

Portlet Tutorial

Note: This tutorial applies to the **Jetspeed 1.4b3 release**. The tutorial covers how to:

[Portlet Tutorial](#)

[Getting Started: Installation and Building](#)

[Downloading Jetspeed](#)

[Developing Without Source](#)

[Development Cycle: Deployment to the Jetspeed Portlet Container](#)

[Building the Examples](#)

[Planning for Your Portal Site](#)

[The Jetspeed Portlet API](#)

[Restarting the Tutorials](#)

[Tutorial 1 – Tailoring the Portal Web Design](#)

[1.1 Change the Portal Logo](#)

[1.2 Modify the Top Navigation](#)

[1.3 Remove the Left Navigation](#)

[1.4 Replace the Bottom Navigation](#)

[1.5 Change the Web Page Title](#)

[1.6 Change the Skin and Cascading Style Sheet](#)

[1.7 Deploy](#)

[Tutorial 2 - Localisation](#)

[2.1 Java Resource Bundles](#)

[2.2 Modules and Bundles](#)

[2.3 Localisation of PSML Resources](#)

[2.4 Deploy](#)

[Tutorial 3 – Site Map](#)

[3.1 The Anonymous User's Resource](#)

[3.2 Navigational PSML Components](#)

[3.3 PSML References](#)

[3.4 An Authenticated User's Resource](#)

[3.5 Layout PSML Components for Portlets](#)

[3.6 Customizing Menus with the Customizer](#)

[3.7 Customizing Panes with the Customizer](#)

[3.8 Deploy](#)

[Tutorial 4 – Site Security](#)

[4.1 Jetspeed Security Concepts](#)

[4.2 Managing Users and Roles](#)

[4.3 Security Features](#)

[4.4 Using the Customizer to Secure Portal Resources](#)

[4.5 The Security Example Portlet](#)

[4.6 Deploy](#)

[Tutorial 5 – Portlet 101](#)

[5.1 Hello World Portlet](#)

[5.2 Adding A Portlet to the Registry](#)

[5.3 Adding A Portlet to a Page](#)

[5.4 Hello User Portlet](#)

[5.5 Deploy](#)

[Tutorial 6 - The Portlet Interface](#)

[6.1 Media Types](#)

[6.2 Portlet Life Cycle](#)

[6.3 Init Parameters, Attributes and Titles](#)

[6.4 Portlet Modes](#)

[6.5 Deploy](#)

[Tutorial 7 – Velocity Portlet](#)

[7.1 Introduction to Velocity](#)

[7.2 Velocity Portlets in the Registry](#)

[7.3 The Velocity Template](#)

[7.4 Template Resolution](#)

[7.5 Velocity Actions and the Context](#)

[7.6 Portlet Customization and Parameter Styles](#)

[7.7 Action Events](#)

[7.8 Deploy](#)

[Tutorial 8 – JSP Portlet](#)

[8.1 Introduction to JSP](#)

[8.2 JSP Portlets in the Registry](#)

[8.3 The JSP Template](#)

[8.4 Template Resolution](#)

[8.5 JSP Actions](#)

[8.6 Portlet Customization and Parameter Styles](#)

[8.7 JSP Events](#)

[8.8 Deploy](#)

[Tutorial 9 – DatabaseBrowser Portlet](#)

[9.1 Configuration in the Registry](#)

[9.2 Linking to Actions](#)

[9.3 Implementing the Action Events](#)

[9.4 Advanced Parameters](#)

[9.5 Deploy](#)

[Tutorial 10 – XSLT Portlet](#)

[9.1 Configuration in the Registry](#)

[9.2 The Transform](#)

[10.3 Deploy](#)

[Tutorial 11 – RSS Portlet](#)

[Tutorial 12 – Parameter Styles](#)

[12.1 Parameter styles architecture](#)

[Future Possible Tutorials](#)

[Appendix A - How Property Merging Works in the Tutorial](#)

[Removing Properties](#)

[Appendix B - Turbine and ECS](#)

Getting Started: Installation and Building

Before we get started, we have a few pre-requisite installations:

1. Install a [java compiler](#) (1.3 or greater)
2. Install [Ant](#) (1.4 or greater)
3. Install [Catalina](#) (4.1.12 or higher)
4. Install [Jetspeed](#) (1.4b3 release or higher)
5. [Build Jetspeed](#) WAR file (ant war) **
6. Download the [Tutorial Examples](#)

** optional – if you want to work in the default build mode, with source code. You can also work without source by downloading the WAR file only.

Downloading Jetspeed

You can get Jetspeed from the Jakarta CVS, or from the Apache website.

To checkout from the CVS head, follow the instructions here:

<http://jakarta.apache.org/jetspeed/site/install.html>

and then build the war target (ant war).

To download from the Apache website, go here:

<http://jakarta.apache.org/builds/jakarta-jetspeed/release/v1.4b3/>

If you want to work with the source code, download [jetspeed-1.4b3-release-src.zip](#)

Unzip it and build the war target (ant war).

Developing Without Source

You can also download the release WAR files, but this will limit your development environment, since you won't have the source code to Jetspeed.

[jetspeed-1.4b3-release-war.zip](#)

[jetspeed-1.4b3-release-fullwar.zip](#)

If you choose to develop without source, you will need to modify the settings in your build.properties as described [here](#).

All of the examples in this document are available for download at

<http://www.bluesunrise.com/jetspeed-docs/jportal.jar>

Download this jar file, and expand it to a directory on your file system with the command:

```
jar -xf jportal.jar
```

Development Cycle: Deployment to the Jetspeed Portlet Container

During development of the examples, we will often deploy our portlets into a Jetspeed web application – a portlet container. This is accomplished with the several ant targets provided with the tutorial:

ant tutorial-n

Configure the distribution to match the given tutorial, where *n* is the number of the tutorial from this document.

ant war

Creates the war file which you can manually copy to your application server. This war file is a combination of the Jetspeed distribution with you're the example code. The war file combines the Jetspeed container with the example portlets and their configuration. The war file is written to `jportal/dist/jportal.war`.

ant deploy

Dependent on the **war** target. After building the war file, it automatically deploys the war file into the application server of your choice. For this tutorial, Jakarta Catalina is required. NOTE: you should shutdown Catalina before invoking this target.

ant hotdeploy

A quicker version of deploy. Only deploys classes and configuration files that are newer than the files in the deployed web application. Requires that deploy target is ran once, since it will not expand the entire application to the web application directory. This target is used most often in the everyday development routine.

Jetspeed 2.0 will support the [Java Standard Portlet API](#), which defines a standard deployment descriptor and portlet archive for deploying portlets to portlet containers. Version 1.x of Jetspeed does not support the standard, thus a custom deployment procedure is necessary.

Building the Examples

The tutorial requires **ant** (<http://jakarta.apache.org/ant/>)

You will need **ant** to build all the examples. Please download and install **ant** before continuing.

Here are the available target commands:

| | |
|---------|---|
| all | Clean build and dist, then compile |
| clean | Delete old build and dist directories |
| compile | Compile Java sources |
| deploy | Deploy application to servlet container |
| dist | Create binary distribution |

```
hotdeploy    Hot Deploy application to portlet container
javadoc      Create Javadoc API documentation
om           Generate Object Model sources
war          Create merged war
tutorial-0   Resets the web application.
tutorial-n   Configure the distribution for the nth tutorial examples.
tutorial-all Configurations the distribution to include all tutorials.
```

Depending on where Jetspeed and Tomcat are installed, you may need to edit a few of the properties in the **build.properties**. The first set of properties are for developing with the source code. The second set are for developing without the source code, using a WAR file that was downloaded, in this example, to a directory **/apache/Jetspeed-1.4b3-release-war/Jetspeed-1.4b3**.

```
#
# typical settings with source
#
jetspeed_home=/apache/jakarta-jetspeed
catalina_home=/apache/catalina
portlet_app_name=jportal
company=com.bluesunrise.jportal
jetspeed_jar=/bin/jetspeed-1.4b3-dev.jar
jetspeed_war=/bin/jetspeed.war
jetspeed_lib=/lib
jetspeed_conf=/webapp/WEB-INF/conf/
#
# typical settings without source
#
# jetspeed_home=/apache/jetspeed-1.4b3-release-war/jetspeed-1.4b3
# catalina_home=/apache/catalina
# portlet_app_name=jportal
# company=com.bluesunrise.jportal
# jetspeed_jar=/WEB-INF/lib/jetspeed-1.4b3-dev.jar
# jetspeed_war=/jetspeed.war
# jetspeed_lib=/WEB-INF/lib
```



```
# jetspeed_conf=/WEB-INF/conf/
```

To build the examples for a specific tutorial example, type:

```
ant tutorial-n      (where n = the number of the tutorial)
ant deploy         (first time, and then)
ant hotdeploy
```

These commands build and deploy the examples to your Catalina distribution as a web application. NOTE: this process will also work with Tomcat.

The tutorials must be run in order. If you want to deploy tutorial-5, you must first run tutorial-1 thru tutorial-4.

If you'd like to skip ahead and see the result of all the tutorials, then run:

```
ant tutorial-all
ant deploy
```

That's it. You are now ready to start up the JPortal web application. Start up Catalina and point a browser at:

<http://localhost:8080/jportal/portal>

You will see the Home Page for JPortal. To see JPortal customized for a user, logon to the system as:

Username = turbine
Password = turbine

NOTE: you should shutdown Catalina before invoking the deploy target.

Planning for Your Portal Site

No matter what portal or web development tool you're using, you should always first spend some time on the specification of your portal. You should not even start looking at the technology before answering questions like:

- How many users?
- Is there already an existing authentication database for these users, do I want to use it?
- How many languages?
- How many media types do I want to support?
- How many and which segments/target groups do I have to serve
- What content do I want to publish?
- How can I get access to this content (disk, db, syndication, etc...)?
- Who is going to update the site content and with what editorial process?
- What applications do I need to connect to?
- Are they already using a single authentication db?
- How do I bridge to these apps (servlet, http, SOAP, CORBA, etc...)?
- For each user target group, what application/content do they need to access?
- Can they personalize their portal view?
- What access rights should they have?
- What will be your portal navigation?
- Do you plan to set up different thematic portal views or a single integrated workspace?

Once you have written down your answers on these points, you should have a much better view on:

- What kind of software tools and features you'll need.
- What kind of skills you'll require.

- An estimate of the amount of work required to setup your portal.
- What budget you'll need to complete your portal transition.
- Where to start

Setting up and running a portal is typically a lot of work, mainly in terms of technical integration, process engineering and project management. Open source software does not magically cancel these, just ensure that your budget can be spent on these items rather than being split between software licenses and integration.

Jumping directly from a static website to a portal-driven system without any defined specification and expected benefits is a recipe for frustration, lost time, lost money and a very slow website...

The Jetspeed Portlet API

The Jetspeed Portal API is a set of interfaces describing how a portlet interacts with a portlet container. There is a soon to be released Java Standard Portlet API. Jetspeed 2.0 will support that standard.

Every portlet has to implement the Portlet interface

org.apache.jetspeed.portal.Portlet

either directly, or by extending a class that in turn implements the Portlet interface. Review the Portlet interface at the link above. This tutorial will cover how to implement most methods on the interface.

It is usually recommended to extend one of the higher level foundation portlet classes, such as the [CustomizerVelocity](#) portlet or [DatabaseBrowserPortlet](#) portlet.

The [AbstractPortlet](#) and [AbstractInstancePortlet](#) classes provide an abstract portlet implementation of the Portlet interface with the most common functionality, leaving only a few methods to be implemented by you.

Restarting the Tutorials

If you'd like to clean up the JPortal Tutorial examples and reset to a default Jetspeed deployment, type:

```
ant clean
ant deploy
```

This ant target removes all classes and configuration files from the deployment, leaving you with a web application that is exactly the same as the Jetspeed web application. No changes are made to the Jetspeed distribution. When you point your browser at <http://localhost:8080/jportal/portal>, you will see the default Jetspeed portal.

This target is useful if you want to restart the tutorials from the beginning.

Tutorial 1 – Tailoring the Portal Web Design

Tutorial 1 shows you how to tailor the web design of the JPortal portal. This is accomplished by overriding and merging the Jetspeed portal configuration property files.

In this tutorial, we will:

1. Change the Portal Logo
2. Modify the Top Navigation
3. Remove the Left Navigation
4. Replace the Bottom Navigation
5. Change the Web Page Title
6. Change the Skin, CSS
7. How Property Merging Works in the Tutorial
8. Deploy

Let's get started. From the JPortal distribution root directory, type:

```
ant tutorial-1
```

Recommend bringing up these configuration files in your editor:

1. `webapp/WEB-INF/conf/JPortalJetspeed.properties`
2. `webapp/WEB-INF/conf/jportal-skins.xreg`
3. `webapp/WEB-INF/conf/templates/vm/navigations/html/bottom-jportal.vm`
4. `webapp/css/jportal.css`

since we will reference them in tutorial 1.

1.1 Change the Portal Logo

Looking at `JPortalJetspeed.properties`:

```
topnav.logo.file=images/jportal.gif
```

The `topnav.logo.file` property points to the new image that we want to display on the JPortal site. Note that all properties relating to files are specified relative to the web application root. The images directory can be found directly under the web application root.

You could also use a logo that exists on another server with the `topnav.logo.url` property. The `topnav.logo.file` and `topnav.logo.url` properties are mutually exclusive, you should one or the other.

Of course you don't have to use any logo. You're top navigation could be customized to use a completely different layout.

1.2 Modify the Top Navigation

Navigations are carried over from the [Turbine-2](#) classic web application design. A website generally has a top and bottom navigation scheme. This is generally defined as the header and footer of the website. Jetspeed allows three navigations: top, bottom, and left. For JPortal, we override navigation properties in `JPortalJetspeed.properties`. Here are the top navigation settings:

```
# topnav.enable=true
# topnav.vm=top.vm
# topnav.logo.url=
# topnav.logo.file=images/jportal.gif
```

```
# topnav.user_login.enable=true  
  
topnav.user_creation.enable=false
```

topnav.enable turns on or off the top navigation. Try turning it off. You won't see the logo or login edit fields.

We've already described **topnav.logo.file** and **topnav.logo.url** in the previous section.

topnav.user_login.enable turns on or off the single-sign-on edit fields for entering a username and password.

topnav.user_creation.enable turns on or off the link to the Enter New User portlet. We will modify the default setting and set it to false.

topnav.vm is the most powerful setting. You can use your own JSP or Velocity template here to completely customize your top navigation.

1.3 Remove the Left Navigation

Next we show how to remove the left navigation.

```
leftnav.enable=false  
  
# leftnav.vm=left.vm  
# leftnav.width=10%
```

leftnav.enable turns on or off the left navigation. The JPortal tutorial site doesn't require a left navigation, so we turn it off.

leftnav.vm You can use your own JSP or Velocity template here to customize your left navigation.

leftnav.width specifies the percentage of the page width to be covered by the left navigation.

1.4 Replace the Bottom Navigation

Next we show how to replace the bottom navigation.

```
bottomnav.enable=true
bottomnav.vm=bottom-jportal.vm
```

bottomnav.enable turns on or off the bottom navigation.

bottomnav.vm You can use your own JSP or Velocity template here to completely customize your bottom navigation. That is exactly what we do here using a Velocity template tailored specifically for the JPortal tutorial.

```
<br/>
<hr/>
<table width="100%">
  <tr>
    <td align="left">
      <small>
        JPortal Tutorial for Jetspeed - Version 1.4b3<br/>
      </small>
    </td>
    #if ($config.getString("mail.support"))
    <td align="center">
      <a href="mailto:$config.getString("mail.support")">Support and Additional Information</a>
    </td>
```



```
#end
<td align="right">
  <a href="http://www.bluesunrise.com">
    
  </a>
</td>
</tr>
</table>
```

1.5 Change the Web Page Title

The web page title (in the browser title-bar) can have a text string prefix added:

```
portalpage.title_prefix=JPportal Tutorial:
```

1.6 Change the Skin and Cascading Style Sheet

Skins are used to define color, font, and borders for various components in Jetspeed. The skin definitions is referenced in portal resources such as controls, controllers, portlets and templates to create a centralized design theme. Review the registry fragment called `jportal-skins.xreg`, where we define a new skin to match our really gaudy colour theme for the Jportal site.

```
<?xml version="1.0" encoding="UTF-8"?>
<registry>
  <skin-entry name="jportal-skin" hidden="false">
    <property name="text-color" value="#ffffff" hidden="false"/>
    <property name="background-color" value="#ffffff" hidden="false"/>
  </skin-entry>
</registry>
```

```
<property name="title-text-color" value="#000000" hidden="false"/>
<property name="title-background-color" value="#ceff63"
  hidden="false"/>
<property name="title-style-class" value="TitleStyleClass"
  hidden="false"/>
<property name="highlight-text-color" value="#ffffff"
  hidden="false"/>
<property name="highlight-background-color" value="#6331ff"
  hidden="false"/>
</skin-entry>
</registry>
```

Then in the `JPortalJetspeed.properties`, the default skin for the portal site is defined:

```
services.PortalToolkit.default.skin=jportal-skin
```

The default skin is applied to the portal whenever a specific skin is not declared for a resource. The default skin facilitates quickly creating a site look and feel with minimal effort.

See the [Jetspeed Skin documentation](#) for all available skin parameters.

Skins are used in combination with [Cascading Style Sheets](#). CSS is a simple mechanism for adding style (e.g. fonts, colors, spacing) to Web documents, defined as a standard by the [World Wide Web Consortium \(W3C\)](#).

In the `JPortalJetspeed.properties`, the portal site stylesheet is defined:

```
site.stylesheet=css/jportal.css
```

Looking at the cascading style sheet for the JPortal site, we see that the Jetspeed style sheet, `default.css`, is included into this style sheet with the import at-rule:

```
@import "default.css";

A:link    {
    text-decoration : none;
    color : #333366;
}

A:visited {
    text-decoration : none;
    color : #666699;
}

A:hover   {
    color : #CC3300;
}

A.index:link {
    font-size : 11px;
    color : #333366;
}

A.index:visited {
    font-size : 11px;
    color : #333366;
}

A.index:hover {
    color : #CC3300;
}
```

We then go on to define several example styles overriding the settings in default.css. Styles declared in the style sheet can also be referenced in a skin, as we do in our skin registry:

```
<property name="title-style-class" value="TitleStyleClass"  
          hidden="false" />
```

1.7 Deploy

To deploy the system type:

```
ant deploy          (or hotdeploy)
```

Use hotdeploy if you have already deployed the system once. This simply saves some time in packaging the JPortal deployment. Next point your browser at:

<http://localhost:8080/jportal/portal>

You should see the new web design for the JPortal site. The portlets inside are still the same as the default Jetspeed deployment.

Development

JPortal

Welcome to Jetspeed

Username:

Password:

[Login Help](#)

Home Page [RSS](#) [Dynamic](#)

Jetspeed Content Example

Jetspeed
POWERED
It works

If you see this item, you've successfully installed and configured your Jetspeed portal. Congratulations

XML at Jetspeed

XML.com has published an article about the Jetspeed project. Positive press is a good thing :)

Jetspeed Documentation

An Open Source Enterprise Information Portal.

- **Jetspeed Documentation**
Jetspeed is an Open Source implementation of an Enterprise Information Portal. Jetspeed attempts to consume information from multiple resources on the Internet and helps the user manage large amounts of data. This information can come from multiple content sources: local files, local applications or remote HTTP sources.

Welcome

2 default accounts are available in this Jetspeed installation:

Login: turbine
Password: turbine

Login: admin
Password: jetspeed

Make sure to try them out.

JPortal Tutorial for Jetspeed - Version 1.4b2

[Support and Additional Information](#)

Bluesunrise

Tutorial 2 - Localisation

When completing the last tutorial, you may have noticed the text right at the top of the screen “Welcome to Jetspeed”. The text comes from a Java localised resource file. This tutorial will show you how to create your own localised resources to easily support multiple languages without hard-coding messages in your Java source code.

In this tutorial, we will:

1. Edit the resource files and change the Welcome string in two languages
2. Review how Jetspeed finds PSML resources by language and country code
3. Localisation of PSML Resources
4. Deploy

Let's get started. From the JPortal distribution root directory, type:

```
ant tutorial-2
```

Recommend bringing up these files in your editor:

1. `webapp/WEB-INF/conf/JPortalTurbine.properties`
2. `src/java/com/bluesunrise/jportal/modules/localization/JportalLocalization_en.properties`
3. `src/java/com/bluesunrise/jportal/modules/localization/JportalLocalization_fr.properties`
4. `webapp/WEB-INF/templates/vm/navigations/html/bottom-jportal.vm`
5. `webapp/WEB-INF/conf/JPortalJetspeed.properties`

since we will reference them in tutorial 2.

