.xml`.

3.1.7.1.1. Portal Skin Configuration

The `gatein-resources.xml` will need to specify the new portal skin. This will include specifying the name of the new skin, where to locate its css stylesheet file and whether to overwrite an existing portal theme with the same name.

```
<gatein-resources>
  <portal-skin>
    <skin-name>MySkin</skin-name>
    <css-path>/skin/myskin.css</css-path>
    <overwrite>false</overwrite>
  </portal-skin>
</gatein-resources>
```

The default portal skin and window styles are defined in `01eXoResources.war/WEB-INF/gatein-resources.xml`.

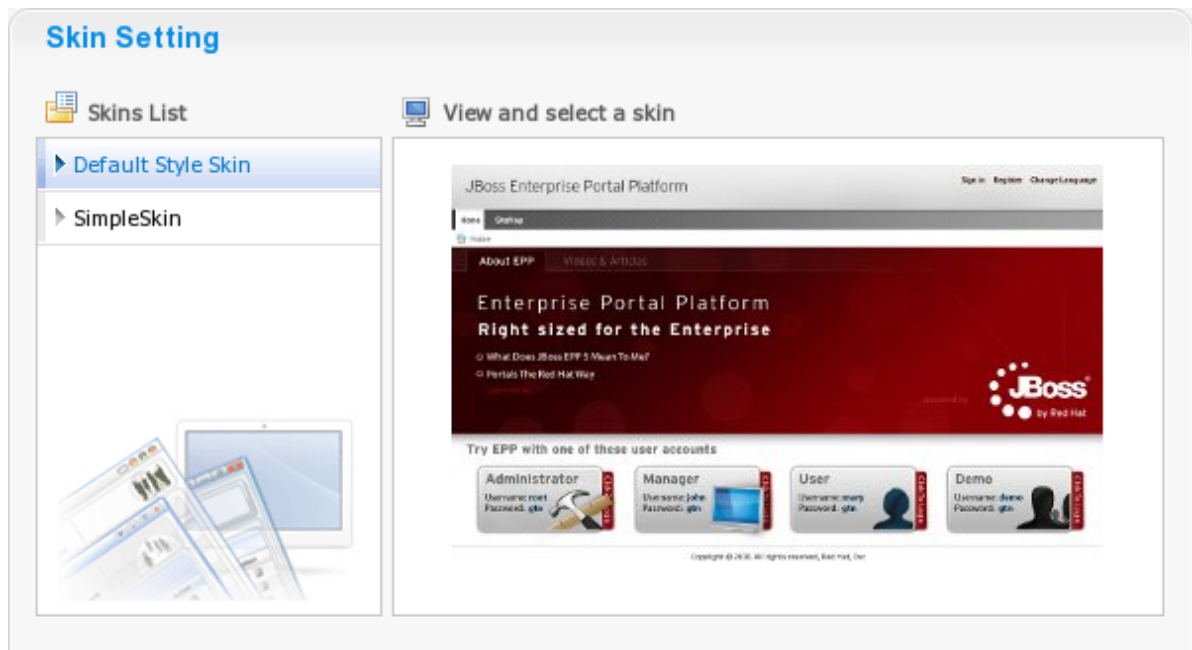


Note

The css for the portal skin needs to contain the css for all the window decorations and the portlet specification css classes.

3.1.7.1.2. Portal Skin Preview Icon

When selecting a skin it is possible to see a preview of what the skin will look like. The current skin needs to know about the skin icons for all the available skins, otherwise it will not be able to show the previews. When creating a new portal it is recommended to include the preview icons of the other skins and to update the other skins with your new portal skin preview.



The portal skin preview icon is specified through the CSS of the portal skin. In order for the current portal skin to be able to display the preview it must specify a specific CSS class and set the icon as the background.

For a portal named **MySkin** in must define the following CSS class:

```
.UIChangeSkinForm .UIItemSelector .TemplateContainer .MySkinImage
```

In order for the default skin to know about the skin icon for a new portal skin, the preview screenshot needs to be place in:

```
01eXoResources.war:/skin/DefaultSkin/portal/webui/component/customization/  
UIChangeSkinForm/background
```

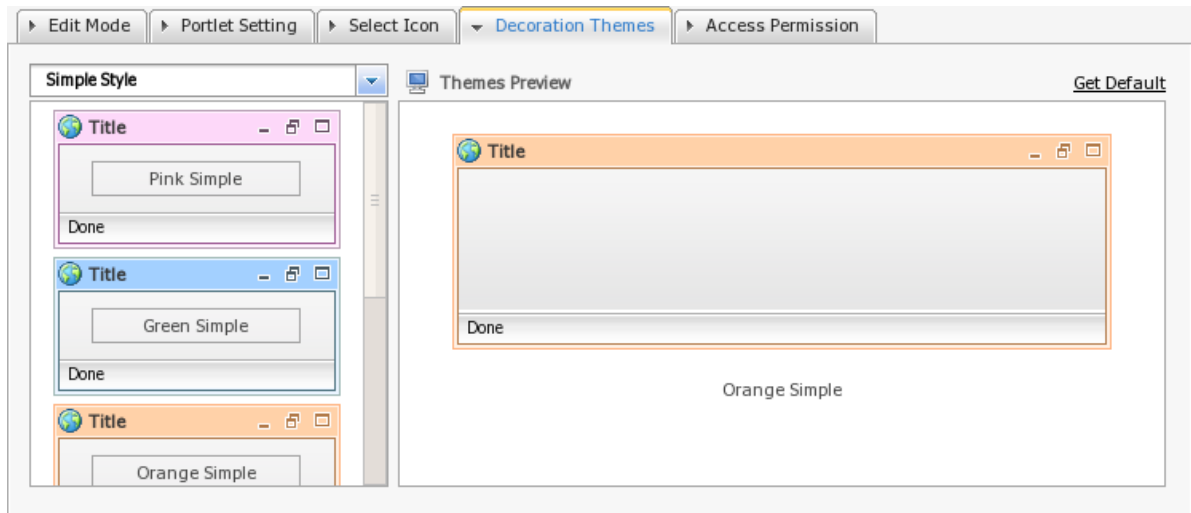
The CSS stylesheet for the default portal needs to have the following updated with the preview icon css class. For a skin named **MySkin** then the following needs to be updated:

```
01eXoResources.war:/skin/DefaultSkin/portal/webui/component/customization/  
UIChangeSkinForm/Stylesheet.css
```

```
.UIChangeSkinForm .UIItemSelector .TemplateContainer .MySkinImage {  
    margin: auto;  
    width: 329px; height:204px;  
    background: url('background/MySkin.jpg') no-repeat top;  
    cursor: pointer ;  
}
```

3.1.7.2. Creating a New Window Style

Window styles are the CSS applied to window decoration. When an administrator choose a new application to add on a page he can decide which style of decoration would go around the window if any.



3.1.7.2.1. Window Style Configuration

Window Styles are defined within a `gatein-resources.xml` file which is used by the skin service to deploy the window style into the portal. Window styles can belong in with a window style category, this category and the window styles will need to be specified in resources file.

The following `gatein-resource.xml` fragment will add `MyThemeBlue` and `MyThemeRed` to the `MyTheme` category.

```
<window-style>
  <style-name>MyTheme</style-name>
  <style-theme>
    <theme-name>MyThemeBlue</theme-name>
  </style-theme>
  <style-theme>
    <theme-name>MyThemeRed</theme-name>
  </style-theme>
</window-style>
```

The windows style configuration for the default skin is configured in:

`01eXoResources.war/WEB-INF/gatein-resources.xml`



Note

When a window style is defined in `gatein-resources.xml` file, it will be available to all portlets regardless if the current portal skin support the window decorator or not. It is recommended that when a new window decorator is added that it is added to all portal skins or that portal skins share a common stylesheet for window decorators.

3.1.7.2.2. Window Style CSS

In order for the skin service to display the window decorators, it must have CSS classes with specific naming in relation to the window style name. The service will try and display css based on this naming. The css class must be included as part of the current portal skin for the window decorators to be displayed.

The location of the window decorator css classes for the default portal theme is located at:

```
01eXoResources.war/skin/PortletThemes/Stylesheet.css
```

Create the CSS file:

```
/*---- MyTheme ----*/
.MyTheme .WindowBarCenter .WindowPortletInfo {
    margin-right: 80px; /* orientation=lt */
    margin-left: 80px; /* orientation=rt */
}
.MyTheme .WindowBarCenter .Controllcon {
    float: right; /* orientation=lt */
    float: left; /* orientation=rt */
    width: 24px;
    height: 17px;
    cursor: pointer;
    background-image: url('background/MyTheme.png');
}
.MyTheme .ArrowDownIcon {
    background-position: center 20px;
}
.MyTheme .OverArrowDownIcon {
    background-position: center 116px;
}
.MyTheme .MinimizedIcon {
    background-position: center 44px;
}
.MyTheme .OverMinimizedIcon {
```

```
background-position: center 140px;
}
.MyTheme .MaximizedIcon {
background-position: center 68px;
}
.MyTheme .OverMaximizedIcon {
background-position: center 164px;
}
.MyTheme .RestoreIcon {
background-position: center 92px;
}
.MyTheme .OverRestoreIcon {
background-position: center 188px;
}
.MyTheme .NormalIcon {
background-position: center 92px;
}
.MyTheme .OverNormalIcon {
background-position: center 188px;
}
.UIPageDesktop .MyTheme .ResizeArea {
float: right;/* orientation=lt */
float: left;/* orientation=rt */
width: 18px; height: 18px;
cursor: nw-resize;
background: url('background/ResizeArea18x18.gif') no-repeat left top; /* orientation=lt */
background: url('background/ResizeArea18x18-rt.gif') no-repeat right top; /* orientation=rt */
}
.MyTheme .Information {
height: 18px; line-height: 18px;
vertical-align: middle; font-size: 10px;
padding-left: 5px;/* orientation=lt */
padding-right: 5px;/* orientation=rt */
margin-right: 18px;/* orientation=lt */
margin-left: 18px;/* orientation=rt */
}
.MyTheme .WindowBarCenter .WindowPortletIcon {
background-position: left top; /* orientation=lt */
background-position: right top; /* orientation=rt */
padding-left: 20px; /* orientation=lt */
padding-right: 20px; /* orientation=rt */
height: 16px;
line-height: 16px;
}
```

```
.MyTheme .WindowBarCenter .PortletName {
    font-weight: bold;
    color: #333333;
    overflow: hidden;
    white-space: nowrap;
    width: 100%;
}
.MyTheme .WindowBarLeft {
    padding-left: 12px;
    background-image: url('background/MyTheme.png');
    background-repeat: no-repeat;
    background-position: left -148px;
}
.MyTheme .WindowBarRight {
    padding-right: 11px;
    background-image: url('background/MyTheme.png');
    background-repeat: no-repeat;
    background-position: right -119px;
}
.MyTheme .WindowBarCenter {
    background-image: url('background/MyTheme.png');
    background-repeat: repeat-x;
    background-position: left -90px;
}
.MyTheme .WindowBarCenter .FixHeight {
    height: 21px;
    padding-top: 8px;
}
.MyTheme .MiddleDecoratorLeft {
    padding-left: 12px;
    background: url('background/MyTheme.png') repeat-y left;
}
.MyTheme .MiddleDecoratorRight {
    padding-right: 11px;
    background: url('background/MyTheme.png') repeat-y right;
}
.MyTheme .MiddleDecoratorCenter {
    background: #ffffff;
}
.MyTheme .BottomDecoratorLeft {
    height: 12px;
    background-image: url('background/MyTheme.png');
    background-repeat: no-repeat;
    background-position: left -60px;
```



```

}
.MyTheme .BottomDecoratorRight {
padding-right: 11px;
background-image: url('background/MyTheme.png');
background-repeat: no-repeat;
background-position: right -30px;
}
.MyTheme .BottomDecoratorCenter {
background-image: url('background/MyTheme.png');
background-repeat: repeat-x;
background-position: left top;
}
.MyTheme .BottomDecoratorCenter .FixHeight {
height: 30px;
}

```

3.1.7.2.3. How to Set the Default Window Style

To set the default window style to be used for a portal, you will to specify the css classes for a theme called `DefaultTheme`.



Note

You do not need to specify the `DefaultTheme` in `gatein-resources.xml`

3.1.7.3. How to Create New Portlet skins

Portlets often require additional styles that may not be defined by the portal skin. GateIn 3.1 allows portlets to define additional stylesheets for each portlet and will append the corresponding `link` tags to the `head`.

The link ID will be of the form `{portletAppName}{PortletName}`. For example: `ContentPortlet` in `content.war`, will give `id="contentContentPortlet"`

To define a new CSS file to include whenever a portlet is available on a portal page, the following fragment needs to be added in `gatein-resources.xml`

```

<portlet-skin>
  <application-name>portletAppName</application-name>
  <portlet-name>PortletName</portlet-name>
  <skin-name>Default</skin-name>
  <css-path>/skin/DefaultStylesheet.css</css-path>
</portlet-skin>

```

```
<portlet-skin>
  <application-name>portletAppName</application-name>
  <portlet-name>PortletName</portlet-name>
  <skin-name>OtherSkin</skin-name>
  <css-path>/skin/OtherSkinStylesheet.css</css-path>
</portlet-skin>
```

This will load the DefaultStylesheet.css when the Default skin is used and the OtherSkinStylesheet.css when the OtherSkin is used.



Note

If the current portal skin is not defined as part of the supported skins, then the portlet css class will not be loaded. It is recommended to update portlet skins whenever a new portal skin is created.

3.1.7.3.1. Change portlet icons

Each portlet can be represented by an unique icon that you can see in the portlet registry or page editor. This icon can be changed by adding an image to the directory of the portlet webapplication:

- `skin/DefaultSkin/portletIcons/icon_name.png`.

To be used correctly the icon must be named after the portlet.

For example, the icon for an account portlet named AccountPortlet would be located at:

- `skin/DefaultSkin/portletIcons/AccountPortlet.png`



Note

You must use `skin/DefaultSkin/portletIcons/` for the directory to store the portlet icon regardless of what skin is going to be used.

3.1.7.4. How to create a new Portlet Specification CSS Classes

The portlet specification defines a set of default css classes that should be available for portlets. These classes are included as part of the portal skin. Please see the portlet specification for a list of the default classes that should be available.

For the default portal skin, the portlet specification CSS classes are defined in :

```
eXoResources.war/skin/Portlet/Stylesheet.css
```

3.1.8. Tips and Tricks

3.1.8.1. Easier css debugging

By default, CSS files are cached and their imports are merged into a single CSS file at the server side. This reduces the number of HTTP requests from the browser to the server.

The optimization code is quite simple as all the CSS files are parsed at the server startup time and all the `@import` and `url(...)` references are rewritten to support a single flat file. The result is stored in a cache directly used from the `ResourceRequestFilter`.

Although the optimization is useful for a production environments, it may be easier to deactivate this optimization while debugging stylesheets. To do so, set the java system property `exo.product.developing` to `true`.

For example, the property can be passed as a JVM parameter with `-D` option when running GateIn.

```
sh $JBOSS_HOME/bin/run.sh -Dexo.product.developing=true
```

1. warning("This option may cause display bugs with certain browsers like Internet Explorer")

3.1.8.2. Some CSS techniques

It is recommended that users have some experience with CSS before studying GateIn 3.1 CSS.

GateIn 3.1 relies heavily on CSS to create the layout and effects for the UI. Some common techniques for customizing GateIn 3.1 CSS are explained below.

3.1.8.2.1. Decorator pattern

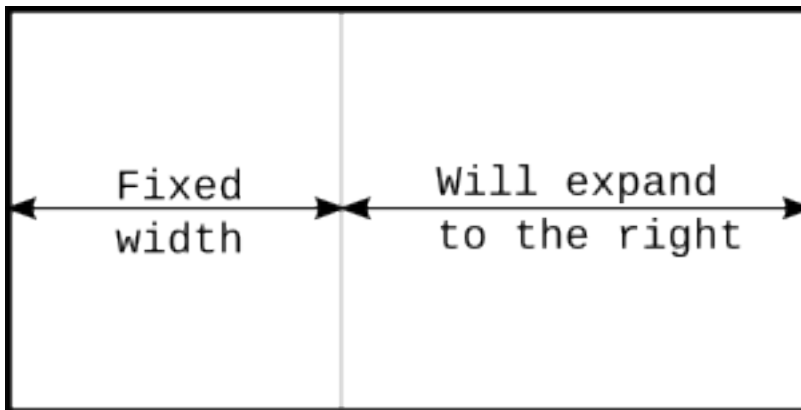
The decorator is a pattern to create a contour or a curve around an area. In order to achieve this effect you need to create 9 cells. The BODY is the central area that you want to decorate. The other 8 cells are distributed around the BODY cell. You can use the width, height and background image properties to achieve any decoration effect that you want.

TopLeft	TopCenter	TopRight
CenterLeft	BODY	CenterRight
BottomLeft	BottomCenter	BottomRight

```
<div class="Parent">
  <div class="TopLeft">
    <div class="TopRight">
      <div class="TopCenter"><span></span></div>
    </div>
  </div>
  <div class="CenterLeft">
    <div class="CenterRight">
      <div class="CenterCenter">BODY</div>
    </div>
  </div>
  <div class="BottomLeft">
    <div class="BottomRight">
      <div class="BottomCenter"><span></span></div>
    </div>
  </div>
</div>
```

3.1.8.2.2. Left margin left pattern

Left margin left pattern is a technique to create 2 blocks side by side. The left block will have a fixed size and the right block will take the rest of the available space. When the user resizes the browser the added or removed space will be taken from the right block.



```
<div class="Parent">
  <div style="float: left; width: 100px">
  </div>
  <div style="margin-left: 105px;">
  <div>
  <div style="clear: left"><span></span></div>
</div>
```

3.2. Portal Lifecycle

3.2.1. Overview

This chapter describes the portal lifecycle from the application server start to its stop as well as how requests are handled.

3.2.2. Application Server start and stop

A portal instance is simply a web application deployed as a WAR in an application server. Portlets are also part of an enhanced WAR called a portlet application.

GateIn 3.1 doesn't require any particular setup for your portlet in most common scenarios and the `web.xml` file can remain without any GateIn 3.1 specific configuration.

During deployment, GateIn 3.1 will automatically and transparently inject a servlet into the portlet application to be able to interact with it. This feature is dependent on the underlying servlet container but will work out of the box on the proposed bundles.

3.2.3. The Command Servlet

The servlet is the main entry point for incoming requests, it also includes some init code when the portal is launched. This servlet (`org.gatein.wci.command.CommandServlet`) is automatically added during deployment and mapped to `/tomcatgateinservlet`.

This is equivalent to adding the following into `web.xml`.



Note

As the servlet is already configured this example is for information only.

```
<servlet>
  <servlet-name>TomcatGateInServlet</servlet-name>
  <servlet-class>org.gatein.wci.command.CommandServlet</servlet-class>
  <load-on-startup>0</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>TomcatGateInServlet</servlet-name>
  <url-pattern>/tomcatgateinservlet</url-pattern>
</servlet-mapping>
```

It is possible to filter on the `CommandServlet` by filtering the URL pattern used by the Servlet mapping.

The example below would create a servlet filter that calculates the time of execution of a portlet request.

The filter class:

```
package org.example;

import java.io.IOException;

import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class MyFilter implements javax.servlet.Filter {

    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException
    {
```

```

    long beforeTime = System.currentTimeMillis();
    chain.doFilter(request, response);
    long afterTime = System.currentTimeMillis();
    System.out.println("Time to execute the portlet request (in ms): " + (afterTime - beforeTime));
}

public void init(FilterConfig config) throws ServletException
{
}

public void destroy()
{
}
}

```

The Java EE web application configuration file (`web.xml`) of the portlet on which we want to know the time to serve a portlet request. As mentioned above nothing specific to GateIn 3.1 needs to be included, only the URL pattern to set has to be known.

```

<?xml version="1.0"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd"
    version="2.5">

    <filter>
        <filter-name>MyFilter</filter-name>
        <filter-class>org.example.MyFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>MyFilter</filter-name>
        <url-pattern>/tomcatgateinservlet</url-pattern>
        <dispatcher>INCLUDE</dispatcher>
    </filter-mapping>

</web-app>

```



INCLUDE dispatcher

It is important to set `INCLUDE` as dispatcher as the portal will always hit the `CommandServlet` through a request dispatcher. Without this, the filter will not be triggered, unless direct access to a resource (such as an image).

3.3. Default Portal Configuration

3.3.1. Overview

GateIn 3.1 default home page URL is `http://{hostname}:{port}/portal/`. There may be multiple independent portals deployed in parallel at any given time, each of which has its root context (i.e.: `http://{hostname}:{port}/sample-portal/`). Each portal is internally composed of one or more, what we again call 'portals'. There needs to be at least one such portal - the default one is called 'classic'. When accessing GateIn 3.1 default home page URL, you are automatically redirected to 'classic' portal. The default portal performs another important task. When starting up GateIn 3.1 for the first time, its JCR database will be empty (that's where portals keep their runtime-configurable settings). It's the default portal that's used to detect this, and to trigger automatic data initialization.

3.3.2. Configuration

The following example configuration can be found at: `"02portal.war:/WEB-INF/conf/portal/portal-configuration.xml"`.

```
<component>
  <key>org.exoplatform.portal.config.UserPortalConfigService</key>
  <type>org.exoplatform.portal.config.UserPortalConfigService</type>
  <component-plugins>
    <component-plugin>
      <name>new.portal.config.user.listener</name>
      <set-method>initListener</set-method>
      <type>org.exoplatform.portal.config.NewPortalConfigListener</type>
      <description>this listener init the portal configuration</description>
      <init-params>
        <value-param>
          <name>default.portal</name>
          <description>The default portal for checking db is empty or not</description>
          <value>classic</value>
        </value-param>
        ...
      </init-params>
    </component-plugin>
  </component-plugins>
</component>
```



```
</component-plugins>
</component>
```

In this example the **classic** portal has been set as the default.



Note

Components, component-plugins, and init-params are explained in Foundations chapter. For now just note how `NewPortalConfigListener` component-plugin is used to add configuration to `UserPortalConfigService`, which is designed in this way to allow other components to add configuration to it.

3.4. Portal Default Permission Configuration

3.4.1. Overview

The default permission configuration for the portal is defined through `org.exoplatform.portal.config.UserACL` component configuration in the file `02portal.war:WEB-INF/conf/portal/portal-configuration.xml`.

It defines 8 permissions types:

`super.user`

The super user has all the rights on the platform, this user is referred to as *root*.

`portal.administrator.groups`

Any member of those groups are considered administrators. Default value is `/platform/administrators`.

`portal.administrator.mstype`

Any user with that membership type would be considered administrator or the associated group. Default value is `manager`.

`portal.creator.groups`

This list defines all groups that will be able to manage the different portals. Members of this group also have the permission to create new portals. The format is `membership:/group/subgroup`.

`navigation.creator.membership.type`

Defines the membership type of group managers. The group managers have the permission to create and edit group pages and they can modify the group navigation.

`guests.group`

Any anonymous user automatically becomes a member of this group when they enter the public pages.

mandatory.groups

Groups that can't be deleted.

mandatory.mstypes

Membership types that can't be deleted.

```
<component>
  <key>org.exoplatform.portal.config.UserACL</key>
  <type>org.exoplatform.portal.config.UserACL</type>
  <init-params>
    <value-param>
      <name>super.user</name>
      <description>administrator</description>
      <value>root</value>
    </value-param>

    <value-param>
      <name>portal.creator.groups</name>
      <description>groups with membership type have permission to manage portal</description>
      <value>*:/platform/administrators,*:/organization/management/executive-board</value>
    </value-param>

    <value-param>
      <name>navigation.creator.membership.type</name>
      <description>specific membership type have full permission with group navigation</
description>
      <value>manager</value>
    </value-param>
    <value-param>
      <name>guests.group</name>
      <description>guests group</description>
      <value>/platform/guests</value>
    </value-param>
    <value-param>
      <name>access.control.workspace</name>
      <description>groups with memberships that have the right to access the User Control
Workspace</description>
      <value>*:/platform/administrators,*:/organization/management/executive-board</value>
    </value-param>
  </init-params>
</component>
```

3.4.2. Overwrite Portal Default Permissions

When creating custom portals and portal extensions it's possible to override the default configuration by using `org.exoplatform.portal.config.PortalACLPlugin`, configuring it as an external-plugin of `org.exoplatform.portal.config.UserACL` service:

```
<external-component-plugins>
  <target-component>org.exoplatform.portal.config.UserACL</target-component>
  <component-plugin>
    <name>addPortalACLPlugin</name>
    <set-method>addPortalACLPlugin</set-method>
    <type>org.exoplatform.portal.config.PortalACLPlugin</type>
    <description>setting some permission for portal</description>
    <init-params>
      <values-param>
        <name>access.control.workspace.roles</name>
        <value>*:/platform/administrators</value>
        <value>*:/organization/management/executive-board</value>
      </values-param>
      <values-param>
        <name>portal.creation.roles</name>
        <value>*:/platform/administrators</value>
        <value>*:/organization/management/executive-board</value>
      </values-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

3.5. Portal Navigation Configuration

3.5.1. Overview

There are three types of navigation available to portal users:

- [Section 3.5.2, "Portal Navigation"](#)
- [Section 3.5.3, "Group Navigation"](#)
- [Section 3.5.4, "User Navigation"](#)

These navigations are configured using standard XML syntax in the file; `"02portal.war:/WEB-INF/conf/portal/portal-configuration.xml"`.

```
<component>
```

```
<key>org.exoplatform.portal.config.UserPortalConfigService</key>
<type>org.exoplatform.portal.config.UserPortalConfigService</type>
<component-plugins>
<component-plugin>
  <name>new.portal.config.user.listener</name>
  <set-method>initListener</set-method>
  <type>org.exoplatform.portal.config.NewPortalConfigListener</type>
  <description>this listener init the portal configuration</description>
  <init-params>
    <value-param>
      <name>default.portal</name>
      <description>The default portal for checking db is empty or not</description>
      <value>classic</value>
    </value-param>
    <object-param>
      <name>portal.configuration</name>
      <description>description</description>
      <object type="org.exoplatform.portal.config.NewPortalConfig">
        <field name="predefinedOwner">
          <collection type="java.util.HashSet">
            <value><string>classic</string></value>
            <value><string>webos</string></value>
          </collection>
        </field>
        <field name="ownerType"><string>portal</string></field>
        <field name="templateLocation"><string>war:/conf/portal</string></field>
      </object>
    </object-param>
    <object-param>
      <name>group.configuration</name>
      <description>description</description>
      <object type="org.exoplatform.portal.config.NewPortalConfig">
        <field name="predefinedOwner">
          <collection type="java.util.HashSet">
            <value><string>platform/administrators</string></value>
            <value><string>platform/users</string></value>
            <value><string>platform/guests</string></value>
            <value><string>organization/management/executive-board</string></value>
          </collection>
        </field>
        <field name="ownerType"><string>group</string></field>
        <field name="templateLocation"><string>war:/conf/portal</string></field>
      </object>
    </object-param>
  </init-params>
</component-plugin>
</component-plugins>
```

```

<object-param>
  <name>user.configuration</name>
  <description>description</description>
  <object type="org.exoplatform.portal.config.NewPortalConfig">
    <field name="predefinedOwner">
      <collection type="java.util.HashSet">
        <value><string>root</string></value>
        <value><string>john</string></value>
        <value><string>mary</string></value>
        <value><string>demo</string></value>
      </collection>
    </field>
    <field name="ownerType"><string>user</string></field>
    <field name="templateLocation"><string>war:/conf/portal</string></field>
  </object>
</object-param>
</init-params>
</component-plugin>
</component-plugins>

```

This XML configuration defines where in the portal's war to look for configuration, and what portals, groups, and user specific views to include in portal/group/user navigation. Those files will be used to create an initial navigation, the first time the portal is launched. That information will then be stored in JCR content repository, and can then be modified, and managed from the portal UI.

3.5.2. Portal Navigation

The portal navigation incorporates the pages that can be accessed even when a user is not logged in (assuming the applicable permissions allow public access). For example; several portal navigations are used when a company owns multiple trademarks, and sets up a web site for each of them.

The **classic** portal is configured by four XML files in `02portal.war:/WEB-INF/conf/portal/portal/classic` directory:

portal.xml

This file describes the layout and portlets that will be shown on all pages. Usually the layout contains the banner, footer, menu and breadcrumbs portlets. GateIn 3.1 is extremely configurable as every view element (even the banner and footer) is a portlet.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<portal-config>
  <portal-name>classic</portal-name>
  <locale>en</locale>

```

```
<factory-id>office</factory-id>
<access-permissions>Everyone</access-permissions>
<edit-permission>*/platform/administrators</edit-permission>
<creator>root</creator>

<portal-layout>
<application>
  <instance-id>portal#classic:/web/BannerPortlet/banner</instance-id>
  <show-info-bar>false</show-info-bar>
</application>
<application>
  <instance-id>portal#classic:/web/NavigationPortlet/toolbar</instance-id>
  <show-info-bar>false</show-info-bar>
</application>

<application>
  <instance-id>portal#classic:/web/BreadcrumbsPortlet/breadcrumbs</instance-id>
  <show-info-bar>false</show-info-bar>
</application>

<page-body> </page-body>

<application>
  <instance-id>portal#classic:/web/FooterPortlet/footer</instance-id>
  <show-info-bar>false</show-info-bar>
</application>
</portal-layout>

</portal-config>
```

It is also possible to apply a nested container that can also contain portlets. Row, column or tab containers are then responsible for the layout of their child portlets.

