JetExpress Tutorial Portal v.1.0

Project Documentation

Apache Software Foundation **22 December 2007**

Table of Contents

1.1 Welcome

Welcome to the Jetspeed Tutorial!

The Jetspeed Tutorial is a step-by-step set of instruction and source code for creating a custom Jetspeed Portal from scratch.

When starting a new Jetspeed portal project, we strongly recommend that you create a custom portal project, and do not edit the Jetspeed-2 source and resources directly.

This tutorial will guide you through the steps to create a sample portal named **jetexpress**. The goal of this tutorial is to prepare you for creating your own custom Jetspeed portal, complete with your own set of portal pages, your company logos and text, your own set of portlet applications, and any special integration required to run inside Jetspeed.

Prerequisites

The tutorial requires the following software installed on your system:

- Java 1.4 or higher
- Maven-2
- An internet connection so that Maven can download plugins

Before getting started with the tutorial, review the Custom Build commands available to you:

- Build Commands
- Project Generation Archetypes
- Project Directory Overview

Tutorial Resources

During the tutorial, at times you will be asked to copy files from the *resources* directory. You can cut and paste the *copy | cp* commands (for Windows and Linux) into a shell, and copy the resources into your new custom portal project. These commands are not really a part of the normal development cycle, but are there to demonstrate the incremental changes to the portal as we progress through the tutorial.

To get started with the tutorial, click here: Start Tutorial

1.2 Build Commands

Build Commands for Maven-2 and Jetspeed

Here is a quick summary of the current Jetspeed2 Maven2 build commands:

mvn	performs base build and installs J2 artifacts in the Maven2 repository, (install is the default Maven2 goal for this build)
mvn -P test	performs base build and includes execution of test cases, (includes test database setup)
mvn -P tomcat	performs base build and deploys to Tomcat app server, (includes prodution database setup). Other app servers will be supported in the future, (see below for details)
mvn clean	cleans all build target directories

Please note that the Maven2 repositories used by default with this build are generally saturated and the Jetspeed2 Maven2 repository does not yet contain a full mirror of the artifacts required to build Jetspeed2. If you experience download or connection failures, simply restart the build by issuing your last command again to retry.

Additional Profiles

The following profiles are optional and can be supplied when building/deploying to Tomcat

mvn -P tomcat,full	at the moment, this is the same as the default build: deploys a full J2 set of portlets and PSML pages.
mvn -P tomcat,nodb	specifies that configuration of the production database should be skipped.
mvn -P tomcat,hot	indicates that a portlet app, component, or content should be directly written to the deployed Jetspeed2 webapp.
mvn -P tomcat,dbpsml	specifies portal build and database deployment/import to include configuration of database PSML.
mvn -P prod	executes deployment to application server specified in settings.xml, (see below).

Currently, only Tomcat 5 and Tomcat 5.5 application servers are supported by this build. More platforms are to be supported in the near future.

1.3 Project Generation Archetypes

Project Generation Commands for Maven-2 and Jetspeed

We support several kinds of project generation commands in the build. These commands are based on *Maven Archetypes*. The project generation commands create a template for building certain types of Jetspeed projects. The most common archetype is the *portal-archetype*, which creates a new Jetspeed Custom Portal. The complete list of project generation commands are:

portal-archetype	Generate a Maven project for a new custom Jetspeed Portal Application
application-archetype	Generate a Maven project for a new JSR 168 Portlet Application
component-archetype	Generate a Maven project for a new general Java component
shared-component-archetype	Generate a Maven project for a new shared Jetspeed component
portal-component-archetype	Generate a Maven project for a new Jetspeed component (installed in Jetspeed core)

1.4 Project Directory Overview

Overview of the Custom Project Directory Structure

The **portal-archetype** generates a complete Maven-2 project directory structure for developing a custom Jetspeed portal as well as JSR 168 portlet applications. Here is an overview of directories created by the portal-archetype (directories are relative to the custom portal root):

/applications	conventional subdirectory location for one or more portal application projects
/app-servers	contains portal deployment builds and resources.
/components	conventional subdirectory for one or more portal component projects
enterprise	maven-2 build to create an J2EE enterprise archive (EAR) deployable file
etc/assembly	custom portal application component Spring assemblies
etc/conf	portal application context configuration files
etc/decorations	custom decorations in images, layout, and portlet subdirectories
etc/pages	custom portal PSML pages to augment/override minimal defaults, (e.g. /Administrative/**, /default-page.psml, /myaccount.psml, /page.security, and /system/**)
etc/schema	Jetspeed2 database schema definitions
etc/sql	Jetspeed base database configuration scripts
etc/templates	custom overrides for Jetspeed2 layout portlet templates
etc/webapp	
portal	portal application war build scripts and webapp resource overrides
src	these directories are an artifact of archetype expansion and should be deleted

2.1 Maven First Steps

First Steps with Maven-2

Minimal configuration is required to get started with your Jetspeed custom build and Maven-2.

Copying in the settings.xml

The first step to building with Maven-2 is to setup your **settings.xml**. These settings hold all the information necessary to build with Maven-2 and Jetspeed.