2.1 Java Resource Bundles

It is recommended that you use resource bundles for all of strings that will be displayed to the end user. This is called 'localising' your application.

The tutorial comes with two sample resource files in English and French. Let's look at the English version:

```
TOP_TITLE>Welcome to JPortal, the Jetspeed Tutorial  
  
CONTACT_US=Contact Us
```

The first string, `TOP_TITLE`, replaces a resource string already defined in Jetspeed. The second string, `CONTACT_US` is a new string that is referenced in the `bottom-jportal.vm` file.

Referencing localised strings in Velocity templates is accomplished with the localisation global tool, with a rather odd name of `$l10n`. It stands for the word 'localisation', where the 10 middle letters are cleverly (sic) represented by the number 10.

Any string in the resource bundle can be referenced as shown below:

```
<td align="center">
  <h2>$l10n.TOP_TITLE</h2>
</td>
```

In this tutorial, we will change the bottom navigation defined in the last tutorial, to use a localised string:

```
<td align="center">
  <a href="mailto:$config.getString("mail.support")">$l10n.CONTACT_US</a>
</td>
```

In the `JPortalJetspeed.properties`, setup your email support account:

```
mail.support=support@bluesunrise.com
```

2.2 Modules and Bundles

Turbine has a concept called [modules](#), which is a special class loader path. Multiple module class paths can be configured by simply adding a path to the root directory of your modules. Take a look at the `JPortalTurbine.properties`:

```
module.packages=com.bluesunrise.jpportal.modules
```

Modules is made up of different kinds of modules: actions, layouts, screens, navigations, pages, and localisations. Jetspeed uses the module path to find the resource bundles.

Resource bundles and the default language are also specified in the `JPortalTurbine.properties`. Notice that the JPortal resource bundle is listed first, overriding the Jetspeed resource bundle.

```
locale.default.bundles=  
    com.bluesunrise.jpportal.modules.localization.JPortalLocalization,  
    org.apache.jetspeed.modules.localization.JetspeedLocalization  
  
locale.default.language=en  
locale.default.country=US
```

2.3 Localisation of PSML Resources

PSML resources may also be optionally localised. PSML Resources are localized by placing them in sub-directories based on language and country code abbreviations. The language-code sub-directory contains one or more country-code subdirectories.

The language-code directory name is specified with an ISO-639 standard two-character language abbreviation. The

country-code subdirectory is optional, and is specified with an ISO-3166 standard two-character country code abbreviation.

An example:

```
user
  |-- david
    |-- html
      |-- fr                // french language
        |-- FR              // France country-code
        |-- BE              // Belgium country-code
```

NOTE: The country codes must be in upper-case

For a given locale of fr_FR, the search order for the default accounting resource would be:

```
groups/accounting/html/fr/FR/default.psml
groups/accounting/html/fr/default.psml
groups/accounting/html/default.psml
groups/accounting/default.psml
```

The Jetspeed profiler looks at the "Content Language" HTML header for locale specific settings. If there are multiple settings, all settings will be searched until the first resource is found. (This is currently not supported)

For a complete list of ISO-639 standard language abbreviations, see:

<http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt>

For a complete list of ISO-3166 standard country code abbreviations, see:

http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

In the `JPortalJetspeed.properties`, you can enable or disable the usage of language and country codes during PSML resource resolution with the following settings. In the tutorial we choose to turn off both language and country code fallback.

```
# Consider the language as part of the fallback?  
services.Profiler.fallback.language=false  
  
# Consider the country code as part of the fallback?  
services.Profiler.fallback.country=false
```

2.4 Deploy

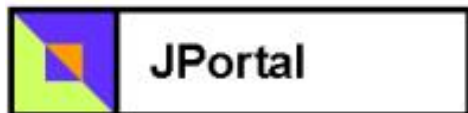
To deploy the system type:

```
ant deploy      (or hotdeploy)
```

Use hotdeploy if you have already deployed the system once. This simply saves some time in packaging the JPortal deployment. Next point your browser at:

<http://localhost:8080/jportal/portal>

You should see the new localised title. The portlets inside are still the same as the default Jetspeed deployment.



Tutorial 3 – Site Map

The next step is for us to layout the portal site map. We do this using a specialized portal XML-derivative called [PSML](#) (Portlet Structure Mark-up Language).

When designing your portal, you should consider how users will navigate around your site. In our example portal, we will allow for anonymous users to access the site, and also for authenticated users to access areas of the site where they are authorized (more on Site Security in the next tutorial).

In this tutorial, we will:

1. Create the anonymous PSML resource with a Tab control
2. Learn about PSML navigational components
3. PSML References
4. Create an authenticated user's PSML resource with a Menu control
5. Learn about PSML layout components for portlets
6. Customize Menus with the Customizer
7. Customizing Panes with the Customizer
8. Deploy

Let's get started. From the JPortal distribution root directory, type:

```
ant tutorial-3
```

Recommend bringing up these files in your editor:

1. `webapp/WEB-INF/psml/user/anon/html/default.psml`
2. `webapp/WEB-INF/psml/user/turbine/html/default.psml`

since we will reference them in tutorial 3.

3.1 The Anonymous User's Resource

Anonymous users are users who have not been authenticated. The anonymous user interface is the default user interface displayed to users who have not yet signed on.

Some sites do not allow anonymous users to see anything on the site. Perhaps you only want to display a single portlet, or perhaps a portlet with static company information, a Signup portlet, and a Returning-User logon portlet. It all depends on your requirements.

The requirements for JPortal are very simple: it's a tutorial. We want to show all the tutorial portlets to everyone. We also want to show off some of the portlets that come with Jetspeed out of the box. It would be confusing to put all the portlets on one page all at once, so let's group together the portlets based on categories such as 'Basic tutorials', 'Advanced tutorials', 'Jetspeed portlets', and 'Referenced portlets'. It would be nice to group these four categories into a manageable user interface components. Here are our four menu choices:

- Basic Tutorials
- Advanced Tutorials
- Jetspeed Portlets
- Referenced Portlets

3.2 Navigational PSML Components

Jetspeed provides layout components for defining the layout of a portal page. In this tutorial we will concentrate on components used to create navigational menus. Components are defined by editing the PSML file with a text editor, or using the Jetspeed Customizer, which makes portal layout much easier.

Before seeing how easy it is with the customizer, look at the PSML file for the anonymous user.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<portlets id="100" xmlns="http://xml.apache.org/jetspeed/2000/psml">

  <metainfo>
    <title>Welcome Page</title>
  </metainfo>
  <control name="TabControl"/>
  <controller name="CardPortletController"/>

  <portlets id="101">
    <metainfo>
      <title>Basic Tutorials</title>
    </metainfo>
  </portlets>

  <portlets id="102">
    <metainfo>
      <title>Advanced Tutorials</title>
    </metainfo>
  </portlets>
```

```
<portlets id="103">
  <metainfo>
    <title>Jetspeed Portlets</title>
  </metainfo>
</portlets>

<portlets id="104">
  <controller name="OneColumn"/>

  <metainfo>
    <title>Referenced Portlets</title>
  </metainfo>
  <reference path="group/apache/media-type/html/page/default"/>
  <reference path="group/Jetspeed/media-type/html/page/default"/>
</portlets>

</portlets>
```

There are several types of elements in the layout of a PSML file. Here are the common navigational components:

<portlets> defines a pane

<controller> defines the type of layout. Must be a card controller for menus

<control> defines the type of menu . Must be used with a card controller

For each menu option, we specify a pane. **Panes** hold collections of portlets. The portlets on a pane have their own layout. We will discuss that in the next section. A pane is specified like this:

```
<portlets id="101">
  <metainfo>
    <title>Basic Tutorials</title>
  </metainfo>
</portlets>
```

The `<title>` tag is used as the menu option text.

Panes are a special kind of a controller. There is one important difference between a pane (`PanedPortletController`) and other controllers: the pane does not render all of its managed portlets at the same time. That is why it needs to be used with a menu control, allowing us to navigate to the appropriate pane while hiding the other panes.

Controls are decorators around a pane or portlet. In the case of a pane, it is always used in combination with a portlet controller. There are two types of controls for panes:

1. `TabControl` – puts menu options across the top of the page
2. `MenuControl` – puts menu options on the left-side of the page

```
<control name="TabControl"/>
<controller name="CardPortletController"/>
```

Controls are sometimes used in combination with controllers. Controllers define the layout of a pane, such as the number of columns and rows. When defining menus, you must select the `CardPortletController`, which enables collections of portlets to occupy the same page, and is used in combination with the a `TabControl` or `MenuControl` to create menu components. These collections of portlets are called **panes**.

Each of the four menu options correspond to a pane. A pane is configured in PSML using the `<portlets>` tag. The `<metainfo>` tag defines the title of the pane which is displayed in the menu.

3.3 PSML References

For this is example, 3 of the panes are empty. The fourth pane contains PSML References.

[PSML References](#) are used to include an entire PSML resource into another PSML resource. This is useful for defining groups of markup once, and then sharing that markup in one or more PSML resources. The reference path is called a PSML resource locator. In our example, we reference two shared group resources as the content of the fourth pane.

```
<portlets id="104">
  <controller name="OneColumn"/>

  <metainfo>
    <title>Referenced Portlets</title>
  </metainfo>
  <reference path="group/apache/media-type/html/page/default"/>
  <reference path="group/Jetspeed/media-type/html/page/default"/>
</portlets>
```

These two PSML resources are included with the Jetspeed distribution. In the first case, the locator specifies that it is a group resource, for the media type HTML, and the name of the resource page is *default*. Shared PSML resources are useful for defining common layout definitions that can be used by all users of the system.

3.4 An Authenticated User's Resource

When a user logs on, the portal displays content customized specific to the user.

Logon as username **turbine**, with the password = **turbine**. You will see basically the same panes as defined for the anonymous user. However, this time we use a menu that is displayed on the left-side of the page.


```
<control name="MenuControl"/>  
<controller name="CardPortletController"/>
```

3.5 Layout PSML Components for Portlets

We saw how menus and panes define the navigational layout of a site. Here we will look at how to layout portlets on a particular pane. A pane is the container for portlets. Inside this container we have the choice of laying out portlets in a number of designs. This is accomplished using portlet **controllers**. In Tutorial 3, we only define portlets in the third pane "Jetspeed Portlets".

There are a fair number of controllers available for portlets:

1. One Column – all portlets flow down a single column on the pane
2. Single Row – all portlets flow across the pane in a single row
3. Three Column - The portlets flow down 3 columns, given column width percentages:
 - a. 25/50/25
 - b. 33/33/33
4. Two Column – The portlets flow down 2 columns, given column width percentages:
 - a. 50/50
 - b. 25/75
 - c. 75/25

Let's look at the pane definition for the Turbine default resource. Here we define four portlets to occupy this pane using a **TwoColumns** controller:

```
<portlets id="103">
  <metainfo>
    <title>Jetspeed Portlets</title>
  </metainfo>

  <controller name="TwoColumns"/>

  <entry parent="BBCFrontPage">
    <layout>
      <property name="column" value="0"/>
      <property name="row" value="0"/>
    </layout>
  </entry>

  <entry parent="WeatherPortlet">
    <layout>
      <property name="column" value="0"/>
      <property name="row" value="1"/>
    </layout>
    <parameter name="weather_city_info" value="US/AZ/Phoenix"/>
    <parameter name="weather_style" value="infobox"/>
  </entry>

  <entry parent="StockQuote">
    <layout>
      <property name="column" value="1"/>
      <property name="row" value="0"/>
    </layout>
  </entry>

  <entry parent="DatabaseBrowserTest">
    <layout>
      <property name="column" value="1"/>
      <property name="row" value="1"/>
    </layout>
    <parameter name="sql" value="select * from coffees"/>
    <parameter name="windowSize" value="10"/>
  </entry>
</portlets>
```

```
</portlet>
```

Notice that you can specify the location of the portlet using the `<layout>` element and the `column` and `row` properties. This is a task that is normally better handled by the customizer:

```
<layout>
  <property name="column" value="1" />
  <property name="row" value="0" />
</layout>
```

If its not already obvious, the `<entry>` element is used to reference a portlet. We will have a closer look at the entry tag in tutorial 5.

There are controls that can be applied to portlet entries as decorators. When applied to a portlet entry, controls define the window around the portlet. Here are some typical controls:

1. **Boxed Title Control** – draws a box around the portlet, adds a title bar and action icons.
2. **Clear Portlet Control** – draws no decorators
3. **Simple Titled Control** – adds a title bar and action icons, but no box

Finally there are default settings for controls, controllers and skins which are defined in the **`JPortalJetspeed.properties`**.

```
services.PortalToolkit.default.control=TitlePortletControl
services.PortalToolkit.default.controller=OneColumn
services.PortalToolkit.default.skin=orange-grey
```

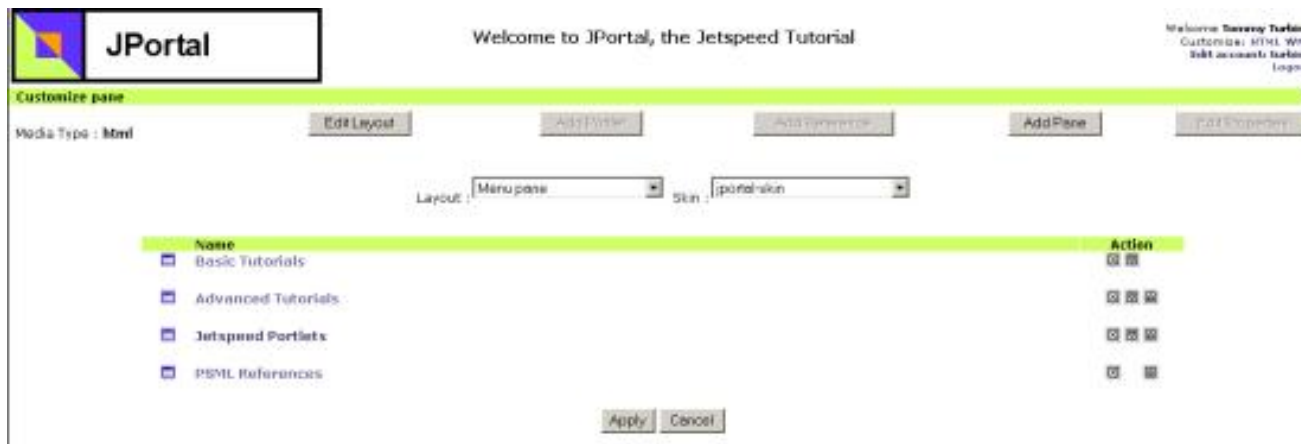
Portlet entries which do not have a control specified will use the default control. In the default Jetspeed deployment, this is the control described as #3 above, the Simple Title Control.

3.6 Customizing Menus with the Customizer

The customizer is a lot easier to use. Let's look at a screen shot of the customizer. Here we customizing the Turbine user page. You can see the four menu options listed.

When customizing menus, you can add panes to the menu.

Clicking the add pane button adds another menu option. The layout has MenuPane selected, which could be switched to TabPane (combination of TabControl + CardPortletController). With the Customizer, menu options can be moved up or down, added and deleted.



Once the menus and panes are defined, you can then move on to placing portlets on the panes by clicking on the specific menu option, which will then bring up a pane customizer for placing portlets. But first, let's have a look at site security.

3.7 Customizing Panes with the Customizer

When customizing panes, you manipulate the layout of portlets (or references) on the pane.

Here is a screen shot of the Jetspeed Portlets pane from the Turbine user page, containing four portlets:

The screenshot shows the JPortal interface with the following components:

- Basic Tutorials**: A sidebar menu with links for Basic Tutorials, Advanced Tutorials, Setup and Portlets, and PAM References.
- BBC Front Page News**: A news portlet titled "Updated every minute of every day" featuring a "BBC NEWS" logo and a list of news items:
 - Forces chief issues strike warning: A continuing firefighters strike would seriously undermine any possible military action against Iraq, according to the UK's most senior military chief.
 - Spanish coast faces second slick: Strong winds threaten to push an oil slick from the wreck of a sunken tanker towards Spain's north-western coast - it could hit land within two days.
 - Controversial autopsy goes ahead: The first public autopsy for 170 years goes ahead before a paying crowd on Wednesday - but under police scrutiny.
 - Diana paparazzi face privacy trial: Three photographers who took pictures of Princess Diana and Dodi Al Fayed at the time of their fatal crash face privacy charges.
 - Hindley cremated in private funeral: The body of Moors murderer Myra Hindley is cremated after a 20-minute service in front of 12 friends.
- Weather**: A weather portlet for Phoenix, AZ, showing 85 °F, Clear, and 2:53 PM. It includes a "Click for Forecast" link.
- Stack Portlets**: A table displaying stock market data:

| Symbol | Price | Change | Volume |
|--------|-------|--------|----------|
| SUNW | 3.62 | +0.04 | 55143384 |
| MSFT | 26.62 | +1.76 | 38147672 |
| ORCL | 10.74 | +0.37 | 48388948 |
| F | 8.93 | +0.38 | 8629800 |
| DVIN | 1.77 | +0.01 | 42556 |
| MNG | 0.08 | -0.01 | 169000 |
| IBM | 81.61 | +2.24 | 8949300 |
| HTEI | 4.19 | -0.01 | 13100 |
| ARBA | 4.21 | +0.75 | 6803898 |
- DatabaseBrowserTest**: A table displaying database test results:

| CODE_NAME | SUP_ID | PRICE | SALES | TOTAL |
|------------------|--------|-------|-------|-------|
| ColosseGrade | 5 | 7.99 | 1 | 2 |
| KonaGrade | 6 | 7.99 | 1 | 2 |
| FrenchRoastGrade | 7 | 7.99 | 1 | 2 |
| MochaNoctGrade | 8 | 7.99 | 1 | 2 |
| VanillaGrade | 9 | 7.99 | 1 | 2 |
| JavaGrade | 10 | 7.99 | 1 | 2 |
| IndonesiaGrade | 11 | 7.99 | 1 | 2 |
| OobaGrade | 1 | 7.99 | 1 | 2 |
| KappaGrade | 2 | 7.99 | 1 | 2 |
| JaxGrade | 3 | 7.99 | 1 | 2 |

The footer of the page includes "JPortal Tutorial for Jetspeed - Version 1.4b2", "Contact Us", and a "Bluesunrise" logo.

And here is the customizer for that pane:

Clicking the add portlet button allows you to select a portlet from the Portlet Browser.(we will cover the Portlet browser in tutorial 5). The layout can be one of the controllers described above (Single Column, Single Row, Two Column, Three Column).

The position of portlet entries can be moved up or down. Portlets may also be deleted from the pane. The pane may have a different skin assigned to it. You can change the control associated with each portlet entry.

3.8 Deploy

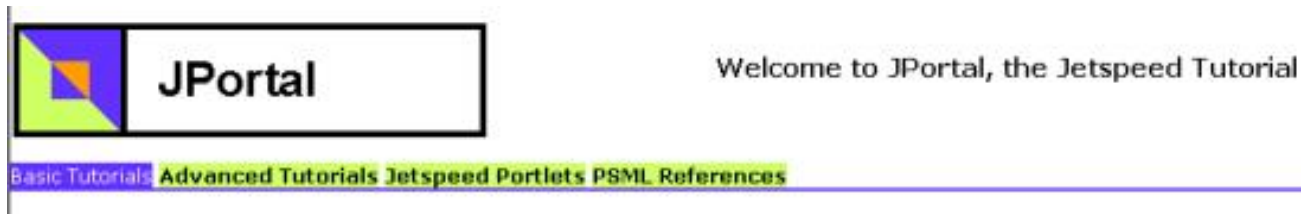
To deploy the system, type:

```
ant deploy          (or hotdeploy)
```

Use hotdeploy if you have already deployed the system once. This simply saves some time in packaging the JPortal deployment. Next point your browser at:

<http://localhost:8080/jportal/portal>

You should see the new site menus for the anonymous user:



Tutorial 4 – Site Security

Securing portal resources is a very important part of defining your portal site. You do not want non-authorized users accessing critical resources. Jetspeed has a declarative security registry for doing so. This section will cover:

1. Jetspeed Security Concepts
2. Managing Users and Roles
3. Security Features
4. Using the Customizer to Secure Portal Resources
5. The Security Example Portlet
6. Deploy

Let's get started. From the JPortal distribution root directory, type:

```
ant tutorial-4
```

Recommend bringing up these files in your editor:

1. `webapp/WEB-INF/conf/jportal-security.xreg`
2. `webapp/WEB-INF/templates/vm/portlets/html/simple-security.vm`
3. `webapp/WEB-INF/psml/user/anon/html/default.psml`
4. `webapp/WEB-INF/psml/user/turbine/html/default.psml`
5. `webapp/WEB-INF/conf/t4-portlets.xreg`

6. `webapp/WEB-INF/conf/JPortalJetspeed.properties`

since we will reference them in tutorial 4.

4.1 Jetspeed Security Concepts

The Jetspeed Security services are defined at the Jetspeed web site: [here](#). It is recommended that you review the concepts there before getting started. The purpose of portal security is to authenticate users of the portal, and to authorize access by those users to portal resources. All security in Jetspeed is defined through pluggable services. Jetspeed provides a default security policy and services. The default security service has a user database along with a security constraint registry. First let's review the security database and the object model.

Jetspeed Security Objects:

| Interface | Description |
|------------------------------|--|
| JetspeedUser | Defines the minimal attributes of a user in the portal system. |
| Role | Defines the minimal attributes of a role in the portal system. |
| Group | Defines the minimal attributes of a group in the portal system. |
| Permission | Defines the minimal attributes of a permission in the portal system. |

The default Jetspeed deployment comes with a populated sample database of users, roles, groups and permissions. This database is conveniently distributed with the webapp to simplify the first time experience. The database is Hypersonic SQL. For production systems, it is recommended to switch to a more robust database.

The default Security service uses [Jakarta Torque](#) to manage object-relational mapping of objects to and from relational tables. This default service can be configured to work with your own database. With version 1.4b3, there is now a LDAP Security service.

4.2 Managing Users and Roles

Jetspeed provides several administrative portlets to manage users, groups, roles and permissions. To see them, logon with the username/password `admin/jetspeed`. Click on the Security menu item, and should see something like this:

The screenshot shows the JPortal administrative interface. At the top left is the JPortal logo. To its right is the text "Welcome to JPortal, the Jetspeed Tutorial". Below the logo are three tabs: "Content", "Security" (which is selected), and "Admin". On the left side, there is a vertical menu with the following items: "User Browser" (highlighted in blue), "Security Role Browser", "Permission Browser", "Group Browser", and "Security browser". The main content area displays the "User Browser" portlet. It features a "Filter string (case sensitive):" input field, a "User Name" dropdown menu, and a "Filter" button. Below this is a table with the following data:

| Username | First Name | Last Name | Email | Actions |
|----------|------------|-----------|----------------------------|--------------------------|
| admin | Jetspeed | Admin | admin@jakarta-jetspeed.com | Edit Roles Groups Remove |
| turbine | Tommy | Turbine | tommy@jakarta-jetspeed.com | Edit Roles Groups Remove |
| anon | Anonymous | User | anon@jakarta-jetspeed.com | Edit Roles Groups Remove |

Below the table is an "Add User" button.

The User Browser allows you to add/edit/delete users, and associate relationships between users-groups and user-roles. The other browsers provide add/edit/delete maintenance for roles, groups and permissions.

User Roles

Roles for Jetspeed Admin

| Role Name | Assign |
|-----------|-------------------------------------|
| user | <input checked="" type="checkbox"/> |
| admin | <input checked="" type="checkbox"/> |
| guest | <input type="checkbox"/> |

Update

The user/group association isn't used specifically by the default security system. User/role associations are used throughout the system to perform role-based security checks. A security constraint in Jetspeed is defined between a role and a resource for a given action (permission). In this example from a security registry, we are declaring a role-based declarative security constraint called **requires-accountManager**:

```
<security-entry name="requires-accountManager">
  <meta-info>
    <title>Account Manager</title>
    <description>Grant full access to Account Manager Role, read
      access to Support Role.</description>
  </meta-info>
  <access action="*">
    <allow-if role="accountManager"/>
    <allow-if role="admin"/>
  </access>
</security-entry>
```

```
<access action="view">
  <allow-if role="guest"/>
</access>
</security-entry>
```

We are granting full access to users with the role **accountManager** or **admin** for all actions(permissions), but only granting **view** access to users with the role **guest**. All other users are denied access to the resource protected by this constraint.

We will look at how to associate a security registry entry with a portal resource in the section after next.

4.3 Security Features

Jetspeed has a number of configurable security settings. You can find most of these in the **JetspeedSecurity.properties** file. We will cover the ones that will probably be most used in a standard portal configuration.

Programmatic Security

Jetspeed can perform programmatic cascade deletes when deleting security objects. For example, when a user is deleted, all role and group associations will be automatically deleted with that user. This setting should be set to true for your database if it doesn't support cascading deletes. You would want to set this to true for the Hypersonic database.

```
services.JetspeedSecurity.programmatic.cascade.delete=false
```

Secure Passwords

Make the password checking secure. When enabled, passwords are transformed by a one-way function into a sequence of bytes that is base64 encoded. When a user logs in, the entered password is transformed the same way and then compared with stored the value. The algorithm property let's you choose what digest algorithm will be used for encrypting

passwords. Check documentation of your JRE for available algorithms.

```
services.JetspeedSecurity.secure.passwords=false  
services.JetspeedSecurity.secure.passwords.algorithm=SHA
```

New User Roles

When a new user is created, one or more roles can be assigned to that user. Multiple Role must be comma separated.

```
services.JetspeedSecurity.newuser.roles=user
```

Default Permissions

When a security resource has no permissions defined, these are the default permissions(actions) that are applied to the security check. The following permissions are defined in the default Jetspeed installation: view, customize, minimize, maximize, close, info, detach. Specifying * denotes all permissions. The default settings can differ can be set for both anonymous access and authenticated (logged in) access.

```
services.JetspeedSecurity.permission.default.anonymous=view  
services.JetspeedSecurity.permission.default.loggedin=*
```

Case Insensitive Usernames and Passwords

These options configure the logon username and password to be case sensitive or insensitive. When enabled, the **logon.casesensitive.upper** property controls whether the username and password are converted to upper or lower case before passing them on to the database.

```
services.JetspeedSecurity.caseinsensitive.username=false  
services.JetspeedSecurity.caseinsensitive.password=false  
services.JetspeedSecurity.caseinsensitive.upper=true
```

Auto-Account Disable

The Auto-Account-Disable Feature combines with the Logon-Strike-Count feature to disable accounts that may be under hacker attack. The strike count is over the strike interval. In the example below, 3 failed logons over five minutes would result in the account being disabled.

```
services.JetspeedSecurity.logon.auto.disable=false  
services.JetspeedSecurity.logon.strike.count=3  
services.JetspeedSecurity.logon.strike.interval=300  
services.JetspeedSecurity.logon.strike.max=10
```

Password Expiration

Number of days until password expires. To disable this feature, set it to 0.

```
services.JetspeedSecurity.password.expiration.period=0
```

Anonymous User Account

The anonymous user is actually stored in the database. The username is configurable.

```
services.JetspeedSecurity.user.anonymous=anon
```

Disabling the Portlet Action Buttons

Portlets are decorated with window controls (portlet action buttons). Some of these (the default) controls display action buttons. These action buttons can be enabled or disabled for all authenticated users, or for the anonymous user.

```
services.JetspeedSecurity.actions.anon.disable=true
```

```
services.JetspeedSecurity.action.allusers.disable=false
```

4.4 Using the Customizer to Secure Portal Resources

The customizer can be used to secure access to:

1. Portlet Entries(instances), using the Portlet Customizer
2. Portlet Pages, Panes, or Sets

The Customizer allows for the editing of the security constraint for any portlet page, pane or set. However, the currently logged on user must have the **admin** role in order to see this dropdown:



The default Portlet Customizer supports the editing of security constraints on any portlet entry(instance):



The screenshot shows the 'JPortal' logo on the left and the text 'Welcome to JPortal, the Jetspeed Tutorial' on the right. Below this is a 'Customize portlet' section with a light green header. The form contains three fields: 'Title' with the value 'Administrative Portlets', 'Skin' with a dropdown menu showing '- Default -', and 'Security ID' with a dropdown menu showing 'Owner-only'.

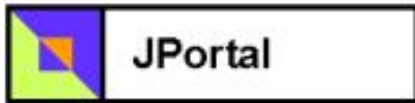
Portlets can also have security references set, but only the Security ID drop-down widget is only displayed if the current user has the admin role. See more details about the default Portlet Customizer in [here](#).

Security refs may also be added directly to the PSML file with a text editor:

```
<portlet-entry name="GroupForm" hidden="false" type="ref"
  parent="CustomizerVelocity" application="false">
  <security-ref parent="admin-only"/>
```

4.5 The Security Example Portlet

Tutorial 4 comes with one portlet, the Security Example portlet.



Welcome to JPortal, the Jetspeed Tutorial

[Basic Tutorials](#) [Advanced Tutorials](#) [Jetspeed Portlets](#) [PSML References](#)

Security Example

| Security Example Portlet | |
|--------------------------|---------------------------|
| Page Requires Admin Role | Page Requires User Role |
| Page Requires No Role | *Pane* Requires User Role |

This portlet allows for you to click on four different portal links. These links were created with the `$jslink` tool, which is used in templates to generate links to portal resources.

| Link Name | Link to Resource | Works with Anon | Works with Authenticated User (Turbine) |
|--------------------------|--|-----------------|---|
| Page Requires Admin Role | <code>\$jslink.setUser('admin')</code> link to default admin user page | NO | NO |
| Page Requires User Role | <code>\$jslink.setGroup('apache','news')</code> link to apache group, news page | NO | YES |
| Page Requires No Role | <code>\$jslink.setGroup('apache')</code> link to default apache group page | YES | YES |
| Pane Requires User Role | <code>\$jslink.getLink(\$jslink.USER, 'anon','default', \$jslink.PANE_ID,'105')</code> link to anonymous user default page, pane id = 105 | NO | YES |

When you test it out, you will see that there is a fifth pane on the anonymous page, but you cannot see it when you are logged on as the anonymous user. However, when you logon as the Turbine user, and click on the “Pane Requires User

Role”, then the pane does show up, since the Turbine user does have the **User** role.

The pane in the anonymous page has a security constraint:

```
<portlets id="105">
  <security-ref parent="user-only"/>
  <metainfo>
    <title>Secured</title>
  </metainfo>
  <entry parent="HelloVelocity"/>
</portlets>
```

4.6 Deploy

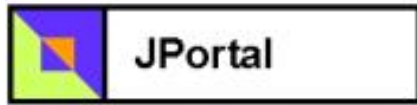
To deploy the system, type:

```
ant deploy      (or hotdeploy)
```

Use hotdeploy if you have already deployed the system once. This simply saves some time in packaging the JPortal deployment. Next point your browser at:

<http://localhost:8080/jportal/portal>

You should see the security example portlet in the default pane:



Welcome to JPortal, the Jetspeed Tutorial

[Basic Tutorial](#) [Advanced Tutorials](#) [Jetspeed Portlets](#) [PSML References](#)

[Security Example](#)

| Security Example Portlet | |
|--------------------------|---------------------------|
| Page Requires Admin Role | Page Requires User Role |
| Page Requires No Role | *Pane* Requires User Role |

Tutorial 5 – Portlet 101

Tutorial 5 guides you through the creation of a portlet, starting with a simple examples and then introduces you to the fundamental concepts of portlet development.. This section covers:

1. HelloWorld Portlet
2. Adding A Portlet to the Registry
3. Adding A Portlet to a Page
4. HelloUser Portlet
5. Deploy

Let's get started. From the JPortal distribution root directory, type:

```
ant tutorial-5
```

Recommend bringing up these files in your editor:

1. `webapp/WEB-INF/conf/t5-portlets.xreg`
2. `webapp/WEB-INF/psml/user/anon/default.psml`
3. `webapp/WEB-INF/psml/user/turbine/default.psml`
4. `src/java/com/bluesunrise/jportal/portal/portlets/`

`HelloWorldPortlet.java`
5. `src/java/com/bluesunrise/jportal/portal/portlets/HelloUserPortlet.java`
6. `webapp/WEB-INF/conf/JPortalJetspeed.properties`

since we will reference them in tutorial 5.

5.1 Hello World Portlet

To start with, the very first portlet is the simplest, the obligatory “Hello World”, it extends [AbstractInstancePortlet](#). All the code necessary is shown below:

```
package com.bluesunrise.jportal.portal.portlets;

import org.apache.jetspeed.portal.portlets.AbstractInstancePortlet;
import org.apache.turbine.util.RunData;
import org.apache.ecs.ConcreteElement;
import org.apache.ecs.StringElement;

public class HelloWorldPortlet extends AbstractInstancePortlet
{
    public ConcreteElement getContent (RunData runData)
    {
        return (new StringElement ("Hello World! #" + getID()));
    }
}
```

When you have compiled this portlet and configured it into the portlet registry, the user can choose the portlet from the list of available portlets when customizing their home page. As you can see, the most important method to override is the `getContent()` method, which provides the content of your portlet to the response payload. We will cover this method in detail in the section on the [Portlet interface](#).

In future chapters we will discover better methods for generating your content, using the set of MVC portlets provided with the Jetspeed distribution. See tutorials 7 onwards for examples these abstractions where it is not necessary to override the `getContent` method. For the purpose of learning the very basics of portlet operations, tutorials 5 and 6 will demonstrate how the `getContent` method works without any MVC abstraction.

5.2 Adding A Portlet to the Registry

In order for Jetspeed to know about your portlet class, two things must happen:

1. Put your class in the classpath.
2. Add a reference to your portlet to Jetspeed's Portlet Registry.

The [Jetspeed Portlet Registry](#) contains the definitions of all portlets known to Jetspeed.

We've already completed both steps for you when you built and deployed the examples for tutorial 5. The example class files are copied to the Jetspeed web application's class directory. This is normally where classes specific to your web application are placed (under `/WEB-INF/classes`).

Secondly, a [registry fragment](#) file was created and deployed. See the source:

```
webapp/WEB-INF/conf/t5-portlets.xreg
```

Registry fragments contain portlet definitions. Any file in the `/WEB-INF/conf` directory that has the `xreg` extension is included in the Jetspeed Registry. Here is what a registry entry looks like for our example:

```
<?xml version="1.0" encoding="UTF-8"?>
<registry>
  <portlet-entry name="HelloWorld" hidden="false" type="instance" application="false">
    <meta-info>
      <title>HelloWorld</title>
      <description>Portlet How To Example 1 - Hello World</description>
    </meta-info>
    <classname>com.bluesunrise.portal.portlets.HelloWorldPortlet</classname>
    <media-type ref="html" />
  </portlet-entry>
</registry>
```

We won't cover all the registry syntax here, but it's important to give it a name, enter the class name correctly, and limit the `<media-type>` entry to only the media types that you will support. In our example, we only support HTML.

5.3 Adding A Portlet to a Page

After successfully deploying the examples to Jetspeed, you can then customize your page. The Anonymous User and the Turbine User already include the Hello World portlets. Let's go ahead and add a second Hello World portlet and see what happens. Start-up Jetspeed, and logon with the username/password of

Username = turbine

Password = turbine

Click on the 'Pencil Icon' in the Title Bar (right below the menu). This will start the Jetspeed Customizer.

Basic Tutorials
 Advanced Tutorials
 Jetspeed Portlets
 PSML References



Click the “Basic Tutorials” menu option. From the Pane Customizer, click the “Add Portlet” button.

You should see a screen like below. Select “HelloWorld” and then press the “Apply” button.

Filter portlets by category: All Portlets

| Add | Title | Description |
|-------------------------------------|------------------------------|--|
| <input type="checkbox"/> | Administrative Portlets | List of most useful Administrative portlets. |
| <input type="checkbox"/> | Apache News from Apache Week | The essential resource for anyone running an Apache server o |
| <input type="checkbox"/> | Apacheweek | Description not available |
| <input type="checkbox"/> | BBC Front Page News | Description not available |
| <input type="checkbox"/> | BBC Technology News | Description not available |
| <input type="checkbox"/> | BBC UK News | Description not available |
| <input type="checkbox"/> | BBC World News | Description not available |
| <input type="checkbox"/> | Barrapunto | Slashdot clone in Spanish |
| <input type="checkbox"/> | Change language portlet | Change language portlet |
| <input type="checkbox"/> | DatabaseBrowserTest | Simple Test Database Browser Portlet Example |
| <input type="checkbox"/> | HelloJSP | Simple JSP Portlet Example |
| <input checked="" type="checkbox"/> | HelloUser | JPortal Tutorial - Example 2 - Hello User |
| <input type="checkbox"/> | HelloVelocity | Simple Velocity Portlet Example |
| <input type="checkbox"/> | HelloVelocityCached | Simple Cached Velocity Portlet Example |
| <input checked="" type="checkbox"/> | HelloWorld | JPortal Tutorial - Example 1 - Hello World |

Apply Cancel Next >>

Then from the next screen, press “Save and Apply” and “Apply” again.

Finally, you should return to the newly customized screen and see the output of your portlet:

HelloWorld

Hello World! #P-f1a5704d9e-10001

HelloUser

Hello Tommy!

Security Example**HelloWorld**

Hello World! #P-f1a572d4a4-1000b

So what are those funny numbers after Hello World? They are the portlet instance ids, which are automatically assigned to the portlet. This id uniquely identifies your portlet instance in the entire portal site. Portlet Instances, are instances of the your portlet on a PSML page. The two different instances use the same Java class instance, which saves memory, however they use different parameters and state, which is important when you have two or more instances of the same portlet on a page.

5.4 Hello User Portlet

The next step in your portlet example is to make the world more specific. That is, instead of writing out a static string, the greeting is personalized according to the user settings.

```
package com.bluesunrise.portal.portlets;

import org.apache.jetspeed.portal.portlets.AbstractInstancePortlet;
import org.apache.turbine.util.RunData;
import org.apache.turbine.om.security.User;
import org.apache.ecs.ConcreteElement;
import org.apache.ecs.StringElement;
```

```

public class HelloUserPortlet extends AbstractInstancePortlet
{
    public ConcreteElement getContent(RunData runData)
    {
        StringBuffer text =new StringBuffer();

        text.append("Hello ");

        String name = runData.getUser().getFirstName();

        if (name ==null)
            name =" Anonymous";

        text.append (name);
        text.append ("!");
        return (new StringElement(text.toString()));
    }
}

```

With this example, you will be using the classes inherited from another Apache project: [Turbine](#). Turbine is a web application framework based on the Java Servlet API. Jetspeed was written with Turbine as its framework, and also upon the Java Servlet API. Let's take a closer look at the interfaces supplied by Turbine: [RunData](#) and [User](#).

The **RunData** interface provides you with access to the servlets request state and other session-context information. Examples are security (access control), Turbine actions, cookies, servlet parameters (parsed), and the current user, to name a few. The current user is accessible via the **getUser()** or **getJetspeedUser()** methods as shown in the example above.

With the **user** object, you can then access typical user information, and also store temporary values for your session (**getTemp()**, **setTemp()**). Here's as summary of the standard user information provided:

| User Property/Methods | Description |
|-----------------------|-------------|
|-----------------------|-------------|

| | |
|-------------------------------|---|
| AccessCounter | Keep track of how many times the user has accessed the portal |
| Confirmed | Returns the confirmation value used during automatic account sign up. |
| CreateDate | The creation date of this account. |
| Email | The email address of the user. |
| FirstName | The first name of the user. |
| LastAccessDate | The last access time stamp for the user. |
| LastLogin | The last login time stamp for the user. |
| LastName | The last name of the user. |
| Password | The password of the user. |
| Perm | Jetspeed stores a hash table of name/value pairs in a binary column in the database. |
| Temp | A temporary hash table of values for the user stored in the session. These values go away after user logs off or after the session times out. |
| UserName | The user name (primary credential) of the user. |
| hasLoggedIn | Indicates if the user is currently logged on. |
| incrementAccessCounter | Increment the user's access counter. |
| isConfirmed | Indicates if this user has yet been confirmed. |
| removeTemp | Remove a name/value pair from the temporary hash table. |

| Jetspeed User Property | Description |
|------------------------|---|
| Disabled | An account can be disabled with this property |

| | |
|---------------|--|
| UserId | The user id |
| isNew | Boolean indicating if the user has been stored yet |

Remember that the set methods of the properties may not store the changes to the database immediately. There is a setting in the `JPortalJetspeed.properties` file that must be set to true to automatically save all modifications to the user on logout.