Each application references a portlet using the id `portal#{portalName}:{portletWarName}/{portletName}/{uniqueId}`

Use the `page-body` tag to define where GateIn 3.1 should render the current page.

The defined **classic** portal is accessible to "Everyone" (at `/portal/public/classic`) but only members of the group `/platform/administrators` can edit it.

navigation.xml

This file defines all the navigation nodes the portal will have. The syntax is simple, using nested node tags. Each node references a page defined in `pages.xml` file (explained next).

When `#{ . . . }` syntax is used, the enclosed property name serves as a key that is automatically passed to internationalization mechanism so the literal property name is replaced by a localized value taken from the associated properties file matching the current locale.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<node-navigation
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_objects_1_0 http://
www.gatein.org/xml/ns/gatein_objects_1_0"
  xmlns="http://www.gatein.org/xml/ns/gatein_objects_1_0">
  <priority>1</priority>
  <page-nodes>
    <node>
      <uri>home</uri>
      <name>home</name>
      <label>#{portal.classic.home}</label>
      <page-reference>portal::classic::homepage</page-reference>
    </node>
    <node>
      <uri>sitemap</uri>
      <name>sitemap</name>
      <label>#{portal.classic.sitemap}</label>
      <visibility>DISPLAYED</visibility>
      <page-reference>portal::classic::sitemap</page-reference>
    </node>
  </page-nodes>
</node-navigation>
```

This navigation tree can have multiple views inside portlets (such as the breadcrumbs portlet) that render the current view node, the site map or the menu portlets.



Warning

For top nodes, the **uri** and the **name** of your navigation nodes must have the *same* value. For other nodes the **uri** is a relative path. For example, *contentmanagement/fileexplorer* where 'contentmanagement' is the name of the parent node and 'fileexplorer' is the name of the node (`<name>fileexplorer</name>`).

pages.xml

This configuration file structure is very similar to `portal.xml` and it can also contain container tags. Each application can decide whether to render the portlet border, the window state, the icons or portlet's mode.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<page-set
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_objects_1_0 http://
www.gatein.org/xml/ns/gatein_objects_1_0"
  xmlns="http://www.gatein.org/xml/ns/gatein_objects_1_0">

  <page>
    <name>homepage</name>
    <title>Home Page</title>
    <access-permissions>Everyone</access-permissions>
    <edit-permission>*:/platform/administrators</edit-permission>
    <portlet-application>
      <portlet>
        <application-ref>web</application-ref>
        <portlet-ref>HomePagePortlet</portlet-ref>
        <preferences>
          <preference>
            <name>template</name>
            <value>system:/templates/groovy/webui/component/UIHomePagePortlet.gtmpl</
value>
            <read-only>>false</read-only>
          </preference>
        </preferences>
      </portlet>
      <title>Home Page portlet</title>
      <access-permissions>Everyone</access-permissions>
      <show-info-bar>>false</show-info-bar>
      <show-application-state>>false</show-application-state>
      <show-application-mode>>false</show-application-mode>
    </portlet-application>
  </page>
  <page>
    <name>sitemap</name>
    <title>Site Map</title>
    <access-permissions>Everyone</access-permissions>
    <edit-permission>*:/platform/administrators</edit-permission>
```



```

<portlet-application>
  <portlet>
    <application-ref>web</application-ref>
    <portlet-ref>SiteMapPortlet</portlet-ref>
  </portlet>
  <title>SiteMap</title>
  <access-permissions>Everyone</access-permissions>
  <show-info-bar>false</show-info-bar>
</portlet-application>
</page>
</page-set>

```

portlet-preferences.xml

Portlet instances can be associated with `portlet-preferences` that override the ones defined in `portlet.xml` of the portlet application war (TODO: clarify which file in which war).

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<portlet-preferences-set>
  <portlet-preferences>
    <owner-type>portal</owner-type>
    <owner-id>classic</owner-id>
    <window-id>portal#classic:/web/BannerPortlet/banner</window-id>
    <preference>
      <name>template</name>
      <value>par:/groovy/groovy/webui/component/UIBannerPortlet.gtmpl</value>
      <read-only>false</read-only>
    </preference>
  </portlet-preferences>
  <portlet-preferences>
    <owner-type>portal</owner-type>
    <owner-id>classic</owner-id>
    <window-id>portal#classic:/web/NavigationPortlet/toolbar</window-id>
    <preference>
      <name>useAJAX</name>
      <value>true</value>
      <read-only>false</read-only>
    </preference>
  </portlet-preferences>
  <portlet-preferences>
    <owner-type>portal</owner-type>
    <owner-id>classic</owner-id>
    <window-id>portal#classic:/web/FooterPortlet/footer</window-id>
    <preference>

```

```
<name>template</name>
<value>par:/groovy/groovy/webui/component/UIFooterPortlet.gtmpl</value>
<read-only>>false</read-only>
</preference>
</portlet-preferences>

<portlet-preferences>
  <owner-type>portal</owner-type>
  <owner-id>classic</owner-id>
  <window-id>portal#classic:/web/GroovyPortlet/groovypportlet</window-id>
  <preference>
    <name>template</name>
    <value>par:/groovy/groovy/webui/component/UGroovyPortlet.gtmpl</value>
    <read-only>>false</read-only>
  </preference>
</portlet-preferences>
</portlet-preferences-set>
```

3.5.3. Group Navigation

Group navigations are dynamically added to the user navigation at login. This allows users to see in the menu all the pages assigned to any groups they belong to.

The group navigation menu is configured by three XML files (`navigation.xml`, `pages.xml` and `portlet-preferences.xml`). The syntax used in these files is the same as those covered in [Section 3.5.2, “Portal Navigation”](#).

They are located in `portal.war/WEB-INF/conf/portal/group/group-name-path/` directory (For example; `portal.war/WEB-INF/conf/portal/group/platform/administrators/`).

3.5.4. User Navigation

User navigation is the set of nodes and pages that are owned by a user. They are part of the user's dashboard.

Three files configure the user navigation (`navigation.xml`, `pages.xml` and `portlet-preferences.xml`). They are located in the directory `"portal.war/WEB-INF/conf/portal/users/{userName}"`.

This directory also contains a `gadgets.xml` file (formerly called `widgets.xml`). This file defines the gadgets located in the user's workspace.

The user's workspace is located at the left hand side of the page and access is restricted to some privileged users, see [Section 6.1, “Predefined User Configuration”](#)

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<widgets>
  <owner-type>user</owner-type>
  <owner-id>root</owner-id>

  <container id="Information">
    <name>Information</name>
    <description>Information's Description</description>
    <application>
      <instance-id>user#root:/GateInWidgetWeb/WelcomeWidget/WelcomeWidget1</instance-id>
      <application-type>GateInWidget</application-type>
    </application>

    <application>
      <instance-id>user#root:/GateInWidgetWeb/StickerWidget/StickerWidget</instance-id>
      <application-type>GateInWidget</application-type>
    </application>

    <application>
      <instance-id>user#root:/GateInWidgetWeb/InfoWidget/InfoWidget1</instance-id>
      <application-type>GateInWidget</application-type>
    </application>
  </container>

  <container id="Calendar">
    <name>Calendar</name>
    <description>Calendar's Description</description>
    <application>
      <instance-id>user#root:/GateInWidgetWeb/CalendarWidget/CalendarWidget</instance-id>
      <application-type>GateInWidget</application-type>
    </application>
  </container>

</widgets>

```

3.5.5. Tips

3.5.5.1. Direct External Links

If you wish to add a link to a URL outside the portal you first have to define a "page" that will only serve as a navigation placeholder for external redirect - it will not be used for any portlets. Then add the URL to the navigation. (TODO: check for correctness)

pages.xml

```
<page>
  <owner-type>portal</owner-type>
  <owner-id>website</owner-id>
  <name>documentation</name>
  <title>Documentation</title>
  <access-permissions>Everyone</access-permissions>
  <edit-permission>*/platform/administrators</edit-permission>
</page>
```

navigation.xml

```
<node>
  <uri>http://wiki.exoplatform.com/xwiki/bin/view/Main/WebHome</uri>
  <name>documentation</name>
  <label>#{portal.classic.documentation}</label>
  <page-reference>portal::website::documentation</page-reference>
</node>
```



Direct external links were not a design goal

Currently you cannot modify the URL using the portal interface, you must change it in the configuration files or modify the underlying database table.

3.6. Internationalization Configuration

3.6.1. Overview



Assumed Knowledge

GateIn 3.1 is fully configurable for internationalization, however users should have a general knowledge of Internationalization in Java products before attempting these configurations.

Sun Java hosts a comprehensive guide to internationalizing java products at <http://java.sun.com/docs/books/tutorial/i18n/TOC.html>.

All GateIn 3.1 applications contain property files for various languages. They are packaged with the portlets applications in a `WEB-INF/classes/locale/` directory.

These files are located in the `classes` folder of the WEB-INF directory, so as to be loaded by the ClassLoader.

All resource files are in a subfolder named `locale`.

For instance; the translations for the *NavigationPortlet* are located in `web.war/WEB-INF/classes/locale/portlet/portal`

```
NavigationPortlet_de.properties
NavigationPortlet_en.properties
NavigationPortlet_es.properties
NavigationPortlet_fr.properties
NavigationPortlet_nl.properties
NavigationPortlet_ru.properties
NavigationPortlet_uk.properties
NavigationPortlet_ar.xml
```

Inside those file are typical `key=value` Java EE properties. For example the French one:

```
javax.portlet.title=Portlet Navigation
```

There are also properties files in the portal itself. They form the **portal resource bundle**.

From a portlet you can then access translations from the portlet itself or shared at the portal level, both are aggregated when you need them.



Translation in XML format

It is also possible to use a proprietary XML format to define translations. This is a more convenient way to translate a document for some languages such as Japanese, Arabic or Russian. Property files have to be ASCII encoded, while the XML file can define its encoding. As a result it's easier for a human being to read (and fix) a translation in XML instead of having to decode and encode the property file.

For more information refer to: [Section 3.8, "XML Resources Bundles"](#)

3.6.2. Locales configuration

Various languages are available in the portal package. The configuration below will define which languages are shown in the "Change Language" section and made available to users.

The `02portal.war:/WEB-INF/conf/common/common-configuration.xml` file of your installation contains the following section:

```
<component>
  <key>org.exoplatform.services.resources.LocaleConfigService</key>
  <type>org.exoplatform.services.resources.impl.LocaleConfigServiceImpl</type>
  <init-params>
    <value-param>
      <name>locale.config.file</name>
      <value>war:/conf/common/locales-config.xml</value>
    </value-param>
  </init-params>
</component>
```

This configuration points to the locale configuration file.

The locale configuration file (`02portal.war:/WEB-INF/conf/common/locales-config.xml`) contains the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<locales-config>
  <locale-config>

    <locale>en</locale>                                1

    <output-encoding>UTF-8</output-encoding>            2

    <input-encoding>UTF-8</input-encoding>              3

    <description>Default configuration for english locale</description> 4
  </locale-config>

  <locale-config>
    <locale>fr</locale>
    <output-encoding>UTF-8</output-encoding>
    <input-encoding>UTF-8</input-encoding>
    <description>Default configuration for the french locale</description>
  </locale-config>

  <locale-config>
    <locale>ar</locale>
    <output-encoding>UTF-8</output-encoding>
    <input-encoding>UTF-8</input-encoding>
    <description>Default configuration for the arabic locale</description>
```

```

    <orientation>rt</orientation>
  </locale-config>
</locales-config>

```

5

- ① *locale* The locale has to be defined such as defined here <http://ftp.ics.uci.edu/pub-ietf-http-related-iso639.txt>. In this example "ar" is Arabic.
- ② *output-encoding* deals with character encoding. It is recommended that **UTF-8** be used.
- ③ *input-encoding* In the java implementation, the encoding parameters will be used for the request response stream. The input-encoding parameter will be used for request `setCharacterEncoding(..)`.
- ④ *description* Description for the language
- ⑤ *orientation* The default orientation of text and images is Left-To-Right. GateIn 3.1 supports **Right-To-Left** orientation. Modifying text orientation is explained in [Section 3.7, "RTL \(Right To Left\) Framework"](#).

3.6.3. ResourceBundleService

The resource bundle service is configured in: `02portal.war:/WEB-INF/conf/common/common-configuration.xml`:

```

<component>
  <key>org.exoplatform.services.resources.ResourceBundleService</key>
  <type>org.exoplatform.services.resources.impl.SimpleResourceBundleService</type>
  <init-params>
    <values-param>
      <name>classpath.resources</name>
      <description>The resources that start with the following package name should be load from
file system</description>
      <value>locale.portlet</value>
    </values-param>
    <values-param>
      <name>init.resources</name>
      <description>Initiate the following resources during the first launch</description>
      <value>locale.portal.expression</value>
      <value>locale.portal.services</value>
      <value>locale.portal.webui</value>
      <value>locale.portal.custom</value>
      <value>locale.navigation.portal.classic</value>
      <value>locale.navigation.group.platform.administrators</value>
      <value>locale.navigation.group.platform.users</value>
    </values-param>
  </init-params>
</component>

```

1

2

```
<value>locale.navigation.group.platform.guests</value>
<value>locale.navigation.group.organization.management.executive-board</value>
</values-param>
<values-param>

<name>portal.resource.names</name>
<description>The properties files of the portal, those files will be merged
into one ResourceBundle properties </description>
<value>locale.portal.expression</value>
<value>locale.portal.services</value>
<value>locale.portal.webui</value>
<value>locale.portal.custom</value>
</values-param>
</init-params>
</component>
```

- ❶ *classpath.resources* are discussed in a later section.
- ❷ *init.resources* TODO
- ❸ *portal.resource.names* Defines all resources that belong to the *Portal Resource Bundle*.

These resources are merged to a single resource bundle which is accessible from anywhere in GateIn 3.1. All these keys are located in the same bundle, which is separated from the navigation resource bundles.

3.6.4. Navigation Resource Bundles

There is a resource bundle for each navigation. A navigation can exist for user, groups, and portal.

The previous example shows bundle definitions for the navigation of the classic portal and of four different groups. Each of these resource bundles occupies a different sphere, they are independent of each other and they are not included in the *portal.resource.names* parameter.

The properties for a group must be in the `WEB-INF/classes/locale/navigation/group/` folder. `/WEB-INF/classes/locale/navigation/group/organization/management/executive-board_en.properties`, for example.

The folder and file names must correspond to the group hierarchy. The group name "*executive-board*" is followed by the iso 639 code.

For each language defined in *LcalesConfig* must have a resource file defined. If the name of a group is changed the name of the folder and/or files of the correspondent navigation resource bundles must also be changed.

Content of `executive-board_en.properties`:


```
organization.title=Organization
organization.newstaff=New Staff
organization.management=Management
```

This resource bundle is only accessible for the navigation of the *organization.management.executive-board* group.

3.6.5. Portlets

Portlets are independent applications and deliver their own resource files.

All shipped portlet resources are located in the **locale/portlet** subfolder. The `ResourceBundleService` parameter **classpath.resources** defines this subfolder.

Procedure 3.1. Example

1. To add a Spanish translation to the *GadgetPortlet*.
2. Create the file `GadgetPortlet_es.properties` in: `WEB-INF/classes/locale/portlet/gadget/GadgetPortlet`.
3. In `portlet.xml`, add *Spanish* as a **supported-locale** ('es' is the 2 letters code for Spanish), the **resource-bundle** is already declared and is the same for all languages :

```
<supported-locale>en</supported-locale>
<supported-locale>es</supported-locale>
<resource-bundle>locale.portlet.gadget.GadgetPortlet</resource-bundle>
```

See the portlet specification for more details about portlet internationalization.

3.6.5.1. Standard portlet resource keys

The portlet specifications defines three standard keys: Title, Short Title and Keywords. Keywords is formatted as a comma-separated list of tags.

```
javax.portlet.title=Breadcrumbs Portlet
javax.portlet.short-title=Breadcrumbs
javax.portlet.keywords=Breadcrumbs, Breadcrumb
```

3.6.5.2. Debugging resource bundle usage

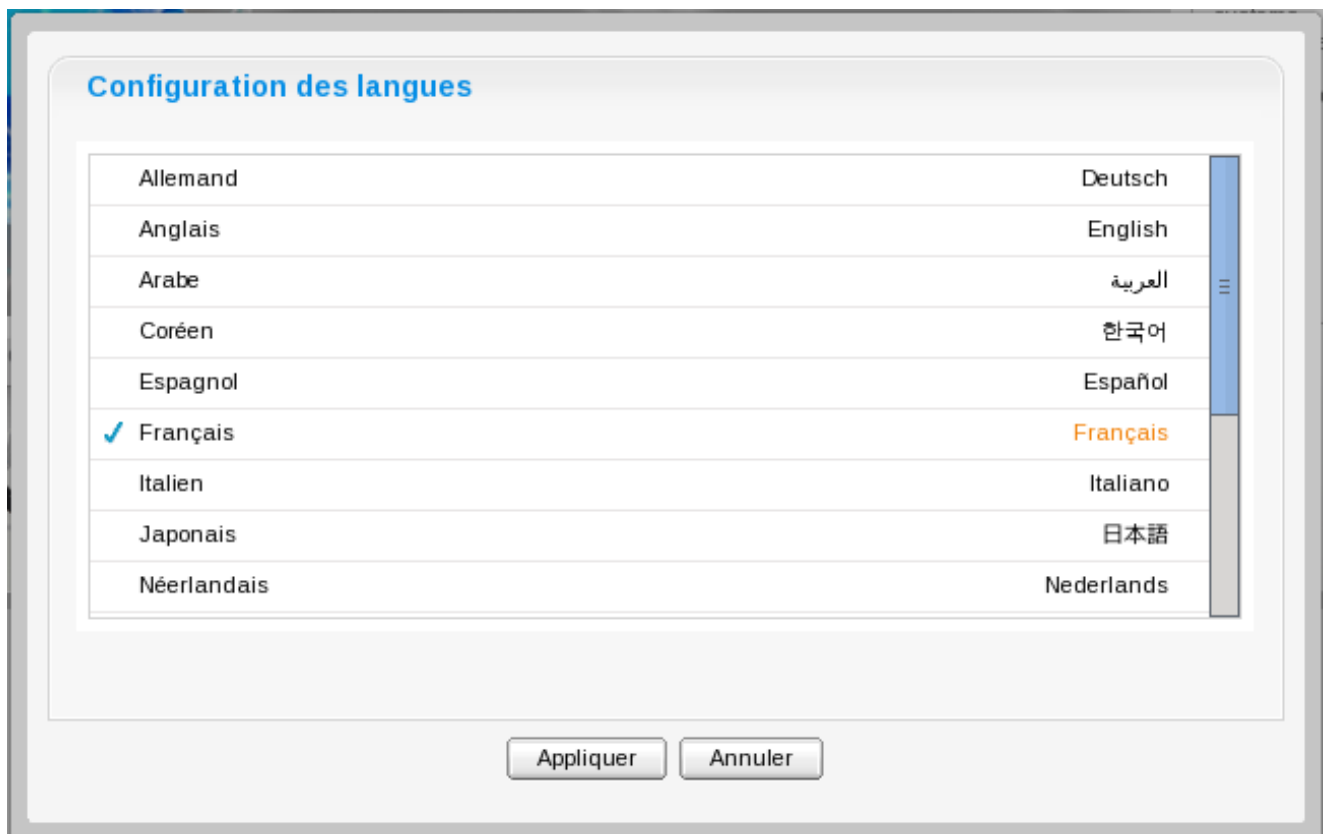
When translating an application it can sometimes be difficult to find the right key for a given property.

Execute the portal in **debug mode** and select, from the available languages, select the special language; **Magic locale**.

This feature translates a key to the same key value.

For example, the translated value for the key `"organization.title"` is simply the value `"organization.title"`. Selecting that language allows use of the portal and its applications with all the keys visible. This makes it easier to find out the correct key for a given label in the portal page.

3.6.6. Translating the language selection form



When choosing a language as on the screenshot above, the user is presented with a list of languages on the left side in the current chosen language and on the right side, the same language translated into its own language. Those texts are obtained from the JDK API `java.util.Locale.getDisplayedLanguage()` and `java.util.Locale.getDisplayedCountry()` (if needed) and all languages may not be translated and can also depend on the JVM currently used. It is still possible to override those values by editing the `locale.portal.webui` resource bundle, to do so edit the file `gatein.ear/02portal.war/WEB-INF/classes/locale/portal/webui_xx_yy.properties` where `xx_yy` represents the country code of the language in which you want to translate a particular language. In that file, add or modify a key such as `Locale.xx_yy` with the value being the translated string.

Example 3.1. Changing the displayed text for Traditional Chinese in French

First edit `gatein.ear/02portal.war/WEB-INF/classes/locale/portal/webui_fr.properties` where `ne` is the country code for French, and add the following key into it:

```
Locale.zh_TW=Chinois traditionnel
```

After a restart the language will be updated in the user interface when a user is trying to change the current language.

3.7. RTL (Right To Left) Framework

The text orientation depends on the current locale setting. The orientation is a Java 5 enum that provides a set of functionalities:

```
LT, // Western Europe
RT, // Middle East (Arabic, Hebrew)
TL, // Japanese, Chinese, Korean
TR; // Mongolian
public boolean isLT() { ... }
public boolean isRT() { ... }
public boolean isTL() { ... }
public boolean isTR() { ... }
```

The object defining the Orientation for the current request is the `UIPortalApplication`. However it should be accessed at runtime using the `RequestContext` that delegates to the `UIPortalApplication`.

In the case of a `PortalRequestContext` it is a direct delegate as the `PortalRequestContext` has a reference to the current `UIPortalApplication`.

In the case of a different context such as the `PortletRequestContext`, it delegates to the parent context given the fact that the root `RequestContext` is always a `PortalRequestContext`.

3.7.1. Groovy templates

Orientation is defined by implicit variables in the groovy binding context:

Orientation

The current orientation as an Orientation

isLT

The value of orientation.isLT()

isRT

The value of orientation.isRT()

dir

The string 'ltr' if the orientation is LT or the string 'rtl' if the orientation is RT.

3.7.2. Stylesheet

The skin service handles stylesheet rewriting to accommodate the orientation. It works by appending -lt or -rt to the stylesheet name.

For instance: `/web/skin/portal/webui/component/UIFooterPortlet/DefaultStylesheet-rt.css` will return the same stylesheet as `/web/skin/portal/webui/component/UIFooterPortlet/DefaultStylesheet.css` but processed for the RT orientation. The `-lt` suffix is optional.

Stylesheet authors can annotate their stylesheet to create content that depends on the orientation.

Example 1. In the example we need to use the orientation to modify the float attribute that will make the horizontal tabs either float on left or on right:

```
float: left; /* orientation=lt */
float: right; /* orientation=rt */
font-weight: bold;
text-align: center;
white-space: nowrap;
```

The LT produced output will be:

```
float: left; /* orientation=lt */
font-weight: bold;
text-align: center;
white-space: nowrap;
```

The RT produced output will be:

```
float: right; /* orientation=rt */
```

```
font-weight: bold;  
text-align: center;  
white-space: nowrap;
```

Example 2. In this example we need to modify the padding according to the orientation:

```
color: white;  
line-height: 24px;  
padding: 0px 5px 0px 0px; /* orientation=lt */  
padding: 0px 0px 0px 5px; /* >orientation=rt */
```

The LT produced output will be:

```
color: white;  
line-height: 24px;  
padding: 0px 5px 0px 0px; /* orientation=lt */
```

The RT produced output will be:

```
color: white;  
line-height: 24px;  
padding: 0px 0px 0px 5px; /* orientation=rt */
```

3.7.3. Images

Sometimes it is necessary to create an RT version of an image that will be used from a template or from a stylesheet. However symmetric images can be automatically generated avoiding the necessity to create a mirrored version of an image and furthermore avoiding maintenance cost.

The web resource filter uses the same naming pattern as the skin service. When an image ends with the -rt suffix the portal will attempt to locate the original image and create a mirror of it.

For instance: requesting the image `/GateInResources/skin/DefaultSkin/webui/component/UITabSystem/UITabs/background/NormalTabStyle-rt.gif` returns a mirror of the image `/GateInResources/skin/DefaultSkin/webui/component/UITabSystem/UITabs/background/NormalTabStyle.gif`.



Note

It is important to consider whether the image to be mirrored is symmetrical as this will impact it's final appearance.

Here is an example combining stylesheet and images:

```
line-height: 24px;
background: url('background/NavigationTab.gif') no-repeat right top; /* orientation=lt */
background: url('background/NavigationTab-rt.gif') no-repeat left top; /* orientation=rt */
padding-right: 2px; /* orientation=lt */
padding-left: 2px; /* orientation=rt */
```

3.7.4. Client side JavaScript

The `eXo.core.I18n` object provides the following parameters for orientation:

`getOrientation()`

Returns either the string `lt` or `rt`

`getDir()`

Returns either the string `ltr` or `rtl`

`isLT()`

Returns true for `LT`

`isRT()`

Returns true of `RT`

3.8. XML Resources Bundles

3.8.1. Motivation

Resource bundles are usually stored in property files. However, as property files are plain files, issues with the encoding of the file may arise. The XML resource bundle format has been developed to provide an alternative to property files.

- The XML format declares the encoding of the file. This avoids use of the `native2ascii` program which can interfere with encoding.
- Property files generally use ISO 8859-1 character encoding which does not cover the full unicode charset. As a result, languages such as Arabic would not be natively supported.

- Tooling for XML files is better supported than the tooling for Java property files and thus the XML editor copes well with the file encoding.

3.8.2. XML format

The XML format is very simple and has been developed based on the *DRY* (Don't Repeat Yourself) principle. Usually resource bundle keys are hierarchically defined and we can leverage the hierarchic nature of the XML for that purpose. Here is an example of turning a property file into an XML resource bundle file:

```
UIAccountForm.tab.label.AccountInputSet = ...
UIAccountForm.tab.label.UIUserProfileInputSet = ...
UIAccountForm.label.Profile = ...
UIAccountForm.label.HomeInfo= ...
UIAccountForm.label.BusinessInfo= ...
UIAccountForm.label.password= ...
UIAccountForm.label.Confirmpassword= ...
UIAccountForm.label.email= ...
UIAccountForm.action.Reset= ...
```

```
<?xml version="1.0" encoding="UTF-8"?>
<bundle>
  <UIAccountForm>
    <tab>
      <label>
        <AccountInputSet>...</AccountInputSet>
        <UIUserProfileInputSet>...</UIUserProfileInputSet>
      </label>
    </tab>
    <label>
      <Profile>...</Profile>
      <HomeInfo>...</HomeInfo>
      <BusinessInfo>...</BusinessInfo>
      <password>...</password>
      <Confirmpassword>...</Confirmpassword>
      <email>...</email>
    </label>
    <action>
      <Reset>...</Reset>
    </action>
  </UIAccountForm>
</bundle>
```

3.8.3. Portal support

In order to be loaded by the portal at runtime (actually the resource bundle service), the name of the file must be the same as a property file and it must use the **.xml** suffix.

For example; for the Account Portlet to be displayed in Arabic, the resource bundle would be **AccountPortlet_ar.xml** rather than **AccountPortlet_ar.properties**.

3.9. JavaScript Inter Application Communication

3.9.1. Overview

JavaScript Inter Application Communication is designed to allow applications within a page to exchange data. This library is made for broadcasting messages on topic.

It is based on 3 functions:

- Subscribe.
- Publish.
- Unsubscribe.

A subscription to a topic will receive any subtopic messages. For example; An application subscribed to `/eXo/application` will receive messages sent on the `/eXo/application/map` topic. A message sent on `/eXo`, however, would not be received.

Subscription Topics

`/eXo`

This topic contains all the events generated by the platform.

`/eXo/portal/notification`

A message is sent on this topic will prompt a popup notification in the top right of the screen.

3.9.2. Library

The Inter Application Communication library is found in `01eXoResources.war:/javascript/eXo/core/Topic.js`

```
/**
 * publish is used to publish an event to the other subscribers to the given channels
 * @param {Object} senderId is a string that identify the sender
```



```

* @param {String} topic is the topic that the message will be published
* @param {Object} message is the message that's going to be delivered to the subscribers to
the topic
*/
Topic.prototype.publish = function(/*Object*/ senderId, /*String*/ topicName, /*Object*/ message
) { ... }

/**
* isSubscribed is used to check if a function receive the events from a topic
* @param {String} topic The topic.
* @param {Function} func is the name of the function of obj to call when a message is received
on the topic
*/
Topic.prototype.isSubscribed = function(/*String*/ topic, /*Function*/ func) { ... }