There is a *settings.xml.jetexpress* file in the */JetspeedTraining/resources/maven/* directory. Copy this file into your *Maven Home* directory. The Maven home directory is located in a directory named **.m2** found under your *User Home* directory. So for example that would be *~/.m2/* on Linux, or *"%USERPROFILE%\.m2\"* on Windows.

Linux: paste into Command Line:

Windows: paste into Command Line:

Editing the settings.xml

The settings file is primarily for configuring your application server and database. The settings are preconfigured for an embedded Derby database, so more changes are required if another database solution is to be used, covered later in this tutorial.

Edit the *~/.m2/settings.xml* file.

Change the Jetspeed Server (Tomcat) Location

For the purpose of the training, lets all use the same Tomcat home, point it to the */JetspeedTraining/tomcat-express* directory.

```
<org.apache.jetspeed.server.home>c:/JetspeedTraining/tomcat-express</org.apache.jetspeed.server.home>
```

Change the Derby Database location

Next, lets change the location of our Derby database to the */JetspeedTraining/database/jetexpress* directory.

```
<org.apache.jetspeed.production.database.url>jdbc:derby:/JetspeedTraining/database/jetexpress</org.apache
```

Change the Repository Location

Change the Maven local repository location by modifying the <*localRepository*> element in the settings.xml file. If you are using the training material, we have pre-configured a Maven-2 repository /JetspeedTraining/maven/repository. Lets configure Maven to point there, edit the *settings.xml*:

```
        <localRepository>c:/JetspeedTraining/maven/repository/</localRepository>
```

Go on and save that file.

Next

2.2 Generating a Jetspeed Portal

Generating a Jetspeed Portal

To create a new custom portal named **jetexpress**, enter the following commands:

```
# Linux
cd /JetspeedTraining/workspace

# Windows
cd \JetspeedTraining\workspace

mvn archetype:create -DarchetypeGroupId=org.apache.portals.jetspeed-2
                     -DarchetypeArtifactId=portal-archetype
                     -DarchetypeVersion=2.1
                     -DgroupId=org.apache.portals.tutorials
                     -DartifactId=jetexpress
                     -Dversion=1.0
```

Paste into Command Line:

A directory named **jetexpress** under */JetspeedTraining/workspace* should have been created. Notice that **jetexpress** will be the name of your portal, not **jetspeed**. The idea is that you can create a customized portal based upon Jetspeed, but with a different name, customized to meet your organization requirements.

Previous Next

2.3 Build and Deploy Custom Portal

Building and Deploying a Custom Jetspeed Portal

Now that you have generated a custom portal named **jetexpress**, we are ready to build the portal. We are going to build the portal from the command line:

```
# Linux
cd /JetspeedTraining/workspace/jetexpress

# Windows
cd \JetspeedTraining\workspace\jetexpress

mvn -P tomcat,min
```

Please take note that there is no space between the comma and the word ***min***

We are building a **min** (minimal) deployment. The Min deployment only creates a Jetspeed core portal and a Jetspeed Admin portlet application (j2-admin). When the build process completes, you should see the message **BUILD SUCCESSFUL** at the end of a large amount of build logging information The portal is built into a temp directory named *target* found directory under your *jetexpress* project directory. As well as building, the *tomcat,min* goals also:

- deploys Jetspeed to the Tomcat application server. If you are following the training material, the Tomcat directory is located under */JetspeedTraining/tomcat-express/*
- creates the Jetspeed core schema tables in the Derby database
- populates the database with a minimal set of portal information, including default roles, groups, profiles, and administrative users

Now that we've built and deployed the portal, lets start up the application server...

Previous Next

2.4 Run the Custom Portal

Running a Custom Jetspeed Portal

Running the portal is very straight forward. In this tutorial we are running our Jetspeed portal in a Tomcat application server. To start up the portal, go to the command line and enter:

```
cd tomcat-express
cd bin
# Windows
catdebug
# Linux
./catdebug.sh
```

The first time we run the portal, its going to take a minute or so to complete the initial deployment of the portal and Admin portlet application.

Open up a browser and navigate to http://localhost:8080/jetexpress/portal

You should see the minimal core Jetspeed portal as shown below. Login with the credentials **admin/admin**. You will be prompted to change the admin password. As you can see, this is a boiler plate, minimal portal. Press the Next button to get started on customizing Jetspeed.

Previous Next

3.1 Decorators and Themes

Decorator and Theme Customization

Now that we have created our custom Jetspeed project, lets start customizing the portal design. The portal design, or skins, are known in Jetspeed as *decorators and themes*. With decorators and themes, you can customize the portal experience to the branding of your organization. In this tutorial, we will simply change a few images, CSS styles and colors to get you on your way.

It is important to note that all of the changes made in this section are made in the build environment. The changes we are making here could just as easily be made using the portal's live customization features. For example, you can drop a decorator or theme into the portal while its running. Jetspeed will pick it up automatically. Or, to customize a page, you can use the portlet customizer or desktop customizer, portlet selector, and site manager: all administrative portlets that work on your live portal. However, the point of configuring everything in a Maven build is to be able to easily reproduce portal environments for development, testing, and new deployments.