```
# automatically save user state on logout
automatic.logout.save = false
```

Otherwise, you will need to explicitly save the user with the User Management service. Here is an example of disabling the current user.

```
JetspeedUser user = user = rundata.getJetspeedUser();
user.setDisabled(true);
JetspeedUserManagement.saveUser(user);
```

Jetspeed actually extends the `RunData` interface as [JetspeedRunData](#). You can safely cast `RunData` to `JetspeedRunData` from anywhere within Jetspeed. The extended class gives you access to some important Jetspeed request state such as the [Profile](#) the [Capability Map](#).

To summarize, you can always get the current user from the `RunData`. `RunData` is used all over Jetspeed as a common mechanism for passing request information.

5.5 Deploy

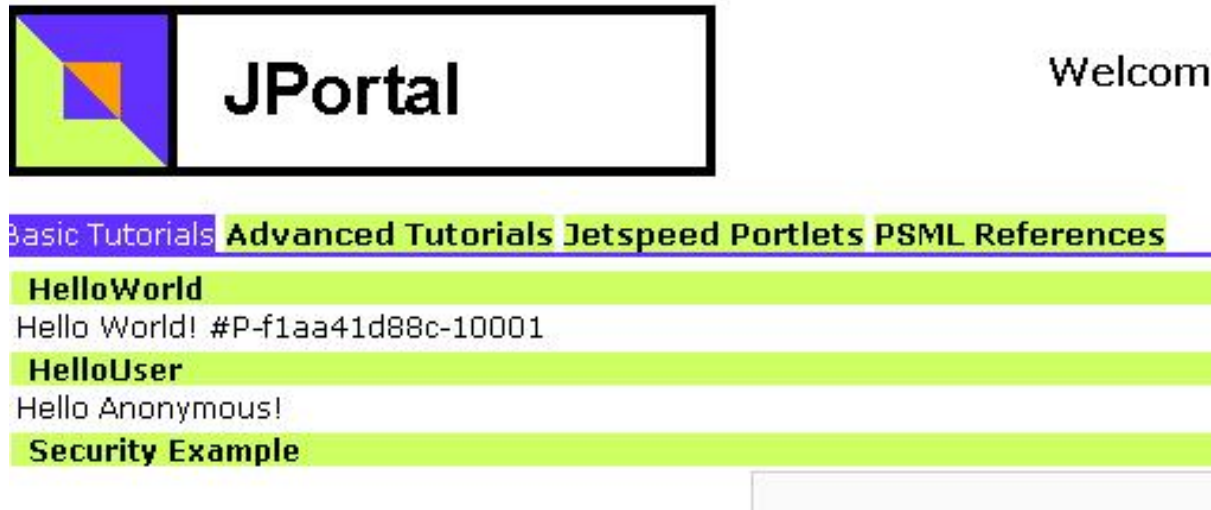
To deploy the system, type:

```
ant deploy      (or hotdeploy)
```

Use hotdeploy if you have already deployed the system once. This simply saves some time in packaging the JPortal deployment. Next point your browser at:

<http://localhost:8080/jportal/portal>

You should see the new Hello portlets in the default pane:



Tutorial 6 - The Portlet Interface

Tutorial 6 introduces the Portlet interface. This section covers:

1. Media Types
2. Portlet Life Cycle
3. Init Parameters, Attributes and Titles
4. Portlet Modes
5. Deploy

Let's get started. From the JPortal distribution root directory, type:

```
ant tutorial-6
```

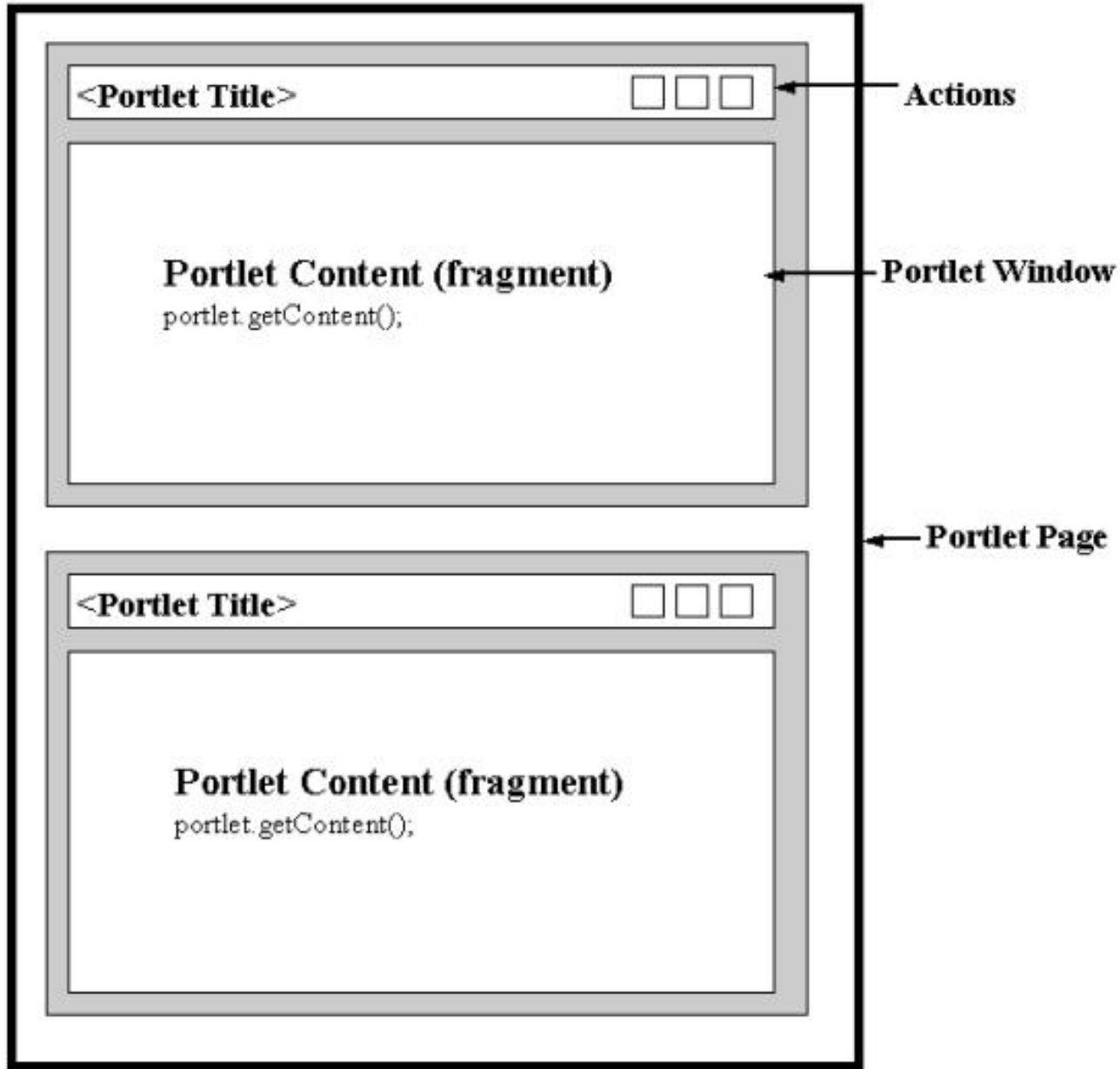
Recommend bringing up these files in your editor:

1. `webapp/WEB-INF/conf/t6-portlets.xreg`
2. `webapp/WEB-INF/psml/user/anon/html/default.psml`
3. `webapp/WEB-INF/psml/user/turbine/html/default.psml`
4. `src/java/com/bluesunrise/jportal/portal/portlets/HelloPortletInterface.java`
5. `webapp/WEB-INF/conf/JPortalJetspeed.properties`

since we will reference them in tutorial 6.

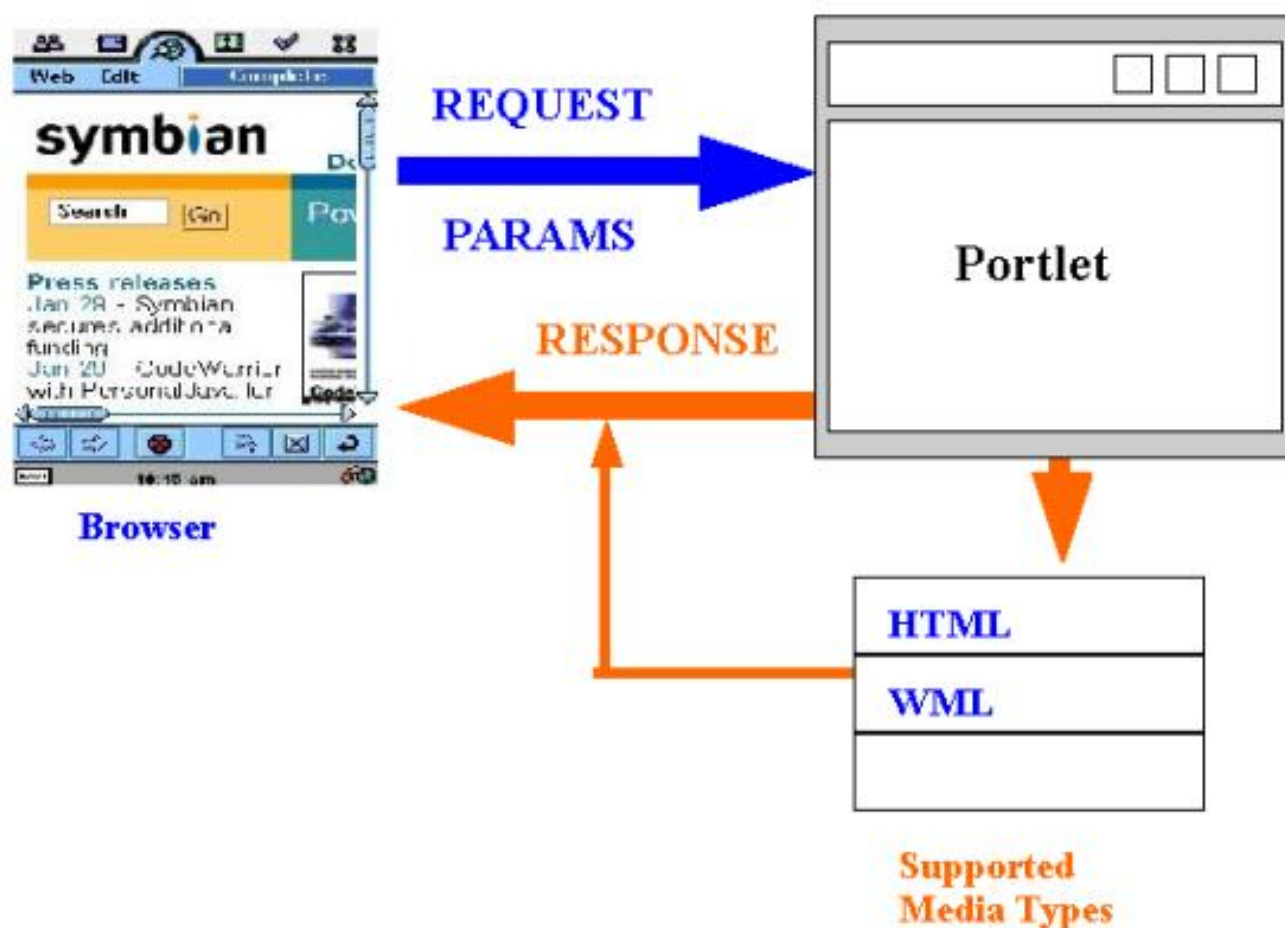
Portlets generate content fragments (from the `getContent()` method) that are aggregated together in a portal page as shown below. Jetspeed takes the content generated by each portlet, and includes it in a portlet window. The location on the page for the portlet window is controlled by the administrator or user when customizing a portlet page. The Portlet interface is the contract between the portlet developer and Jetspeed defining how a portlet behaves and interacts with the

Jetspeed environment.



6.1 Media Types

The content generated is dependent on the media type supported by the browser. Browsers (devices) supply request parameters describing the capabilities of device. For instance, a web browser would provide in its request parameters a description of which [MIME types](#) and languages it supports. Jetspeed packages this all up in a request object call [JetspeedRunData](#). You can check the request for the media type by getting the [CapabilityMap](#) associated with the request (`rundata.getCapabilityMap`) and the checking the media type against the supported media types for your portlet.



Let's take a look at how the `HelloPortletInterface` portlet checks the media type during content fragment generation:

```
public ConcreteElement getContent(RunData rundata)
{
    JetspeedRunData jrun = (JetspeedRunData) rundata;

    CapabilityMap map = jrun.getCapability();

    StringBuffer text = new StringBuffer();
    String mimeType = map.getPreferredType().toString();

    if (this.supportsType(map.getPreferredType()))
    {
        text.append("Supports preferred MimeType: " + mimeType);
    }
    else
    {
        text.append("Doesn't support preferred MimeType: "
            + mimeType);
    }
}
...

```

Every portlet has the `supportsType` method for checking a browser device's MIME type. The Media Types are looked up from the portlet definition in the registry. Looking at the registry entry for `HelloPortletInterface`, we see the supported media types are directly in the registry definition:

```
<portlet-entry name="HelloPortletInterface"
    hidden="false" type="instance" application="false">
    <meta-info>
        <title>Hello Portlet Interface</title>
        <description>JPortal Tutorial - Hello Portlet Interface
        </description>
    </meta-info>
    <classname>com.bluesunrise.jportal.portal.portlets.HelloPortletInterface

```

```
</classname>
  <parameter name="version" value="1.4b3" hidden="false"/>
  <media-type ref="html"/>
  <media-type ref="wml"/>
  <category>tutorial</category>
  <category>portlet</category>
</portlet-entry>
```

There can be multiple `<media-type>` entries per portlet.

Media types are defined in the [Media Type Registry](#). Four media types are currently supported in the registry:

- HTML
- WML
- XML
- VXML

```
<media-type-entry name="html">
  <mime-type>text/html</mime-type>
  <character-set>UTF-8</character-set>
  <meta-info>
    <title>HTML</title>
    <description>Rich HTML for HTML 4.0 compliants browsers
  </description>
  </meta-info>
</media-type-entry>
```

Also note that the `getContent` method returns a class called `ConcreteElement`.

```
import org.apache.ecs.ConcreteElement;
import org.apache.ecs.StringElement;
```


Unfortunately, Jetspeed is coupled to the Element Construction Set, a Java-based HTML mark-up generator. As you will see in the tutorials 7 and onwards, you shouldn't have to work much with ECS or the low level methods of the Portlet interface. Jetspeed provides a better way of getting content using templating engines such as Velocity or JSP, along with RSS and XSLT ready-written portlets that you can extend.

For more information on `RunData` class and ECS elements , see [Appendix B – Turbine and ECS](#).

6.2 Portlet Life Cycle

All portlets can be automatically created when Jetspeed first starts up. The default and recommended behaviour is not to create the portlets until they are accessed on a PSML page. Once a page where a portlet exists is requested, then at this point, the portlet is created and added to the portlet cache.

```
# Jetspeed can automatically create/instantiate all your Portlets
# and place them in the cache when Jetspeed starts up.
autocreate.portlets=false
```

At the time of portlet creation, a portlet's [init\(\)](#) method is called. The init method is only called once, during the portlet creation phase. The lifetime of a portlet is controlled by how long a portlet stays in the cache. When a portlet is removed from the cache, the portlet is effectively removed from memory (left to the garbage collector) and is no longer accessible. If a portlet is removed from the cache, and then requested again, the portlet will have its init method called again, since a new instance is created of the portlet. Both the system administrator and programmer can control the lifetime of a portlet by controlling how long it lives in the cache.

A portlet is managed through a not so well defined life cycle that defines how it is initialized, how it handles requests for content, and how long it lives based on caching behaviour described later in this section. Thus you could consider there to be 3 phases in the life cycle of a portlet:

1. Init
2. Render
3. Destroy

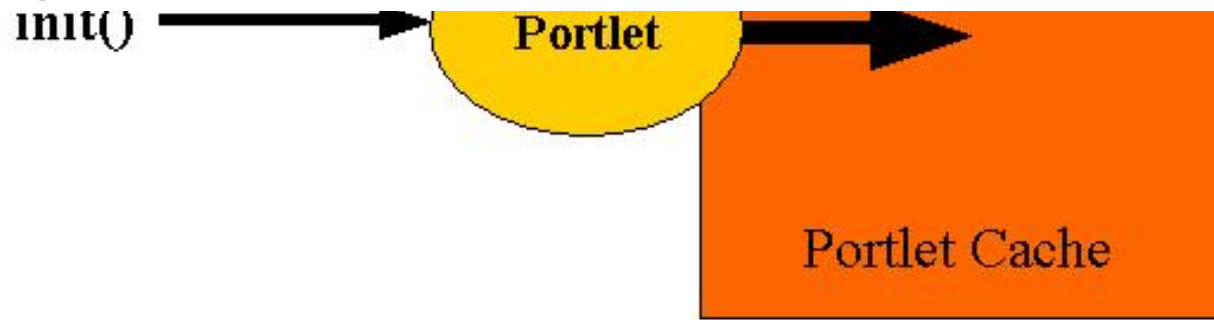
Jetspeed does support another phase in between phase 1 and 2. It is called the processAction phase. Unfortunately, the processAction phase does not manifest itself directly in the portlet interface. Instead, actions must be handled in action classes. Action classes are inherited from Turbine, and they bleed through into the design of Jetspeed, coupling the Jetspeed architecture to Turbine's in a very bad way. Velocity portlets attempt to rectify this design flaw, but the bottom line is that is still a kludge and the action processing phase is not a part of the portlet interface, but must be handled on another action interface. We will cover actions in more depth in the section on [Velocity Portlets](#).

To match the phases with interface methods, we have:

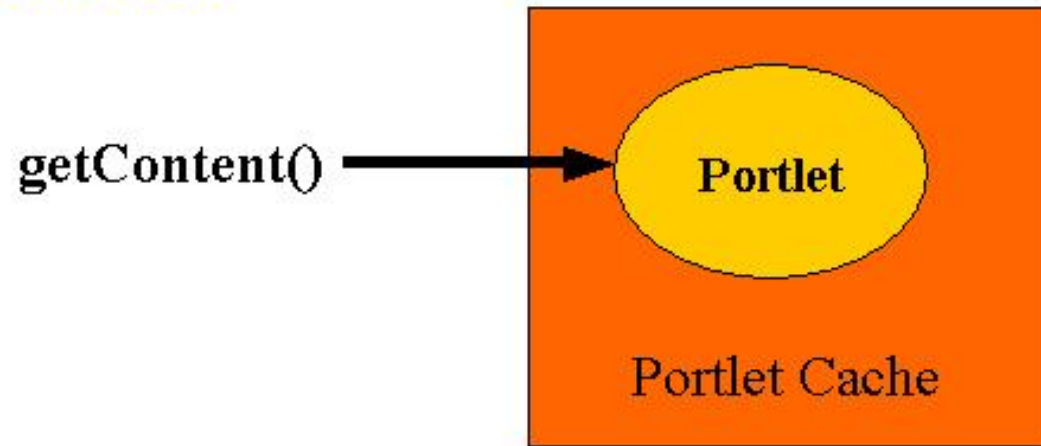
| Phase | Method |
|---------------|-----------------|
| Init | init |
| ProcessAction | Turbine Actions |
| Render | getContent |
| Destroy | --none-- |

During the init phase, it is recommended that you do any one-time initialisation of the portlet. Usually this involves the initialisation of costly resources such as database connections or other costly activities. The render phase is called per request. Every time a portlet's content is requested, the getContent method is called.

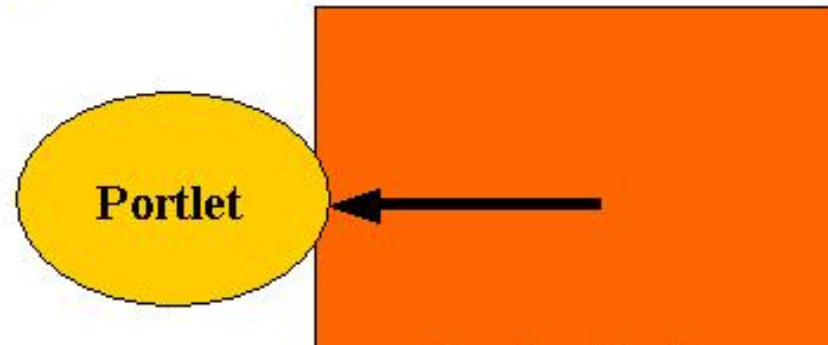
Init Phase



Render Phase



Destroy Phase



Portlet Cache

The destroy phase is dependent on the life time definition of the portlet. Unfortunately, your portlet is never notified when the destroy phase occurs. You can control the life time of your portlet. If your portlet inherits from [AbstractPortlet](#), then the lifetime of your portlet is controlled by how long it lives in the cache, and by how many instances of the portlet there are in the cache.

First let's look at how long your portlet will live in the cache. By default, this is controlled by a global portal setting for all portlets:

```
# TimeToLive.default =  
# number of milliseconds an unused portlet will remain in cache.  
# Default 2700000 which is 45 minutes (45 * 60 * 1000)  
services.PortletCache.TimeToLive.default=2700000
```

The default setting is 45 minutes. After a portlet has been in the cache for 45 minutes, it will be reloaded. This means the init method is called again. Since there is no destroy method, your portlet doesn't know when it is getting destroyed.

If your portlet inherits from [AbstractPortlet](#) or [AbstractInstancePortlet](#), then your portlet is cacheable, but it will not be evicted from the cache, since the base implementation [AbstractPortlet](#) never allows for your portlet to expire.