/**
* subscribe is used to subscribe a callback to a topic
* @param {String} topic is the topic that will be listened
* @param {Function} func is the name of the function of obj to call when a message is received
on the topic
*
* func is a function that take a Object in parameter. the event received have this format:
* {senderId:senderId, message:message, topic: topic}
*
*/
Topic.prototype.subscribe = function(/*String*/ topic, /*Function*/ func) { ... }

/**
* unsubscribe is used to unsubscribe a callback to a topic
* @param {String} topic is the topic
* @param {Object} id is the id of the listener we want to unsubscribe
*/
Topic.prototype.unsubscribe = function(/*String*/ topic, /*Object*/ id) { ... }

Topic.prototype.initCometdBridge = function() { ... }

```

3.9.3. Syntax

The three messaging functions require particular objects and definitions in their syntax:

Subscribe

The `subscribe` function is used to subscribe a callback to a topic. It uses the following parameters:

topic

The topic that will be listened for.

func

The name of the object function to call when a message is received on the topic. It has to be a function that takes an Object parameter. The event received will have this format:

```
{
  senderId:senderId,
  message:message,
  topic: topic
}
```

Publish

The `publish` function is used to publish an event to the other subscribed applications through the given channels. Its parameters are:

senderId

This is a string that identifies the sender.

topicName

The topic that the message will be published.

message

This is the message body to be delivered to the subscribers to the topic.

Unsubscribe

The `unsubscribe` function is used to unsubscribe a callback to a topic. The required parameters are:

topic

The topic that will be unsubscribed from.

id

This is the context object.

3.9.4. Example of Javascript events usage

```
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>
<div>
  <p>
    Received messages:
    <div id="received_<portlet:namespace/>">

  </div>
```

```

</p>

<p>
  Send message:
      <input type="text" id="msg_<portlet:namespace/>"/> <a href="#"
onclick="send_<portlet:namespace/>();">send</a>
</p>
</div>

<script type="text/javascript">

Function.prototype.bind = function(object) {
  var method = this;
  return function() {
    method.apply(object, arguments);
  }
}

function send_<portlet:namespace/>() {
  var msg = document.getElementById("msg_<portlet:namespace/>").value;
  eXo.core.Topic.publish("<portlet:namespace/>", "/demo", msg);
}

function Listener_<portlet:namespace/>(){

}

Listener_<portlet:namespace/>.prototype.receiveMsg = function(event) {
  document.getElementById("received_<portlet:namespace/>").innerHTML =
    document.getElementById("received_<portlet:namespace/>").innerHTML + "<br />* " +
    event.senderId + ": " + event.message;
}

function init_<portlet:namespace/>() {
  var listener_<portlet:namespace/> = new Listener_<portlet:namespace/>();
  eXo.core.Topic.subscribe("/demo", listener_<portlet:namespace/>
>.receiveMsg.bind(listener_<portlet:namespace/>));
}

init_<portlet:namespace/>();
</script>

```

3.10. Upload Component

3.10.1. Upload Service

The service is defined by the class: `org.exoplatform.upload.UploadService`;

This can be configured with the following xml code:

```
<component>
  <type>org.exoplatform.upload.UploadService</type>
  <init-params>
    <value-param>
      <name>upload.limit.size</name>
      <description>Maximum size of the file to upload in MB</description>
      <value>10</value>
    </value-param>
  </init-params>
</component>
```

This code allows for a default upload size limit for the service to be configured. The value unit is in MegaBytes.

This limit will be used by default by all applications if no application-specific limit is set. Setting a different limit for applications is discussed in a later section.

If the value is set at *0* the upload size is unlimited.

Procedure 3.2. How to use the upload component

1. Create an object type `org.exoplatform.webui.form.UIFormUploadInput`.

Two constructors are available for this:

```
public UIFormUploadInput(String name, String bindingExpression)
```

or:

```
public UIFormUploadInput(String name, String bindingExpression, int limit)
```

This is an example using the second form :

```

PortletRequestContext pcontext =
(PortletRequestContext)WebuiRequestContext.getCurrentInstance();
PortletPreferences portletPref = pcontext.getRequest().getPreferences();
int limitMB = Integer.parseInt(portletPref.getValue("uploadFileSizeLimitMB", "").trim());
UIFormUploadInput uiInput = new UIFormUploadInput("upload", "upload", limitMB);

```

2. To obtain the limit from the `xml` configuration, this piece of code can be added to the either `portlet.xml` or `portlet-preferences.xml` :

```

<preference>
  <name>uploadFileSizeLimitMB</name>
  <value>30</value>
  <read-only>false</read-only>
</preference>

```

Again, a `0` value means an unlimited upload size, and the value unit is set in MegaBytes.

3. Use the `getUploadDataAsStream()` method to get the uploaded data:

```

UIFormUploadInput input = (UIFormUploadInput)uiForm.getUIInput("upload");
InputStream inputStream = input.getUploadDataAsStream();
...
jcrData.setValue(inputStream);

```

4. The upload service stores a temporary file on the filesystem during the upload process. When the upload is finished, the service must be cleaned in order to:

1. Delete the temporary file.
2. Delete the classes used for the upload.

Use the `removeUpload()` method defined in the upload service to purge the file:

```

UploadService uploadService = uiForm.getApplicationComponent(UploadService.class) ;
UIFormUploadInput uiChild = uiForm.getChild(UIFormUploadInput.class) ;
uploadService.removeUpload(uiChild.getUploadId()) ;

```



Saving the uploaded file

Ensure the file is saved **before** the service is cleaned.

3.11. Deactivation of the Ajax Loading Mask Layer

3.11.1. Purpose

The loading mask layer is deployed after an ajax-call. Its purpose is to block the GUI in order to prevent further user actions until the the ajax-request has been completed.

However, the mask layer may need to be deactivated in instances where the portal requires user instructions before previous instructions have been carried out.

Procedure 3.3. How to deactivate the ajax-loading mask

1. Generate a script to make an asynchronous ajax-call. Use the `uicomponent.doAsync()` method rather than the `uicomponent.event()` method.

For example:

```
<a href="<%=uicomponent.doAsync(action, beanId, params)%>" alt="">Asynchronous</a>
```

2. The `doAsync()` method automatically adds the following new parameter into the parameters list; `asynccparam = new Parameter(AJAX ASYNC,"true"); (AJAX ASYNC == "ajax async")`

This request is asynchronous and the ajax-loading mask will not deployed.



Note

An asynchronous request can still be made using the `uicomponent.event()`. When using this method, however, the `asynccparam` must be added manually.

The GUI will be blocked to ensure a user can only request one action at a time and while the request seems to be synchronous, all ajax requests are, in fact always asynchronous. For further information refer to [Section 3.11.2, "Synchronous issue"](#).

3.11.2. Synchronous issue

Most web browsers support ajax requests in two modes: *Synchronous* and *Asynchronous*. This mode is specified with a boolean `bAsync` parameter.

```
var bAsync = false; // Synchronous
request.open(instance.method, instance.url, bAsync);
```

However, in order to work with browsers that do not support *Synchronous* requests, `bAsync` is set to always be true (Ajax request will always be asynchronous).

```
// Asynchronous request
request.open(instance.method, instance.url, true);
```

3.12. JavaScript Configuration

Managing JavaScript in an application like GateIn 3.1 is a critical part of the configuration work. Configuring the scripts correctly will result in a faster response time from the portal.

Every portlet can have its own JavaScript code but in many cases, it is more convenient to reuse some existing shared libraries. For that reason, GateIn 3.1 has a mechanism to easily register the libraries that will be loaded when every page is rendered.

To do so, every WAR deployed in GateIn 3.1 can register the `.js` files with the **gatein-resources.xml** configuration file.

The example code snippet below is found in the **gatein-resources.xml** in the **eXoResources.war** file.

```
<javascript>
  <param>
    <js-module>eXo</js-module>
    <js-path>/javascript/eXo.js</js-path>
    <js-priority>0</js-priority>
  </param>
</javascript>

<!-- CORE Javascripts -->
<javascript>
  <param>
    <js-module>eXo.core.Utills</js-module>
    <js-path>/javascript/eXo/core/Util.js</js-path>
    <js-priority>1</js-priority>
  </param>
  <param>
    <js-module>eXo.core.DOMUtil</js-module>
    <js-path>/javascript/eXo/core/DOMUtil.js</js-path>
```

```
<js-priority>1</js-priority>
</param>
<param>
  <js-module>eXo.core.Browser</js-module>
  <js-path>/javascript/eXo/core/Browser.js</js-path>
  <js-priority>2</js-priority>
</param>
<param>
  <js-module>eXo.core.MouseEventManager</js-module>
  <js-path>/javascript/eXo/core/EventManager.js</js-path>
</param>
<param>
  <js-module>eXo.core.UIMaskLayer</js-module>
  <js-path>/javascript/eXo/core/UIMaskLayer.js</js-path>
</param>
<param>
  <js-module>eXo.core.Skin</js-module>
  <js-path>/javascript/eXo/core/Skin.js</js-path>
</param>
<param>
  <js-module>eXo.core.DragDrop</js-module>
  <js-path>/javascript/eXo/core/DragDrop.js</js-path>
</param>
<param>
  <js-module>eXo.core.DragDrop2</js-module>
  <js-path>/javascript/eXo/core/DragDrop2.js</js-path>
</param>
</javascript>
```

Note that registered JavaScript files will be merged into a single `merged.js` file when the server loads. This reduces the number of HTTP calls as seen in the home page source code:

```
<script type="text/javascript" src="/portal/javascript/merged.js"></script>
```

Although this optimization is useful for a production environment, it may be easier to deactivate this optimization while debugging JavaScript problems.

To do this, set the java system property `exo.product.developing` to `true`. GateIn provides two startup scripts that define this property in `gatein-dev.sh` (for Linux, Mac) and `gatein-dev.bat` (for Windows).

To generate the `merged.js` file, set this property to `false`. If the property is not set, the default value is `false`.

The property can be passed as a JVM parameter with the `-D` option in your `GateIn.sh` or `GateIn.bat` startup script.

Every JavaScript file is associated with a module name which acts as a namespace.

Inside the associated JavaScript files, the eXo JavaScript objects are exposed as global variables named after the module.

For example:

```
eXo.core.DragDrop = new DragDrop();
```

It is also possible to use the `eXo.require()` method to lazy load and evaluate some JavaScript code. This is quite useful for the portlet or gadget applications that will use this JavaScript only once. Otherwise, if the library is reusable in several places, it is better to define it in the **gatein-resources.xml** file.

Portlet development

4.1. Portlet Primer

4.1.1. JSR-168 and JSR-286 overview

The Java Community Process (JCP) uses Java Specification Requests (JSRs) to define proposed specifications and technologies designed for the Java platform.

The Portlet Specifications aim at defining portlets that can be used by any [JSR-168 \(Portlet 1.0\)](http://www.jcp.org/en/jsr/detail?id=168) [http://www.jcp.org/en/jsr/detail?id=168] or [JSR-286 \(Portlet 2.0\)](http://www.jcp.org/en/jsr/detail?id=286) [http://www.jcp.org/en/jsr/detail?id=286] portlet container.

Most Java EE (Enterprise Edition) portals include at least one compliant portlet container, and GateIn 3.1 is no exception. In fact, GateIn 3.1 includes a container that supports both versions.

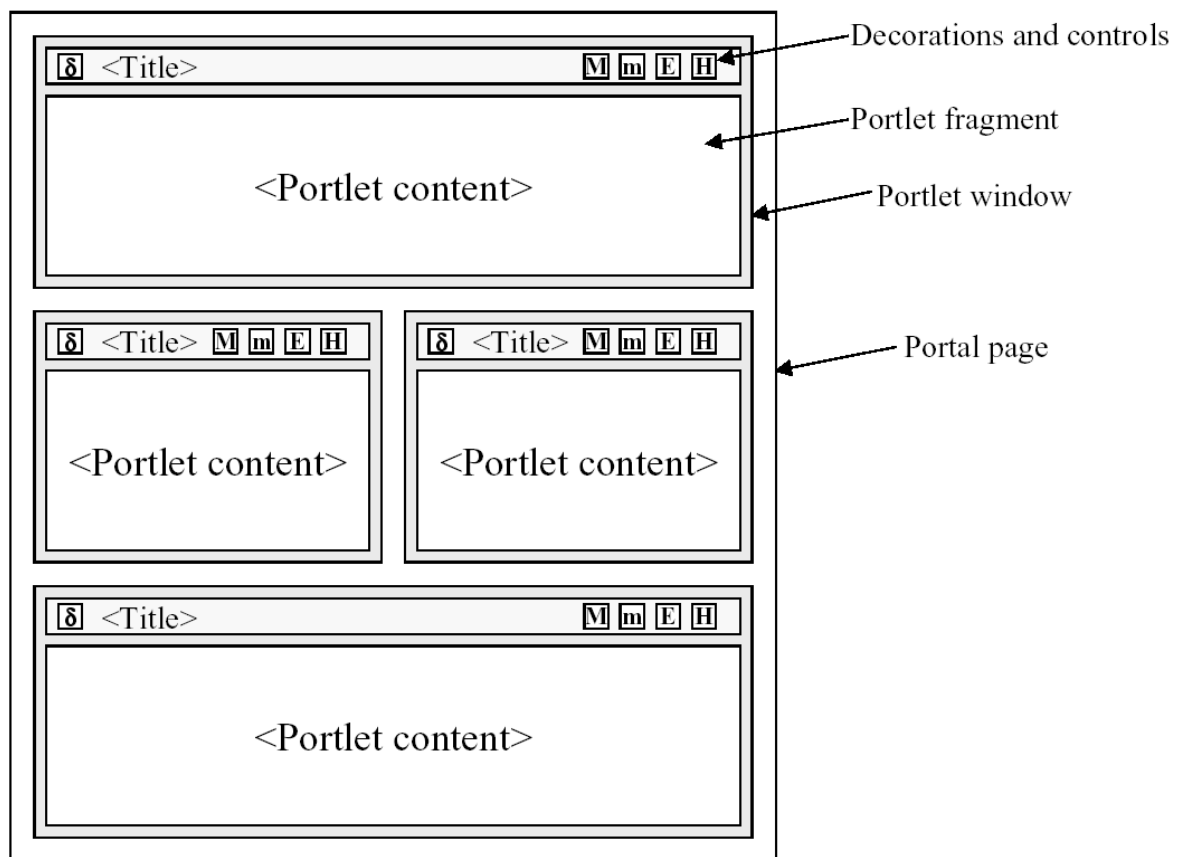
This chapter gives a brief overview of the Portlet Specifications but portlet developers are strongly encouraged to read the [JSR-286 Portlet Specification](http://www.jcp.org/en/jsr/detail?id=286) [http://www.jcp.org/en/jsr/detail?id=286] .

GateIn 3.1 is fully JSR-286 compliant. Any JSR-168 or JSR-286 portlet operates as it is mandated by the respective specifications inside the portal.

4.1.1.1. Portal Pages

A portal can be considered as a series of web pages with different *areas* within them. Those areas contain different *windows* and each *window* contains portlet:

The diagram below visually represents this nesting:



4.1.1.2. Rendering Modes

A portlet can have different view modes. Three modes are defined by the JSR-286 specification:

View

Generates markup reflecting the current state of the portlet.

Edit

Allows a user to customize the behavior of the portlet.

Help

Provides information to the user as to how to use the portlet.

4.1.1.3. Window States

Window states are an indicator of how much page space a portlet consumes on any given page. The three states defined by the JSR-168 specification are:

Normal

A portlet shares this page with other portlets.

Minimized

A portlet may show very little information, or none at all.

Maximized

A portlet may be the only portlet displayed on this page.

4.1.2. Tutorials

The tutorials contained in this chapter are targeted toward portlet developers. It is also recommend that developers read and understand the [JSR-286 Portlet Specification](http://www.jcp.org/en/jsr/detail?id=286) [http://www.jcp.org/en/jsr/detail?id=286] .



Maven

This example is using Maven to compile and build the web archive. Maven versions can be downloaded from [maven.apache.org](http://maven.apache.org/download.html) [http://maven.apache.org/download.html]

4.1.2.1. Deploying your first Portlet

This section describes how to deploy a portlet in GateIn 3.1. A sample portlet called SimplestHelloWorld is located in the `examples` directory at the root of your GateIn 3.1 binary package. This sample is used in the following examples.

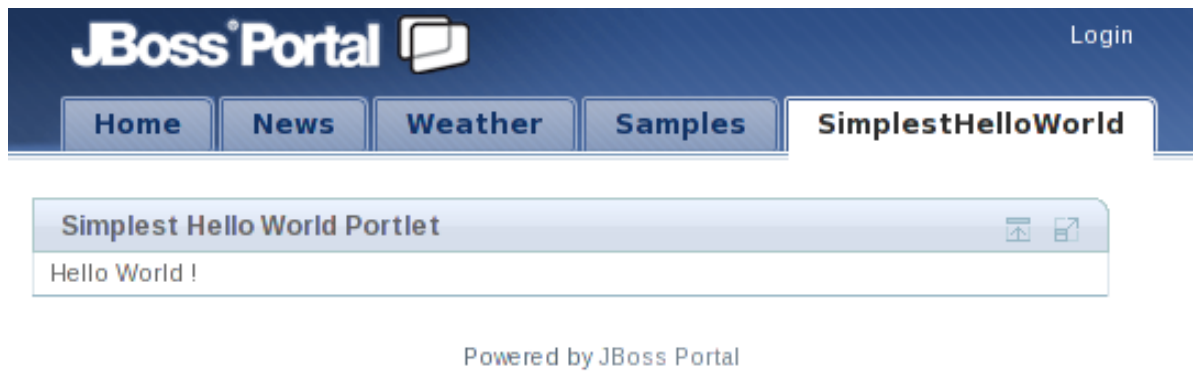
4.1.2.1.1. Compiling

To compile and package the application:

1. Navigate to the `SimplestHelloWorld` directory and execute:

```
mvn package
```

2. If the compile is successfully packaged the result will be available in: `SimplestHelloWorld/target/SimplestHelloWorld-0.0.1.war` .
3. Copy the package file into `JBOSS_HOME/server/default/deploy`.
4. Start JBoss Application Server (if it is not already running).
5. Create a new portal page and add the portlet to it.



4.1.2.1.2. Package Structure

Like other Java EE applications, GateIn 3.1 portlets are packaged in WAR files. A typical portlet WAR file can include servlets, resource bundles, images, HTML, JavaServer Pages (JSP), and other static or dynamic files.

The following is an example of the directory structure of the `SimplestHelloWorld` portlet:

```
|-- SimplestHelloWorld-0.0.1.war
|  |-- WEB-INF
|    |-- classes
|    |   |-- org
|    |     |-- gatein
|    |       |-- portal
|    |         |-- examples
|    |           |-- portlets
|    |             |-- SimplestHelloWorldPortlet.class ①
|    |-- portlet.xml ②
|    |-- web.xml ③
```

- ① The compiled Java class implementing `javax.portlet.Portlet` (through `javax.portlet.GenericPortlet`)
- ② This is the mandatory descriptor files for portlets. It is used during deployment..
- ③ This is the mandatory descriptor for web applications.

4.1.2.1.3. Portlet Class

Below is the `SimplestHelloWorldPortlet/src/main/java/org/gatein/portal/examples/portlets/SimplestHelloWorldPortlet.java` Java source:

```
package org.gatein.portal.examples.portlets;

import java.io.IOException;
import java.io.PrintWriter;

import javax.portlet.GenericPortlet;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;

public class SimplestHelloWorldPortlet extends GenericPortlet
{
    public void doView(RenderRequest request,
                       RenderResponse response) throws IOException
    {
        PrintWriter writer = response.getWriter();
        writer.write("Hello World !");
        writer.close();
    }
}
```

- 1 All portlets must implement the `javax.portlet.Portlet` interface. The portlet API provides a convenient implementation of this interface.

The `javax.portlet.Portlet` interface uses the `javax.portlet.GenericPortlet` class which implements the `Portlet render` method to dispatch to abstract mode-specific methods. This makes it easier to support the standard portlet modes.

`Portlet render` also provides a default implementation for the `processAction`, `init` and `destroy` methods. It is recommended to extend `GenericPortlet` for most cases.

- 2 If only the `view` mode is required, then only the `doView` method needs to be implemented. The `GenericPortletrender` implementation calls our implementation when the `view` mode is requested.
- 3 Use the `RenderResponse` to obtain a writer to be used to produce content.
- 4 Write the markup to display.
- 5 Closing the writer.



Markup Fragments

Portlets are responsible for generating markup fragments, as they are included on a page and are surrounded by other portlets. This means that a portlet outputting HTML must not output any markup that cannot be found in a `<body>` element.

4.1.2.1.4. Application Descriptors

GateIn 3.1 requires certain descriptors to be included in a portlet WAR file. These descriptors are defined by the Java EE (`web.xml`) and Portlet Specification (`portlet.xml`).

Below is an example of the `SimplestHelloWorldPortlet/WEB-INF/portlet.xml` file. This file must adhere to its definition in the JSR-286 Portlet Specification. More than one portlet application may be defined in this file:

```
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
    http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  version="2.0">
  <portlet>

    <portlet-name>SimplestHelloWorldPortlet</portlet-name>           ①

    <portlet-class>                                                    ②
      org.gatein.portal.examples.portlets.SimplestHelloWorldPortlet
    </portlet-class>

    <supports>                                                         ③
      <mime-type>text/html</mime-type>
    </supports>

    <portlet-info>                                                     ④
      <title>Simplest Hello World Portlet</title>
    </portlet-info>
  </portlet>
</portlet-app>
```

- ① Define the portlet name. It does not have to be the class name.
- ② The Fully Qualified Name (FQN) of your portlet class must be declared here.

- 3 The `<supports>` element declares all of the markup types that a portlet supports in the render method. This is accomplished via the `<mime-type>` element, which is required for every portlet.

The declared MIME types must match the capability of the portlet. It allows administrators to pair which modes and window states are supported for each markup type.

This does not have to be declared as all portlets must support the `view` portlet mode.

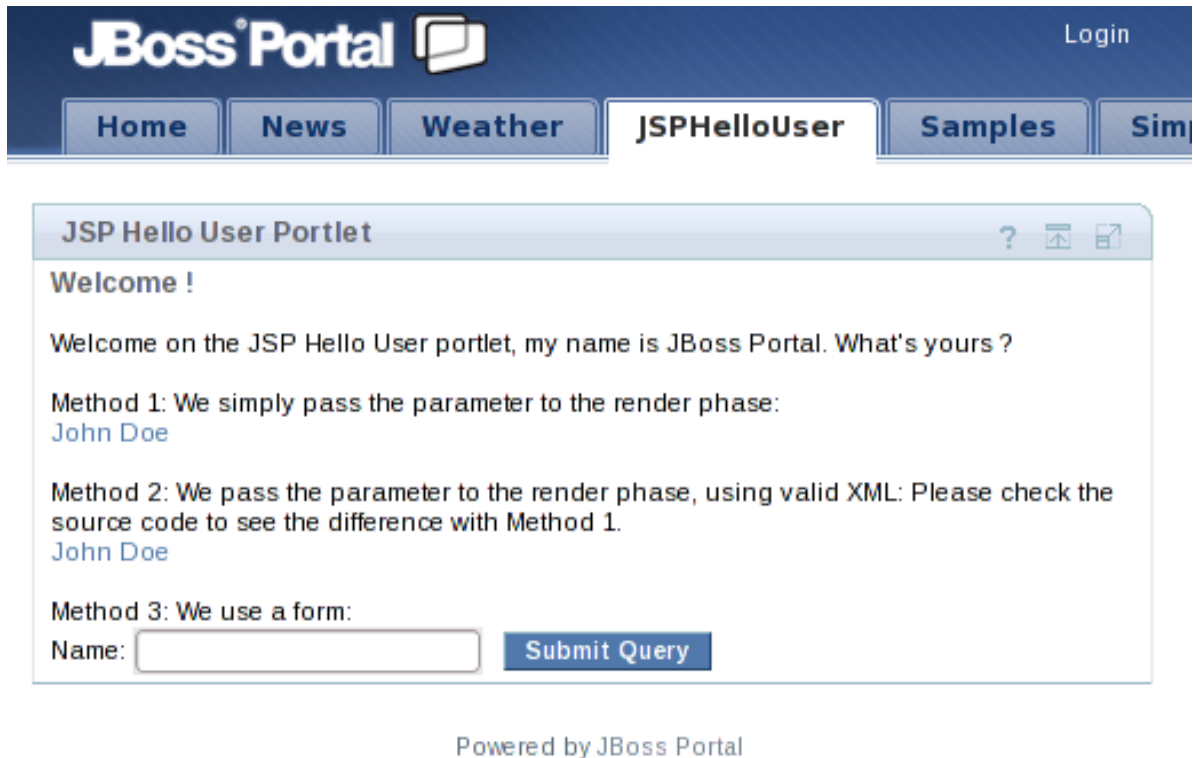
Use the `<mime-type>` element to define which markup type the portlet supports. In the example above this is `text/html`. This section tells the portal to only output HTML.

- 4 When rendered, the portlet's title is displayed as the header in the portlet window, unless it is overridden programmatically. In the example above the title would be `Simplest Hello World Portlet`.

4.1.2.2. JavaServer Pages Portlet Example

This section discusses:

1. Adding more features to the previous example.
 2. Using a JSP page to render the markup.
 3. Using the portlet tag library to generate links to the portlet in different ways.
 4. Using the other standard portlet modes.
-
1. The example used in this section can be found in the `JSPHelloUser` directory.
 2. Execute `mvn package` in this directory.
 3. Copy `JSPHelloUser/target/JSPHelloUser-0.0.1.war` to the `deploy` directory of JBoss Application Server.
 4. Select the new `JSPHelloUser` tab in your portal.



Note

The `EDIT` button only appears with logged-in users, which is not the case in the screenshot.

4.1.2.2.1. Package Structure

The package structure in this tutorial does not differ greatly from the previous example, with the exception of adding some JSP files detailed later.

The JSPHelloUser portlet contains the mandatory portlet application descriptors. The following is an example of the directory structure of the JSPHelloUser portlet:

```
JSPHelloUser-0.0.1.war
|-- META-INF
| |-- MANIFEST.MF
|-- WEB-INF
| |-- classes
| | `-- org
| |   |-- gatein
| |     |-- portal
| |       |-- examples
| |       `-- portlets
```

```
| |          |-- JSPHelloUserPortlet.class
| |-- portlet.xml
| |-- web.xml
|-- jsp
| |-- edit.jsp
| |-- hello.jsp
| |-- help.jsp
|-- welcome.jsp
```

4.1.2.2.2. Portlet Class

The code below is from the `JSPHelloUser/src/main/java/org/gatein/portal/examples/portlets/JSPHelloUserPortlet.java` Java source. It is split in different pieces.

```
package org.gatein.portal.examples.portlets;

import java.io.IOException;

import javax.portlet.ActionRequest;
import javax.portlet.ActionResponse;
import javax.portlet.GenericPortlet;
import javax.portlet.PortletException;
import javax.portlet.PortletRequestDispatcher;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
import javax.portlet.UnavailableException;

public class JSPHelloUserPortlet extends GenericPortlet
{

    public void doView(RenderRequest request, RenderResponse response)
        throws PortletException, IOException 1
    {
        String sYourName = (String) request.getParameter("yourname");

        if (sYourName != null) 2
        {
            request.setAttribute("yourname", sYourName);
            PortletRequestDispatcher prd =

                getPortletContext().getRequestDispatcher("/jsp/hello.jsp"); 3

            prd.include(request, response); 4
        }
    }
}
```

```
else
{
    PortletRequestDispatcher prd = getPortletContext().getRequestDispatcher("/jsp/
welcome.jsp");
    prd.include(request, response);
}
}
...
```

- 1 Override the *doView* method (as in the first tutorial).
- 2 This entry attempts to obtain the value of the render parameter named `yourname`. If defined it should redirect to the `hello.jsp` JSP page, otherwise to the `welcome.jsp` JSP page.
- 3 Get a request dispatcher on a file located within the web archive.
- 4 Perform the inclusion of the markup obtained from the JSP.

As well as the `VIEW` portlet mode, the specification defines two other modes; `EDIT` and `HELP`.

These modes need to be defined in the `portlet.xml` descriptor. This will enable the corresponding buttons on the portlet's window.

The generic portlet that is inherited dispatches the different views to the methods: `doView`, `doHelp` and `doEdit`.

```
...
protected void doHelp(RenderRequest rRequest, RenderResponse rResponse) throws
PortletException, IOException,
    UnavailableException
{
    rResponse.setContentType("text/html");
    PortletRequestDispatcher prd = getPortletContext().getRequestDispatcher("/jsp/help.jsp");
    prd.include(rRequest, rResponse);
}

protected void doEdit(RenderRequest rRequest, RenderResponse rResponse) throws
PortletException, IOException,
    UnavailableException
{
    rResponse.setContentType("text/html");
    PortletRequestDispatcher prd = getPortletContext().getRequestDispatcher("/jsp/edit.jsp");
    prd.include(rRequest, rResponse);
}
...
```

Portlet calls happen in one or two phases. One when the portlet is rendered and two when the portlet is actioned *then* rendered.

An action phase is a phase where some state changes. The render phase will have access to render parameters that will be passed each time the portlet is refreshed (with the exception of caching capabilities).

The code to be executed during an action has to be implemented in the *processAction* method of the portlet.

```
...
    public void processAction(ActionRequest aRequest, ActionResponse aResp1onse)
    throws PortletException, IOException,
        UnavailableException
    {
        String sYourname = (String) aRequest.getParameter("yourname"); 2
        aResponse.setRenderParameter("yourname", sYourname); 3
    }
...

```

- ¹ `processAction` is the method from `GenericPortlet` to override for the *action* phase.
- ² Here the parameter is retrieved through an *action URL*.
- ³ The value of `yourname` is kept to make it available in the rendering phase. The previous line simply copies an action parameters to a render parameter for this example.

4.1.2.2.3. JSP files and the Portlet Tag Library

The `help.jsp` and `edit.jsp` files are very simple. Note that CSS styles are used as defined in the portlet specification. This ensures that the portlet will render well within the theme and across portal vendors.

```
<div class="portlet-section-header">Help mode</div>
<div class="portlet-section-body">This is the help mode, a convenient place to give the user some
help information.</div>
```

```
<div class="portlet-section-header">Edit mode</div>
<div class="portlet-section-body">This is the edit mode, a convenient place to let the user change
his portlet preferences.</div>
```

The landing page contains the links and form to call our portlet:

```
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %> 1

<div class="portlet-section-header">Welcome !</div>

<br/>

<div class="portlet-font">Welcome on the JSP Hello User portlet,
my name is GateIn Portal. What's yours ?</div>

<br/>

<div class="portlet-font">Method 1: We simply pass the parameter to the render phase:<br/>
<a href="<portlet:renderURL><portlet:param name="yourname" value="John Doe"/> 2
    </portlet:renderURL>">John Doe</a></div>

<br/>

<div class="portlet-font">Method 2: We pass the parameter to the render phase, using valid XML:
Please check the source code to see the difference with Method 1.

<portlet:renderURL var="myRenderURL"> 3
    <portlet:param name="yourname" value='John Doe' />
</portlet:renderURL>
<br/>

<a href="<%= myRenderURL %>">John Doe</a></div> 4

<br/>

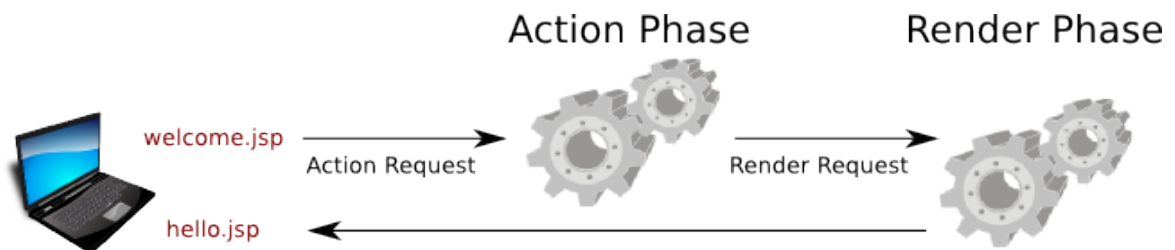
<div class="portlet-font">Method 3: We use a form:<br/>

<portlet:actionURL var="myActionURL"/> 5

<form action="<%= myActionURL %>" method="POST"> 6
    <span class="portlet-form-field-label">Name:</span>
    <input class="portlet-form-input-field" type="text" name="yourname"/>
    <input class="portlet-form-button" type="Submit"/>
</form>
</div>
```

- 1 The portlet taglib, needs to be declared.
- 2 The first method showed here is the simplest one. `portlet:renderURL` will create a URL that calls the render phase of the current portlet and append the result at the place of the markup (within a tag). A parameter is also added directly to the URL.
- 3 In this method the `var` attribute is used. This avoids having one XML tag within another. Instead of printing the url the `portlet:renderURL` tag will store the result in the referenced variable (`myRenderURL`).
- 4 The variable `myRenderURL` is used like any other JSP variable.
- 5 The third method mixes form submission and action request. Again, a temporary variable is used to put the created URL into.
- 6 The action URL is used in HTML form.

In the third method the action phase is triggered first then the render phase is triggered, which outputs some content back to the web browser based on the available render parameters.



4.1.2.2.4. JSF example using the JBoss Portlet Bridge

In order to write a portlet using JSF a 'bridge' is needed. This software allows developers to write a portlet application as if it was a JSF application. The bridge then negotiates the interactions between the two layers.

An example of the JBoss Portlet Bridge is available in `examples/JSFHelloUser`. The configuration is slightly different from a JSP application. This example can be used as a base to configure instead of creating a new application.

As in any JSF application, the file `faces-config.xml` is required. It must contain the following information:

```
<faces-config>
...
<application>
  <view-handler>org.jboss.portletbridge.application.PortletViewHandler</view-handler>
  <state-manager>org.jboss.portletbridge.application.PortletStateManager</state-manager>
</application>
...
</faces-config>
```

The portlet bridge libraries must be available and are usually bundled with the `WEB-INF/lib` directory of the web archive.

The other difference compare to a regular portlet application, can be found in the portlet descriptor. All details about it can be found in the JSR-301 specification that the JBoss Portlet Bridge implements.

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
    http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  version="2.0">
  <portlet>
    <portlet-name>JSFHelloUserPortlet</portlet-name>

    <portlet-class>javax.portlet.faces.GenericFacesPortlet</portlet-class> ①
    <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>view</portlet-mode>
      <portlet-mode>edit</portlet-mode>
      <portlet-mode>help</portlet-mode>
    </supports>
    <portlet-info>
      <title>JSF Hello User Portlet</title>
    </portlet-info>

    <init-param>
      <name>javax.portlet.faces.defaultViewId.view</name> ②
      <value>/jsf/welcome.jsp</value>
    </init-param>

    <init-param>
      <name>javax.portlet.faces.defaultViewId.edit</name> ③
      <value>/jsf/edit.jsp</value>
    </init-param>

    <init-param>
      <name>javax.portlet.faces.defaultViewId.help</name> ④
      <value>/jsf/help.jsp</value>
    </init-param>

  </portlet>
```



```
</portlet-app>
```

- ① All JSF portlets define `javax.portlet.faces.GenericFacesPortlet` as portlet class. This class is part of the JBoss Portlet Bridge
- ② This is a mandatory parameter to define what's the default page to display.
- ③ This parameter defines which page to display on the 'edit' mode.
- ④ This parameter defines which page to display on the 'help' mode.

Gadget development

5.1. Gadgets

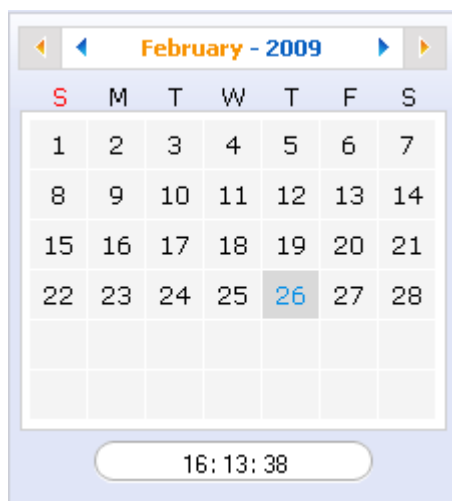
A gadget is a mini web application, embedded in a web page and running on an application server platform. These small applications help users perform various tasks.

GateIn 3.1 supports gadgets such as: Todo gadget, Calendar gadget, Calculator gadget, Weather Forecasts and and RSS Reader.

Default Gadgets:

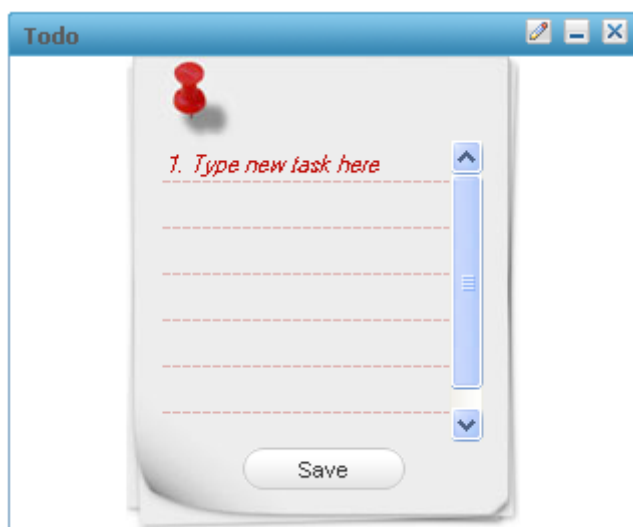
Calendar

The calendar gadget allows users to switch easily between daily, monthly and yearly view and, again, is customizable to match your portal's theme.



ToDo

This application helps you organize your day and work group. It is designed to keep track of your tasks in a convenient and transparent way. Tasks can be highlighted with different colors.



Calculator

This mini-application lets you perform most basic arithmetic operations and can be themed to match the rest of your portal.



RSS Reader

An RSS reader, or aggregator, collates content from various, user-specified feed sources and displays them in one location. This content can include, but isn't limited to, news headlines, blog posts or email. The RSS Reader gadget displays this content in a single window on your Portal page.

More Gadgets

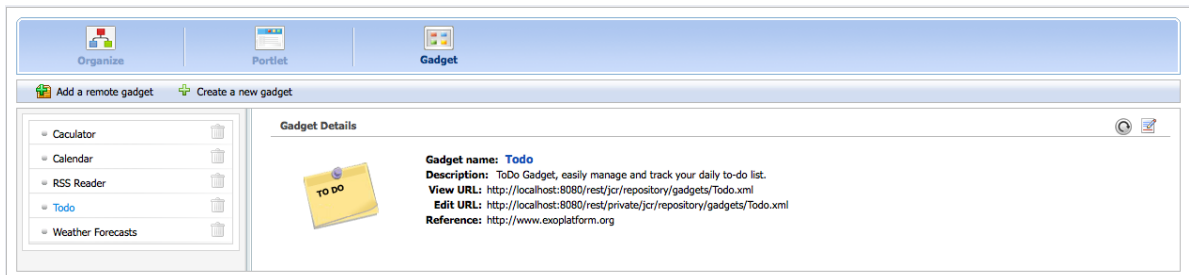
Further gadgets can be obtained from the [Google Gadget](http://www.google.com/ig/directory?synd=open) [http://www.google.com/ig/directory?synd=open] site. GateIn 3.1 is compatible with most of the gadgets available here.



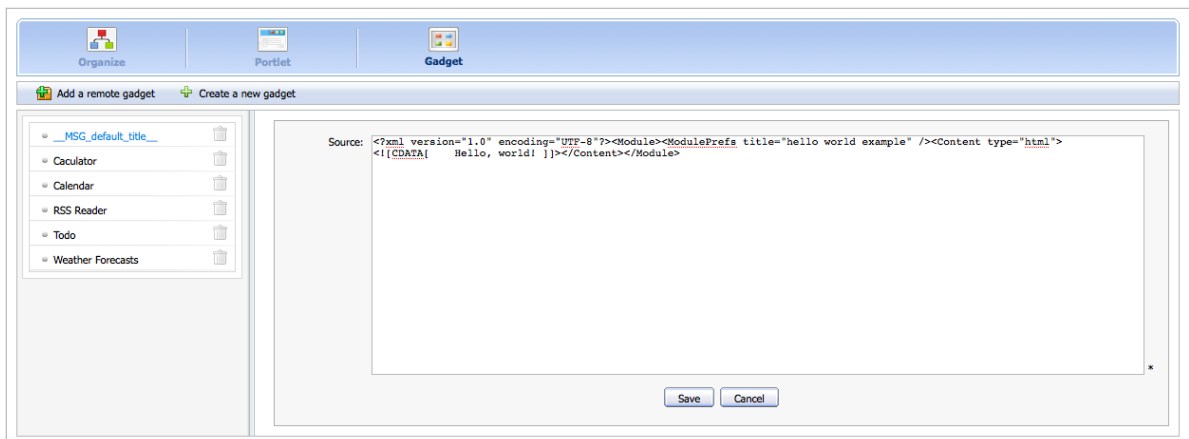
Important

The following sections require more textual information.

5.1.1. Existing Gadgets

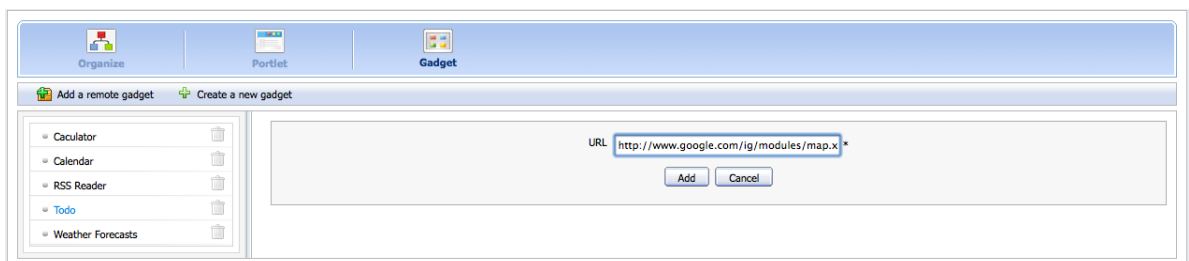


5.1.2. Create a new Gadget



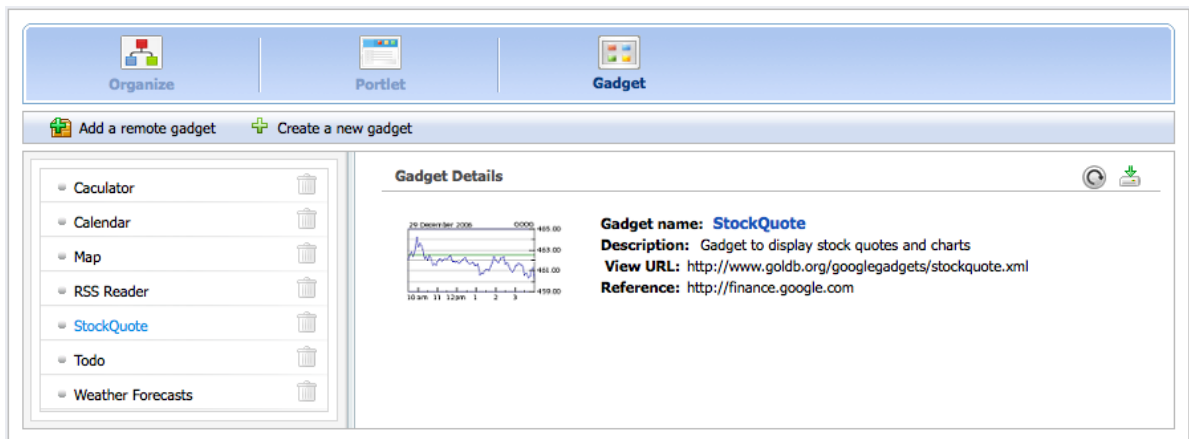
5.1.3. Remote Gadget

This is the reference to a remote gadget (stock one).



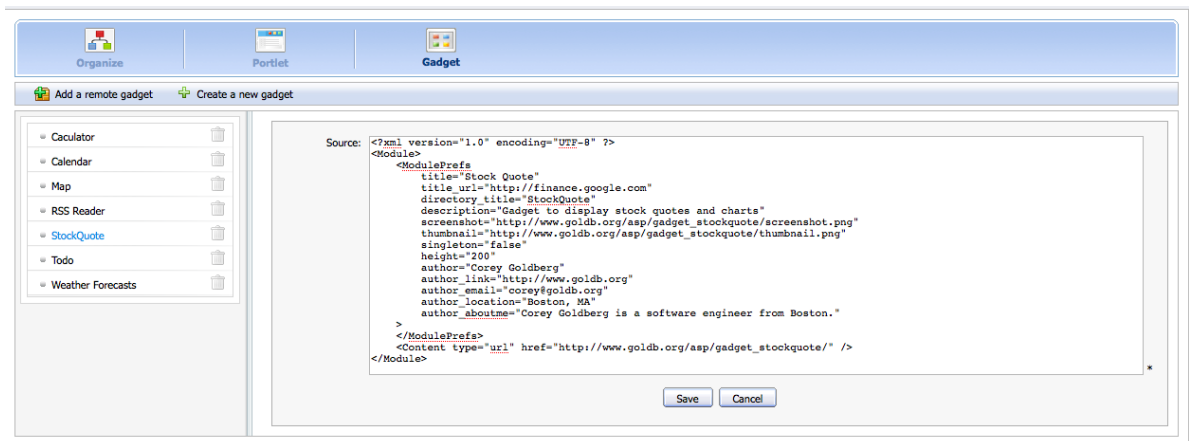
5.1.4. Gadget Importing

After referencing the gadget successfully, then import it into the local repository.



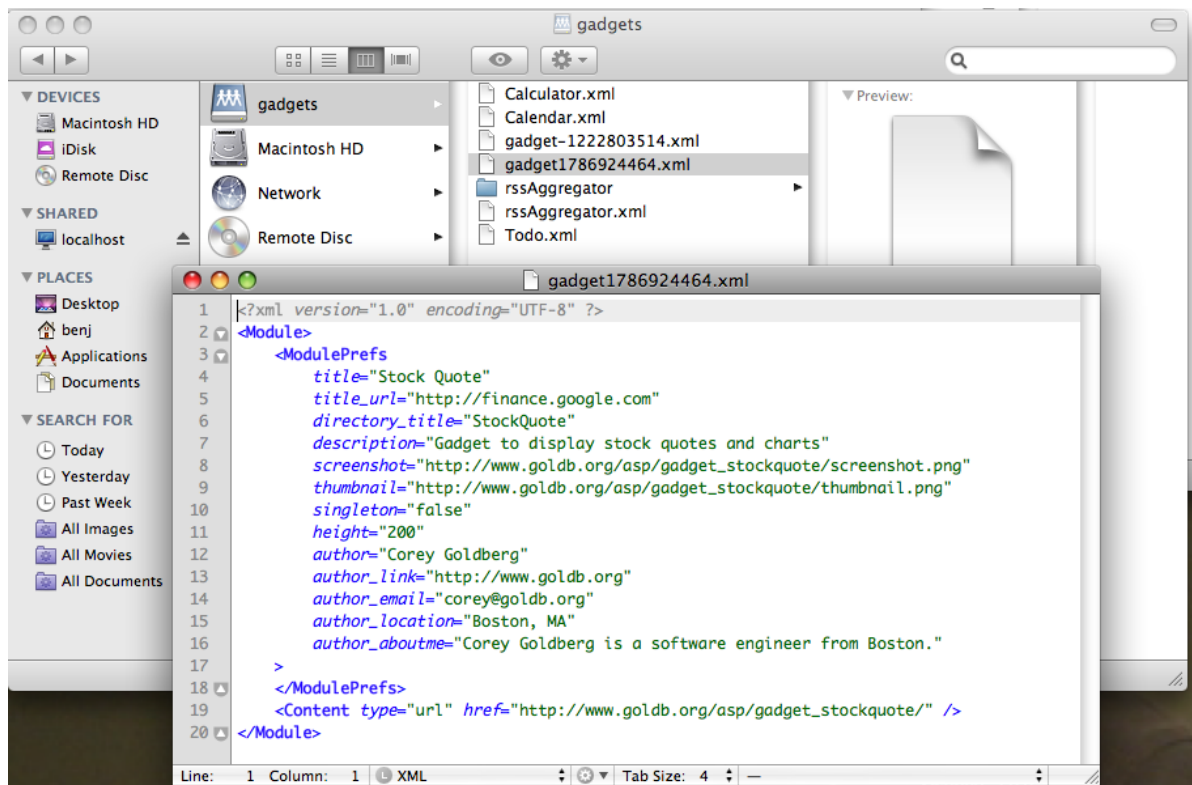
5.1.5. Gadget Web Editing

Edit it from the Web the imported Gadget to modify it:



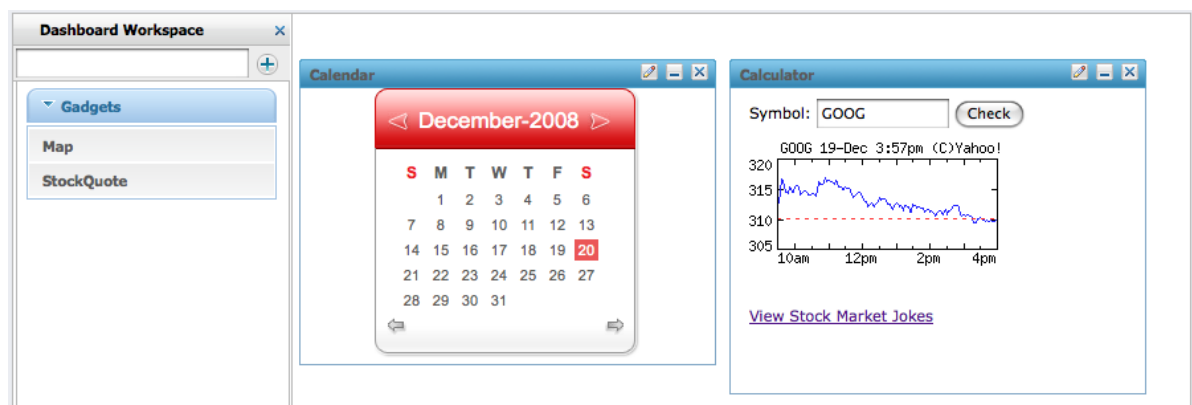
5.1.6. Gadget IDE Editing

Edit it from your IDE thanks to the WebDAV protocol:



5.1.7. Dashboard Viewing

View it from the Dashboard when you drag and drop the Gadget from listing to the dashboard.



5.2. Setup a Gadget Server

5.2.1. Virtual servers for gadget rendering

GateIn 3.1 recommends using two virtual hosts for security. If the gadget is running on a different domain than the container (the website that 'contains' the app), it is unable to interfere with the portal by modifying code or cookies.

An example would hosting the portal from <http://www.sample.com> and the gadgets from <http://www.samplemodules.com>.

To do this, configure a parameter called *gadgets.hostName*. The value is the *path/to/gadgetServer* in *GadgetRegistryService*:

```
<component>
  <key>org.exoplatform.application.gadget.GadgetRegistryService</key>
  <type>org.exoplatform.application.gadget.jcr.GadgetRegistryServiceImpl</type>
  <init-params>
    <value-param>
      <name>gadgets.hostName</name>
      <description>Gadget server url</description>
      <value>http://localhost:8080/GateInGadgetServer/gadgets</value>
    </value-param>
  </init-params>
</component>
```

It is also possible to have multiple rendering servers. This helps to balance the rendering load across multiple servers.

When deploying on the same server ensure the gadget initiates before anything that calls it (for example; the webapp *GateInGadgets* which uses *org.exoplatform.application.gadget.GadgetRegister*).

5.2.2. Configuration

5.2.2.1. Security key

A file called **key.txt** has to be generated for every installation of *GateIn 3.1* to be secure. This file contains a secret key used to encrypt the security token used for authenticating the user.

In Linux systems this file can be generated with:

```
dd if=/dev/random bs=32 count=1 | openssl base64 > /tmp/key.txt
```

5.2.2.2. Gadget proxy and concat configuration

These servers have to be on the same domain as the gadget server. You can configure the container in *eXoGadgetServer: /WEB-INF/classes/containers/default/container.js*.

```
"gadgets.content-rewrite" : {
  "include-urls": ".*",
  "exclude-urls": "",
  "include-tags": ["link", "script", "embed", "img", "style"],
  "expires": "86400",
```



```
"proxy-url": "http://localhost:8080/eXoGadgetServer/gadgets/proxy?url=",  
"concat-url": "http://localhost:8080/eXoGadgetServer/gadgets/concat?"  
},
```

5.2.2.3. Proxy

To allow external gadgets when the server is behind a proxy, add the following code to the beginning of the JVM:

```
-Dhttp.proxyHost=proxyhostURL -Dhttp.proxyPort=proxyPortNumber -  
Dhttp.proxyUser=someUserName -Dhttp.proxyPassword=somePassword
```

Authentication and Identity

6.1. Predefined User Configuration

6.1.1. Overview

To specify the initial Organization configuration, the content of `02portal.war:/WEB-INF/conf/organization/organization-configuration.xml` should be edited. This file uses the portal XML configuration schema. It lists several configuration plugins.

6.1.2. Plugin for adding users, groups and membership types

The plugin of type `org.exoplatform.services.organization.OrganizationDatabaseInitializer` is used to specify a list of membership types, a list of groups, and a list of users to be created.

The **checkDatabaseAlgorithm** initialization parameter determines how the database update is performed.

If its value is set to **entry** it means that each user, group and membership listed in the configuration is checked each time GateIn 3.1 is started. If the entry doesn't yet exist in the database, it is created. If **checkDatabaseAlgorithm** parameter value is set to **empty**, the configuration data will be updated to the database only if the database is empty.

6.1.3. Membership types

The predefined membership types are specified in the **membershipType** field of the **OrganizationConfig** plugin parameter.



Note

See `02portal.war:/WEB-INF/conf/organization/organization-configuration.xml` for the full content.

```
<field name="membershipType">
  <collection type="java.util.ArrayList">
    <value>
      <object type="org.exoplatform.services.organization.OrganizationConfig$MembershipType">
        <field name="type">
          <string>member</string>
        </field>
        <field name="description">
          <string>member membership type</string>
        </field>
      </object>
    </value>
  </collection>
</field>
```

```
</field>
</object>
</value>
<value>
  <object type="org.exoplatform.services.organization.OrganizationConfig$MembershipType">
    <field name="type">
      <string>owner</string>
    </field>
    <field name="description">
      <string>owner membership type</string>
    </field>
  </object>
</value>
<value>
  <object type="org.exoplatform.services.organization.OrganizationConfig$MembershipType">
    <field name="type">
      <string>validator</string>
    </field>
    <field name="description">
      <string>validator membership type</string>
    </field>
  </object>
</value>
</collection>
</field>
```

6.1.4. Groups

The predefined groups are specified in the **group** field of the **OrganizationConfig** plugin parameter.

```
<field name="group">
  <collection type="java.util.ArrayList">
    <value>
      <object type="org.exoplatform.services.organization.OrganizationConfig$Group">
        <field name="name">
          <string>portal</string>
        </field>
        <field name="parentId">
          <string></string>
        </field>
        <field name="type">
          <string>hierachy</string>
        </field>
      </object>
    </value>
  </collection>
</field>
```

```

    </field>
    <field name="description">
      <string>the /portal group</string>
    </field>
  </object>
</value>
<value>
  <object type="org.exoplatform.services.organization.OrganizationConfig$Group">
    <field name="name">
      <string>community</string>
    </field>
    <field name="parentId">
      <string>/portal</string>
    </field>
    <field name="type">
      <string>hierachy</string>
    </field>
    <field name="description">
      <string>the /portal/community group</string>
    </field>
  </object>
</value>
...
</collection>
</field>

```

6.1.5. Users

The predefined users are specified in the **membershipType** field of the **OrganizationConfig** plugin parameter.

```

<field name="user">
  <collection type="java.util.ArrayList">
    <value>
      <object type="org.exoplatform.services.organization.OrganizationConfig$User">
        <field name="userName"><string>root</string></field>
        <field name="password"><string>exo</string></field>
        <field name="firstName"><string>root</string></field>
        <field name="lastName"><string>root</string></field>
        <field name="email"><string>exoadmin@localhost</string></field>
        <field name="groups"><string>member:/admin,member:/user,owner:/portal/admin</string></field>
      </object>
    </value>
  </collection>
</field>

```

```
</value>
<value>
  <object type="org.exoplatform.services.organization.OrganizationConfig$User">
    <field name="userName"><string>exo</string></field>
    <field name="password"><string>exo</string></field>
    <field name="firstName"><string>site</string></field>
    <field name="lastName"><string>site</string></field>
    <field name="email"><string>exo@localhost</string></field>
    <field name="groups"><string>member:/user</string></field>
  </object>
</value>
...
</collection>
</field>
```

6.1.6. Plugin for monitoring user creation

The plugin of type `org.exoplatform.services.organization.impl.NewUserEventListener` specifies which groups all the newly created users should become members of. It specifies the groups and the memberships to use (while group is just a set of users, a membership type represents a user's role within a group). It also specifies a list of users that should not be processed (i.e. administrative users like 'root').