```
public Expire getExpire()  
{  
    try  
    {  
        return ExpireFactory.getExpire( this,  
                                         ExpireFactory.NO_EXPIRE );  
    } catch ( JetspeedException e ) {
```

```
        Log.error( e );
        return null;
    }
}
```

Some portlets will expire, such as the [FileWatchPortlet](#) and [NewRSSPortlet](#) which uses the cache to control refreshing of its content.

Whenever Jetspeed goes through the initialisation phase for a portlet, it will first check to see if the portlet already exists in the cache. It does this by looking up the portlet using a unique cache handle. The cache handle method can be used to control the number of instances of the same portlet in the cache. [AbstractPortlet](#) has a method [getHandle](#). For the base class, a rather odd algorithm is used to determine whether a new portlet instance should be created:

1. the URL parameter, but only if the `cachedOnURL` attribute is set to true
2. all other parameters, both their names and values

The algorithm takes all of these strings and combines them to make a unique string. Thus the creation of multiple instances of possibly the same portlet is controlled by the uniqueness of the portlets parameters. This seems like a rather odd solution, but works perfectly fine if all of your portlets are RSS feeds, as Jetspeed was once designed but has since evolved into much more. In the example below of an RSS portlet, the key would be combined to create:

<http://www.mozilla.org/news.rdf?itemdisplayed-10>. This isn't really very useful, since the items displayed shouldn't have any affect on the number of instances created of a portlet. Basing the cache on the URL makes a little more sense, since we control the number of portlets to be equal to the number of feeds in the system.

```
<portlet-entry name="Mozilla" hidden="false"
              type="ref" parent="RSS" application="false">
  <meta-info>
    <title>Mozilla</title>
  </meta-info>
```

```

    <classname>org.apache.jetspeed.portal.portlets.NewRSSPortlet
    </classname>
    <parameter name="itemdisplayed" value="10" hidden="false"
        cachedOnName="true" cachedOnValue="true"/>
    <url cachedOnURL="true">http://www.mozilla.org/news.rdf</url>
    <category group="Jetspeed">news.software.opensource</category>
</portlet-entry>

```

The [AbstractInstancePortlet](#) has a different algorithm to create the uniqueness of a portlet:

```

StringBuffer handle = new StringBuffer(256);
handle.append(pc.getPageId());
handle.append('/');
handle.append(pc.getPortletId());

return handle.toString();

```

New portlets are created for every portlet instance. A portlet instance is defined as each instance of a portlet referenced in a PSML file. Thus for each portlet defined on a PSML page, a new Java instance of the portlet is created and put in the cache with a unique handle of: PSML-PAGE-ID + PORTLET-ID. The important gain with this approach is that you can now have the same portlet twice on the page, and those two portlets will have their own private parameters.

This approach will have scalability issues. Think about the case where each user has his own page. Since we are creating new portlets for every user in the system with her own page, if there are thousands of simultaneous users, we have the same portlet instantiated thousands of time. It's a waste of memory. This approach was necessary in order to rectify a bug in Jetspeed's design: it shares all instance parameters across all instances. If you find this approach to be unsatisfactory, just override the [getHandle](#) method for your portlet and have its uniqueness based on the name of your portlet as shown in the commented out method in the tutorial example (its commented out in order for the tutorial example in 6.3 to work properly with multiple instances of the same portlet):

```

public static Object getHandle(Object config)
{

```

```
    PortletConfig pc = null;

    if (!(config instanceof PortletConfig))
    {
        return null;
    }
    return pc.getName();
}
```

6.3 Init Parameters, Attributes and Titles

The Portlet interface allows for you to set two different types of configuration variables for your portlet and its instances. Init Parameters are defined in the registry. Attributes are defined in a PSML file. Init Parameters are associated with all portlets of a given class and are read-only. Attributes are associated with only one instance of a portlet class on a given PSML page, and can be modified and persisted. Given our example portlet, we have the Init Parameter named ‘version’, which is set to the value ‘1.4b3’. We also have two instances of the same portlet on our page. Each of these portlets instances have the same parameter defined on the same PSML page, called ‘city’ with the value of ‘Tokyo’ and ‘Sidney’.

Here is the Init Parameter defined in the registry. Init Parameters can be hidden. This means they will not be displayed in certain areas of the application such as portlet customization:

```
<parameter name="version" value="1.4b3" hidden="false"/>
```

And the attributes for each portlet instance:

```
<entry parent="HelloPortletInterface">
  <parameter name='city' value='Tokyo' />
</entry>
<entry parent="HelloPortletInterface">
  <parameter name='city' value='Sidney' />
</entry>
```

The portlet interface supports working with portlet instances directly using the methods:

getID() – returns the unique Portlet instance id
getInstance() – returns the instance associated with this portlet.
getAttribute() – returns a persistent attribute from page storage
setAttribute() – stores an attribute to a persistent page

For accessing read-only Init Parameters, you must first get the [PortletConfig](#) object for a portlet, and from there you can get the Init Parameters. From the getContent method:

```
text.append("Portlet id = " + this.getID());
text.append("<BR/>");
text.append("Init Parameter (version): " +
           pc.getInitParameter("version", "NOT FOUND!"));
text.append("<BR/>");
text.append("Page Attribute (city): "
           + this.getAttribute("city", "NOT FOUND!", rundata));
```

The Jetspeed Portlet customizer only displays Init Parameters in the edit window. Thus the “city” attribute described above will not show up in the customizer, since it doesn’t have an Init Parameter associated with it. The portlet customizer associates Attributes with Init Parameters, and allows you to edit the parameter, but when it is saved, it is saved as an

attribute to the current page, and is not saved to the common registry entry.

Let's try customizing the first Hello Portlet Interface portlet:

Customize portlet

| | |
|----------------|--|
| Title | <input type="text" value="Hello New Title"/> |
| Skin | <input type="text" value="- Default -"/> |
| version | <input type="text" value="2.0"/> |

Press update and we see:

Hello New Title

```
Supports preferred mimeType: text/html
Tommy!
Portlet id = P-f1bfb0a363-10001
Init Parameter (version): 2.0
Page Attribute (city): Tokyo
MODE = VIEW
```

Hello Portlet Interface

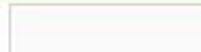
```
Supports preferred mimeType: text/html
Tommy!
Portlet id = P-f1bfb0a51c-10002
Init Parameter (version): 1.4b2
Page Attribute (city): Sidney
MODE = VIEW
```

Hello World

```
Hello World! #P-f1bfb0a51c-10003
```

Hello User

```
Hello Tommy!
```

Security Example

If you want to use the default customizer, it is recommended that all of your parameters be defined in the portlet registry when you deploy your portlets. Also notice that the title was changed. Titles are maintained with the Portlet interface methods [getTitle](#) and [setTitle](#).

6.4 Portlet Modes

There are several portlet modes that Jetspeed supports:

1. View
2. Customize (Edit)
3. Print-Friendly

4. Info
5. Minimize
6. Maximize

View mode is just the normal mode of operation. A request to display a portal page is considered the default or view mode. The `getContent` method will be called on the portlet.

Customize mode is an important mode of operation. The Portlet interface method `providesCustomization` can be overridden if you want to provide your own portlet customizer. Otherwise, Jetspeed provides a default customizer when you go into edit mode. If you are providing your own customization, you will need to check for the current mode of the request in your `getContent` method. Velocity portlets provide a more elegant way of handling customization covered in the next section.

Maximize Mode displays the selected portlet on the entire page. All other portlets are not displayed. **Print-Friendly** is the same as View Mode, but the portal engine does not display any controls around your portlet. The other modes: **Info**, **Close** and **Minimize** do not make calls directly to your portlet. Unfortunately, your portlet is not notified of this event, but must check the request parameters to detect if it is being minimized. **Info** mode simply displays some runtime information about your portlet in a maximized view. The content generated for the portlet is controlled by Jetspeed. Minimizing a portlet will only show the control of the portlet, but not the content. There is also the Close action, which removes the portlet from the page. All six of these mode actions and the close action can be displayed on the control of a portlet:



In order from right to left we have: Customize, Print-Friendly, Info, Close, Minimize and Maximize.

By checking [JetspeedRunData.getMode](#), you can determine the mode of the current request:

```
switch ( jrun.getMode() )
{
case JetspeedRunData.NORMAL:
    text.append( "MODE = VIEW" );
    break;
case JetspeedRunData.CUSTOMIZE:
    text.append( "MODE = CUSTOMIZE" );
    break;
case JetspeedRunData.MAXIMIZE:
    text.append( "MODE = MINIMIZE" );
    break;
default:
    text.append( "MODE = UNKNOWN" );
    break;
}
```

6.5 Deploy

To deploy the system, type:

```
ant deploy      (or hotdeploy)
```

Use hotdeploy if you have already deployed the system once. This simply saves some time in packaging the JPortal deployment. Next point your browser at:

<http://localhost:8080/jportal/portal>

You should see the new Hello Portlet Interface portlets in the default pane:

[Basic Tutorials](#) [Advanced Tutorials](#) [Jetspeed Portlets](#)

Hello Portlet Interface

Supports preferred mimeType: text/html
Anonymous!
Portlet id = P-f1bfb0c5b2-1000c
Init Parameter (version): 1.4b2
Page Attribute (city): Tokyo
MODE = VIEW

Hello Portlet Interface

Supports preferred mimeType: text/html
Anonymous!
Portlet id = P-f1bfb0c5b2-1000d
Init Parameter (version): 1.4b2
Page Attribute (city): Sidney
MODE = VIEW

Hello World

Hello World! #P-f1bfb0c5b2-1000e

Hello User

Hello Anonymous!

Security Example

Tutorial 7 – Velocity Portlet

Tutorial 7 introduces the Velocity Portlet. This section covers:

1. Introduction to Velocity
2. Velocity Portlets in the Registry

3. The Velocity Template
4. Template Resolution
5. Velocity Actions and the Context
6. Portlet Customization and Parameter Styles
7. Action Events
8. Deploy

Let's get started. From the JPortal distribution root directory, type:

```
ant tutorial-7
```

Recommend bringing up these files in your editor:

1. `webapp/WEB-INF/conf/t7-portlets.xreg`
2. `webapp/WEB-INF/psml/user/anon/html/default.psml`
3. `webapp/WEB-INF/psml/user/turbine/html/default.psml`
4. `src/java/com/bluesunrise/jportal/modules/actions/portlets/ TutorialStockQuoteAction1.java`
5. `src/java/com/bluesunrise/jportal/modules/actions/portlets/ TutorialStockQuoteAction2.java`
6. `src/java/com/bluesunrise/jportal/modules/actions/portlets/ CobiJonesPortletAction.java`
7. `webapp/WEB-INF/templates/vm/portlet/html/stock-quotel.vm`
8. `webapp/WEB-INF/templates/vm/portlet/html/stock-quote2.vm`
9. `webapp/WEB-INF/templates/vm/portlet/html/cobi-jones-form.vm`
10. `webapp/WEB-INF/conf/JPortalTurbine.properties`

since we will reference them in tutorial 7.

7.1 Introduction to Velocity

[Velocity](#) is a Java-based template engine. It permits web page designers to reference methods defined in Java code. Web designers can work in parallel with Java programmers to develop web sites according to the Model-View-Controller (MVC) model, meaning that web page designers can focus solely on creating a well-designed site, and programmers can focus solely on writing top-notch code. Velocity separates Java code from the web pages, making the web site more maintainable over the long run and providing a viable alternative to [Java Server Pages](#) (JSP) or [PHP](#).

Jetspeed has a Velocity templating service built directly into the Jetspeed engine. Many of the controls and controllers are Velocity-driven to create the layout of the portal. You can also base your portlets on Velocity. This means that your portlets will also follow the MVC design pattern, separating content from code.

The examples in tutorials 5 and 6 were useful for you to learn the Portlet interface. However, overriding the `getContent` method of the portlet interface is not good practice. We recommend abstracting the content generation phase by basing your portlets on one of the MVC-based portlets provided in the Jetspeed distribution; such as [JSPPortlet](#), [XSLTPortlet](#), [RSSPortlet](#), [HTMLPortlet](#) or of course the [VelocityPortlet](#).

A Velocity portlet is made up of the typical MVC components:

| MVC Component | Velocity Component |
|---------------|---------------------------------|
| Model | Java Objects put in the context |
| View | Template |
| Controller | Your Velocity Action |

The controller is your Velocity Action class. The base [VelocityPortlet](#) class should rarely have to be modified. Your view is a Velocity template, which will generate the content of your portlet by pulling out dynamic model information from the

Velocity context. The getContent method of the [VelocityPortlet](#) should never be edited. All content is generated by the template, as designed in our MVC design pattern.

The Life Cycle phases of a portlet are also enhanced with the Velocity portlet.

| Phase | Method |
|---------------|--|
| Init | init |
| ProcessAction | Velocity Action and Action Events |
| Render | Template is called by Velocity Portlet |
| Destroy | --none-- |

Velocity portlets are really about dynamic content. If you have static content, there is no real benefit to using a Velocity portlet; take a look at one of the static content generation portlets such as the [HTMLPortlet](#) instead. The basic function of Velocity is really very simple, it substitutes live Java objects into a Velocity template. There is an online tutorial with great examples [here](#).

Let's look at simple introductory example, but we recommend visiting the Velocity site and investing some time in their well-written tutorials.

```
<HTML>
<BODY>
Hello $customer.Name!
<br/>
Here is a list of your current subscriptions:<br/><br/>
<table>
#foreach( $subscription in $subscriptions )
  #if ( $customer.isSubscribed($subscription) )
    <tr>
      <td>
        $subscription.Name
```



```
        </td>
    </tr>
#end
#end
</table>
```

This example is not included with the examples source code distribution.

The above is an example Velocity template that displays all subscriptions for a given customer. There are two dynamic model objects that we are working with: the customer and subscriptions. Velocity uses a very simple syntax for model variables, simply prefix the variable with a \$ sign. The designer can then access all public methods and accessors of that Java object. Velocity uses Java reflection to find the methods. Thus in our example above, the customer object has a getter and setter for the Name attribute. You don't need to specify the get prefix, Velocity will figure it out.

Velocity provides a small set of directives for logic control. Most commonly used are #if and #foreach. #foreach will iterate over any Java 2 collection or array. The #if statement above calls a public method to check if the customer is subscribed to a subscription. Finally, the subscription name is displayed in a table column by getting the Name attribute.

So where did these two variables (\$customer, \$subscriptions) come from?

They came from your action class. We will look into how to program an action class in more detail later in this section. Right now it's important to understand that your action class puts your model objects into a Velocity construct called the 'context'. The context is where all variables are made accessible to a template. The context is the common liaison between your model and view. Here is how the Java code would put an object into the context:

```
Customer customer = CustomerPeer.retrieveByPK(primaryKey);
context.put("customer", customer);

List subscriptions = SubscriptionPeer.doSelect(criteria);
context.put("subscriptions", subscriptions);
```

This example is not included with the examples source code distribution.

There is another tutorial on Velocity at [Java World](#). Now that we have a basic understanding of the Velocity context and templates, let's get back to the Velocity Portlet. The MVC components are all configured in the portlet registry.

7.2 Velocity Portlets in the Registry

Velocity Portlets are defined like any other portlet: in the portlet registry. In tutorial 7 we define two new portlets:

```
<portlet-entry name="TutorialStockQuote1" hidden="false" type="ref"
    parent="Velocity" application="false">
...
    <parameter name="template" hidden="true"
        value="tutorial-stock-quote1"/>
    <parameter name="action" hidden="true"
        value="portlets.TutorialStockQuoteAction1" />
...
</portlet-entry>

<portlet-entry name="TutorialStockQuote2" hidden="false" type="ref"
    parent="Velocity" application="false">
...
    <parameter name="template" hidden="true"
        value="tutorial-stock-quote2" />
    <parameter name="action" hidden="true"
        value="portlets.TutorialStockQuoteAction2" />
...
</portlet-entry>
```

When defining a Velocity portlet, there are two required parameters: the template and the action. The template defines the Velocity template which will generate the portlet content. It is your MVC View component. The action is the controller, and it has several responsibilities including handling action events and populating the context. The templates should be

placed in the portlets subdirectory of one of your Velocity template paths. The action is placed in the module path, conventionally under the portlets directory of the root actions directory.

Also note that a Velocity portlet can derive from one of two Velocity portlets: [Velocity](#) or [CustomizerVelocity](#). The only difference being that the Velocity portlet uses the default Jetspeed portlet customizer, where as the CustomizerVelocity portlet is expected to provide its own customization. Look at the Weather Portlet as an example of a CustomizerVelocity portlet providing its own customization.

```
<portlet-entry name="TutorialStockQuote1" hidden="false" type="ref"
              parent="Velocity" application="false">
```

...

```
<portlet-entry name="WeatherPortlet" hidden="false" type="ref"
              parent="CustomizerVelocity" application="false" >
```

7.3 The Velocity Template

Let's have a look at the Velocity template for our first example: `tutorial-stock-quote1.vm`. It's a very simple example of displaying live stock quotes from a web service. The stock quotes are returned in a collection of quote records, called `$quotes`. Also, the column headers are in a collection of strings called `$columns`.

```
<table>
  <tr>
    <td>
      <table border="true" cellspacing="1" cellpadding="3">
        <tr>
```

```
        #foreach ($column in $columns)
            #headerCell ($column)
        #end
    </tr>

    #foreach ($quote in $quotes)
    <tr>
        #entryCell ($quote.Symbol)
        #entryCell ($quote.Price)
        #entryCell ($quote.Change)
        #entryCell ($quote.Volume)
    </tr>
    #end
</table>
</td>
</tr>
</table>
```

Accessors get the attributes: symbol, price, change and volume. Also notice that there are some macros: `#headerCell` and `#entryCell`. The macros are defined in a common [Velocimacro](#) file that is shared across Jetspeed. [Velocimacro](#) are great for defining little snippets of templates that can be included into other templates.

```
services.VelocityService.velocimacro.library = GlobalMacros.vm
```

The macros in question are defined in the `GlobalMacros.vm` file. A macro definition takes parameters. Macros can include macros:

```
#macro (formCell $label $name $value)
    #formLabel ("$label")
    #formTextField("$name" "$value")
#end
```

```
#end
```

Similarly, common snippets can simple be parsed into the template directly:

```
#parse ("/portlets/html/tree-row-browser.vm")
```

7.4 Template Resolution

Where should templates be placed in the web application? By convention, Jetspeed looks for Velocity Portlet templates in the Jetspeed deployment under `/WEB-INF/templates/vm/portlets/html` (for HTML portlets). Similarly, WML portlets would be found under `/WEB-INF/templates/vm/portlets/wml`. The resolution of portlet templates is based on several configuration files and a resolution algorithm.

The `TurbineResources.properties` is where the first part of the configuration goes:

```
services.VelocityService.file.resource.loader.path =  
    /WEB-INF/templates/vm, /WEB-INF/mytemplates/vm
```

You can specify multiple paths by comma-separating the paths. Secondly, the `JetspeedResources.properties` must be updated with the same path, but to root directory containing all templates (both JSP and VM).The list may also be comma-separated.