Note

The terms 'membership' and 'membership type' refer to the same thing, and are used interchangeably.

```
<component-plugin>
  <name>new.user.event.listener</name>
  <set-method>addListenerPlugin</set-method>
  <type>org.exoplatform.services.organization.impl.NewUserEventListener</type>
  <description>this listener assign group and membership to a new created user</description>
  <init-params>
    <object-param>
      <name>configuration</name>
      <description>description</description>
      <object type="org.exoplatform.services.organization.impl.NewUserConfig">
        <field name="group">
          <collection type="java.util.ArrayList">
            <value>
              <object type="org.exoplatform.services.organization.impl.NewUserConfig$JoinGroup">
```

```

    <field name="groupId"><string>/user</string></field>
    <field name="membership"><string>member</string></field>
  </object>
</value>
</collection>
</field>
<field name="ignoredUser">
  <collection type="java.util.HashSet">
    <value><string>exo</string></value>
    <value><string>root</string></value>
    <value><string>company</string></value>
    <value><string>community</string></value>
  </collection>
</field>
</object>
</object-param>
</init-params>
</component-plugin>

```

6.2. Authentication Token Configuration

6.2.1. What is Token Service?

Token Service is used in authentication.

The token system prevents user account information being sent in clear text mode within inbound requests. This increases authentication security.

Token service allows administrators to create, delete, retrieve and clean tokens as required. The service also defines a validity period of any given token. The token becomes invalid once this period expires.

6.2.2. Implementing the Token Service API

All token services used in GateIn 3.1 authentication must be implemented by subclassing an **AbstractTokenService** abstract class. The following **AbstractTokenService** methods represent the contract between authentication runtime, and a token service implementation.

```

public Token getToken(String id) throws PathNotFoundException, RepositoryException;
public Token deleteToken(String id) throws PathNotFoundException, RepositoryException;
public String[] getAllTokens();
public long getNumberTokens() throws Exception;

```

```
        public String createToken(Credentials credentials) throws
        IllegalArgumentException,NullPointerException;

        public Credentials validateToken(String tokenKey, boolean remove) throws
        NullPointerException;
```

6.2.3. Configuring token services

Token services configuration includes specifying the token validity period. The token service is configured as a portal component (in portal scope, as opposed to root scope - more about that in Foundations chapter).

In the example below, *CookieTokenService* is a subclass of **AbstractTokenService** so it has a property which specifies the validity period of the token.

The token service will initialize this validity property by looking for an *init-param* named **service.configuration**.

This property must have three values.

```
<component>
  <key>org.exoplatform.web.security.security.CookieTokenService</key>
  <type>org.exoplatform.web.security.security.CookieTokenService</type>
  <init-params>
    <values-param>
      <name>service.configuration</name>

      <value>jcr-token</value>                                1
      <value>7</value>                                         2
      <value>DAY</value>                                       3
    </values-param>
  </init-params>
</component>
```

- 1 Service name
- 2 Amount of time
- 3 Unit of time

In this case, the service name is **jcr-token** and the token expiration time is one week.

GateIn 3.1 supports *four* time units:

1. *SECOND*

2. *MINUTE*

3. *HOURL*

4. *DAY*

6.3. PicketLink IDM integration

Gateln 3.1 uses PicketLink IDM component to keep the necessary identity information (users, groups, memberships, etc.). While legacy interfaces are still used (org.exoplatform.services.organization) for identity management, there is a wrapper implementation that delegates to PicketLink IDM framework.

This section doesn't provide information about PicketLink IDM and its configuration. Please, refer to the appropriate project documentation (<http://jboss.org/picketlink/IDM.html>) for further information.



Note

It is important to fully understand the concepts behind this framework design before changing the default configuration.

The identity model represented in '**org.exoplatform.services.organization**' interfaces and the one used in **PicketLink IDM** have some major differences.

TODO: tell more about org.exoplatform.services.organization

For example: **PicketLink IDM** provides greater abstraction. It is possible for groups in **IDM** framework to form memberships with many parents (which requires recursive ID translation), while Gateln model allows only pure tree-like membership structures.

Additionally, Gateln *membership* concept needs to be translated into the IDM *Role* concept. Therefore **PicketLink IDM** model is used in a limited way. All these translations are applied by the integration layer.

6.3.1. Configuration files

The main configuration file is **idm-configuration.xml**:

```
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_0.xsd http://
www.exoplatform.org/xml/ns/kernel_1_0.xsd"
    xmlns="http://www.exoplatform.org/xml/ns/kernel_1_0.xsd">

    <component>
        <key>org.exoplatform.services.organization.idm.PicketLinkIDMService</key>
```

1

```
<type>org.exoplatform.services.organization.idm.PicketLinkIDMServiceImpl</type>
<init-params>
  <value-param>
    <name>config</name>
    <value>war:/conf/organization/idm-config.xml</value>
  </value-param>
  <value-param>
    <name>portalRealm</name>
    <value>realm${container.name.suffix}</value>
  </value-param>
</init-params>
</component>

<component>

  <key>org.exoplatform.services.organization.OrganizationService</key> ②
  <type>org.exoplatform.services.organization.idm.PicketLinkIDMOrganizationServiceImpl</
type>
  <init-params>
    <object-param>
      <name>configuration</name>
      <object type="org.exoplatform.services.organization.idm.Config">
        <field name="useParentIdAsGroupType">
          <boolean>true</boolean>
        </field>

        <field name="forceMembershipOfMappedTypes">
          <boolean>true</boolean>
        </field>

        <field name="pathSeparator">
          <string>.</string>
        </field>

        <field name="rootGroupName">
          <string>GTN_ROOT_GROUP</string>
        </field>

        <field name="groupTypeMappings">
          <map type="java.util.HashMap">
            <entry>
              <key><string></string></key>
              <value><string>root_type</string></value>
            </entry>
```

```

    <!-- Sample mapping -->
    <!--
    <entry>
      <key><string>/platform/*</string></key>
      <value><string>platform_type</string></value>
    </entry>
    <entry>
      <key><string>/organization/*</string></key>
      <value><string>organization_type</string></value>
    </entry>
    -->

  </map>
</field>

<field name="associationMembershipType">
  <string>member</string>
</field>

<field name="ignoreMappedMembershipType">
  <boolean>>false</boolean>
</field>
</object>
</object-param>
</init-params>

</component>

</configuration>

```

- 1 The **org.exoplatform.services.organization.idm.PicketLinkIDMServiceImpl** service has the following options:

config

(value-param)

PicketLink IDM configuration file

hibernate.properties

(properties-param)

A list of hibernate properties used to create SessionFactory that will be injected to JBoss Identity IDM configuration registry.

hibernate.annotations

A list of annotated classes that will be added to Hibernate configuration.

hibernate.mappings

A list of xml files that will be added to hibernate configuration as mapping files.

jndiName

(value-param)

If the 'config' parameter is not provided, this parameter will be used to perform JNDI lookup for IdentitySessionFactory

portalRealm

(value-param)

The realm name that should be used to obtain proper IdentitySession. The default is 'PortalRealm'.

2

The

org.exoplatform.services.organization.idm.PicketLinkIDMOrganizationServiceImpl key is a main endpoint implementing **org.exoplatform.services.organization.OrganizationService** and is dependant on **org.exoplatform.services.organization.idm.PicketLinkIDMService**

org.exoplatform.services.organization.idm.PicketLinkIDMOrganizationServiceImpl service has the following options defined as fields of object-param of type **org.exoplatform.services.organization.idm.Config**:

defaultGroupType

The name of the PicketLink IDM GroupType that will be used to store groups. The default is 'GTN_GROUP_TYPE'.

rootGroupName

The name of the PicketLink IDM Group that will be used as a root parent. The default is 'GTN_ROOT_GROUP'

passwordAsAttribute

This parameter specifies if a password should be stored using PicketLink IDM Credential object or as a plain attribute. The default is false.

useParentIdAsGroupType

This parameter stores the parent ID path as a group type in PicketLink IDM for any IDs not mapped with a specific type in 'groupTypeMappings'. If this option is set to false, and no mappings are provided under 'groupTypeMappings', then only one group with the given name can exist in the GateIn 3.1 group tree.

pathSeparator

When 'userParentIdAsGroupType' is set to true, this value will be used to replace all "/" characters in IDs. The "/" character is not allowed to be used in group type name in PicketLink IDM.

associationMembershipType

If this option is used, then each Membership, created with MembershipType that is equal to the value specified here, will be stored in PicketLink IDM as simple Group-User association.

groupTypeMappings

This parameter maps groups added with GateIn 3.1 API as children of a given group ID, and stores them with a given group type name in PicketLink IDM.

If the parent ID ends with "/*", then all child groups will have the mapped group type. Otherwise, only direct (first level) children will use this type.

This can be leveraged by LDAP if LDAP DN is configured in PicketLink IDM to only store a specific group type. This will then store the given branch in GateIn 3.1 group tree, while all other groups will remain in the database.

forceMembershipOfMappedTypes

Groups stored in PicketLink IDM with a type mapped in 'groupTypeMappings' will automatically be members under the mapped parent. Group relationships linked by PicketLink IDM group association will not be necessary.

This parameter can be set to false if all groups are added via GateIn 3.1 APIs. This may be useful with LDAP configuration as, when set to true, it will make every entry added to LDAP appear in GateIn 3.1. This, however, is not true for entries added via GateIn 3.1 management UI.

ignoreMappedMembershipType

If "associationMembershipType" option is used, and this option is set to true, then Membership with MembershipType configured to be stored as PicketLink IDM association will not be stored as PicketLink IDM Role.

Additionally, **JBossIDMOrganizationServiceImpl** uses those defaults to perform identity management operations

- GateIn 3.1 User interface properties fields are persisted in JBoss Identity IDM using those attributes names: firstName, lastName, email, createdDate, lastLoginTime, organizationId, password (if password is configured to be stored as attribute)
- GateIn 3.1 Group interface properties fields are persisted in JBoss Identity IDM using those attributes names: label, description

- GateIn 3.1 MembershipType interface properties fields are persisted in JBoss Identity IDM using those RoleType properties: description, owner, create_date, modified_date

A sample **PicketLink IDM** configuration file is shown below. To understand all the options it contains, please refer to the PicketLink IDM Reference Guide

```
<jboss-identity xmlns="urn:jboss:identity:idm:config:v1_0_beta"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:jboss:identity:idm:config:v1_0_alpha identity-config.xsd">
  <realms>
    <realm>
      <id>PortalRealm</id>
      <repository-id-ref>PortalRepository</repository-id-ref>
      <identity-type-mappings>
        <user-mapping>USER</user-mapping>
      </identity-type-mappings>
    </realm>
  </realms>
  <repositories>
    <repository>
      <id>PortalRepository</id>
      <class>org.jboss.identity.idm.impl.repository.WrapperIdentityStoreRepository</class>
      <external-config/>
      <default-identity-store-id>HibernateStore</default-identity-store-id>
      <default-attribute-store-id>HibernateStore</default-attribute-store-id>
    </repository>
  </repositories>
  <stores>
    <attribute-stores/>
    <identity-stores>
      <identity-store>
        <id>HibernateStore</id>
        <class>org.jboss.identity.idm.impl.store.hibernate.HibernateIdentityStoreImpl</class>
        <external-config/>
        <supported-relationship-types>
          <relationship-type>JBOSS_IDENTITY_MEMBERSHIP</relationship-type>
          <relationship-type>JBOSS_IDENTITY_ROLE</relationship-type>
        </supported-relationship-types>
        <supported-identity-object-types>
          <identity-object-type>
            <name>USER</name>
            <relationships/>
            <credentials>
```

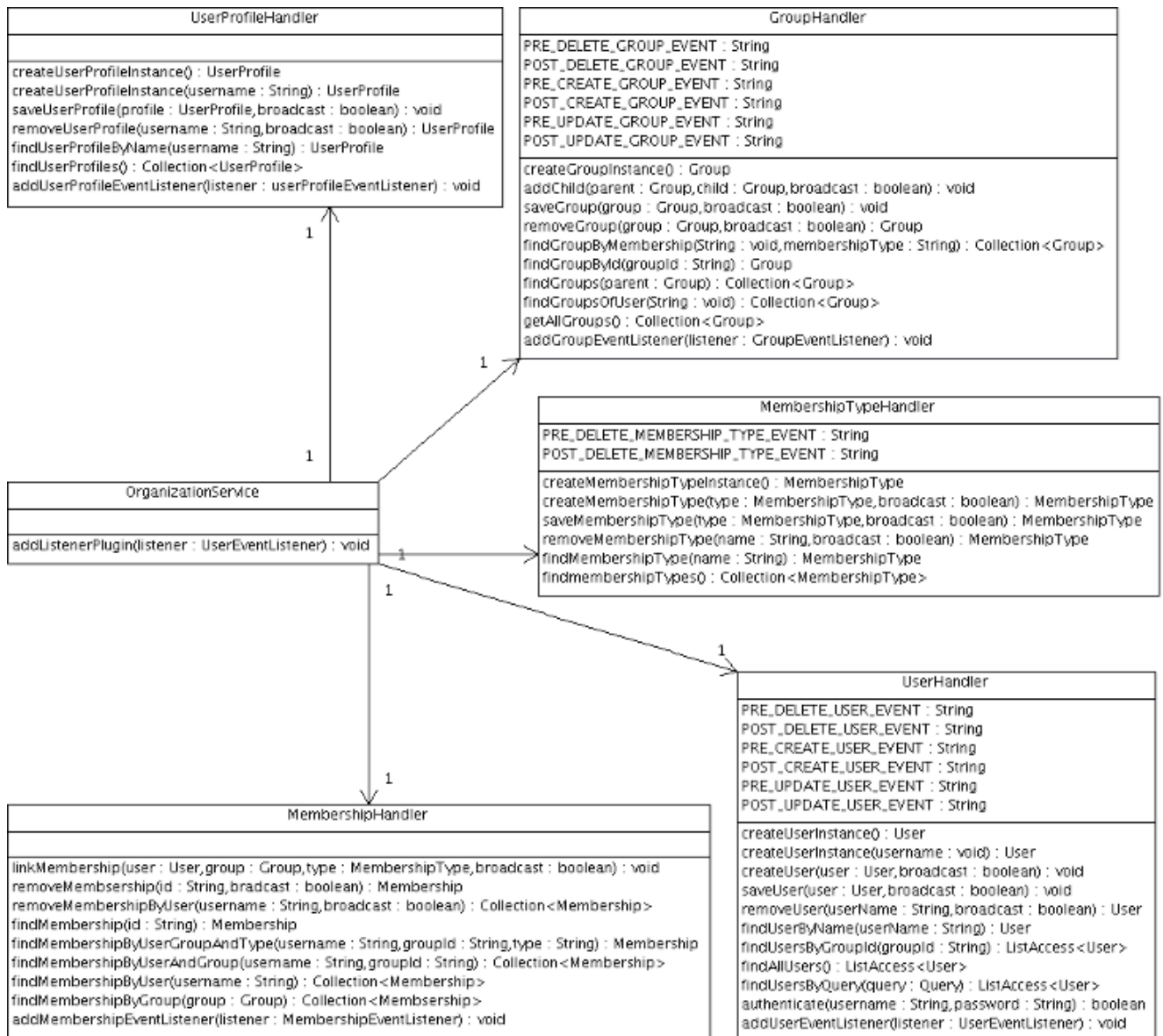
```

        <credential-type>PASSWORD</credential-type>
    </credentials>
    <attributes/>
    <options/>
</identity-object-type>
</supported-identity-object-types>
<options>
    <option>
        <name>hibernateSessionFactoryRegistryName</name>
        <value>hibernateSessionFactory</value>
    </option>
    <option>
        <name>allowNotDefinedIdentityObjectTypes</name>
        <value>true</value>
    </option>
    <option>
        <name>populateRelationshipTypes</name>
        <value>true</value>
    </option>
    <option>
        <name>populateIdentityObjectTypes</name>
        <value>true</value>
    </option>
    <option>
        <name>allowNotDefinedAttributes</name>
        <value>true</value>
    </option>
    <option>
        <name>isRealmAware</name>
        <value>true</value>
    </option>
</options>
</identity-store>
</identity-stores>
</stores>
</jboss-identity>

```

6.4. Organization API

The `exo.platform.services.organization` package has five main components: user, user profile, group, membership type and membership. There is an additional component that serves as an entry point into Organization API - `OrganizationService` component, that provides handling functionality for the five components.



The `User` component contains basic information about a user - such as username, password, first name, last name, and email. The `User Profile` component contains extra information about a user, such as user's personal information, and business information. You can also add additional information about a user if your application requires it. The `Group` component contains a group graph. The `Membership Type` component contains a list of predefined membership types. Finally, the `Membership` component connects a `User`, a `Group` and a `Membership Type`.

A user can have one or more memberships within a group, for example: user A can have the 'member' and 'admin' memberships in group /user. A user belongs to a group if he has at least one membership in that group.

Exposing the Organization API to developers the `OrganizationService` component provides developers with access to handler objects for managing each of the five components - `UserHandler`, `UserProfileHandler`, `GroupHandler`, `MembershipTypeHandler`, and `MembershipHandler`.

The five central API components are really designed like persistent entities, and handlers are really specified like data access objects (DAO).

Organization API simply describes a contract, meaning it is not a concrete implementation. The described components are interfaces, allowing for different concrete implementations. In practical terms that means, you can replace the existing implementation with a different one.

6.5. Accessing User Profile

The following code retrieves the details for a logged-in user:

```
// Alternative context: WebuiRequestContext context =
WebuiRequestContext.getCurrentInstance() ;
PortalRequestContext context = PortalRequestContext.getCurrentInstance() ;
// Get the id of the user logged
String userId = context.getRemoteUser();
// Request the information from OrganizationService:
OrganizationService orgService = getApplicationComponent(OrganizationService.class) ;
if (userId != null)
{
    User user = orgService.getUserHandler().findUserByName(userId) ;
    if (user != null)
    {
        String firstName = user.getFirstName();
        String lastName = user.getLastName();
        String email = user.getEmail();
    }
}
```

Below are two alternatives for retrieving the Organization Service:

```
OrganizationService service = (OrganizationService)
```

```
ExoContainerContext.getCurrentContainer().getComponentInstanceOfType(OrganizationService.class);
```

```
OrganizationService service = (OrganizationService)
```

```
PortalContainer.getInstance().getComponentInstanceOfType(OrganizationService.class);
```

6.6. SSO - Single Sign On

6.6.1. Overview

GateIn 3.1 provides some form of Single Sign On (SSO) as an integration and aggregation platform.

When logging into the portal users gain access to many systems through portlets using a single identity. In many cases, however, the portal infrastructure must be integrated with other SSO enabled systems. There are many different Identity Management solutions available. In most cases each SSO framework provides a unique way to plug into a Java EE application.

6.6.1.1. Prerequisites

In this tutorial, the SSO server is installed in a Tomcat installation. Tomcat can be obtained from <http://tomcat.apache.org>.

All the packages required for setup can be found in a zip file located at: <http://repository.jboss.org/maven2/org/gatein/ssso/ssso-packaging>. In this document we will call the directory where the file is extracted \$GATEIN_SSO_HOME.

Users are advised to not run any portal extensions that could override the data when manipulating the `gatein.ear` file directly.

Remove `$JBOSS_HOME/server/default/deploy/gatein-sample-extension.ear` and `$JBOSS_HOME/server/default/deploy/gatein-sample-portal.ear` which are packaged by default with GateIn 3.1.

6.6.2. CAS - Central Authentication Service

This Single Sign On plugin enables seamless integration between GateIn 3.1 and the CAS Single Sign On Framework. Details about CAS can be found [here](http://www.ja-sig.org/products/cas/) [http://www.ja-sig.org/products/cas/].

The integration consists of two parts; the first part consists of installing or configuring a CAS server, the second part consists of setting up the portal to use the CAS server.

6.6.2.1. CAS server

First, set up the server to authenticate against the portal login module. In this example the CAS server will be installed on Tomcat.

6.6.2.1.1. Obtaining CAS

CAS can be downloaded from <http://www.jasig.org/cas/download>.

Extract the downloaded file into a suitable location. This location will be referred to as \$CAS_HOME in the following example.

6.6.2.1.2. Modifying CAS server

To configure the web archive as desired, the simplest way is to make the necessary changes directly in CAS codebase.



Note

To complete these instructions, and perform the final build step, you will need the Apache Maven 2. You can get it [here](http://maven.apache.org/download.html) [http://maven.apache.org/download.html].

First, we need to change the default authentication handler with the one provided by GateIn 3.1.

The CAS Server Plugin makes secure authentication callbacks to a RESTful service installed on the remote GateIn server in order to authenticate a user.

In order for the plugin to function correctly, it needs to be properly configured to connect to this service. This configuration is done via the `cas.war/WEB-INF/deployerConfigContext.xml` file.

1. Open `CAS_HOME/cas-server-webapp/src/main/webapp/WEB-INF/deployerConfigContext.xml`
2. Replace:

```
<!--
  | Whereas CredentialsToPrincipalResolvers identify who it is some Credentials might
  | authenticate,
  | AuthenticationHandlers actually authenticate credentials. Here e declare the
  | AuthenticationHandlers that
  | authenticate the Principals that the CredentialsToPrincipalResolvers identified. CAS will
  | try these handlers in turn
  | until it finds one that both supports the Credentials presented and succeeds in
  | authenticating.
  +-->
<property name="authenticationHandlers">
  <list>
    <!--
      | This is the authentication handler that authenticates services by means of callback via
      | SSL, thereby validating
      | a server side SSL certificate.
      +-->
                                                                 <bean
class="org.jasig.cas.authentication.handler.support.HttpBasedServiceCredentialsAuthenticationHandler"
  p:httpClient-ref="httpClient" />
    <!--
```

| This is the authentication handler declaration that every CAS deployer will need to change before deploying CAS

| into production. The default SimpleTestUsernamePasswordAuthenticationHandler authenticates UsernamePasswordCredentials

| where the username equals the password. You will need to replace this with an AuthenticationHandler that implements your

| local authentication strategy. You might accomplish this by coding a new such handler and declaring

| edu.someschool.its.cas.MySpecialHandler here, or you might use one of the handlers provided in the adaptors modules.

```
+-->
<bean

/>
</list>
</property>
```

3. With the following (Make sure to set the host, port and context with the values corresponding to your portal). Also available in `GATEIN_SSO_HOME/cas/plugin/WEB-INF/deployerConfigContext.xml`.

```
<!--
| Whereas CredentialsToPrincipalResolvers identify who it is some Credentials might
| authenticate,
| AuthenticationHandlers actually authenticate credentials. Here we declare the
| AuthenticationHandlers that
| authenticate the Principals that the CredentialsToPrincipalResolvers identified. CAS will
| try these handlers in turn
| until it finds one that both supports the Credentials presented and succeeds in
| authenticating.
+-->
<property name="authenticationHandlers">
  <list>
    <!--
    | This is the authentication handler that authenticates services by means of callback via
    | SSL, thereby validating
    | a server side SSL certificate.
    +-->
    <bean
      class="org.jasig.cas.authentication.handler.support.HttpBasedServiceCredentialsAuthenticationHandler"
      p:httpClient-ref="httpClient" />
    <!--
```

| This is the authentication handler declaration that every CAS deployer will need to change before deploying CAS

| into production. The default SimpleTestUsernamePasswordAuthenticationHandler authenticates UsernamePasswordCredentials

| where the username equals the password. You will need to replace this with an AuthenticationHandler that implements your

| local authentication strategy. You might accomplish this by coding a new such handler and declaring

| edu.someschool.its.cas.MySpecialHandler here, or you might use one of the handlers provided in the adaptors modules.

```
+-->
```

```
<!-- Integrates with the GateIn Authentication Service to perform authentication -->
```

```
<!--
```

| Note: Modify the Plugin Configuration based on the actual information of a GateIn instance.

| The instance can be anywhere on the internet...Not necessarily on localhost where CAS is running

```
+-->
```

```
<bean class="org.gatein.sso.cas.plugin.AuthenticationPlugin">
```

```
  <property name="gateInHost"><value>localhost</value></property>
```

```
  <property name="gateInPort"><value>8080</value></property>
```

```
  <property name="gateInContext"><value>portal</value></property>
```

```
</bean>
```

```
</list>
```

```
</property>
```

4. Copy `GATEIN_SSO_HOME/cas/plugin/WEB-INF/lib/sso-cas-plugin-<VERSION>.jar` and `GATEIN_SSO_HOME/cas/plugin/WEB-INF/lib/commons-httpclient-<VERSION>.jar` into the `CAS_HOME/cas-server-webapp/src/main/webapp/WEB-INF/lib` created directory.
5. Get an installation of Tomcat and extract it into a suitable location (which will be called `TOMCAT_HOME` for these instructions).

Change the default port to avoid a conflict with the default GateIn 3.1 (for testing purposes). Edit `TOMCAT_HOME/conf/server.xml` and replace the 8080 port to 8888.



Note

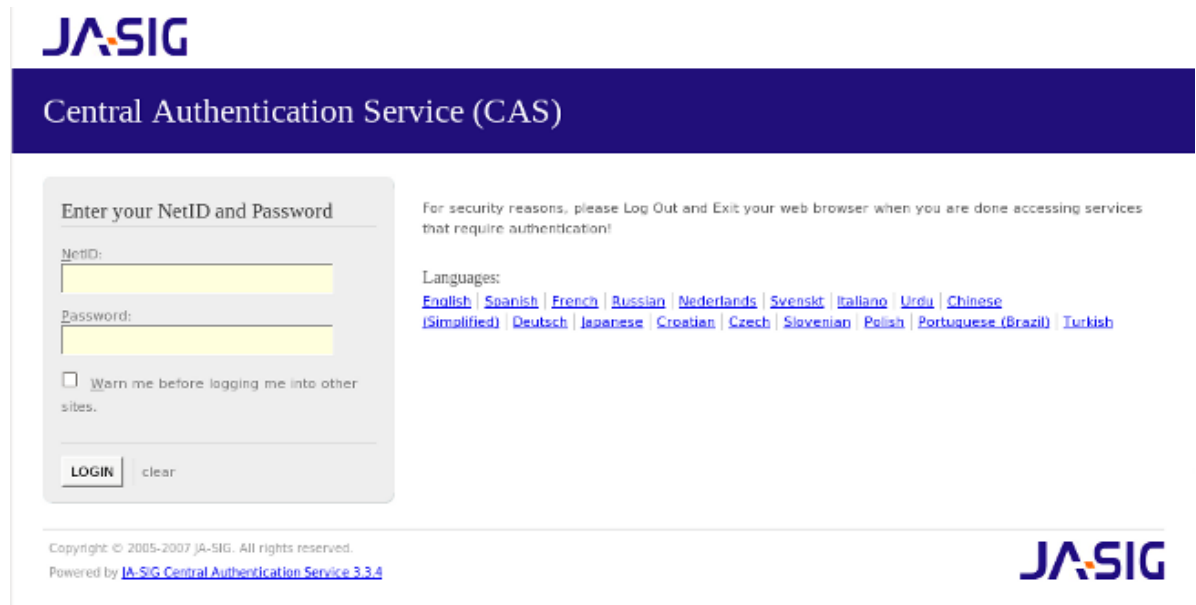
If GateIn 3.1 is running on the same machine as Tomcat, other ports need to be changed in addition to 8080 in order to avoid port conflicts. They can be changed to any free port. For example, you can change admin port from 8005 to 8805, and AJP port from 8009 to 8809.

6. Go to `CAS_HOME/cas-server-webapp` and execute the command:

```
mvn install
```

7. Copy `CAS_HOME/cas-server-webapp/target/cas.war` into `TOMCAT_HOME/webapps`.

Tomcat should start and be accessible at <http://localhost:8888/cas>. Note that at this stage login won't be available.



6.6.2.2. Setup the CAS client

1. Copy all libraries from `GATEIN_SSO_HOME/cas/gatein.ear/lib` into `JBoss_HOME/server/default/deploy/gatein.ear/lib` (Or in Tomcat, into `$GATEIN_HOME/lib`)
2. • In JBoss AS, edit `gatein.ear/META-INF/gatein-jboss-beans.xml` and uncomment this section:

```
<authentication>
  <login-module code="org.gatein.sso.agent.login.SSOLoginModule" flag="required">
  </login-module>
    <login-module code="org.exoplatform.services.security.j2ee.JbossLoginModule"
flag="required">
      <module-option name="portalContainerName">portal</module-option>
      <module-option name="realmName">gatein-domain</module-option>
    </login-module>
</authentication>
```

- In Tomcat, edit `GATEIN_HOME/conf/jaas.conf` and uncomment this section:

```
org.gatein.sso.agent.login.SSOLoginModule required
org.exoplatform.services.security.j2ee.TomcatLoginModule required
portalContainerName=portal
realmName=gatein-domain
```

3. The installation can be tested at this point:

1. Start (or restart) GateIn 3.1, and (assuming the CAS server on Tomcat is running) direct your browser to <http://localhost:8888/cas>.
2. Login with the username `root` and the password `gtin` (or any account created through the portal).

6.6.2.3. Redirect to CAS

To utilize the Central Authentication Service, GateIn 3.1 needs to redirect all user authentication to the CAS server.

Information about where the CAS is hosted must be properly configured within the GateIn 3.1 instance. The required configuration is done by modifying three files:

- In the `gatein.ear/web.war/groovy/groovy/webui/component/UIBannerPortlet.gtmpl` file modify the 'Sign In' link as follows:

```
<!--
<a class="Login" onclick="$signInAction"><%= _ctx.appRes("UILoginForm.label.Signin")%></a>
-->
<a class="Login" href="/portal/sso"><%= _ctx.appRes("UILoginForm.label.Signin")%></a>
```

- In the `gatein.ear/web.war/groovy/portal/webui/component/UILogoPortlet.gtmpl` file modify the 'Sign In' link as follows:

```
<!--
<a onclick="$signInAction"><%= _ctx.appRes("UILogoPortlet.action.signin")%></a>
-->
```

```
<a href="/portal/sso"><%= _ctx.appRes("UILogoPortlet.action.signin")%></a>
```

- Replace the entire contents of `gatein.ear/02portal.war/login/jsp/login.jsp` with:

```
<html>
<head>
  <script type="text/javascript">
    window.location = '/portal/sso';
  </script>
</head>
<body>
</body>
</html>
```

- Add the following Filters at the top of the filter chain in `gatein.ear/02portal.war/WEB-INF/web.xml`:

```
<filter>
<filter-name>LoginRedirectFilter</filter-name>
<filter-class>org.gatein.sso.agent.filter.LoginRedirectFilter</filter-class>
<init-param>
  <!-- This should point to your SSO authentication server -->

  <param-name>LOGIN_URL</param-name>
  <!--
    If casRenewTicket param value of InitiateLoginServlet is: not specified or false
  -->
  <param-value>http://localhost:8888/cas/login?service=http://localhost:8080/portal/private/
classic</param-value>
  <!--
    If casRenewTicket param value of InitiateLoginServlet is : true
  -->
  <!--
  <param-value>http://localhost:8888/cas/login?service=http://localhost:8080/portal/private
/classic&renew=true</param-value>
  -->
</init-param>
</filter>
</filter>
```



```
<filter-name>CASLogoutFilter</filter-name>
<filter-class>org.gatein.sso.agent.filter.CASLogoutFilter</filter-class>

<init-param>
  <!-- This should point to your JOSSO authentication server -->

  <param-name>LOGOUT_URL</param-name>

  <param-value>http://localhost:8888/cas/logout</param-value>

</init-param>
</filter>

<!-- Mapping the filters at the very top of the filter chain -->
<filter-mapping>
  <filter-name>LoginRedirectFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>CASLogoutFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

- Replace the `InitiateLoginServlet` declaration in `gatein.ear/02portal.war/WEB-INF/web.xml` with:

```
<servlet>
  <servlet-name>InitiateLoginServlet</servlet-name>
  <servlet-class>org.gatein.sso.agent.GenericSSOAgent</servlet-class>
  <init-param>
    <param-name>ssoServerUrl</param-name>
    <param-value>http://localhost:8888/cas</param-value>
  </init-param>
  <init-param>
    <param-name>casRenewTicket</param-name>
    <param-value>>false</param-value>
  </init-param>
</servlet>
```

Once these changes have been made, all links to the user authentication pages will redirect to the CAS centralized authentication form.

6.6.3. JOSSO

This Single Sign On plugin enables seamless integration between GateIn 3.1 and the JOSSO Single Sign On Framework. Details about JOSSO can be found [here](http://www.josso.org) [http://www.josso.org].

Setting up this integration involves two steps. The first step is to install or configure a JOSSO server, and the second is to set up the portal to use the JOSSO server.

6.6.3.1. JOSSO server

This section details setting up the JOSSO server to authenticate against the GateIn 3.1 login module.

In this example the JOSSO server will be installed on Tomcat.

6.6.3.1.1. Obtaining JOSSO

JOSSO can be downloaded from <http://sourceforge.net/projects/josso/files/>. Use the package that embeds Apache Tomcat. The integration was tested with JOSSO-1.8.1.

Once downloaded, extract the package into what will be called `JOSSO_HOME` in this example.

6.6.3.1.2. Modifying JOSSO server

1. Copy the files from `GATEIN_SSO_HOME/josso/plugin` into the Tomcat directory (`JOSSO_HOME`).

This action should replace or add the following files to the `JOSSO_HOME/webapps/josso/WEB-INF/lib` directory:

- `JOSSO_HOME/lib/josso-gateway-config.xml`
- `JOSSO_HOME/lib/josso-gateway-gatein-stores.xml`

and

- `JOSSO_HOME/webapps/josso/WEB-INF/classes/gatein.properties`

2. Edit `TOMCAT_HOME/conf/server.xml` and replace the 8080 port to 8888 to change the default Tomcat port and avoid a conflict with the default GateIn 3.1 port (for testing purposes).



Port Conflicts

If GateIn 3.1 is running on the same machine as Tomcat, other ports need to be changed in addition to 8080 in order to avoid port conflicts. They can be changed to any free port. For example, you can change admin port from 8005 to 8805, and AJP port from 8009 to 8809.

- Tomcat should now start and allow access to <http://localhost:8888/josso/signon/login.do> but at this stage login will not be available.

6.6.3.2. Setup the JOSSO client

- Copy the library files from `GATEIN_SSO_HOME/josso/gatein.ear/lib` into `gatein.ear/lib` (or into `GATEIN_HOME/lib` if GateIn 3.1 is running in Tomcat)
- Copy the file `GATEIN_SSO_HOME/josso/gatein.ear/portal.war/WEB-INF/classes/josso-agent-config.xml` into `gatein.ear/02portal.war/WEB-INF/classes` (or into `GATEIN_HOME/webapps/portal.war/WEB-INF/classes`, or `GATEIN_HOME/conf` if GateIn 3.1 is running in Tomcat)
- In JBoss AS, edit `gatein.ear/META-INF/gatein-jboss-beans.xml` and uncomment this section:

```
<authentication>
  <login-module code="org.gatein.sso.agent.login.SSOLoginModule" flag="required">
  </login-module>
    <login-module    code="org.exoplatform.services.security.j2ee.JbossLoginModule"
flag="required">
      <module-option name="portalContainerName">portal</module-option>
      <module-option name="realmName">gatein-domain</module-option>
    </login-module>
  </authentication>
```

- In Tomcat, edit `GATEIN_HOME/conf/jaas.conf` and uncomment this section:

```
org.gatein.sso.agent.login.SSOLoginModule required
org.exoplatform.services.security.j2ee.TomcatLoginModule requiredtm
portalContainerName=portal
realmName=gatein-domain
```

4. The installation can be tested at this point.

1. Start (or restart) GateIn 3.1, and (assuming the JOSSO server on Tomcat is running) direct your browser to <http://localhost:8888/josso/signon/login.do>.
2. Login with the username `root` and the password `gtm` or any account created through the portal.

6.6.3.3. Setup the portal to redirect to JOSSO

The next part of the process is to redirect all user authentication to the JOSSO server.

Information about where the JOSSO server is hosted must be properly configured within the GateIn 3.1 instance. The required configuration is done by modifying four files:

- In the `gatein.ear/web.war/groovy/groovy/webui/component/UIBannerPortlet.gtmpl` file modify the 'Sign In' link as follows:

```
<!--
<a class="Login" onclick="$signInAction"><%= _ctx.appRes("UILoginForm.label.Signin")%></a>
-->
<a class="Login" href="/portal/sso"><%= _ctx.appRes("UILoginForm.label.Signin")%></a>
```

- In the `gatein.ear/web.war/groovy/portal/webui/component/UILogoPortlet.gtmpl` file modify the 'Sign In' link as follows:

```
<!--
<a onclick="$signInAction"><%= _ctx.appRes("UILogoPortlet.action.signin")%></a>
-->
```

```
<a href="/portal/sso"><%= _ctx.appRes("UILogoPortlet.action.signin")%></a>
```

- Replace the entire contents of `gatein.ear/02portal.war/login/jsp/login.jsp` with:

```
<html>
<head>
  <script type="text/javascript">
    window.location = '/portal/sso';
  </script>
</head>
<body>
</body>
</html>
```

- Add the following Filters at the top of the filter chain in `gatein.ear/02portal.war/WEB-INF/web.xml`:

```
<filter>
  <filter-name>LoginRedirectFilter</filter-name>
  <filter-class>org.gatein.sso.agent.filter.LoginRedirectFilter</filter-class>
  <init-param>
    <!-- This should point to your SSO authentication server -->

    <param-name>LOGIN_URL</param-name>
    <param-value>http://localhost:8888/josso/signon/login.do?josso_back_to=http://
localhost:8080/portal
/private/classic</param-value>
  </init-param>
</filter>
<filter>
  <filter-name>JOSSOLogoutFilter</filter-name>

  <filter-class>org.gatein.sso.agent.filter.JOSSOLogoutFilter</filter-class>

  <init-param>
    <!-- This should point to your JOSSO authentication server -->

    <param-name>LOGOUT_URL</param-name>
```

```
<param-value>http://localhost:8888/josso/signon/logout.do</param-value>

</init-param>
</filter>

<!-- filters should be placed at the very top of the filter chain -->
<filter-mapping>
  <filter-name>LoginRedirectFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>JOSSOLogoutFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

- Replace the `InitiateLoginServlet` declaration in `gatein.ear/02portal.war/WEB-INF/web.xml` with:

```
<servlet>
  <servlet-name>InitiateLoginServlet</servlet-name>
  <servlet-class>org.gatein.sso.agent.GenericSSOAgent</servlet-class>
  <init-param>
    <param-name>ssoServerUrl</param-name>
    <param-value>http://localhost:8888/josso/signon/login.do</param-value>
  </init-param>
</servlet>
```

- Remove the `PortalLoginController` servlet declaration and mapping in `gatein.ear/02portal.war/WEB-INF/web.xml`

From now on, all links redirecting to the user authentication pages will redirect to the JOSSO centralized authentication form.

6.6.4. OpenSSO - The Open Web SSO project

This Single Sign On plugin enables seamless integration between GateIn 3.1 and the OpenSSO Single Sign On Framework. Details about OpenSSO can be found [here](https://opensso.dev.java.net/) [https://opensso.dev.java.net/].

Setting up this integration involves two steps. The first step is to install or configure an OpenSSO server, and the second is to set up the portal to use the OpenSSO server.

6.6.4.1. OpenSSO server

This section details the setting up of OpenSSO server to authenticate against the GateIn 3.1 login module.

In this example the OpenSSO server will be installed on Tomcat.

6.6.4.1.1. Obtaining OpenSSO

OpenSSO can be downloaded from <https://opensso.dev.java.net/public/use/index.html>.

Once downloaded, extract the package into a suitable location. This location will be referred to as `OPENSZO_HOME` in this example.

6.6.4.1.2. Modifying OpenSSO server

To configure the web server as desired, it is simpler to directly modify the sources.

The first step is to add the GateIn 3.1 Authentication Plugin:

The plugin makes secure authentication callbacks to a RESTful service installed on the remote GateIn 3.1 server in order to authenticate a user.

In order for the plugin to function correctly, it needs to be properly configured to connect to this service. This configuration is done via the `opensso.war/config/auth/default/AuthenticationPlugin.xml` file.

1. Obtain a copy of Tomcat and extract it into a suitable location (this location will be referred to as `TOMCAT_HOME` in this example).
2. Change the default port to avoid a conflict with the default GateIn 3.1 port (for testing purposes). Do this by editing `TOMCAT_HOME/conf/server.xml` and replacing the 8080 port to 8888.



Note

If GateIn 3.1 is running on the same machine as Tomcat, other ports need to be changed in addition to 8080 in order to avoid port conflicts. They can be changed to any free port. For example, you can change admin port from 8005 to 8805, and AJP port from 8009 to 8809.

3. Ensure the `TOMCAT_HOME/webapps/opensso/config/auth/default/AuthenticationPlugin.xml` file looks like this:

```
<?xml version='1.0' encoding="UTF-8"?>

<!DOCTYPE ModuleProperties PUBLIC "-//iPlanet//Authentication Module Properties XML
Interface 1.0 DTD//EN"
    "jar://com/sun/identity/authentication/Auth_Module_Properties.dtd">

<ModuleProperties moduleName="AuthenticationPlugin" version="1.0" >
  <Callbacks length="2" order="1" timeout="60"
    header="GateIn OpenSSO Login" >
    <NameCallback>
      <Prompt>
        Username
      </Prompt>
    </NameCallback>
    <PasswordCallback echoPassword="false" >
      <Prompt>
        Password
      </Prompt>
    </PasswordCallback>
  </Callbacks>
</ModuleProperties>
```

4. Copy `GATEIN_SSO_HOME/opensso/plugin/WEB-INF/lib/sso-opensso-plugin-
<VERSION>.jar`, `GATEIN_SSO_HOME/opensso/plugin/WEB-INF/lib/commons-
httpclient-<VERSION>.jar`, and `GATEIN_SSO_HOME/opensso/plugin/WEB-INF/lib/
commons-logging-<VERSION>.jar` into the Tomcat directory at `TOMCAT_HOME/webapps/
opensso/WEB-INF/lib`.
5. Copy `GATEIN_SSO_HOME/opensso/plugin/WEB-INF/classes/gatein.properties` into
`TOMCAT_HOME/webapps/opensso/WEB-INF/classes`
6. Tomcat should start and be able to access [http://localhost:8888/opensso/UI/
Login?realm=gatein](http://localhost:8888/opensso/UI/Login?realm=gatein). Login will not be available at this point.



Configure "gatein" realm:

1. Direct your browser to <http://localhost:8888/opensso>
2. Create default configuration
3. Login as `amadmin` and then go to tab "Configuration" -> tab "Authentication" -> link "Core" -> add new value and fill in the class name "org.gatein.sso.opensso.plugin.AuthenticationPlugin". This step is really important. Without it AuthenticationPlugin is not available among other OpenSSO authentication modules.
4. Go to tab "Access control" and create new realm called "gatein".
5. Go to "gatein" realm and click on "Authentication" tab. At the bottom in the section "Authentication chaining" click on "IdapService". Here change the selection from "Datastore", which is the default module in the authentication chain, to "AuthenticationPlugin". This enables authentication of "gatein" realm by using GateIn REST service instead of the OpenSSO LDAP server.
6. Go to "Advanced properties" and change UserProfile from "Required" to "Dynamic". This step is needed because GateIn 3.1 users are not in OpenSSO Datastore (LDAP server), so their profiles can't be obtained if "Required" is active. By using "Dynamic" all new users are automatically created in OpenSSO datastore after successful authentication.

7. Increase the user privileges to allow REST access. Go to "Access control" -> Top level realm -> "Privileges" tab -> All authenticated users, and check the last two checkboxes:

- Read and write access only for policy properties
- Read and write access to all realm and policy properties

8. Do the same for "gatein" realm.

TODO: The above OpenSSO manual configuration could be replaced by configuration files prepared in advance

6.6.4.2. Setup the OpenSSO client

1. Copy all libraries from `GATEIN_SSO_HOME/opensso/gatein.ear/lib` into `JBOSS_HOME/server/default/deploy/gatein.ear/lib` (Or, in Tomcat, into `GATEIN_HOME/lib`)
2. • In JBoss AS, edit `gatein.ear/META-INF/gatein-jboss-beans.xml` and uncomment this section

```
<authentication>
  <login-module code="org.gatein.sso.agent.login.SSOLoginModule" flag="required">
  </login-module>
    <login-module code="org.exoplatform.services.security.j2ee.JbossLoginModule"
flag="required">
      <module-option name="portalContainerName">portal</module-option>
      <module-option name="realmName">gatein-domain</module-option>
    </login-module>
  </authentication>
```

- If you are running GateIn in Tomcat, edit `$GATEIN_HOME/conf/jaas.conf` and uncomment this section

```
org.gatein.sso.agent.login.SSOLoginModule required
org.exoplatform.services.security.j2ee.TomcatLoginModule required
portalContainerName=portal
realmName=gatein-domain
```

At this point the installation can be tested:

1. Access GateIn 3.1 by going to <http://localhost:8888/opensso/UI/Login?realm=gatein> (assuming that the OpenSSO server using Tomcat is still running).

2. Login with the username `root` and the password `gtm` or any account created through the portal.

6.6.4.3. Setup the portal to redirect to OpenSSO

The next part of the process is to redirect all user authentication to the OpenSSO server.

Information about where the OpenSSO server is hosted must be properly configured within the Enterprise Portal Platform instance. The required configuration is done by modifying three files:

- In the `gatein.ear/web.war/groovy/groovy/webui/component/UIBannerPortlet.gtmpl` file modify the 'Sign In' link as follows:

```
<!--  
<a class="Login" onclick="$signInAction"><%= _ctx.appRes("UILoginForm.label.Signin")%></a>  
-->  
<a class="Login" href="/portal/sso"><%= _ctx.appRes("UILoginForm.label.Signin")%></a>
```

- In the `gatein.ear/web.war/groovy/portal/webui/component/UILogoPortlet.gtmpl` file modify the 'Sign In' link as follows:

```
<!--  
<a onclick="$signInAction"><%= _ctx.appRes("UILogoPortlet.action.signin")%></a>  
-->  
<a href="/portal/sso"><%= _ctx.appRes("UILogoPortlet.action.signin")%></a>
```

- Replace the entire contents of `gatein.ear/02portal.war/login/jsp/login.jsp` with:

```
<html>  
<head>  
  <script type="text/javascript">  
    window.location = '/portal/sso';  
  </script>  
</head>
```

```
<body>
</body>
</html>
```

- Add the following Filters at the top of the filter chain in `gatein.ear/02portal.war/WEB-INF/web.xml`:

```
<filter>
  <filter-name>LoginRedirectFilter</filter-name>
  <filter-class>org.gatein.sso.agent.filter.LoginRedirectFilter</filter-class>
  <init-param>
    <!-- This should point to your SSO authentication server -->

    <param-name>LOGIN_URL</param-name>
    <param-value>http://localhost:8888/opensso/UI/Login?realm=gatein&goto=http://
localhost:8080
/portal/private/classic</param-value>
  </init-param>
</filter>
<filter>
  <filter-name>OpenSSOLogoutFilter</filter-name>

  <filter-class>org.gatein.sso.agent.filter.OpenSSOLogoutFilter</filter-class>

  <init-param>
    <!-- This should point to your OpenSSO authentication server -->

    <param-name>LOGOUT_URL</param-name>

    <param-value>http://localhost:8888/opensso/UI/Logout</param-value>

  </init-param>
</filter>

<!-- place the filters at the top of the filter chain -->
<filter-mapping>
  <filter-name>LoginRedirectFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>OpenSSOLogoutFilter</filter-name>
```

```
<url-pattern>/*</url-pattern>  
</filter-mapping>
```

- Replace the `InitiateLoginServlet` declaration in `gatein.ear/02portal.war/WEB-INF/web.xml` with:

```
<servlet>  
  <servlet-name>InitiateLoginServlet</servlet-name>  
  <servlet-class>org.gatein.sso.agent.GenericSSOAgent</servlet-class>  
  <init-param>  
    <param-name>ssoServerUrl</param-name>  
    <param-value>http://localhost:8888/opensso</param-value>  
  </init-param>  
  <init-param>  
    <param-name>ssoCookieName</param-name>  
    <param-value>iPlanetDirectoryPro</param-value>  
  </init-param>  
</servlet>
```

From now on, all links redirecting to the user authentication pages will redirect to the OpenSSO centralized authentication form.

6.6.5. SPNEGO

SPNEGO (Simple and Protected GSSAPI Negotiation Mechanism) is used to authenticate transparently through the web browser after the user has been authenticated when logging-in his session.

A typical use case is the following:

1. Users logs into his desktop (Such as a Windows machine).
2. The desktop login is governed by Active Directory domain.
3. User then uses his browser (IE/Firefox) to access a web application (that uses JBoss Negotiation) hosted on JBoss EPP.
4. The Browser transfers the desktop sign on information to the web application.
5. JBoss EAP/AS uses background GSS messages with the Active Directory (or any Kerberos Server) to validate the user.
6. The User has seamless SSO into the web application.

6.6.5.1. Configuration

GateIn uses JBoss Negotiation to enable SPNEGO based desktop SSO for the Portal. Here are the steps to integrate SPNEGO with GateIn.

1. Activate the Host authentication Under `conf/login-config.xml`, add the following host login module:

```
<!-- SPNEGO domain -->
<application-policy name="host">
  <authentication>
    <login-module code="com.sun.security.auth.module.Krb5LoginModule"
      flag="required">
      <module-option name="storeKey">true</module-option>
      <module-option name="useKeyTab">true</module-option>
      <module-option name="principal">HTTP/server.local.network@LOCAL.NETWORK</
module-option>
      <module-option name="keyTab">/home/user/krb5keytabs/jboss.keytab</module-
option>
      <module-option name="doNotPrompt">true</module-option>
      <module-option name="debug">true</module-option>
    </login-module>
  </authentication>
</application-policy>
```

the 'keyTab' value should point to the keytab file that was generated by the `kadmin` kerberos tool. See the [Setting up your Kerberos Development Environment](#) guide for more details.

2. Extend the core authentication mechanisms to support SPNEGO Under `deployers/jbossweb.deployer/META-INF/war-deployers-jboss-beans.xml`, add 'SPNEGO' authenticators property

```
<property name="authenticators">
  <map keyClass="java.lang.String" valueClass="java.lang.String">
    <entry>
      <key>BASIC</key>
      <value>org.apache.catalina.authenticator.BasicAuthenticator</value>
    </entry>
    <entry>
      <key>CLIENT-CERT</key>
      <value>org.apache.catalina.authenticator.SSLAuthenticator</value>
    </entry>
    <entry>
```

```

    <key>DIGEST</key>
    <value>org.apache.catalina.authenticator.DigestAuthenticator</value>
  </entry>
  <entry>
    <key>FORM</key>
    <value>org.apache.catalina.authenticator.FormAuthenticator</value>
  </entry>
  <entry>
    <key>NONE</key>
    <value>org.apache.catalina.authenticator.NonLoginAuthenticator</value>
  </entry>

  <!-- Add this entry -->
  <entry>
    <key>SPNEGO</key>
    <value>org.jboss.security.negotiation.NegotiationAuthenticator</value>
  </entry>
</map>
</property>

```

3. Add the JBoss Negotiation binary copy \$GATEIN_SSO_HOME/spnego/jboss-negotiation-2.0.3.GA.jar to lib
4. Add the Gatein SSO module binaries Add \$GATEIN_SSO_HOME/spnego/gatein.ear/lib/sso-agent.jar, and \$GATEIN_SSO_HOME/spnego/gatein.ear/lib/sso-spnego.jar to deploy/gatein.ear/lib
5. Activate SPNEGO LoginModule for GateIn Modify deploy/gatein.ear/META-INF/gatein-jboss-beans.xml, so that it looks like this:

```

<deployment xmlns="urn:jboss:bean-deployer:2.0">
  <application-policy xmlns="urn:jboss:security-beans:1.0" name="gatein-domain">
    <!-- Uncomment this for Kerberos based SSO integration -->
    <authentication>
      <login-module
        code="org.gatein.sso.spnego.SPNEGOLoginModule"
        flag="required">
        <module-option name="password-stacking">useFirstPass</module-option>
        <module-option name="serverSecurityDomain">host</module-option>
      </login-module>
      <login-module
        code="org.gatein.sso.agent.login.SPNEGORolesModule"
        flag="required">
        <module-option name="password-stacking">useFirstPass</module-option>

```

```
<module-option name="portalContainerName">portal</module-option>
<module-option name="realmName">gatein-domain</module-option>
</login-module>
</authentication>
</application-policy>
</deployment>
```

6. Integrate SPNEGO support into the Portal web archive Switch GateIn authentication mechanism from the default "FORM" based to "SPNEGO" based authentication as follows:
Modify gatein.ear/02portal.war/WEB-INF/web.xml

```
<!--
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>gatein-domain</realm-name>
  <form-login-config>
    <form-login-page>/initiatelogin</form-login-page>
    <form-error-page>/errorlogin</form-error-page>
  </form-login-config>
</login-config>
-->
<login-config>
  <auth-method>SPNEGO</auth-method>
  <realm-name>SPNEGO</realm-name>
</login-config>
```

Integrate request pre-processing needed for SPNEGO via filters. Add the following filters to the web.xml at the top of the Filter chain:

```
<filter>
  <filter-name>LoginRedirectFilter</filter-name>
  <filter-class>org.gatein.sso.agent.filter.LoginRedirectFilter</filter-class>
  <init-param>
    <!-- This should point to your SSO authentication server -->

    <param-name>LOGIN_URL</param-name>

    <param-value>/portal/private/classic</param-value>

  </init-param>
</filter>
<filter>
```



```

    <filter-name>SPNEGOFilter</filter-name>
    <filter-class>org.gatein.sso.agent.filter.SPNEGOFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>LoginRedirectFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <filter-mapping>
    <filter-name>SPNEGOFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

```

7. Modify the Portal's 'Sign In' link to perform SPNEGO authentication Modify the 'Sign In' link on gatein.war/web.war/groovy/groovy/webui/component/UIBannerPortlet.gtmpl as follows:

```

<!--
<a onclick="$signInAction"><%= _ctx.appRes("UILoginForm.label.Signin")%></a>
-->
<a href="/portal/sso"><%= _ctx.appRes("UILoginForm.label.Signin")%></a>

```

8. Start the GateIn Portal

```

sudo ./run.sh -Djava.security.krb5.realm=LOCAL.NETWORK -
Djava.security.krb5.kdc=server.local.network -c spnego -b server.local.network

```

9. Login to Kerberos

```

kinit -A demo

```

You should be able to click the 'Sign In' link on the GateIn Portal and the 'demo' user from the GateIn portal should be automatically logged in

Web Services for Remote Portlets (WSRP)

7.1. Introduction

The Web Services for Remote Portlets specification defines a web service interface for accessing and interacting with interactive presentation-oriented web services. It has been produced through the efforts of the Web Services for Remote Portlets (WSRP) OASIS Technical Committee. It is based on the requirements gathered and on the concrete proposals made to the committee.

Scenarios that motivate WSRP functionality include:

- Content hosts, such as portal servers, providing Portlets as presentation-oriented web services that can be used by aggregation engines.
- Aggregating frameworks, including portal servers, consuming presentation-oriented web services offered by content providers and integrating them into the framework.

More information on WSRP can be found on the [official website for WSRP](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp) [http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp]. We suggest reading the [primer](http://www.oasis-open.org/committees/download.php/10539/wsrp-primer-1.0.html) [http://www.oasis-open.org/committees/download.php/10539/wsrp-primer-1.0.html] for a good, albeit technical, overview of WSRP.

7.2. Level of support in GateIn 3.1

The WSRP Technical Committee defined [WSRP Use Profiles](http://www.oasis-open.org/committees/download.php/3073) [http://www.oasis-open.org/committees/download.php/3073] to help with WSRP interoperability. We will refer to terms defined in that document in this section.

GateIn provides a Simple level of support for our WSRP Producer except that out-of-band registration is not currently handled. We support in-band registration and persistent local state (which are defined at the Complex level).

On the Consumer side, GateIn provides a Medium level of support for WSRP, except that we only handle HTML markup (as GateIn itself doesn't handle other markup types). We do support explicit portlet cloning and we fully support the PortletManagement interface.

As far as caching goes, we have Level 1 Producer and Consumer. We support Cookie handling properly on the Consumer and our Producer requires initialization of cookies (as we have found that it improved interoperability with some consumers). We don't support custom window states or modes, as Portal doesn't either. We do, however, support CSS on both the Producer (though it's more a function of the portlets than inherent Producer capability) and Consumer.

While we provide a complete implementation of WSRP 1.0, we do need to go through the [Conformance statements](http://www.oasis-open.org/committees/download.php/6018) [http://www.oasis-open.org/committees/download.php/6018] and

perform more interoperability testing (an area that needs to be better supported by the WSRP Technical Committee and Community at large).



Note

As of version 3.1 of GateIn, WSRP is only activated and supported when GateIn is deployed on JBoss Application Server.

7.3. Deploying GateIn's WSRP services

GateIn provides a complete support of WSRP 1.0 standard interfaces and offers both consumer and producer services. WSRP support is provided by the following files, assuming `$GATEIN_HOME` is where GateIn has been installed, `$WSRP_VERSION` (at the time of the writing, it was 1.1.0-GA) is the version of the WSRP component and `$PORTAL_VERSION` (at the time of the writing, it was 3.0.1-GA) is the current GateIn version:

- `$GATEIN_HOME/wsrp-admin-gui.war`, which contains the WSRP Configuration portlet with which you can configure consumers to access remote servers and how the WSRP producer is configured.
- `$GATEIN_HOME/wsrp-producer.war`, which contains the WSRP producer web application.
- `$GATEIN_HOME/lib/wsrp-common-$WSRP_VERSION.jar`, which contains common classes needed by the different WSRP libraries.
- `$GATEIN_HOME/lib/wsrp-consumer-$WSRP_VERSION.jar`, which contains the WSRP consumer.
- `$GATEIN_HOME/lib/wsrp-integration-api-$WSRP_VERSION.jar`, which contains the API classes needed to integrate the WSRP component into portals.
- `$GATEIN_HOME/lib/wsrp-producer-lib-$WSRP_VERSION.jar`, which contains the classes needed by the WSRP producer.
- `$GATEIN_HOME/lib/wsrp-wsrp1-ws-$WSRP_VERSION.jar`, which contains the generated JAX-WS classes for WSRP version 1.
- `$GATEIN_HOME/lib/wsrp-wsrp2-ws-$WSRP_VERSION.jar`, which contains the generated JAX-WS classes for WSRP version 2.
- `$GATEIN_HOME/lib/gatein.portal.component.wsrp-$PORTAL_VERSION.jar`, which contains the code to integrate the WSRP service into GateIn.