```
services.TemplateLocator.templateRoot=
```

```
/WEB-INF/templates, /WEB-INF/mytemplates
```

The template resolution algorithm finds the template. The resolver is a service and is pluggable. The default service has a resolution algorithm much like how the Profiler service resolves PSML resources.

Templates resources may be optionally localised. Templates are localized by placing them in sub-directories based on language and country code abbreviations. The language-code sub-directory contains one or more country-code subdirectories.

The language-code directory name is specified with an ISO-639 standard two-character language abbreviation. The country-code subdirectory is optional, and is specified with an ISO-3166 standard two-character country code abbreviation.

An example:

```
vm
|-- portlets
    |-- html
        |-- fr // french language
            |-- FR // France country-code
            |-- BE // Belgium country-code
```

NOTE: The country codes must be in upper-case

For a given locale of fr_FR, and an HTML request, the search order for a template named grenouille.vm would be:

```
vm/portlets/html/fr/FR/grenouille.vm
vm/portlets/html/fr/grenouille.vm
vm/portlets/html/grenouille.vm
vm/portlets/grenouille.vm
```

The template locator looks at the "Content Language" HTML header for locale specific settings. If there are multiple settings, all settings will be searched until the first resource is found. (This is currently not supported)

For a complete list of ISO-639 standard language abbreviations, see:

<http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt>

For a complete list of ISO-3166 standard country code abbreviations, see:

http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

7.5 Velocity Actions and the Context

Velocity Actions are Java classes, they are where you put your controlling logic for your code. Here you will do any backend processing necessary to retrieve or store dynamic information, and then populate the Velocity context with your model so that the template may display the dynamic content.

First, let's look at the code for the `TutorialStockQuoteAction1`. This portlet retrieves stock quotes from a web service. There is only one method in this Velocity action, the `buildNormalContext` method. The `StockQuoteService` comes with the Jetspeed deployment. It returns an array of `StockQuote` objects when you ask for a quote on a array of stock symbols.

```
public class TutorialStockQuoteAction1 extends VelocityPortletAction
{
    private static final String SYMBOLS = "symbols";
```

```
private static final String COLUMNS = "columns";
private static final String QUOTES = "quotes";
private static final String[] ALL_COLUMNS =
    {"Symbol", "Price", "Change", "Volume"};

protected void buildNormalContext(VelocityPortlet portlet,
    Context context,
    RunData runda)

{
    try
    {
        // Get reference to stock quote web service
        StockQuoteService service = (StockQuoteService)
            TurbineServices.getInstance().
                getService(StockQuoteService.SERVICE_NAME);

        // Retrieve portlet parameters
        String symbols = PortletConfigState.getParameter(portlet,
            runda, SYMBOLS, "IBM,MSFT,ORCL,SUNW");

        // Request stock quote(s) from the stock quote web service
        String[] symbolArray = StringUtils.stringToArray(
            symbols, ",");
        StockQuote[] quotes = service.fullQuotes(symbolArray);

        // Place appropriate objects in Velocity context
        context.put(QUOTES, quotes);
        context.put(COLUMNS, ALL_COLUMNS);
    }
    catch (Exception e)
    {
        Log.error(e);
    }
}
```

The array of stock quotes is put in the context and is made available to the template. The template can then pull out the

stock records that it needs, iterating over the array of stock quotes:

```
#foreach ($quote in $quotes)
<tr>
  #entryCell ($quote.Symbol)
  #entryCell ($quote.Price)
  #entryCell ($quote.Change)
  #entryCell ($quote.Volume)
</tr>
#end
```

Let's take a closer look at the [VelocityPortletAction](#) class, and see how the `processAction` phase is nicely handled for you by inheriting from this class. There are three important methods than you can implement in your velocity action. Remember from section [6.2 on the Portlet Life Cycle](#), the `processAction` phase occurs before the render phase. So the following methods will be called before the template is processed, allowing for you to cleanly populate the Velocity context first, depending on the [portlet mode](#).

| Method | Portlet Mode |
|------------------------------------|------------------|
| <code>buildNormalContext</code> | View |
| <code>buildConfigureContext</code> | Customize (Edit) |
| <code>buildMaximizedContent</code> | Maximize |


Using this approach you can easily customize your portlet to generate different content depending on the mode of the request. Often, the maximize content is the same as the normal context. In the default implementation of `buildConfigureContext`, the normal context is called. You can override this method to specially handle maximize mode. The `buildConfigureContext` is available if you want to provide your own customization as we will do further on in this chapter. To go with the default portal customizer provided by Jetspeed, just don't override this method.

7.6 Portlet Customization and Parameter Styles


The next portlet example illustrates editing your portlet parameters using the default Portlet Customizer. The first portlet titled “Tutorial Stock Portfolio” in this tutorial uses the default customizer. The second portlet titled “Tutorial Stock Portfolio with parameter styles“ illustrates how to enhance the parameter edit fields in the default Portlet Customizer.

To see these examples working, you will need to login as the turbine/turbine user. First let’s look again at the first portlet that doesn’t have any parameter styles so that we can see the customizer before and after.

The Default Portlet Customizer

You can customize a portlet by clicking a portlet’s customize action button.  With the default deployment, portlets cannot be customized for the anonymous user. Since the anonymous user PSML resource is shared, this is probably a good idea. But for that odd case where you need anonymous customization of portlets, it is possible. Modify the **JetspeedSecurity.properties** file:

```
services.JetspeedSecurity.actions.anon.disable=false
```

For now just logon as turbine/turbine. Click on the customize action button  for the first portlet titled “Tutorial Stock Portfolio”. You will see the Default Portlet Customizer. This customizer will display all of the non-hidden portlet-entry parameters:

Customize portlet

| | |
|----------------|---|
| Title | <input type="text" value="Tutorial Stock Portfolio"/> |
| Skin | <input type="text" value="- Default -"/> |
| Symbols | <input type="text" value="MSFT,IBM,ORCL,SUNW"/> |

List of comma-separated stock symbols

The Title and Skin parameters are always customizable. Remember from [Tutorial 6](#), all parameters that are edited are not stored back to the registry. The edited parameters are stored to the PSML resource for the current logged on user. Thus if we change the title of this portlet, it will only change for the user “turbine” and only for this one particular instance of this portlet on this page. Likewise for all other parameters. If you are logged on as a user with the admin role, you can also change the security constraint for this portlet instance. Only when logged on as a user with the admin role, a third drop-down list of security constraints is displayed. See [Tutorial 4](#) for details.

The third parameter shown above, “Symbols”, is a parameter that we defined for this portlet. It is the list of stock symbols for the portlet instance.

```
<parameter name="symbols" value="MSFT,IBM,ORCL,SUNW" type=""
    hidden="false" cachedOnName="true" cachedOnValue="true">
  <meta-info>
    <title>Symbols</title>
    <description>List of comma-separated stock symbols<
      /description>
  </meta-info>
</parameter>
```

The read-only meta-info parameters title and description are listed in the portlet customizer from the registry entry above.

The value of the parameter obviously comes from the value attribute. Now let's see how we can enhance parameter customization using Parameter Styles.

The Default Portlet Customizer with Parameter Styles

Parameter styles are custom widgets which allow you to present portlet parameter using something other than the default input text box control. These widgets can be as simple as text area control or as complex as a pop up calendar control.

The second portlet titled “Tutorial Stock Portfolio with parameter styles“ illustrates how to enhance the parameter edit fields in the default Portlet Customizer. Its action class is `TutorialStockQuoteAction2`. This portlet also retrieves stock quotes from a web service, just like the first example. The only difference is that we will show you how to use [Parameter Styles](#) to make your portlet's customization more dynamic.

Logon as turbine/turbine. Click on the customize action button  for the second portlet titled “Tutorial Stock Portfolio with parameter styles”. You will see the Default Portlet Customizer with two custom widgets:

The “Symbols” parameter has a **TextArea** parameter style, and a new parameter “Columns” has a **CheckBoxGroup** style. The columns parameter lets us customize which columns are displayed in view mode. If you don’t want to see all stock quote columns, just uncheck the column.

Looking at the registry entry, we have created several parameter styles. First there is the Symbols widget, it has a **TextArea** style:

```
<parameter name="symbols" value="MSFT, IBM, ORCL, SUNW" type="style"
    hidden="false" cachedOnName="true" cachedOnValue="true">
  <meta-info>
    <title>Symbols</title>
    <description>List of comma-separated stock symbols</description>
  </meta-info>
</parameter>
<parameter name="symbols.style" value="TextArea" hidden="true"
    cachedOnName="true" cachedOnValue="true"/>
```

The default value for all portlet instances is provided in the value attribute of the **symbols** parameter. Again, when a user

customizes this portlet, its values are not stored back to the registry but to the PSML resource for the particular portlet instance. The meta-info title and description are displayed by the customizer. The **symbols** parameter is linked to the **symbols.style** by name, and the parameter style **TextArea** is selected there.

The next parameter is the **columns** parameter, which allows you to choose which columns will be displayed in the portlet's view mode.

```
<parameter name="columns" value="Symbols,Price,Change,Volume" type="style"
    hidden="false" cachedOnName="true" cachedOnValue="true">
  <meta-info>
    <title>Columns</title>
    <description>Columns to display</description>
  </meta-info>
</parameter>
<parameter name="columns.style.items" value="Symbol,Price,Change,Volume"
    hidden="true" cachedOnName="true" cachedOnValue="true"/>
<parameter name="columns.style.layout" value="$eastwest" hidden="true"
    cachedOnName="true" cachedOnValue="true"/>
<parameter name="columns.style" value="CheckBoxGroup" hidden="true"
    cachedOnName="true" cachedOnValue="true"/>
```

The column headers strings are defined in the value attribute for the parameter style. The style is set to be a **CheckBoxGroup**. The layout of the check boxes is controlled by the layout style, which automatically aligns the check boxes vertically or horizontally using the keywords: **\$eastwest** or **\$northsouth**. The names of the check box items are defined in the **columns.style.items** parameter.

The action class is responsible for displaying the content in view mode. The action class for our portlet only handles normal (view mode) processing. It let's the base class handle default processing for maximize and customize modes.

```
protected void buildNormalContext(VelocityPortlet portlet,
```

```
Context context,
RunData rundata)
{
    try
    {
        // Get reference to stock quote web service
        StockQuoteService service = (StockQuoteService)
            TurbineServices.getInstance().
            getService(StockQuoteService.SERVICE_NAME);

        // Retrieve portlet parameters
        String symbols = PortletConfigState.getParameter(portlet,
            rundata, SYMBOLS, "IBM,MSFT,ORCL,SUNW");
        String columns = PortletConfigState.getParameter(portlet,
            rundata, COLUMNS,

                StringUtils.arrayToString(ALL_COLUMNS, ","));
        String[] selectedColumnsArray =
            StringUtils.stringToArray(columns, ",");

        // Request stock quote(s) from the stock quote web service
        String[] symbolArray = StringUtils.stringToArray(symbols, ",");
        StockQuote[] quotes = service.fullQuotes(symbolArray);

        // Place appropriate objects in Velocity context
        context.put(QUOTES, quotes);
        context.put(SELECTED_COLUMNS, selectedColumnsArray);
        context.put(COLUMNS, columns);
    }
    ...
}
```

Three objects are put into the Velocity context: \$quotes, \$columns, and \$selected-columns. These objects are then used to generate the content of the portlet in view and maximize modes. Two changes are made in the second example. First only the selected headers from the customizer are displayed:

```
#foreach ($column in $selected-columns)
#headerCell ($column)
```

```
#end
```

Secondly, only the selected column-values are displayed:

```
#foreach ($quote in $quotes)
<tr>
#if ($columns.indexOf("Symbol") >= 0) #entryCell ($quote.Symbol) #end
#if ($columns.indexOf("Price") >= 0) #entryCell ($quote.Price) #end
#if ($columns.indexOf("Change") >= 0) #entryCell ($quote.Change) #end
#if ($columns.indexOf("Volume") >= 0) #entryCell ($quote.Volume) #end
</tr>
#end
```

7.7 Action Events

Action events allow you tie events that occur on the user interface, such as a form being submitted, or a button clicked, to post back to a specific event handler on a [VelocityPortletAction](#) class. We will look at a third Velocity portlet that demonstrates action events to perform user actions. This portlet is a simple data entry form with 5 input fields. The data is stored in the session for the current user, but does not persist past the lifetime of the session.

```
<portlet-entry name="CobiJonesPortlet" hidden="false" type="ref"
    parent="Velocity" application="false">
  <meta-info>
    <title>The Cobi Jones Portlet</title>
    <description>Tutorial showing an Action Event Executing, dedicated
      to the man Cobi</description>
  </meta-info>
```



```
<parameter name="template" value="cobi-jones-form" hidden="true"/>
<parameter name="action" value="portlets.CobiJonesPortletAction"
            hidden="true" />
<media-type ref="html"/>
  <category group="Jetspeed">tutorial</category>
</portlet-entry>
```

And here is what the portlet looks like when running:

| The Cobi Jones Portlet | |
|--|---|
| First Name | <input type="text" value="Cobi"/> |
| Last Name | <input type="text" value="Jones"/> |
| Position | <input type="text" value="Midfielder"/> |
| Caps | <input type="text" value="150"/> |
| Active | <input checked="" type="checkbox"/> |
| <input type="button" value="Save Cobi"/> | |

When you press “Save Cobi”, the values on the form are passed to the Turbine (the MVC controller servlet), from where you can get the values of the input fields into your Java action class. Let’s look at our action event:

```
public void doUpdate(RunData rundata, Context context) throws Exception
{
    Player player = (Player)rundata.getUser().getTemp(PLAYER);
    if (null == player)
    {
        player = createNewPlayer();
        rundata.getUser().setTemp(PLAYER, player);
    }
}
```

```

String cobj = rundata.getParameters().getString(INPUT_FIRST_NAME);
String jones = rundata.getParameters().getString(INPUT_LAST_NAME);
if (!cobj.equalsIgnoreCase("Cobi") ||
    !jones.equalsIgnoreCase("Jones"))
{
    rundata.getRequest().setAttribute(COBI_ERROR,
        "Hey now, you cant substitute Cobi with "
        + cobj + " " + jones + "!");
}
player.setFirstName(cobj);
player.setLastName(jones);
player.setPosition(rundata.getParameters().getString(INPUT_POSITION));
player.setCaps(rundata.getParameters().getInt(INPUT_CAPS));
player.setActive(rundata.getParameters().getBoolean(INPUT_ACTIVE));

rundata.getUser().setTemp(PLOYER, player);
}

```

This code above is from the `CobiJonesPortletAction`. This is the action event. As we found out in [previous section](#), Velocity action have three standard methods: `BuildNormalContext`, `BuildConfigureContext`, and `BuildMaximizedContext`. In addition, you can zero or more action events. Action events are how your portlet code reacts to user interface interactions. In our example, we have a simple data entry form. The “Save Cobi” button is a submit button which posts the contents of the HTML form back to the portlet.

We can get the contents of the form from the `rundata` parameter passed into our action event. Jetspeed puts all the form parameters into a parameter parser object. You can get strings and other data types and then set the values into the portlet session.

```

String cobj = rundata.getParameters().getString(INPUT_FIRST_NAME);
...
player.setActive(rundata.getParameters().getBoolean(INPUT_ACTIVE));

```

In this simple example state is kept in the servlet session. In future examples we can store the values to the database.

Jetspeed provides methods to get and set objects into the session:

```
rundata.getUser().setTemp(PLAYER, player);  
..  
Player player = (Player)rundata.getUser().getTemp(PLAYER);
```

Let's look at how to setup form posting from the Velocity template (`cobi-jones-form.vm`):

```
<form method="post"  
    action="$jslink.setAction("portlets.CobiJonesPortletAction")">  
...  
    <input type="submit" name="eventSubmit_doUpdate" value="Save Cobi"/>
```

First, the form must link back to the current portal page. The `$jslink` tool handles this. Next the action must be specified. We specify the action to be our portlet's action with the `setAction` parameter on the action attribute of the form element. The submit input requires a convention for finding the correct event on the specified action: you must prefix the name attribute of the action with the string `"eventSubmit_"` and then enter the name of the action event method. In this case, it is `"eventSubmit_doUpdate"`.

The input fields are specified using standard Jetspeed data entry macros defined in

```
<tr>  
    #formCell ("First Name" "firstname" $player.FirstName)  
</tr>  
<tr>  
    #formCell ("Last Name" "lastname" $player.LastName)  
</tr>  
<tr>
```

```

        #formCell ("Position" "position" $player.Position)
    </tr>
    <tr>
        #formCell ("Caps" "caps" $player.Caps)
    </tr>
    <tr>
        #formCheckBox2 ("Active" "active" $player.Active)
    </tr>

```

The first parameter to `#formCell` is the input caption, the second is the input's name attribute, and the third is the player object from the Velocity context. Ensure that the input's name attribute is the same as in your Java action event:

```

        #formCell ("Position" "position" $player.Position)

public static final String INPUT_POSITION = "position";

        player.setPosition(rundata.getParameters().getString(INPUT_POSITION));

```

Finally, the `doUpdate` action event shows how to do some basic exception handling by passing an exception string from the action event to the `BuildNormalContext` method.

```

if (!cobi.equalsIgnoreCase("Cobi") ||
    !jones.equalsIgnoreCase("Jones"))
{
    rundata.getRequest().setAttribute(COBI_ERROR,
        "Hey now, you cant substitute Cobi with "
        + cobi + " " + jones + "!");
}

```

We set the error message text into a servlet request attribute (from [the Servlet API](#)), and then later on in the pipeline, in the

BuildNormalContext method, we check for existence of the error message text and display an error message in case an exception occurred in the action event.

```
// make sure they don't sub Cobi!  
String cobiError = (String)rundata.getRequest().getAttribute(COBI_ERROR);  
if (null != cobiError)  
{  
    context.put(COBI_ERROR, cobiError);  
}
```

This error message is then retrieved from the context in the template:

```
#if ($cobierror)  
<tr>  
    <td colspan="2">  
        <table bgcolor="red">  
            <tr>  
                <td>  
                    $cobierror  
                </td>  
            </tr>  
        </table>  
    </td>  
</tr>  
#end
```

7.8 Deploy

To deploy the system, type:

```
ant deploy          (or hotdeploy)
```

Use hotdeploy if you have already deployed the system once. This simply saves some time in packaging the JPortal deployment. Next point your browser at:

<http://localhost:8080/jportal/portal>

You should see the new Velocity portlets in the default pane:

Tutorial 8 – JSP Portlet

Tutorial 8 introduces the JSP Portlet. This section covers:

1. Introduction to JSP
2. JSP Portlets in the Registry
3. The JSP Template
4. Template Resolution
5. JSP Actions
6. Portlet Customization and Parameter Styles
7. JSP Actions
8. Deploy

Let's get started. From the JPortal distribution root directory, type:

```
ant tutorial-8
```

Recommend bringing up these files in your editor:

1. `webapp/WEB-INF/conf/t8-portlets.xreg`
2. `src/java/com/bluesunrise/jportal/modules/actions/portlets/TutorialStockQuoteAction8.java`
3. `webapp/WEB-INF/templates/jsp/portlet/html/TutorialStockQuote8.jsp`

since we will reference them in tutorial 8.

8.1 Introduction to JSP

[JSP](#) (Java Server Pages) is a technology based on the Java language and enables the development of dynamic web sites. JSP was developed by Sun Microsystems to allow server side development. JSP files are HTML files with special Tags containing Java source code that provide the dynamic content. JSP is easy to learn and allows developers to quickly produce web sites and applications in an open and standard way. JSP offers a robust platform for web development. Main reasons to use JSP:

1. Multi platform
2. Component reuse by using JavaBeans and EJB.
3. Separation of content from code.

You can take one JSP file and move it to another platform, web server or JSP Servlet engine.

Jetspeed has a JSP templating service built directly into the Jetspeed engine. Similar to Velocity, you can base your

portlets on JSP. This means that your portlets will also follow the MVC design pattern, separating content from code. The examples in [Tutorial 5](#) and [Tutorial 6](#) were useful for you to learn the Portlet interface. However, overriding the `getContent` method of the portlet interface is not good practice. We recommend abstracting the content generation phase by basing your portlets on one of the MVC-based portlets provided in the Jetspeed distribution; such as [VelocityPortlet](#), [JSPPortlet](#), [XSLTPortlet](#), [RSSPortlet](#) or [HTMLPortlet](#).

A JSP portlet is made up of the typical MVC components:

| MVC Component | JSP Component |
|---------------|---|
| Model | Java Objects put in as request attributes |
| View | Template |
| Controller | Your JSP Action |

The controller is your JSP Action class. The base [JspPortlet](#) class should rarely have to be modified. Your view is a JSP template, which will generate the content of your portlet by pulling out dynamic model information from the request attributes. In addition to the standard JSP variables (request, response, session, etc), the following variables are available in the model:

| Variable | Type | Description |
|----------------------|-------------------------|--------------------------------------|
| <code>runData</code> | RunData | Reference to request run data object |
| <code>js_peid</code> | String | Portlet unique identifier |
| <code>link</code> | JspLink | Instance of JspLink object |
| <code>portlet</code> | Portlet | Reference to this portlet object |

These can be retrieved from request attributes as in the example below:


```
String jspeid = (String) request.getAttribute("js_peid");
```

Also, your JSP template can make use of any of the available tags from the Jetspeed tag library. See **JSP1_1andJetspeedTagLibrary** portlet for comprehensive list of examples.

The getContent method of the [JspPortlet](#) should never be edited. All content is generated by the template, as designed in our MVC design pattern.

The Life Cycle phases of a portlet are also enhanced with the JSP portlet.

| Phase | Method |
|---------------|--------------------------------------|
| Init | Init |
| ProcessAction | JSP Portlet Action and Action Events |
| Render | Template is called by JSP Portlet |
| Destroy | --none-- |

JSP portlets are really about dynamic content. If you have static content, there is no real benefit to using a JSP portlet; take a look at one of the static content generation portlets such as the [HTMLPortlet](#) instead. The basic function of JSP is really very simple, it substitutes live Java objects into a JSP template. There is an online tutorial with great examples [here](#).

8.2 JSP Portlets in the Registry

JSP Portlets are defined like any other portlet: in the portlet registry. In tutorial 8 we define one new portlet:

```
<portlet-entry name="TutorialStockQuote8" hidden="false" type="ref"
    parent="JSP" application="false">
...

```

```
        <parameter name="template" hidden="true"
                value="TutorialStockQuote8.jsp" />
        <parameter name="action" hidden="true"
                value="portlets.TutorialStockQuoteAction8" />
...
</portlet-entry>
```

When defining a JSP portlet, there are two required parameters: the **template** and the **action**. The template defines the JSP template which will generate the portlet content. It is your MVC View component. The action is the controller, and it has several responsibilities including handling action events and populating the request attributes. The templates should be placed in the portlets subdirectory of one of your JSP template paths. The action is placed in the module path, conventionally under the portlets directory of the root actions directory.

8.3 The JSP Template

Let's have a look at the JSP template from our example: `TutorialStockQuote8.jsp`. It's a very simple example of displaying live stock quotes from a web service. The stock quotes are returned in a collection of quote records and stored in request attribute called "quotes". Also, the column headers are in a collection of strings and stored in a request attribute called "columns".