If you're not going to use WSRP in GateIn, you can remove `$GATEIN_HOME/lib/gatein.portal.component.wsrp-$PORTAL_VERSION.jar` from your GateIn distribution to easily deactivate WSRP support. Of course, if you want to trim your installation, you can also remove all the files mentioned above.

7.3.1. Considerations to use WSRP when running GateIn on a non-default port or hostname

JBoss WS (the web service stack that GateIn uses) should take care of the details of updating the port and host name used in WSDL. See the [JBoss WS user guide on that subject](http://community.jboss.org/wiki/JBossWS-UserGuide#Configuration) [http://community.jboss.org/wiki/JBossWS-UserGuide#Configuration] for more details.

Of course, if you have modified you have modified the host name and port on which your server runs, you will need to update the configuration for the consumer used to consume GateIn's 'self' producer. Please refer to the [Section 7.6, "Consuming remote WSRP portlets in GateIn"](#) to learn how to do so.

7.3.2. Considerations to use WSRP with SSL

It is possible to use WSRP over SSL for secure exchange of data. Please refer to the [instructions](http://community.jboss.org/wiki/ConfiguringWSRPforuseoverSSL) [http://community.jboss.org/wiki/ConfiguringWSRPforuseoverSSL] on how to do so from [GateIn's wiki](http://community.jboss.org/wiki/GateIn) [http://community.jboss.org/wiki/GateIn].

7.4. Making a portlet remotable



Note

Only JSR-286 (Portlet 2.0) portlets can be made remotable as the mechanism to expose a portlet to WSRP relies on a JSR-286-only functionality.

GateIn does **NOT**, by default, expose local portlets for consumption by remote WSRP consumers. In order to make a portlet remotely available, it must be made "remotable" by marking it as such in the associated `portlet.xml`. This is accomplished by using a specific `org.gatein.pc.remotable container-runtime-option`. Setting its value to `true` makes the portlet available for remote consumption, while setting its value to `false` will not publish it remotely. As specifying the remotable status for a portlet is optional, you do not need to do anything if you don't need your portlet to be available remotely.

In the following example, the "BasicPortlet" portlet is specified as being remotable.

Example 7.1.

```
<?xml version="1.0" standalone="yes"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd http://
java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
```

```
        version="2.0">
<portlet-app>
  <portlet>
    <portlet-name>BasicPortlet</portlet-name>

    ...

    <container-runtime-option>
      <name>org.gatein.pc.remotable</name>
      <value>true</value>
    </container-runtime-option>
  </portlet>
</portlet-app>
```

It is also possible to specify that all the portlets declared within a given portlet application to be remotable by default. This is done by specifying the `container-runtime-option` at the `portlet-app` element level. Individual portlets can override that value to not be remotely exposed. Let's look at an example:

Example 7.2.

```
<?xml version="1.0" standalone="yes"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd http://
java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  version="2.0">
<portlet-app>

  <portlet>
    <portlet-name>RemotelyExposedPortlet</portlet-name>
    ...
  </portlet>
  <portlet>
    <portlet-name>NotRemotelyExposedPortlet</portlet-name>
    ...
    <container-runtime-option>
      <name>org.gatein.pc.remotable</name>
      <value>false</value>
    </container-runtime-option>
  </portlet>
```

```
<container-runtime-option>
  <name>org.gatein.pc.remotable</name>
  <value>true</value>
</container-runtime-option>
</portlet-app>
```

In the example above, we defined two portlets. The `org.gatein.pc.remotable` `container-runtime-option` being set to `true` at the `portlet-app` level, all portlets defined in this particular portlet application are exposed remotely by GateIn's WSRP producer. Note, however, that it is possible to override the default behavior: specifying a value for the `org.gatein.pc.remotable` `container-runtime-option` at the `portlet` level will take precedence over the default. In the example above, the `RemotelyExposedPortlet` inherits the `remotable` status defined at the `portlet-app` level since it does not specify a value for the `org.gatein.pc.remotable` `container-runtime-option`. The `NotRemotelyExposedPortlet`, however, overrides the default behavior and is not remotely exposed. Note that in the absence of a top-level `org.gatein.pc.remotable` `container-runtime-option` value set to `true`, portlets are NOT remotely exposed.

7.5. Consuming GateIn's WSRP portlets from a remote Consumer

WSRP Consumers vary a lot as far as how they are configured. Most of them require that you specify the URL for the Producer's WSDL definition. Please refer to your Consumer's documentation for specific instructions. For instructions on how to do so in GateIn, please refer to [Section 7.6, "Consuming remote WSRP portlets in GateIn"](#).

GateIn's Producer is automatically set up when you deploy a portal instance with the WSRP service. You can access the WSDL file at `http://{hostname}:{port}/wsrp-producer/MarkupService?wsdl`. The default hostname is `localhost` and the default port is `8080`.

7.6. Consuming remote WSRP portlets in GateIn

7.6.1. Overview

To be able to consume WSRP portlets exposed by a remote producer, GateIn's WSRP consumer needs to know how to access that remote producer. One can configure access to a remote producer using WSRP Producer descriptors. Alternatively, a portlet is provided to configure remote producers.

Once a remote producer has been configured, the portlets that it exposes are then available in the Application Registry to be added to categories and then to pages.

As a way to test the WSRP producer service and to check that the portlets that you want to expose remotely are correctly published via WSRP, a default consumer named `self`, that consumes the portlets exposed by GateIn's producer, has been configured.

7.6.2. Configuring a remote producer walk-through

Let's work through the steps of defining access to a remote producer so that its portlets can be consumed within GateIn. We will configure access to Oracle's public WSRP producer. We will first examine how to do so using the configuration portlet. We will then show how the same result can be accomplished with a producer descriptor, though it is far easier to do so via the configuration portlet.



Note

Some WSRP producers do not support chunked encoding that is activated by default by JBoss WS. If your producer does not support chunked encoding, your consumer will not be able to properly connect to the producer. This will manifest itself with the following error: `Caused by: org.jboss.ws.WSException: Invalid HTTP server response [503] - Service Unavailable`. Please see this GateIn's [wiki page](http://community.jboss.org/wiki/Workaroundwhenchunkedencodingisnotsupported) [http://community.jboss.org/wiki/Workaroundwhenchunkedencodingisnotsupported] for more details.

7.6.2.1. Using the configuration portlet

GateIn provides a portlet to configure access (among other functions) to remote WSRP Producers graphically. It isn't, however, installed by default, so the first thing we will need to do is to install the WSRP configuration portlet using the Application Registry.

Use the usual procedure to log in as a Portal administrator and go to the Application Registry. With the default install, you can just go to <http://localhost:8080/portal/login?initialURI=%2Fportal%2Fprivate%2Fclassic%2Fadministration%2Fregistry&username=root&password=gtn> [http://localhost:8080/portal/login?initialURI=%2Fportal%2Fprivate%2Fclassic%2Fadministration%2Fregistry&username=root&password=gtn] Add the WSRP Configuration portlet to the Administration category. If you use the Import Applications functionality, the WSRP Configuration portlet will be automatically added to the Administration category.

Now that the portlet is added to a category, it can be added to a page and used. We recommend adding it to the same page as the Application Registry as operations relating to WSRP and adding portlets to categories are somewhat related as we will see. Go ahead and add the WSRP Configuration portlet to the page using the standard procedure.

If all went well, you should see something similar to this:

This screen presents all the configured Consumers associated with their status and possible actions on them. A Consumer can be active or inactive. Activating a Consumer means that it is ready to act as a portlet provider. Note also that a Consumer can be marked as requiring refresh meaning that the information held about it might not be up to date and refreshing it from the remote Producer might be a good idea. This can happen for several reasons: the service description for that remote Producer has not been fetched yet, the cached version has expired or modifications have been made to the configuration that could potentially invalidate it, thus requiring re-validation of the information.

Next, we create a new Consumer which we will call `oracle`. Type " `oracle`" in the "Create a consumer named:" field then click on "Create consumer":

Create a consumer named: **Create Consumer**

You should now see a form allowing you to enter/modify the information about the Consumer. Set the cache expiration value to 300 seconds, leave the default timeout value for web services (WS) operations and enter the WSDL URL for the producer in the text field and press the "Refresh & Save" button:

This will retrieve the service description associated with the Producer which WSRP interface is described by the WSDL file found at the URL you just entered. In our case, querying the service description will allow us to learn that the Producer requires registration but didn't request any registration property:

✓ Refresh was successful.

Consumer 'oracle' configuration **active**

Producer id:
Cache expiration: (seconds before expiration)
Timeout for WS operations: (milliseconds before timeout)
Producer WSDL URL:
Registration information:

Current registration information:

Registration is indicated as required without registration properties.

Registration context: Handle:C:148.87.122.191:3fa5ac:1274c6b80c7:1de6

The Consumer for the `oracle` Producer should now be available as a portlet provider and be ready to be used.

Now, assuming that the producer required a value for an `email` registration property, GateIn's WSRP consumer would have informed you that you were missing some information:

❗ **Error: Refresh failed (probably because the registration information was not valid).**

Consumer 'self' configuration **inactive**

Producer id:
Cache expiration: (seconds before expiration)
Timeout for WS operations: (milliseconds before timeout)
Producer WSDL URL:
Registration information:

Current registration information:

Name	Description	Value
email	A valid contact email.	<input type="text"/> ❗ Error: Missing value



Note

At this point, there is no automated way to learn about which possible values (if any) are expected by the remote Producer. Sometimes, the possible values will be indicated in the registration property description but this is not always the case... Please refer to the specific Producer's documentation.

If you entered "example@example.com" as the value for the registration property and press "Save & Refresh" once more, you would have seen something similar to:

✓ Refresh was successful.

Consumer 'self' configuration
active

Producer id:

Cache expiration: (seconds before expiration)

Timeout for WS operations: (milliseconds before timeout)

Producer WSDL URL:

Registration information:

Current registration information:

Name	Description	Value
email	A valid contact email.	<input type="text" value="example@example.com"/>

Update properties

Registration context:

Handle: 03b0a000c0a801327ec0c8c50ffa27db

Erase local registration

Refresh & Save Cancel

7.6.2.2. Using XML

While we recommend you use the WSRP Configuration portlet to configure Consumers, we provide an alternative way to configure consumers by editing the XML file located at `$GATEIN_HOME/lib/wsrp-consumer-$WSRP_VERSION.jar/conf/wsrp-consumers-config.xml`.

```
<?xml version='1.0' encoding='UTF-8' ?>
<deployments xmlns="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0 http://
www.jboss.org/portal/xsd/gatein_wsrp_consumer_1_0.xsd">
  <deployment>
    <wsrp-producer id="self" expiration-cache="300" ws-timeout="30000">
      <endpoint-wsdl-url>http://localhost:8080/wsrp-producer/MarkupService?wsdl</endpoint-
wsdl-url>
      <registration-data>
        <property>
          <name>email</name>
          <lang>en</lang>
          <value>example@example.com</value>
        </property>
      </registration-data>
    </wsrp-producer>
  </deployment>
</deployments>
```

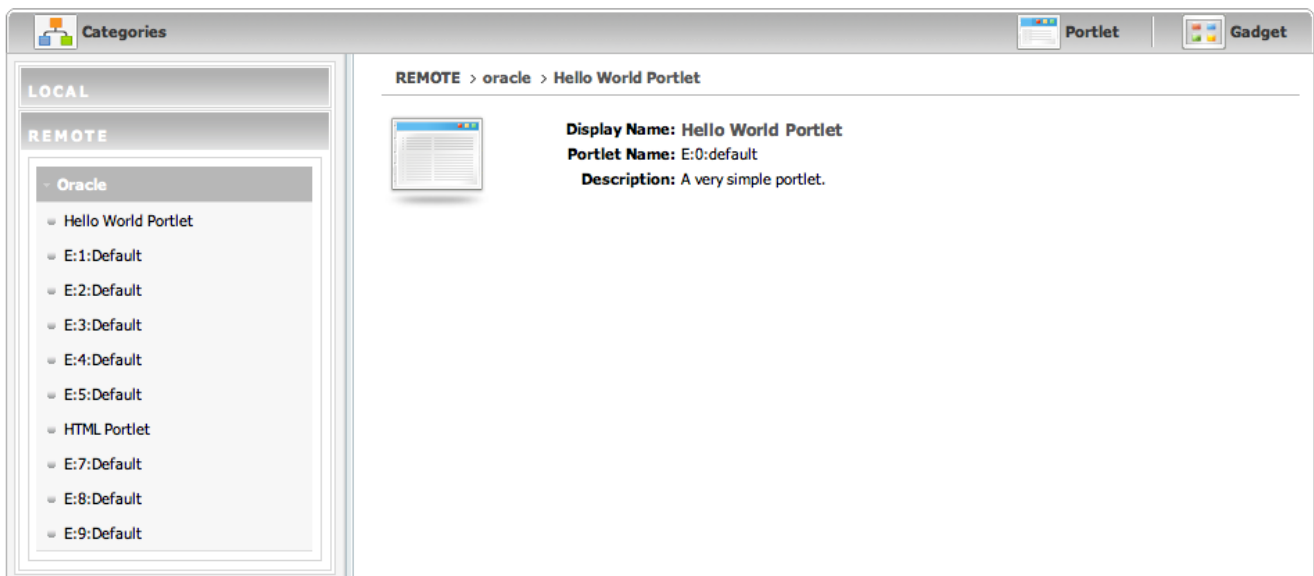
```
</wsrp-producer>
</deployment>
<deployment>
  <wsrp-producer id="oracle" expiration-cache="300">
    <endpoint-wsdl-url>http://portalstandards.oracle.com/portletapp/portlets?WSDL</endpoint-
wsdl-url>
    <registration-data/>
  </wsrp-producer>
</deployment>
</deployments>
```

The file as shown above specifies access to two producers: `self`, which consumes GateIn's own WSRP producer albeit in a version that assumes that the producer requires a value for an `email` registration property, and `oracle`, which consumes Oracle's public producer, both in configurations as shown in the walk-through above.

We will look at the details of the meaning of elements later on.

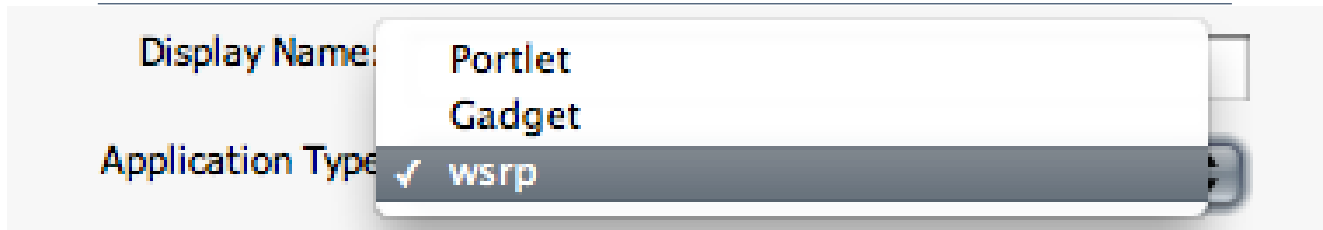
7.6.2.3. Configuring access to a remote portlet

If we go back to the Application Registry and examine the available portlets by clicking on the Portlet link, you will now be able to see the remote portlets if you click on the REMOTE tab in the left column:



These portlets are, of course, available to be used such as regular portlets: they can be used in categories and added to pages. If you use the Import Applications functionality, they will also be automatically imported in categories based on the keywords they define.

More specifically, if you want to add a WSRP portlet to a category, you can access these portlets by selecting `wsrp` in the Application Type drop-down menu:



7.6.3. Configuring access to remote producers via XML

While we recommend you use the WSRP Configuration portlet to configure Consumers, we provide an alternative way to configure consumers by editing the XML file located at `$GATEIN_HOME/lib/wsrp-consumer-$WSRP_VERSION.jar/conf/wsrp-consumers-config.xml`.



Note

An XML Schema defining which elements are available to configure Consumers via XML can be found in `$GATEIN_HOME/lib/wsrp-integration-api-$WSRP_VERSION.jar/xsd/gatein_wsrp_consumer_1_0.xsd`



Note

It is important to note how the XML consumers configuration file is processed. It is read the first time the WSRP service starts and the associated information is then put under control of JCR (Java Content Repository). Subsequent launches of the WSRP service will use the JCR-stored information for all producers already known to GateIn. More specifically, `wsrp-consumers-config.xml` file is scanned for producer identifiers. Any identifier that is already known will be bypassed and the JCR information associated with this remote producer will be used. The information defined at the XML level is only processed for producer definition for which no information is already present in JCR. Therefore, if you wish to delete a producer configuration, you need to delete the associated information in the database (this can be accomplished using the configuration portlet as we saw in [Section 7.6.2.1, "Using the configuration portlet"](#)) AND remove the associated information in `wsrp-consumers-config.xml` (if such information exists) as the producer will be re-created the next time the WSRP is launched if that information is not removed.

7.6.3.1. Required configuration information

Let's now look at which information needs to be provided to configure access to a remote producer.

First, we need to provide an identifier for the producer we are configuring so that we can refer to it afterwards. This is accomplished via the mandatory `id` attribute of the `<wsrp-producer>` element.

GateIn also needs to learn about the remote producer's endpoints to be able to connect to the remote web services and perform WSRP invocations. This is accomplished by specifying the URL for the WSDL description for the remote WSRP service, using the `<endpoint-wsdl-url>` element.

Both the `id` attribute and `<endpoint-wsdl-url>` elements are required for a functional remote producer configuration.

7.6.3.2. Optional configuration

It is also possible to provide additional configuration, which, in some cases, might be important to establish a proper connection to the remote producer.

One such optional configuration concerns caching. To prevent useless roundtrips between the local consumer and the remote producer, it is possible to cache some of the information sent by the producer (such as the list of offered portlets) for a given duration. The rate at which the information is refreshed is defined by the `expiration-cache` attribute of the `<wsrp-producer>` element which specifies the refreshing period in seconds. For example, providing a value of 120 for `expiration-cache` means that the producer information will not be refreshed for 2 minutes after it has been somehow accessed. If no value is provided, GateIn will always access the remote producer regardless of whether the remote information has changed or not. Since, in most instances, the information provided by the producer does not change often, we recommend that you use this caching facility to minimize bandwidth usage.

It is also possible to define a timeout after which WS operations are considered as failed. This is helpful to avoid blocking the WSRP service, waiting forever on the service that doesn't answer. Use the `ws-timeout` attribute of the `<wsrp-producer>` element to specify how many milliseconds the WSRP service will wait for a response from the remote producer before timing out and giving up.

Additionally, some producers require consumers to register with them before authorizing them to access their offered portlets. If you know that information beforehand, you can provide the required registration information in the producer configuration so that the consumer can register with the remote producer when required.



Note

At this time, though, only simple String properties are supported and it is not possible to configure complex registration data. This should, however, be sufficient for most cases.

Registration configuration is done via the `<registration-data>` element. Since GateIn can generate the mandatory information for you, if the remote producer does not require any registration properties, you only need to provide an empty `<registration-data>` element. Values for the registration properties required by the remote producer can be provided via `<property>` elements. See the example below for more details. Additionally, you can override the default

consumer name automatically provided by GateIn via the `<consumer-name>` element. If you choose to provide a consumer name, please remember that this should uniquely identify your consumer.

7.6.4. Examples

Here is the configuration of the `self` producer as found in `default-wsrp.xml` with a cache expiring every five minutes and with a 30 second timeout for web service operations.

Example 7.3.

```
<?xml version='1.0' encoding='UTF-8' ?>
<deployments xmlns="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0 http://
www.jboss.org/portal/xsd/gatein_wsrp_consumer_1_0.xsd">
  <deployment>
    <wsrp-producer id="self" expiration-cache="300" ws-timeout="30000">
      <endpoint-wsdl-url>http://localhost:8080/wsrp-producer/MarkupService?wsdl</endpoint-
wsdl-url>
      <registration-data/>
    </wsrp-producer>
  </deployment>
</deployments>
```

Here is an example of a WSRP descriptor with registration data and cache expiring every minute:

Example 7.4.

```
<?xml version='1.0' encoding='UTF-8' ?>
<deployments xmlns="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0 http://
www.jboss.org/portal/xsd/gatein_wsrp_consumer_1_0.xsd">
<deployments>
  <deployment>
    <wsrp-producer id="AnotherProducer" expiration-cache="60">
      <endpoint-wsdl-url>http://example.com/producer/producer?WSDL</endpoint-wsdl-url>
      <registration-data>
        <property>
```

```

    <name>property name</name>
    <lang>en</lang>
    <value>property value</value>
  </property>
</registration-data>
</wsrp-producer>
</deployment>
</deployments>

```

7.7. Consumers maintenance

7.7.1. Modifying a currently held registration

7.7.1.1. Registration modification for service upgrade

Producers often offer several levels of service depending on consumers' subscription levels (for example). This is implemented at the WSRP level with the registration concept: producers can assert which level of service to provide to consumers based on the values of given registration properties.

There might also be cases where you just want to update the registration information because it has changed. For example, the producer required you to provide a valid email and the previously email address is not valid anymore and needs to be updated.

It is therefore sometimes necessary to modify the registration that concretizes the service agreement between a consumer and a producer. Let's take the example of the producer requiring an email we configured in [Section 7.6.2.1, "Using the configuration portlet"](#). If you recall, the producer was requiring registration and required a value to be provided for the `email` property.

Suppose now that we would like to update the email address that we provided to the remote producer. We will need to tell the producer that our registration data has been modified. Let's see how to do this. Assuming you have configured access to the producer as previously described, please go to the configuration screen for the `self` producer and modify the value of `email` to `foo@example.com` instead of `example@example.com`:

Current registration information:		
Name	Description	Value
email	A valid email.	<input type="text" value="foo@example.com"/>

Now click on "Update properties" to save the change. A "Modify registration" button should now appear to let you send this new data to the remote producer:

Current registration information:

Name	Description	Value
email	A valid email.	<input type="text" value="foo@example.com"/>

Click on this new button and, if everything went well and your updated registration has been accepted by the remote producer, you should see something similar to:

- ✓ **Successfully modified registration!**
- ✓ **Refresh was successful.**

Consumer 'self' configuration **active**

Producer id:

Cache expiration: (seconds before expiration)

Timeout for WS operations: (milliseconds before timeout)

Producer WSDL URL:

Registration information:

Current registration information:

Name	Description	Value
email	A valid email.	<input type="text" value="foo@example.com"/>

Registration context:

Handle: 07d57d29c0a801325a0da57a96c12a32

7.7.1.2. Registration modification on producer error

It can also happen that a producer administrator decided to change its requirement for registered consumers. In this case, invoking operations on the producer will fail with an `OperationFailedFault`. GateIn will attempt to help you in this situation. Let's walk through an example using the `self` producer. Let's assume that registration is requiring a valid value for an `email` registration property (as we have seen so far). If you go to the configuration screen for this producer, you should see:

Consumer 'self' configuration active

Producer id:

Cache expiration: (seconds before expiration)

Timeout for WS operations: (milliseconds before timeout)

Producer WSDL URL:

Registration information:

Current registration information:

Name	Description	Value
email	A valid email.	<input type="text" value="foo@example.com"/>

Registration context: Handle:07d57d29c0a801325a0da57a96c12a32

Now suppose that the administrator of the producer now additionally requires a value to be provided for a `name` registration property. We will actually see how to do perform this operation in GateIn when we examine how to configure GateIn's producer in [Section 7.8, "Configuring GateIn's WSRP Producer"](#). Operations with this producer will now fail. If you suspect that a registration modification is required, you should go to the configuration screen for this remote producer and refresh the information held by the consumer by pressing "Refresh & Save":

Error: Either local or remote information has been changed, you should modify your registration with the remote producer. The new local information will be saved but your current registration data will be used until you successfully modify the registration with the producer.

Consumer 'self' configuration
inactive (refresh needed)

Producer id:

Cache expiration:

(seconds before expiration)

Timeout for WS operations:

(milliseconds before timeout)

Producer WSDL URL:

Registration information:

Current registration information:

Name	Description	Value
email	A valid email.	<input type="text" value="foo@example.com"/>

Expected registration information:

Name	Description	Value
email	A valid email.	<input type="text" value="foo@example.com"/>
name	A contact name.	<input type="text"/> Error: Missing value

Registration context:

Handle: 07d57d29c0a801325a0da57a96c12a32

As you can see, the configuration screen now shows the currently held registration information and the expected information from the producer. Enter a value for the `name` property and then click on "Modify registration". If all went well and the producer accepted your new registration data, you should see something similar to:

- ✓ Successfully modified registration!
- ✓ Refresh was successful.

Consumer 'self' configuration active

Producer id:

Cache expiration: (seconds before expiration)

Timeout for WS operations: (milliseconds before timeout)

Producer WSDL URL:

Registration information:

Current registration information:

Name	Description	Value
email	A valid email.	<input type="text" value="foo@example.com"/>
name	A contact name.	<input type="text" value="Foo Bar"/>

Registration context: Handle:07d57d29c0a801325a0da57a96c12a32



Note

As of WSRP 1, it is rather difficult to ascertain for sure what caused an `OperationFailedFault` as it is the generic exception returned by producers if something didn't quite happen as expected during a method invocation. This means that `OperationFailedFault` can be caused by several different reasons, one of them being a request to modify the registration data. Please take a look at the log files to see if you can gather more information as to what happened. WSRP 2 introduces an exception that is specific to a request to modify registrations thus reducing the ambiguity that currently exists.

7.7.2. Consumer operations

Several operations are available from the consumer list view of the WSRP configuration portlet:

Create a consumer named:

Consumer [status: active , inactive, (refresh needed)]	Actions
self (active)	<input type="button" value="Configure"/> <input type="button" value="Refresh"/> <input type="button" value="Deactivate"/> <input type="button" value="Deregister"/> <input type="button" value="Delete"/>

The available operations are:

- Configure: displays the consumer details and allows user to edit them

- Refresh: forces the consumer to retrieve the service description from the remote producer to refresh the local information (offered portlets, registration information, etc.)
- Activate/Deactivate: activates/deactivates a consumer, governing whether it will be available to provide portlets and receive portlet invocations
- Register/Deregister: registers/deregisters a consumer based on whether registration is required and/or acquired
- Delete: destroys the consumer, after deregistering it if it was registered

7.7.3. Erasing local registration data

There are rare cases where it might be required to erase the local information without being able to deregister first. This is the case when a consumer is registered with a producer that has been modified by its administrator to not require registration anymore. If that ever was to happen (most likely, it won't), you can erase the local registration information from the consumer so that it can resume interacting with the remote producer. To do so, click on "Erase local registration" button next to the registration context information on the consumer configuration screen:

Registration context:

Handle:07d57d29c0a801325a0da57a96c12a32

Erase local registration

Warning: This operation is dangerous as it can result in inability to interact with the remote producer if invoked when not required. A warning screen will be displayed to give you a chance to change your mind:



Delete local registration for 'self' consumer?

Warning: You are about to delete the local registration information for the 'self' consumer! This is only needed if this consumer had previously registered with the remote producer and this producer has been modified to not require registration anymore. Only erase local registration information if you experience errors with the producer due to this particular situation. Erasing local registration when not required might lead to inability to work with this producer anymore.

Are you sure you want to proceed?

Erase local registration

Cancel

7.8. Configuring GateIn's WSRP Producer

7.8.1. Overview

You can configure the behavior of Portal's WSRP Producer by using the WSRP administration interface, which is the preferred way, or by editing the `$GATEIN_HOME/wsrp-producer.war/WEB-INF/conf/producer/config.xml` file. Several aspects can be modified with respects to whether registration is required for consumers to access the Producer's services. An XML Schema for the configuration format is available at `$GATEIN_HOME/lib/wsrp-integration-api-$WSRP_VERSION.jar/xsd/gatein_wsrp_producer_1_0.xsd`.

7.8.2. Default configuration

The default producer configuration is to require that consumers register with it before providing access its services but does not require any specific registration properties (apart from what is mandated by the WSRP standard). It does, however, require consumers to be registered before sending them a full service description. This means that our WSRP producer will not provide the list of offered portlets and other capabilities to unregistered consumers. The producer also uses the default `RegistrationPolicy` paired with the default `RegistrationPropertyValidator`. We will look into property validators in greater detail later in [Section 7.8.3, "Registration configuration"](#). Suffice to say for now that this allows users to customize how Portal's WSRP Producer decides whether a given registration property is valid or not.