```
<%@ taglib uri='/WEB-INF/templates/jsp/tld/template.tld' prefix='jetspeed' %>

<%@ page import = "org.apache.turbine.util.Log" %>
<%@ page import = "org.apache.jetspeed.webservices.finance.stockmarket.StockQuote" %>

<%
try{
    StockQuote[] quotes = (StockQuote[]) request.getAttribute("quotes");
    String[] columns = (String[]) request.getAttribute("columns");
    String jspeid = (String) request.getAttribute("js_peid");
}%>
```

```

<FORM METHOD="POST">
  <INPUT TYPE="hidden" NAME="js_peid" VALUE="<%=jspeid%>">
  Enter symbol(s) separated with commas: <input name="symbols" type="TEXT"><INPUT TYPE="SUBMIT"
NAME="refresh" VALUE="Get Quotes">
</FORM>
<table>
  <tr>
    <td>
      <table border="true" cellspacing="1" cellpadding="3">
        <tr>
          <%for (int i = 0; columns != null && i < columns.length; i++) {%>
            <TH><%=columns[i]%></TH>
          <%}%>
        </tr>

          <%for (int j = 0; quotes != null && j < quotes.length; j++) {%>
        <tr>
          <TD><%=quotes[j].getSymbol()%></TD>
          <TD><%=quotes[j].getPrice()%></TD>
          <TD><%=quotes[j].getChange()%></TD>
          <TD><%=quotes[j].getVolume()%></TD>
        </tr>
          <%}%>
        </table>
      </td>
    </tr>
  </table>
  <%> catch (Exception e) {
    Log.error(e);
    return;
  }%>

```

Also, you will note that one additional request attribute is retrieved: **jspeid**. The value of this attribute is used to define a hidden field “js_peid” which uniquely identifies this portlet when its form is submitted via the “refresh” button:

```

<INPUT TYPE="hidden" NAME="js_peid" VALUE="<%=jspeid%>">

```

Adding the above field in your template assures that when multiple instances of the portlet are used on the same pane, the refresh will only apply to particular portlet instance. This concept applies to Velocity portlets as well.

8.4 Template Resolution

Where should templates be placed in the web application? By convention, Jetspeed looks for JSP Portlet templates in the Jetspeed deployment under `/WEB-INF/templates/jsp/portlets/html` (for HTML portlets). Similarly, WML portlets would be found under `/WEB-INF/templates/jsp/portlets/wml`. The resolution of portlet templates is based on several configuration files and a resolution algorithm.

The `TurbineResources.properties` is where the first part of the configuration goes:

```
services.JspService.templates = /WEB-INF/templates/jsp, /WEB-INF/mytemplates/jsp
```

You can specify multiple paths by comma-separating the paths. Secondly, the `JetspeedResources.properties` must be updated with the same path, but to root directory containing all templates (both JSP and VM). The list may also be comma-separated.

```
services.TemplateLocator.templateRoot=  
/WEB-INF/templates, /WEB-INF/mytemplates
```

The template resolution algorithm finds the template. The resolver is a service and is pluggable. The default service has a resolution algorithm much like how the Profiler service resolves PSML resources.

Templates resources may be optionally localised. Templates are localized by placing them in sub-directories based on language and country code abbreviations. The language-code sub-directory contains one or more country-code subdirectories.

The language-code directory name is specified with an ISO-639 standard two-character language abbreviation. The country-code subdirectory is optional, and is specified with an ISO-3166 standard two-character country code abbreviation.

An example:

```
jsp
|-- portlets
    |-- html
        |-- fr                // french language
            |-- FR           // France country-code
            |-- BE           // Belgium country-code
```

NOTE: The country codes must be in upper-case

For a given locale of fr_FR, and an HTML request, the search order for a template named grenouille.jsp would be:

```
jsp/portlets/html/fr/FR/grenouille.jsp
jsp/portlets/html/fr/grenouille.jsp
jsp/portlets/html/grenouille.jsp
jsp/portlets/grenouille.jsp
```

The template locator looks at the "Content Language" HTML header for locale specific settings. If there are multiple settings, all settings will be searched until the first resource is found. (This is currently not supported)

For a complete list of ISO-639 standard language abbreviations, see:

<http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt>

For a complete list of ISO-3166 standard country code abbreviations, see:

http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

8.5 JSP Portlet Actions

JSP Portlet Actions are Java classes, they are where you put your controlling logic for your code. Here you will do any backend processing necessary to retrieve or store dynamic information, and then populate the JSP request attributes with your model so that the template may display the dynamic content.

First, let's look at the code for the `TutorialStockQuoteAction8`. This portlet retrieves stock quotes from a web service. There is only one method in this JSP portlet action, the [buildNormalContext](#) method. The [StockQuoteService](#) comes with the Jetspeed deployment. It returns an array of [StockQuote](#) objects when you ask for a quote on a array of stock symbols.

```
public class TutorialStockQuoteAction8 extends JspPortletAction
{
    private static final String SYMBOLS = "symbols";
    private static final String COLUMNS = "columns";
    private static final String QUOTES = "quotes";
    private static final String[] ALL_COLUMNS = {"Symbol", "Price", "Change", "Volume"};

    /**
     * Build the normal state content for this portlet.
     *
     * @param portlet The jsp-based portlet that is being built.
     */
}
```

```
    * @param rundata The turbine rundata context for this request.
    */
protected void buildNormalContext(Portlet portlet,
                                  RunData rundata)
{
    try
    {
        // Get reference to stock quote web service
        StockQuoteService service = (StockQuoteService) TurbineServices.getInstance().
            getService(StockQuoteService.SERVICE_NAME);

        // Retrieve portlet parameters
        String symbols = (String) PortletSessionState.getAttributeWithFallback(portlet, rundata,
SYMBOLS);

        // Request stock quote(s) from the stock quote web service
        String[] symbolArray = StringUtils.stringToArray(symbols, ",");
        StockQuote[] quotes = service.fullQuotes(symbolArray);

        // Place appropriate objects in jsp context
        rundata.getRequest().setAttribute(QUOTES, quotes);
        rundata.getRequest().setAttribute(COLUMNS, ALL_COLUMNS);
    }
    catch (Exception e)
    {
        Log.error(e);
    }
}
}
```

The above example is very similar to the one used in [Tutorial 7](#) with the exception that it allows the user to temporarily specify stock quote symbols without having to customize. Note that the symbols are retrieved by the portlet action as follows:

```
String symbols = (String) PortletSessionState.getAttributeWithFallback(portlet, rundata, SYMBOLS);
```

The above method call retrieves value of the symbols parameter using the following algorithm:

1. If the parameter is present in the request, get it from there and store it in the session, otherwise
2. If the parameter is present in the session, get it from the session, otherwise,
3. If the parameter is present in portlet instance attributes (values defined for the portlet in user psml), get it from there, otherwise,
4. Get it from the portlet configuration (values defined for the portlet in registry)

So when the user “overrides” the stock symbols using the input field provided within the body of the portlet, the portlet will temporarily display stock quotes for those symbols until:

1. The user logs off and logs back in
2. The user clicks the “Get Quotes” button with empty input field
3. The user customizes the portlet

Now, let’s take a closer look at the [JspPortletAction](#) class, and see how the processAction phase is nicely handled for you by inheriting from this class. There are three important methods that you can implement in your velocity action. Remember from section [6.2 on the Portlet Life Cycle](#), the processAction phase occurs before the render phase. So the following methods will be called before the template is processed, allowing for you to cleanly populate the JSP request attributes first, depending on the [portlet mode](#).

| Method | Portlet Mode |
|------------------------------------|------------------|
| <code>BuildNormalContext</code> | View |
| <code>buildConfigureContext</code> | Customize (Edit) |
| <code>buildMaximizedContent</code> | Maximize |

Using this approach you can easily customize your portlet to generate different content depending on the mode of the request. Often, the maximize content is the same as the normal context. In the default implementation of `buildConfigureContext`, the normal context is called. You can override this method to specially handle maximize mode. The `buildConfigureContext` is available if you want to provide your own customization as we will do further on in this chapter. To go with the default portal customizer provided by Jetspeed, just don't override this method.

8.6 Portlet Customization and Parameter Styles

Please review the following [section](#) in [Tutorial 7](#) on the process of portlet customization and usage of parameter styles. You will find that Velocity and JSP Portlets work identically in that respect. You will also find additional information about parameter styles in [Tutorial 12](#).

8.7 JSP Events

Action events allow you tie events that occur on the user interface, such as a form being submitted, or a button clicked, to post back to a specific event handler on a [JspPortletAction](#) class.

Although our example uses a single action event (“refresh”), it is possible to use multiple events per portlet. In short, declaring an action event involves:

1. Specifying the name of your `JspPortletAction` as using a hidden input field called “action”,
2. Defining a submit button named with the following convention: “`eventSubmit_{action-name}`”,
3. Declaring an event handler in your `JspPortletAction` called `{action-name}`.

For example:

```
<form method="post"
  action="<jetspeed:dynamicUri/>"
  <input type="hidden" name="js_peid" value="<%=jspeid%>">
  <input type="hidden" name="action" value=" portlets.myJspPortletAction "/>
  <input type="submit" name="eventSubmit_doUpdate" value="Save"/>
```

```
public void doUpdate(RunData runda, Portlet portlet) throws Exception
{
    // action logic goes here
}
```

You may refer to the following [section](#) in [Tutorial 7](#) for an example of using action events.

8.8 Deploy

To deploy the system, type:

```
ant deploy          (or hotdeploy)
```

Use hotdeploy if you have already deployed the system once. This simply saves some time in packaging the JPortal deployment. Next point your browser at:

<http://localhost:8080/jportal/portal>

You should see the new JSP portlet in the default pane:

Tutorial 9 – DatabaseBrowser Portlet

Tutorial 9 introduces the Database Browser Portlet. This section covers:

1. Configuration in the Registry
2. Linking to Actions
3. Implementing the Action Events
4. Advanced Parameters
5. Deploy

Let's get started. From the JPortal distribution root directory, type:

```
ant tutorial-9
```

Recommend bringing up these files in your editor:

1. `webapp/WEB-INF/conf/t9-portlets.xreg`
2. `src/java/com/bluesunrise/jportal/modules/actions/portlets/
TutorialCoffeesAction.java`
3. `src/java/com/bluesunrise/jportal/modules/actions/portlets/`

```
CoffeesBrowserAction.java
```

4. `webapp/WEB-INF/psml/user/anon/html/default.psml`
5. `webapp/WEB-INF/psml/user/turbine/html/default.psml`
6. `webapp/WEB-INF/templates/vm/portlet/html/coffees-form.vm`
7. `webapp/WEB-INF/templates/vm/portlet/html/coffees-browser.vm **`

** not actually used in demo, but you could extend this later on

since we will reference them in tutorial 9. The examples are found under the “Advanced Tutorials” menu selection.

9.1 Configuration in the Registry

The Database Browser Portlet uses an SQL statement and JDBC to create a database browser form inside of a portlet. The browser runs off the Torque connection pools and data sources configured in your `Torque.properties` file. The standard functionality of the browser doesn't require any coding. The portlet is configured with parameters entered in the portlet registry. The minimal parameters and attributes required are highlighted below:

```
<portlet-entry name="TutorialCoffeesBrowser" hidden="false" type="ref"
parent="DatabaseBrowserPortlet" application="false">
<meta-info>
  <title>Coffees Browser</title>
  <description>Browses Over the Coffees Table</description>
</meta-info>
<parameter name="sql" value="select * from coffees"
  hidden="false"/>
<parameter name="windowSize" value="15" hidden="false"/>
...
</portlet-entry>
```

You must derive the **portlet-entry** from the **DatabaseBrowserPortlet**, which is deployed with the Jetspeed distribution. This gives you all the base functionality. The **sql** parameter provides the query string. In our example, the string is not hidden so it can be modified by the user. The **windowSize** parameter controls how many rows to display in the portlet. That is all you need. The portlet generates the rest for you:

| Coffees Browser | | | | | | |
|-----------------|-------------------------|-------------|-------|-------|-------|-------------|
| COFFEE ID | COFFEE NAME | SUPPLIER ID | PRICE | SALES | TOTAL | |
| 1 | ColombianGrade | 5 | 7.99 | 1 | 2 | Edit Delete |
| 2 | KonaGrade | 6 | 7.99 | 1 | 2 | Edit Delete |
| 3 | FrenchRoastGrade | 7 | 7.99 | 1 | 2 | Edit Delete |
| 4 | HazelNutGrade | 8 | 7.99 | 1 | 2 | Edit Delete |
| 5 | VanillaGrade A | 9 | 7.99 | 12 | 2 | Edit Delete |
| 6 | JavaGrade | 10 | 7.99 | 1 | 2 | Edit Delete |
| 7 | IndonesianGrade | 11 | 7.99 | 1 | 2 | Edit Delete |
| 8 | OotyGrade | 13 | 7.99 | 12 | 220 | Edit Delete |
| 9 | KenyanGrade | 2 | 7.99 | 1 | 2 | Edit Delete |
| 10 | JoeGrade | 3 | 7.99 | 1 | 2 | Edit Delete |
| 11 | CantThinkOfAnymoreGrade | 44 | 7.99 | 1 | 2 | Edit Delete |

Add

Refresh

The database browser portlet is a standard [Velocity Portlet](#), and it has an action, template, and customizer template associated with it. This three parameters must be entered. You can select default values or implement your own. In our example, we implement our own action but use the default template and customizer template. You may want to override these to customize the layout of the browser. The default templates can be used as the basis for writing your own templates.

```
<parameter name="template" value=" database-browser-portlet "
  hidden="true"/>
<parameter name="customizeTemplate"
  value="database-browser-customize" hidden="true"/>
<parameter name="action" value="portlets.CoffeesBrowserAction"
```

```
hidden="true"/>
```

Our action class extends the default action class to provide the added functionality of refreshing the browser content by passing a query parameter in the URL string. The base class handles everything else. Here we are overriding the standard Velocity Portlet content generation method, [buildNormalContext](#):

```
protected void buildNormalContext( VelocityPortlet portlet,
                                   Context context,
                                   RunData rundata )
{
    String refresh =
        rundata.getParameters().getString(BROWSER_COMMAND);
    if (refresh != null && refresh.equals(BROWSER_REFRESH))
    {
        this.clearDatabaseBrowserIterator(portlet, rundata);
    }
    super.buildNormalContext(portlet, context, rundata);
}
```

When we find the refresh command, the browser content is re-queried. By default, the browser will not refresh its content unless explicitly invalidated.

Another useful method to overwrite is the `getQueryString()` method. The tutorial example does not do so, but here is an example of how it could be done. This method provides you with a way to generate the query string dynamically. It is called whenever the browser is invalidated, and needs to re-query the database in order to generate a new result set.

```
public String getQueryString(RunData rundata, Context context)
{
    String sql = null;
    try
    {
        sql = PortletConfigState.getConfigParameter(portlet,
                                                    Constants.SQL_QUERY, null);
    }
}
```

```
        SomeObject so = (SomeObject)
            SessionHelper.getSessionAttribute(rundata,
                Constants.SOME_OBJECT, null);

        if(so == null)
        {
            throw new Exception("Failed to get Object from session");
        }

        List parameters = new ArrayList();
        parameters.add(so.getName());
        parameters.add(so.getCity());
        super.setSQLParameters(parameters);

    }
    catch (Exception e)
    {
        Log.error(e);
    }
    return sql;
}
```

Your SQL string in the registry would then need to have JDBC parameters:

```
<parameter name="sql"
    value="select name, city, state from people where name=? and city=?"
    hidden="false"/>
```

9.2 Linking to Actions

Database browsers can be used in combination with edit forms to create a complete database maintenance portlet application. Links can be made to another pane on the current page, or to another page in the portal.

A Database Browser portlet supports row links and table links. Row links are applied to a particular database row in the browser. Table links are actions that do not apply to any particular row. Using the default template, the row links are displayed at the end of each row as shown here we have two links: Edit, Delete.



| COFFEE ID | COFFEE NAME | SUPPLIER ID | PRICE | SALES | TOTAL | |
|-----------|----------------|-------------|-------|-------|-------|-------------|
| 1 | ColombianGrade | 5 | 7,99 | 1 | 2 | Edit Delete |

The table links are displayed at the bottom of the portlet. Here we have one link: Add.

Add

Links are configured in the registry. The **row-link-ids** parameter is a comma separated list of localised string ids. These ids are looked up in the resource bundle, and displayed as hyperlinks. The **row-link-types** determine the type of link. They are also comma separated. Valid values are:

1. pane
2. psml

The pane link will go to another pane in your current PSML resource. Here you can place an edit form for entering the values of the selected record.

The **row-link-types** specifies the name of the resource. With a pane link type, it identifies the name of the pane. The pane name can also be a portlet. When used with the psml link type, it specifies the name of a PSML resource.

```
<parameter name="row-link-ids" value="EDIT,DELETE" hidden="true"/>
<parameter name="row-link-types" value="pane,pane" hidden="true"/>
<parameter name="row-link-targets" value="CoffeesForm,CoffeesForm"
```



```
hidden="true" />
```

```
...
```

To add a new record to the database, the Add link is provided using table links. The **table-link-ids** parameter is a comma separated list of localised string ids. These ids are looked up in the resource bundle, and displayed as hyperlinks. The **table-link-types** determine the type of link. They are also comma separated. Valid values are:

3. pane
4. psml

The pane link will go to another pane in your current PSML resource. Here you can place an edit form for entering the values of a new record.

The **table-link-types** specifies the name of the resource. With a pane link type, it identifies the name of the pane. The pane name can also be a portlet. When used with the psml link type, it specifies the name of a PSML resource.

```
<parameter name="table-link-ids" value="ADD" hidden="true" />
<parameter name="table-link-types" value="pane" hidden="true" />
<parameter name="table-link-targets" value="CoffeesForm"
  hidden="true" />
```

9.3 Implementing the Action Events

The action events link to a data entry form for entering Coffee records. The form supports Add, Edit and Delete modes. Here we see edit mode:

| Coffee | |
|-----------------|----------------|
| Brand of Coffee | ColombianGrade |
| Supplier Id | 5 |
| Unit Price | 7.99 |
| Sales Tax | 1 |
| Total Price | 2 |

Save Cancel

This portlet is also a Velocity Portlet. Like all Velocity portlets, it has a templates and an action.