GateIn provides a web interface to configure the producer's behavior. You can access it by clicking on the "Producer Configuration" tab of the "WSRP" page of the "admin" portal. Here's what you should see with the default configuration:

The screenshot shows a web interface for configuring the WSRP Producer. It includes three checked checkboxes: "Access to full service description requires consumers to be registered.", "Use strict WSRP compliance.", and "Requires registration. Modifying this information will trigger invalidation of consumer registrations." Below these are two text input fields: "Registration policy class name:" with the value "org.gatein.registration.policies.DefaultRegistrationPolicy" and "Registration property validator class name:" with the value "org.gatein.registration.policies.DefaultRegistrationPropertyValidator". At the bottom, there is a section titled "Registration properties" with an "Add property" button. Below this section, it says "No specified required registration properties." On the right side, there are "Save" and "Cancel" buttons.

As would be expected, you can specify whether or not the producer will send the full service description to unregistered consumers, and, if it requires registration, which `RegistrationPolicy` to use (and, if needed, which `RegistrationPropertyValidator`), along with required registration property description for which consumers must provide acceptable values to successfully register.

7.8.3. Registration configuration

In order to require consumers to register with Portal's producer before interacting with it, you need to configure Portal's behavior with respect to registration. Registration is optional, as are registration properties. The producer can require registration without requiring consumers to pass

any registration properties as is the case in the default configuration. Let's configure our producer starting with a blank state:

- ☐ Access to full service description requires consumers to be registered.
- ☒ Use strict WSRP compliance.
- ☐ Requires registration. Modifying this information will trigger invalidation of consumer registrations.

Save

Cancel

We will allow unregistered consumers to see the list of offered portlets so we leave the first checkbox ("Access to full service description requires consumers to be registered.") unchecked. We will, however, specify that consumers will need to be registered to be able to interact with our producer. Check the second checkbox ("Requires registration. Modifying this information will trigger invalidation of consumer registrations."). The screen should now refresh and display:

Consumers Configuration
Producer Configuration

☒ Access to full service description requires consumers to be registered.
☒ Use strict WSRP compliance.
☒ Requires registration. Modifying this information will trigger invalidation of consumer registrations.

Registration policy class name:

Registration property validator class name:

Registration properties
Add property

No specified required registration properties.

Save
Cancel

You can specify the fully-qualified name for your `RegistrationPolicy` and `RegistrationPropertyValidator` there. We will keep the default value. See [Section 7.8.3.1, "Customization of Registration handling behavior"](#) for more details. Let's add, however, a registration property called `email`. Click "Add property" and enter the appropriate information in the fields, providing a description for the registration property that can be used by consumers to figure out its purpose:

- ☐ Access to full service description requires consumers to be registered.
- ☒ Use strict WSRP compliance.
- ☒ Requires registration. Modifying this information will trigger invalidation of consumer registrations.

Registration policy class name:

Registration property validator class name:

Registration properties
Add property

Name	Type	Label	Hint	Action
email	xsd:string	A valid contact email.	A valid contact email.	Remove

Save
Cancel

Press "Save" to record your modifications.



Note

At this time, only String (xsd:string) properties are supported. If your application requires more complex properties, please let us know.



Note

If consumers are already registered with the producer, modifying the configuration of required registration information will trigger the invalidation of held registrations, requiring consumers to modify their registration before being able to access the producer again. We saw the consumer side of that process in [Section 7.7.1.2](#), *"Registration modification on producer error"*.

7.8.3.1. Customization of Registration handling behavior

Registration handling behavior can be customized by users to suit their Producer needs. This is accomplished by providing an implementation of the `RegistrationPolicy` interface. This interface defines methods that are called by Portal's Registration service so that decisions can be made appropriately. A default registration policy that provides basic behavior is provided and should be enough for most user needs.

While the default registration policy provides default behavior for most registration-related aspects, there is still one aspect that requires configuration: whether a given value for a registration property is acceptable by the WSRP Producer. This is accomplished by plugging a `RegistrationPropertyValidator` in the default registration policy. This allows users to define their own validation mechanism.

Please refer to the Javadoc™ for `org.jboss.portal.registration.RegistrationPolicy` and `org.jboss.portal.Registration.policies.RegistrationPropertyValidator` for more details on what is expected of each method.

Defining a registration policy is required for the producer to be correctly configured. This is accomplished by specifying the qualified class name of the registration policy. Since we anticipate that most users will use the default registration policy, it is possible to provide the class name of your custom property validator instead to customize the default registration policy behavior. Note that property validators are only used by the default policy.



Note

Since the policy or the validator are defined via their class name and dynamically loaded, it is important that you make sure that the identified class is available to the application server. One way to accomplish that is to deploy your policy implementation as JAR file in your AS instance deploy directory. Note also that,

since both policies and validators are dynamically instantiated, they must provide a default, no-argument constructor.

7.8.4. WSRP validation mode

The lack of conformance kit and the wording of the WSRP specification leaves room for differing interpretations, resulting in interoperability issues. It is therefore possible to encounter issues when using consumers from different vendors. We have experienced such issues and have introduced a way to relax the validation that our WSRP producer performs on the data provided by consumers to help with interoperability by accepting data that would normally be invalid. Note that we only relax our validation algorithm on aspects of the specification that are deemed harmless such as invalid language codes.

By default, the WSRP producer is configured in strict mode. If you experience issues with a given consumer, you might want to try to relax the validation mode. This is accomplished by unchecking the "Use strict WSRP compliance." checkbox on the Producer configuration screen.

Advanced Development

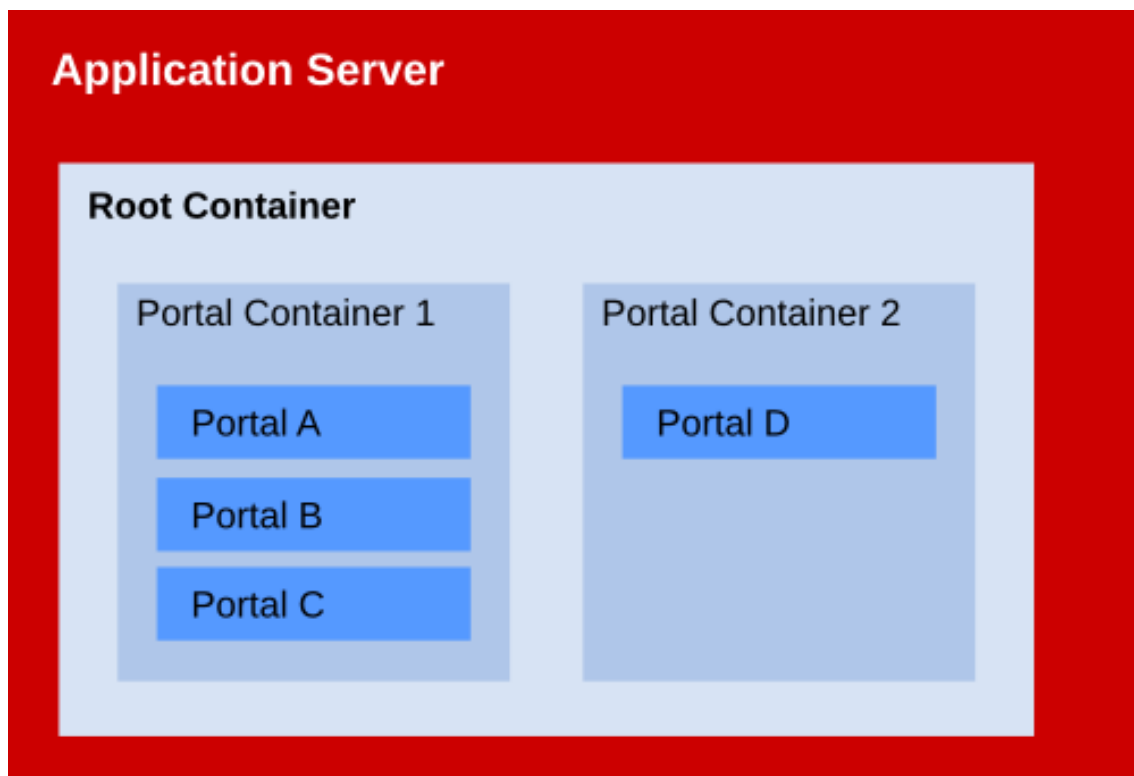
8.1. Foundations

8.1.1. GateIn Kernel

GateIn 3.1 is built as a set of services on top of a dependency injection kernel. The kernel provides configuration, lifecycle handling, component scopes, and some core services.

Service components exist in two scopes. First scope is represented by **RootContainer** - it contains services that exist independently of any portal, and can be accessed by all portals.

Second scope is portal-private in the form of **PortalContainer**. Each portal lives in an instance of PortalContainer. This scope contains services that are common for a set of portals, and services which should not be shared by all portals.



Whenever a specific service is looked up through PortalContainer, and the service is not available, the lookup is delegated further up to RootContainer. We can therefore have default instance of a certain component in RootContainer, and portal specific instances in some or all PortalContainers, that override the default instance.

Whenever your portal application has to be integrated more closely with GateIn services, the way to do it is by looking up these services through PortalContainer. Be careful though - only officially documented services should be accessed this way, and used according to documentation, as most of the services are an implementation detail of GateIn, and subject to change without notice.

8.1.2. Configuring services

GateIn Kernel uses dependency injection to create services based on **configuration.xml** configuration files. The location of the configuration files determines if services are placed into RootContainer scope, or into PortalContainer scope. All configuration.xml files located at **conf/configuration.xml** in the classpath (any directory, or any jar in the classpath) will have their services configured at RootContainer scope. All configuration.xml files located at **conf/portal/configuration.xml** in the classpath will have their services configured at PortalContainer scope. Additionally, **portal extensions** can contain configuration in **WEB-INF/conf/configuration.xml**, and will also have their services configured at PortalContainer scope.



Note

Portal extensions are described later on.

8.1.3. Configuration syntax

8.1.3.1. Components

A service component is defined in **configuration.xml** by using **<component>** element.

There is only one required information when defining a service - the service implementation class, specified using **<type>**

Every component has a **<key>** that identifies it. If not explicitly set, a key defaults to the value of **<type>**. If key can be loaded as a class, a Class object is used as a key, otherwise a String is used.

The usual approach is to specify an interface as a key.

Example 8.1. Example of service component configuration:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd
    http://www.exoplaform.org/xml/ns/kernel_1_0.xsd"
  xmlns="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd">
  <component>
    <key>org.exoplaform.services.database.HibernateService</key>
    <type>org.exoplaform.services.database.impl.HibernateServiceImpl</type>

    ...
```

```

</component>
</configuration>

```

8.1.3.2. External Plugins

GateIn Kernel supports non-component objects that can be configured, instantiated, and injected into registered components, using method calls. The mechanism is called 'plugins', and allows portal extensions to add additional configurations to core services.

External plugin is defined by using **<external-component-plugins>** wrapper element which contains one or more **<component-plugin>** definitions. **<external-component-plugins>** uses **<target-component>** to specify a target service component that will receive injected objects.

Every **<component-plugin>** defines an implementation type, and a method on target component to use for injection (**<set-method>**).

A plugin implementation class has to implement **org.exoplatform.container.component.ComponentPlugin** interface.

In the following example **PortalContainerDefinitionPlugin** implements ComponentPlugin:

Example 8.2. PortalContainerDefinitionPlugin

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_0.xsd
    http://www.exoplatform.org/xml/ns/kernel_1_0.xsd"
  xmlns="http://www.exoplatform.org/xml/ns/kernel_1_0.xsd">

  <external-component-plugins>
    <target-component>org.exoplatform.container.definition.PortalContainerConfig</target-
component>
    <component-plugin>
      <!-- The name of the plugin -->
      <name>Add PortalContainer Definitions</name>

      <!-- The name of the method to call on the PortalContainerConfig
in order to register the PortalContainerDefinitions -->
      <set-method>registerPlugin</set-method>

      <!-- The fully qualified name of the PortalContainerDefinitionPlugin -->
      <type>org.exoplatform.container.definition.PortalContainerDefinitionPlugin</type>
    </component-plugin>
  </external-component-plugins>
</configuration>

```

```
...  
  
    </component-plugin>  
  </external-component-plugins>  
</configuration>
```

8.1.3.3. Includes, and special URLs

It is possible to break **configuration.xml** file into many smaller files, that are then included into a 'master' configuration file. The included files are complete configuration xml documents by themselves - they are not fragments of text.

An example configuration.xml that 'outsources' its content into several files:

```
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd  
    http://www.exoplaform.org/xml/ns/kernel_1_0.xsd"  
  xmlns="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd">  
  
  <import>war:/conf/sample-ext/jcr/jcr-configuration.xml</import>  
  <import>war:/conf/sample-ext/portal/portal-configuration.xml</import>  
  
</configuration>
```

We see a special URL being used to reference another configuration file. URL schema '**war:**' means, that the path that follows is resolved relative to current PortalContainer's servlet context resource path, starting at **WEB-INF** as a root.



Note

Current PortalContainer is really a newly created PortalContainer, as war: URLs only make sense for PortalContainer scoped configuration.

Also, thanks to extension mechanism, the servlet context used for resource loading is a **unified servlet context** (as explained in a later section).

To have include path resolved relative to current classpath (context classloader), use '**jar:**' URL schema.

8.1.3.4. Special variables

Configuration files may contain a **special variable** reference `${container.name.suffix}`. This variable resolves to the name of the current portal container, prefixed by underscore (`_`). This facilitates reuse of configuration files in situations where portal specific unique names need to be assigned to some resources (i.e. JNDI names, Database / DataSource names, JCR repository names, etc ...).

This variable is only defined when there is a current PortalContainer available - only for PortalContainer scoped services.

A good example for this is **HibernateService**:

Example 8.3. HibernateService using variables

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd
    http://www.exoplaform.org/xml/ns/kernel_1_0.xsd"
  xmlns="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd">

  <component>
    <key>org.exoplaform.services.database.HibernateService</key>
    <jmx-name>database:type=HibernateService</jmx-name>
    <type>org.exoplaform.services.database.impl.HibernateServiceImpl</type>
    <init-params>
      <properties-param>
        <name>hibernate.properties</name>
        <description>Default Hibernate Service</description>
        <property name="hibernate.show_sql" value="false" />
        <property name="hibernate.cglib.use_reflection_optimizer" value="true" />
        <property name="hibernate.connection.url"
          value="jdbc:hsqldb:file:../temp/data/exodb${container.name.suffix}" />
        <property name="hibernate.connection.driver_class" value="org.hsqldb.jdbcDriver" />
        <property name="hibernate.connection.autocommit" value="true" />
        <property name="hibernate.connection.username" value="sa" />
        <property name="hibernate.connection.password" value="" />
        <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect" />
        <property name="hibernate.c3p0.min_size" value="5" />
        <property name="hibernate.c3p0.max_size" value="20" />
        <property name="hibernate.c3p0.timeout" value="1800" />
        <property name="hibernate.c3p0.max_statements" value="50" />
      </properties-param>
```

```
</init-params>
</component>
</configuration>
```

8.1.4. InitParams configuration object

`InitParams` is a configuration object that is essentially a map of key-value pairs, where key is always a `String`, and value can be any type that can be described using kernel configuration xml.

Service components that form the GateIn 3.1 infrastructure use `InitParams` object to configure themselves. A component can have one instance of `InitParams` injected at most. If the service component's constructor takes `InitParams` as any of the parameters it will automatically be injected at component instantiation time. The xml configuration for a service component that expects `InitParams` object must include `<init-params>` element (even if an empty one).

Let's use an example to see how the kernel xml configuration syntax looks for creating `InitParams` instances.

Example 8.4. InitParams - properties-param

```
<component>
  <key>org.exoplatform.services.naming.InitialContextInitializer</key>
  <type>org.exoplatform.services.naming.InitialContextInitializer</type>
  <init-params>
    <properties-param>
      <name>default-properties</name>
      <description>Default initial context properties</description>
      <property name="java.naming.factory.initial"
        value="org.exoplatform.services.naming.SimpleContextFactory" />
    </properties-param>
  </init-params>
</component>
```

`InitParams` object description begins with `<init-params>` element. It can have zero or more children elements each of which is one of `<value-param>`, `<values-param>`, `<properties-param>`, or `<object-param>`. Each of these child elements takes a `<name>` that serves as a map entry key, and an optional `<description>`. It also takes a type-specific value specification.

For `<properties-param>` the value specification is in the form of one or more `<property>` elements, each of which specifies two strings - a property name, and a property value. Each `<properties-`

`params`> defines one `java.util.Properties` instance. Also see [Example 8.3, “HibernateService using variables”](#) for an example.

Example 8.5. InitParams - value-param

```
<component>
  <key>org.exoplatform.services.transaction.TransactionService</key>
  <type>org.exoplatform.services.transaction.impl.jotm.TransactionServiceJotmImpl</type>
  <init-params>
    <value-param>
      <name>timeout</name>
      <value>5</value>
    </value-param>
  </init-params>
</component>
```

For `<value-param>` the value specification is in the form of `<value>` element, which defines one `String` instance.

Example 8.6. InitParams - values-param

```
<component>
  <key>org.exoplatform.services.resources.ResourceBundleService</key>
  <type>org.exoplatform.services.resources.impl.SimpleResourceBundleService</type>
  <init-params>
    <values-param>
      <name>classpath.resources</name>
      <description>The resources that start with the following package name should be load from
file system</description>
      <value>locale.portlet</value>
    </values-param>

    <values-param>
      <name>init.resources</name>
      <description>Store the following resources into the db for the first launch </description>
      <value>locale.test.resources.test</value>
    </values-param>

    <values-param>
      <name>portal.resource.names</name>
```

```
<description>The properties files of the portal , those file will be merged
into one ResourceBundle properties </description>
<value>local.portal.portal</value>
<value>local.portal.custom</value>
</values-param>
</init-params>
</component>
```

For `<values-param>` the value specification is in the form of one or more `<value>` elements, each of which represents one `String` instance, where all the `String` values are then collected into a `java.util.List` instance.

Example 8.7. InitParams - object-param

```
<component>
  <key>org.exoplatform.services.cache.CacheService</key>
  <jmx-name>cache:type=CacheService</jmx-name>
  <type>org.exoplatform.services.cache.impl.CacheServiceImpl</type>
  <init-params>
    <object-param>
      <name>cache.config.default</name>
      <description>The default cache configuration</description>
      <object type="org.exoplatform.services.cache.ExoCacheConfig">
        <field name="name">
          <string>default</string>
        </field>
        <field name="maxSize">
          <int>300</int>
        </field>
        <field name="liveTime">
          <long>300</long>
        </field>
        <field name="distributed">
          <boolean>false</boolean>
        </field>
        <field name="implementation">
          <string>org.exoplatform.services.cache.concurrent.ConcurrentFIFOExoCache</string>
        </field>
      </object>
    </object-param>
  </init-params>
</component>
```

```
</component>
```

For `<object-param>` in our case, the value specification comes in a form of `<object>` element, which is used for POJO style object specification (you specify an implementation class - `<type>`, and property values - `<field>`).

Also see [Example 8.8, “Portal container declaration example”](#) for an example of specifying a field of `Collection` type.

The `InitParams` structure - the names and types of entries is specific for each service, as it is the code inside service components's class that decides what entry names to look up and what types it expects to find.

8.1.5. Configuring a portal container

A **portal container** is defined by several attributes.

First, there is a **portal container name**, which is always equal to URL context to which the current portal is bound.

Second, there is a **REST context name**, which is used for REST access to portal application - every portal has exactly one (unique) REST context name.

Then, there is a **realm name** which is the name of security realm used for authentication when users log into the portal.

Finally, there is a list of **Dependencies** - other web applications, whose resources are visible to current portal (via extension mechanism described later), and are searched in the specified order.

Example 8.8. Portal container declaration example

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd
    http://www.exoplaform.org/xml/ns/kernel_1_0.xsd"
  xmlns="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd">

  <external-component-plugins>
    <!-- The full qualified name of the PortalContainerConfig -->
    <target-component>org.exoplaform.container.definition.PortalContainerConfig</target-
component>

    <component-plugin>
```

```
<!-- The name of the plugin -->
<name>Add PortalContainer Definitions</name>

<!-- The name of the method to call on the PortalContainerConfig
      in order to register the PortalContainerDefinitions -->
<set-method>registerPlugin</set-method>

<!-- The full qualified name of the PortalContainerDefinitionPlugin -->
<type>org.exoplatform.container.definition.PortalContainerDefinitionPlugin</type>

<init-params>
  <object-param>
    <name>portal</name>
    <object type="org.exoplatform.container.definition.PortalContainerDefinition">
      <!-- The name of the portal container -->
      <field name="name"><string>portal</string></field>

      <!-- The name of the context name of the rest web application -->
      <field name="restContextName"><string>rest</string></field>

      <!-- The name of the realm -->
      <field name="realmName"><string>exo-domain</string></field>

      <!-- All the dependencies of the portal container ordered by loading priority -->
      <field name="dependencies">
        <collection type="java.util.ArrayList">
          <value>
            <string>eXoResources</string>
          </value>
          <value>
            <string>portal</string>
          </value>
          <value>
            <string>dashboard</string>
          </value>
          <value>
            <string>exoadmin</string>
          </value>
          <value>
            <string>eXoGadgets</string>
          </value>
          <value>
            <string>eXoGadgetServer</string>
          </value>
        </collection>
      </field>
    </object>
  </object-param>
</init-params>
```

```

        <value>
            <string>rest</string>
        </value>
        <value>
            <string>web</string>
        </value>
        <value>
            <string>wsrp-producer</string>
        </value>
        <!-- The sample-ext has been added at the end of the dependency list
            in order to have the highest priority -->
        <value>
            <string>sample-ext</string>
        </value>
    </collection>
</field>
</object>
</object-param>
</init-params>
</component-plugin>
</external-component-plugins>
</configuration>

```



Note

Dependencies are part of the extension mechanism.

Every **portal container** is represented by a **PortalContainer instance**, which contains:

- associated **ExoContainerContext**, which contains information about the portal
- **unified servlet context**, for web-archive-relative resource loading
- **unified classloader**, for classpath based resource loading
- methods for retrieving services

Unified servlet context, and **unified classloader** are part of the **extension mechanism** (explained in next section), and provide standard API (ServletContext, ClassLoader) with specific resource loading behavior - visibility into associated web application archives, configured with Dependencies property of PortalContainerDefinition. Resources from other web applications are queried in the order specified by Dependencies. The later entries in the list override the previous ones.

8.1.6. GateIn Extension Mechanism, and Portal Extensions

Extension mechanism is a functionality that makes it possible to override portal resources in an almost plug-and-play fashion - just drop in a .war archive with the resources, and configure its position on the portal's classpath. This way any customizations of the portal don't have to involve unpacking and repacking the original portal .war archives. Instead, you create your own .war archive with changed resources, that override the resources in the original archive.

A web archive packaged in a way to be used through extension mechanism is called **portal extension**.

There are two steps necessary to create a portal extension.

First, declare **PortalConfigOwner** servlet context listener in web.xml of your web application.

Example 8.9. Example of a portal extension called sample-ext:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN
    http://java.sun.com/dtd/web-app_2_3.dtd>
<web-app>

    <display-name>sample-ext</display-name>

    <listener>
        <listener-class>org.exoplatform.container.web.PortalContainerConfigOwner</listener-class>
    </listener>

    ...

</web-app>
```

Then, add the servlet context name of this web application in proper place in the list of Dependencies of the PortalContainerDefinition of all the portal containers that you want to have access to its resources.

After this step your web archive will be on portal's unified classpath, and unified servlet context resource path. The later in the Dependencies list your application is, the higher priority it has when resources are loaded by portal.



Note

See 'Configuring a portal' section for example of `PortalContainerDefinition`, that has `sample-ext` at the end of its list of `Dependencies`.

8.1.7. Running Multiple Portals

It is possible to run several independent portal containers - each bound to a different URL context - within the same JVM instance. This kind of setup is very efficient from administration and resource consumption aspect. The most elegant way to **reuse** configuration for different coexisting portals is by way of extension mechanism - by **inheriting** resources and configuration from existing web archives, and just **adding** extra resources to it, and **overriding** those that need to be changed by including modified copies.

In order for a portal application to correctly function when deployed in multiple portals, the application may have to dynamically query the information about the current portal container. The application should not make any assumptions about the name, and other information of the current portal, as there are now multiple different portals in play.

At any point during request processing, or lifecycle event processing, your application can retrieve this information through **`org.exoplatform.container.ExoContainerContext`**. Sometimes your application needs to make sure that the proper **`PortalContainer`** - the source of **`ExoContainerContext`** - is associated with the current call.

If you ship servlets or servlet filters as part of your portal application, and if you need to access portal specific resources at any time during the processing of the servlet or filter request, then you need to make sure the servlet/filter is associated with the current container.

The proper way to do that is to make your servlet extend **`org.exoplatform.container.web.AbstractHttpServlet`** class. This will not only properly initialize current **`PortalContainer`** for you, but will also set the current thread's context classloader to one that looks for resources in associated web applications in the order specified by **`Dependencies`** configuration (as explained in Extension mechanism section).

Similarly for filters, make sure your filter class extends **`org.exoplatform.container.web.AbstractFilter`**. Both **`AbstractHttpServlet`**, and **`AbstractFilter`** have a method **`getContainer()`**, which returns the current **`PortalContainer`**. If your servlet handles the requests by implementing a **`service()`** method, you need to rename that method to match the following signature:

```
/**
 * Use this method instead of Servlet.service()
 */
protected void onService(ExoContainer container, HttpServletRequest req,
    HttpServletResponse res) throws ServletException, IOException;
```



Note

The reason is that `AbstractHttpServlet` implements `service()` to perform its interception, and you don't want to overwrite (by overriding) this functionality.

You may also need to access portal information within your **`HttpSessionListener`**. Again, make sure to extend the provided abstract class - **`org.exoplatform.container.web.AbstractHttpSessionListener`**. Also, modify your method signatures as follows:

```
/**  
 * Use this method instead of HttpSessionListener.sessionCreated()  
 */  
protected void onSessionCreated(ExoContainer container, HttpSessionEvent event);  
  
/**  
 * Use this method instead of HttpSessionListener.sessionDestroyed()  
 */  
protected void onSessionDestroyed(ExoContainer container, HttpSessionEvent event);
```

There is another method you have to implement in this case:

```
/**  
 * Method should return true if unified servlet context,  
 * and unified classloader should be made available  
 */  
protected boolean requirePortalEnvironment();
```

If this method returns true, current thread's context classloader is set up according to **Dependencies** configuration, and availability of the associated web applications. If it returns false, the standard application separation rules are used for resource loading (effectively turning off the extension mechanism). This method exists on **`AbstractHttpServlet`** and **`AbstractFilter`** as well, where there is a default implementation that automatically returns true, when it detects there is a current `PortalContainer` present, otherwise it returns false.

We still have to explain how to properly perform **ServletContextListener** based initialization, when you need access to current PortalContainer.

GateIn has no direct control over the deployment of application archives (.war, .ear files) - it is the application server that performs the deployment. For **extension mechanism** to work properly, the applications, associated with the portal via **Dependencies** configuration, have to be deployed before the portal, that depends on them, is initialized. On the other hand, these applications may require an already initialized PortalContainer to properly initialize themselves - we have a recursive dependency problem. To resolve this problem, a mechanism of **initialization tasks**, and **task queues**, was put in place. Web applications that depend on current PortalContainer for their initialization have to avoid performing their initialization directly in some ServletContextListener executed during their deployment (before any PortalContainer was initialized). Instead, a web application should package its initialization logic into an init task of appropriate type, and only use ServletContextListener to insert the init task instance into the proper init tasks queue.

An example of this is Gadgets application which registers Google gadgets with the current PortalContainer:

```
public class GadgetRegister implements ServletContextListener
{
    public void contextInitialized(ServletContextEvent event)
    {
        // Create a new post-init task
        final PortalContainerPostInitTask task = new PortalContainerPostInitTask() {

            public void execute(ServletContext context, PortalContainer portalContainer)
            {
                try
                {
                    SourceStorage sourceStorage =
                        (SourceStorage) portalContainer.getComponentInstanceOfType(SourceStorage.class);
                    ...
                }
                catch (RuntimeException e)
                {
                    throw e;
                }
                catch (Exception e)
                {
                    throw new RuntimeException("Initialization failed: ", e);
                }
            }
        };
    }
};
```

```
// Add post-init task for execution on all the portal containers
// that depend on the given ServletContext according to
// PortalContainerDefinitions (via Dependencies configuration)
PortalContainer.addInitTask(event.getServletContext(), task);
}
}
```

The above example uses **PortalContainerPostInitTask**, which gets executed after the portal container has been initialized. In some situations you may want to execute initialization after portal container was instantiated, but before it was initialized - use **PortalContainerPreInitTask** in that case. Or, you may want to execute initialization after all the post-init tasks have been executed - use **PortalContainerPostCreateTask** in that case.

One more area that may need your attention are **LoginModules**. If you use custom LoginModules, that require current ExoContainer, make sure they extend **org.exoplatform.services.security.jaas.AbstractLoginModule** for proper initialization. AbstractLoginModule also takes care of the basic configuration - it recognizes two initialization options - **portalContainerName**, and **realmName** whose values you can access via protected fields of the same name.