```
<parameter name="template" value="coffees-form" hidden="true"/>
<parameter name="action" value="portlet.TutorialCoffeesAction" />
```

The form posts back to itself by default.

```
<form method="post" action="$jslink.setAction("portlet.TutorialCoffeesAction")">
```

An action event is connected to the button:

```
<input type="submit" name="eventSubmit_doUpdate" value="Save"/>
```

Form values are entered with the standard Jetspeed data entry macros. See the tutorial on [Action Events](#) for more information on programming data entry forms and with Velocity macros and input fields.

```
<tr>
  #formCell ("Brand of Coffee" "coffeeName" $!coffee.CoffeeName)
</tr>
<tr>
  #formCell ("Supplier Id" "supplierId" $!coffee.SupplierId)
</tr>
<tr>
  #formCell ("Unit Price" "price" $!coffee.Price)
</tr>
<tr>
  #formCell ("Sales Tax" "sales" $!coffee.Sales)
</tr>
<tr>
  #formCell ("Total Price" "total" $!coffee.Total)
</tr>
```

Finally we have an action event (**TutorialCoffeesAction**) that stores the row to the database.

```
public void doUpdate(RunData rundata, Context context) throws Exception
{
    Coffees coffee = null;
    Connection con = null;

    try
    {
        con = Torque.getConnection();

        coffee = (Coffees)rundata.getUser().getTemp(SESSION_COFFEE);

        if(coffee == null)
        {
```

```
        Log.error(NO_CURRENT_REC);
        rundata.setMessage(NO_CURRENT_REC);
        return;
    }

    rundata.getParameters().setProperties(coffee);

    validate(coffee);

    coffee.save(con);

    con.commit();

    returnToBrowser(rundata, true);
}
catch (Exception e)
{
    Log.error("error in saving coffee: " + e);
    rundata.setMessage(e.toString());
    if (con != null)
    {
        con.rollback();
    }
}
finally
{
    try
    {
        Torque.closeConnection(con);
    }
    catch (Exception e)
    {}
}
}
```

The update method is used by both Add and Update modes. Take note of the usage of Torque's object-relational programming model. The objects can store themselves. The Coffees objects were automatically generated for you as a task in the tutorial's ant build. First we get a connection:

```
con = Torque.getConnection();
```

then get the object from the session, which was placed there when the link was clicked on from the database browser.

```
coffee = (Coffees)rundata.getUser().getTemp(SESSION_COFFEE);
```

The parameter parser will automatically populate any bean, as long as the names of the inputs on the HTML form match (case-insensitive) the names of the properties on the bean.

```
rundata.getParameters().setProperties(coffee);
```

We then validate and save the input, finally committing it:

```
validate(coffee);  
  
coffee.save(con);  
  
con.commit();
```

If there is an error, we store the error message for later retrieval from our template:

```
rundata.setMessage(e.toString());
```

```
#if ($data.Message)  
<tr>  
  <td colspan="2">  
    <table bgcolor="red">
```

```

        <tr>
            <td>
                $data.Message
            </td>
        </tr>
    </table>
</td>
</tr>
#end

```

Finally, we redirect to the browser on success and invalidate the browser to refresh and pick up the modifications:

```

JetspeedLink link = JetspeedLinkFactory.getInstance(rundata);
DynamicURI duri = link.getPaneByName("TutorialCoffeesBrowser");
if (refresh)
{
    duri.addQueryData(CoffeesBrowserAction.BROWSER_COMMAND,
        CoffeesBrowserAction.BROWSER_REFRESH);
}
rundata.setRedirectURI(duri.toString());
JetspeedLinkFactory.putInstance(link);

```

9.4 Advanced Parameters

Database Connection Pool

By default, the database browser portlet uses the default Torque data source and connection pool. To select another connection pool, add the **poolname** parameter to your portlet's registry entry:

```

<!-- to use an alternate torque pool, set this parameter -->

```

```
<parameter name="poolname" value="otherpool" hidden="true"/>
```

The name of the pool must match the name of the data source in the `Torque.properties` file. See the [Torque documentation](#) for details on configuring data sources.

User Objects

There may be times when you need to attach additional information to each row in the browser. For example, you are implementing a tree view over a joined database query, and you need to track whether a row is expanded or contracted to show the detail lines. The state of each row can be attached to the row with User Objects. A user object can be any Java class. The user object can also be accessed in your Velocity template

The class name of the user object is specified with the `user-object-types` parameter, a comma separated list of Java class names. For each row in result set, a new class of this type is created and attached to the browser row.

```
<parameter name="user-object-types"  
value="com.bluesunrise.jportal.modules.actions.portlets.ExampleBrowserItem" hidden="true"/>  
  
<parameter name="user-object-names" value="userObject" hidden="true"/>
```

9.5 Deploy

To deploy the system, type:

```
ant deploy          (or hotdeploy)
```

Use hotdeploy if you have already deployed the system once. This simply saves some time in packaging the JPortal deployment. Next point your browser at:

<http://localhost:8080/jportal/portal>

The Tutorial 9 examples are found under the “Advanced Tutorials” menu selection.

Tutorial 10 – XSLT Portlet

Tutorial 10 introduces the XSLT Portlet. This section covers:

1. Configuration in the Registry
2. The Transform
3. Deploy

Let’s get started. From the JPortal distribution root directory, type:

```
ant tutorial-10
```

Recommend bringing up these files in your editor:

1. `webapp/WEB-INF/conf/t7-portlets.xreg`
2. `webapp/WEB-INF/psml/user/anon/html/default.psml`

3. `webapp/WEB-INF/psml/user/turbine/html/default.psml`

since we will reference them in tutorial 10

9.1 Configuration in the Registry

The XSLT Portlet ...

9.2 The Transform

TODO: Write this.

10.3 Deploy

To deploy the system, type:

```
ant deploy      (or hotdeploy)
```

Use hotdeploy if you have already deployed the system once. This simply saves some time in packaging the JPortal deployment. Next point your browser at:

<http://localhost:8080/jportal/portal>

Tutorial 11 – RSS Portlet

TODO: write this

Tutorial 12 – Parameter Styles

Parameter styles are custom widgets which allow you to present portlet parameter using something other than the default input text box control. These widgets can be as simple as text area control or as complex as a pop up calendar control. In this tutorial, you will learn how to:

1. Write a simple parameter style
2. Modify the stock quote portlet from the VelocityPortlet tutorial to use parameter styles

Let's get started. From the JPortal distribution root directory, type:

```
ant tutorial-12
```

Recommend bringing up these files in your editor:

1. `webapp/WEB-INF/conf/t12-portlets.xreg`
2. `webapp/WEB-INF/conf/templates/vm/parameters/html/InputWithHelp.vm`
3. `webapp/WEB-INF/conf/templates/vm/portlets/html/tutorial-stock-quote.vm`
4. `webapp/src/java/com/bluesunrise/jportal/modules/actions/portlets/TutorialStockQuoteAction.java`

since we will refer to them in Tutorial 12.

12.1 Parameter styles architecture

Parameter styles have been implemented using the Turbine [module](#) mechanism. For those of you not familiar with Turbine, [modules](#) are webapp resources (screens, actions, layouts, navigations, etc) which may be loaded dynamically based on a predefined module search path.

A Portlet parameter is defined in a registry entry as having a custom presentation style by setting its type attribute to **"style"** and then adding additional portlet parameters to define the style name and any style options. The additional parameters are linked to a parameter via the parameter name.

Popup Calendar Date Style

In the example below, we have a widget named **"date"** whose style is **"PopupCalendar"**, and the style format string is **"mm/dd/yyyy"**.

```
<parameter name="date" value="" type="style" hidden="false">
  <meta-info>
    <title>Date</title>
    <description>Date with popup calendar. Format pattern: mm/dd/yyyy
  </description>
  </meta-info>
</parameter>

<parameter name="date.style" value="PopupCalendar" hidden="true"/>
<parameter name="date.style.format" value="mm/dd/yyyy" hidden="true"/>
```

Date Style with Popup Calendar

Velocity Parameter Style

In the example below, we have a widget named **"password"** whose style is **"VelocityParameterPresentationStyle"**. This is a very flexible parameter style, which allows you to reuse a velocity template to format the presentation of your widget. Velocity Parameter Style templates should be placed in the Jetspeed deployment under **/WEB-INF/templates/vm/parameters/html** (for HTML portlets). Similarly, WML portlets would be found under **/WEB-INF/templates/vm/parameters/wml**. The resolution of portlet templates is based on several configuration files and a resolution algorithm. See the section on [Template Resolution](#) for more details.

```
<parameter name="password" value="secret" type="style" hidden="false"/>
<parameter name="password.style"
           value="VelocityParameterPresentationStyle" hidden="true"/>
<parameter name="password.style.template" value="Password.vm"
           hidden="true"/>
```

Velocity Parameter Style

JSP Parameter Style

In the example below, we have a widget named **"password"** whose style is **"JspParameterPresentationStyle"**. This is a very flexible parameter style, which allows you to reuse a JSP template to format the presentation of your widget. JSP Parameter Style templates should be placed in the Jetspeed deployment under **/WEB-INF/templates/jsp/parameters/html** (for HTML portlets). Similarly, WML portlets would be found under **/WEB-INF/templates/jsp/parameters/wml**. The resolution of portlet templates is based on several configuration files and a resolution algorithm. See the section on [Template Resolution](#) for more details.

```
<parameter name="password" value="secret" type="style" hidden="false"/>
<parameter name="password.style" value="JspParameterPresentationStyle" hidden="true"/>
<parameter name="password.style.template" value="Password.jsp" hidden="true"/>
```

JSP Parameter Style

```
## InputWithHelp.vm
#set ($src = $parms.get("src")) ## Retrieve the "src" parameter from the $parms map
<input type="text" name="$name" value="$value" size="30" disabled>

```

```
<parameter name="about" value="About Bluesunrise Stock Quote Web Service" type="style" hidden="false"/>
<parameter name="about.style" value="VelocityParameterPresentationStyle" hidden="true"/>
<parameter name="about.style.template" value="InputWithHelp.vm" hidden="true"/>
<parameter name="about.style.src" value="images/info.gif" hidden="true"/>
<parameter name="about.style.javascript.onclick" value="alert('For additional info on the stock quote web
service contact info@bluesunrise')" hidden="true"/>
```

```
package com.bluesunrise.jportal.modules.actions.portlets;

// Turbine stuff
import org.apache.turbine.util.Log;
import org.apache.turbine.util.RunData;
import org.apache.turbine.services.TurbineServices;

// Velocity Stuff
import org.apache.velocity.context.Context;

// Jetspeed stuff
import org.apache.jetspeed.portal.portlets.VelocityPortlet;
import org.apache.jetspeed.webservices.finance.stockmarket.StockQuoteService;
import org.apache.jetspeed.webservices.finance.stockmarket.StockQuote;
import org.apache.jetspeed.util.PortletConfigState;
import org.apache.jetspeed.util.StringUtils;

/**
 * This action sets up the template context for retrieving stock quotes.
 *
 */

public class TutorialStockQuoteAction extends VelocityPortletAction
```

```
{
private static final String SYMBOLS = "symbols";
private static final String COLUMNS = "columns";
private static final String SORT = "sort";
private static final String QUOTES = "quotes";
private static final String[] ALL_COLUMNS = {"Symbol", "Price", "Change", "Volume"};
private static final String SELECTED_COLUMNS = "selected-columns";1

/**
 * Build the normal state content for this portlet.
 *
 * @param portlet The velocity-based portlet that is being built.
 * @param context The velocity context for this request.
 * @param rundata The turbine rundata context for this request.
 */

protected void buildNormalContext(VelocityPortlet portlet, Context context, RunData rundata)
{
    try
    {
        // Get reference to stock quote web service
        StockQuoteService service = (StockQuoteService) TurbineServices.getInstance().
            getService(StockQuoteService.SERVICE_NAME);

        // Retrieve portlet parameters
        String symbols = PortletConfigState.getParameter(portlet, rundata, SYMBOLS,
"IBM,MSFT,ORCL,SUNW");
        String columns = PortletConfigState.getParameter(portlet, rundata, COLUMNS,
StringUtils.arrayToString(ALL_COLUMNS, ","));2
        String[] selectedColumnsArray = StringUtils.stringToArray(columns, ",");3
        String sort = PortletConfigState.getParameter(portlet, rundata, SYMBOLS, "Symbol");4

        // Request stock quote(s) from the stock quote web service
        String[] symbolArray = StringUtils.stringToArray(symbols, ",");
        StockQuote[] quotes = service.fullQuotes(symbolArray);

        // Place appropriate objects in Velocity context
        context.put(QUOTES, quotes);
        context.put(SELECTED_COLUMNS, selectedColumnsArray);5
    }
}
```

```

        context.put(COLUMNS, columns);6
    }
    catch (Exception e)
    {
        Log.error(e);
    }
}

```

So far the changes are related to stock quote column selection:

1. The *SELECTED_COLUMN* will be placed in the Velocity context as list of selected column headings.
2. The *columns* variable will hold user customized value of columns selected for display. Please note the usage of `PortletConfigState.getParameter()` method to retrieve the value of *COLUMNS* from user's PSML.
3. Here we convert *columns* which is in a comma-delimited format to a string array *selectedColumnsArray*.
4. For the sake of example, we can also retrieve a custom *sort* setting. We will not be implementing the actual sort though.
5. The *selectedColumnsArray* is placed in the Velocity context so the template can use it as headings.
6. The *columns* is placed in the Velocity context so the template can determine which stock quote fields to display.

Next, here's a modified Velocity template `tutorial-stock-quote.vm` used to display the stock quotes:

```

## tutorial-stock-quote.vm
<table>
  <tr>
    <td>
      <table border="true" cellspacing="1" cellpadding="3">
        <tr>
          #foreach ($column in $selected-columns)
            #headerCell ($column)
          #end
        </tr>
      </table>
    </td>
  </tr>
</table>

```

```

        #foreach ($quote in $quotes)
        <tr>
            #if ($columns.indexOf("Symbol") >= 0) #entryCell ($quote.Symbol) #end
            #if ($columns.indexOf("Price") >= 0) #entryCell ($quote.Price) #end
            #if ($columns.indexOf("Change") >= 0) #entryCell ($quote.Change) #end
            #if ($columns.indexOf("Volume") >= 0) #entryCell ($quote.Volume) #end
        </tr>
        #end
    </table>
</td>
</tr>
</table>

```

```

<portlet-entry name="TutorialStockQuote" hidden="false" type="ref" parent="Velocity" application="false">
  <meta-info>
    <title>Tutorial Stock Portfolio with parameter styles</title>
    <description>Tutorial Stock Portfolio Portlet with parameter styles</description>
  </meta-info>
  <parameter name="template" value="tutorial-stock-quote2" hidden="true"/>
  <parameter name="action" value="portlets.TutorialStockQuoteAction2" hidden="true"/>
  <parameter name="sort" value="Symbol" type="style" hidden="true">
    <meta-info>
      <title>Sort</title>
      <description>Sort is not implemented</description>
    </meta-info>
  </parameter>
  <parameter name="sort.style" value="ListBox" hidden="true"/>
  <parameter name="sort.style.items" value="Symbol,Price,Change,Volume" hidden="true"/>
  <parameter name="columns" value="Symbols,Price,Change,Volume" type="style" hidden="false">
    <meta-info>
      <title>Columns</title>
      <description>Columns to display</description>
    </meta-info>
  </parameter>
  <parameter name="columns.style.items" value="Symbol,Price,Change,Volume" hidden="true"/>
  <parameter name="columns.style.layout" value="$eastwest" hidden="true"/>
  <parameter name="columns.style" value="CheckBoxGroup" hidden="true"/>

```



```
<parameter name="symbols" value="MSFT,IBM,ORCL,SUNW" type="style" hidden="false">
  <meta-info>
    <title>Symbols</title>
    <description>List of comma-separated stock symbols</description>
  </meta-info>
</parameter>
<parameter name="symbols.style" value="TextArea" hidden="true"/>
<parameter name="About" value="About Stock Quote Web Service" type="style" hidden="false"/>
<parameter name="About.style" value="VelocityParameterPresentationStyle" hidden="true"/>
<parameter name="About.style.template" value="InputWithHelp.vm" hidden="true"/>
<parameter name="About.style.src" value="images/info.gif" hidden="true"/>
<parameter name="About.style.javascript:onclick" value="alert('For additional contact
info@bluesunrise.com')" hidden="true"/>
<media-type ref="html"/>
<category group="Jetspeed">tutorial</category>
</portlet-entry>
```

Future Possible Tutorials

If anyone wants to write one of these please feel free to do so.

1. WebPage Portlet
2. Torque OM configuration

Appendix A - How Property Merging Works in the Tutorial

You may be wondering why the tutorial works with only a few properties, and why we have our own versions of the configuration-property files. The reason is that Jetspeed does not have a standard way to deploy portlet applications, thus with version 1.4 we have to merge property files in order for a portal application to override and define its own configuration properties, and also to avoid editing the Jetspeed property files everytime they are modified in the CVS.

The build.xml file in the JPortal source is capable of doing just that. It merges the JPortal configuration files shown below on the left into the master Jetspeed resource files on the right:

| JPortal Configuration File | | Jetspeed Configuration File |
|----------------------------|---|------------------------------|
| JPortalJetspeed.properties | à | JetspeedResources.properties |
| JPortalTurbine.properties | à | TurbineResources.properties |
| JPortalSecurity.properties | à | JetspeedSecurity.properties |
| JPortalTorque.properties | à | Torque.properties |

The merge program is provided with Jetspeed. Let's take a look at one of the Ant targets for merging the JetspeedResources.properties:

```
<target name="merge_jrp" depends="properties">
  <echo> merge_jrp required </echo>
  <java fork="yes"
    classname="org.apache.jetspeed.util.OverwriteProperties" dir=".">
    <classpath refid="test.classpath"/>
    <sysproperty key="DEBUG" value="true"/>
    <arg
      value="\${build.home}/WEB-INF/conf/JetspeedResources.properties"/>
    <arg value="./webapp/WEB-INF/conf/JPortalJetspeed.properties"/>
    <arg value="./webapp/WEB-INF/conf/" />
  </java>
</target>
```

The OverwriteProperties class is run with these arguments:

Argument

| | |
|-------------|---|
| Master File | "\${build.home}/WEB-INF/conf/JetspeedResources.properties |
| Merge File | ./webapp/WEB-INF/conf/JPortalJetspeed.properties |

The general rules for merging properties are:

1. A property in the Master File with the same name as a property in the Merge File will be overwritten by the Merge File property **
2. Properties from the Merge File that are not found in the Master File will be added to the Master File
3. Properties will be removed from the Master File if they match the property name prefix-strings described below.

** with the exception of the include directive and *module.properties* property, which are added since there can be multiple *include directives* or *module.packages* properties.

Removing Properties

There are many properties in the default Jetspeed configuration that aren't used by many sites. For instance, the RSS and OCS support, which requires background threads to be running, can automatically be removed by merging them out of the master properties file. This is accomplished using a remove directive and prefix strings in your merge property file:

```
-contentfeeds.  
-#contentfeeds.  
-daemon.  
-services.DaemonFactory
```

```
-content.  
-#content.
```

Any line that begins with a dash “-“ will instruct the property merger to remove a property starting with the given prefix-string immediately following the dash. The prefix-string can include comments (or all comments).

Appendix B - Turbine and ECS

The `getContent()` method of a portlet takes a `RunData` object, which is passed through from an internal component called [Turbine](#). The `RunData` class is one of the few Turbine classes that you must deal with often. Turbine is a MVC-2 (Model View Controller) Servlet framework that Jetspeed was founded upon.

Another component coupled to in the Portlet interface is the [Element Construction Set](#) (ECS), an object-based mark-up generator. The Jetspeed Portlet interface requires that portlets return ECS elements as their content. It uses Java objects to construct the elements of the respective mark-up. Admittedly, this approach is better than hard coding mark-up tags, but it still places mark-up information into compiled code.

Therefore, we generally recommend using a page template mechanism of your choice, rather than using ECS directly. The majority of the portlets in Jetspeed are based upon the base class [Velocity Portlet](#), a template-based portlet which does not mix Java source with mark-up. Similarly, you can use the [JSP Portlet](#) to create MVC portlets based upon the Java JSP standard.

Note: Starting with version 2.0 of Jetspeed, ECS will be deprecated.