This section of the tutorial covers customizing:

- Portal Skins (or Decorators). We replace the default logo, colors, and page header and footers with our own.
- Portal Themes. Same as Portal Skins, but themes apply to the Jetspeed Desktop.
- Configuring the default portal page to use these new decorators and themes

Lets get started with customization of the default portal page decoration

Previous Next

3.2 Page Decoration

Page Decorators

Each Jetspeed page can be associated with a different page decoration. Page decorations control some important aspect of a portal page:

- The colors, images, CSS styles that skin this page
- The header portion of the page
- The page margins
- The footer portion of the page
- Menus displayed on the page
- Action buttons displayed on the window

Decorators do not control the placement of portlets. That is handled by layouts. Jetspeed comes with several page decorations out of the box. The default page decorator for most pages is called *tigris*. It looks like this:

We are going to create a new decorator for this tutorial. This new decorator can be copied into our project from the */JetspeedTutorial/resources/decorations/layout/express-page/* directory. This will save you the trouble of creating all the logo images and CSS definitions.

```
# Linux
cd /JetspeedTraining/workspace/jetexpress
mkdir portal/src/webapp/decorations/layout/express-page
cp -r ../../resources/decorations/layout/express-page/*
portal/src/webapp/decorations/layout/express-page/

# Windows
cd \JetspeedTraining\workspace\jetexpress
mkdir portal\src\webapp\decorations\layout\express-page
xcopy /s ..\..\resources\decorations\layout\express-page\*
portal\src\webapp\decorations\layout\express-page
```

The Header

Open up the *decorations/layout/express-page/header.vm* This is a Velocity template, very much like JSP but simpler, with no Java compilation required. Jetspeed does support JSP-based decorators. However no one has contributed one yet. We could spend a lot of time teaching you about all the macros available. But lets just concentrate on changing the logos first. Scroll down to the banner content. Here we add our

new left-hand side logo:

```
#GetPageResource('images/blueSunrisePicture.gif')
```

We've added a few more custom images, one in the center area of the banner:

```
<img src="#GetPageResource('images/BlueSunriseTextTahoma.gif')" height="29"
width="165">
```

and one more in the right area of the banner:

```
<img src="#GetPageResource('images/RisingEdgeSlogan.gif')" height="28" width="300">
```

#GetPageResource is a Velocity macro. It retrieves a resource (image, CSS, HTML) from the decoration folder, relative to the root of the express-page decoration folder. Besides the images, the header.vm is pretty much the same as Tigris. In fact we simply cut and paste the Tigris decorator to get us started. This gives you a good start of customizing the page.

Velocity Variables

Velocity makes several variables about the context of a decoration available for dynamic substition of menus and content:

Variable	Desc	Usage
$layoutDecoration	Retrieve layout content from the decoration dir	$layoutDecoration.getResource("decorator-macros.vm")
$site	Retrieve menus by name	$site.getMenu("pages")

Menus

The remainder of the page is HTML markup mixed in with some important macros for displaying Jetspeed Menus. *Jetspeed Menus* are build from a collection of portal resources known as the *Portal Site*. The portal site is a content tree (like a file system) of portal resources. The site can be stored in the file system or in a database. Resources can be a page, folder, or link. Lets look at some of the available macros for

displaying menus on your page.

The $site always has the following menus available to you at any time:

Menu	Desc
pages	relative pages menu of pages in the current folder. Used to define the page tabs above the portal.
breadcrumbs	paths to page used to provide history links below the page tabs
navigations	relative subfolders and root level links menu used to define the navigation pane beside the portal.
back	parent folder menu used to define the single "back" link above the portal page tabs.

You can also define your own menus (not covered in this tutorial).

There are some helper macros for creating different styles of menus. The macros are defined in the decorator-macros.vm file:

Macro	Decription
#includeTabsNavigation($someMenu $LEFT_TO_RIGHT)	Displays a menu in a vertical tabbed navigation style.
#includeLinksNavigation($breadCrumb $LEFT_TO_RIGHT "" $BREADCRUMBS_STYLE "")	Displays a menu of links according to a given style.
#includeNestedLinksWithIconNavigation($standardNavs $TOP_TO_BOTTOM)	Displays a nested top-to-bottom menu navigation of folders, links, and pages.
#PageActionBar()	Not a menu, but the available actions (edit, view, help, print...) for this page

The Footer

Open up the *decorations/layout/express-page/footer.vm*

```
<img src="#GetPageResource('images/jetspeed-powered.gif')" alt="Jetspeed 2 Powered"
border="0" />
```

Previous Next

3.3 Portlet Decoration

Portlet Decorators

Each Jetspeed portlet window on a page can be associated with a different portlet decoration. Portlet decorations control some important aspect of a portlet window:

- The colors, images, CSS styles that skin this window
- The title portion of the portlet
- The borders of the window
- Action buttons displayed on the window

Jetspeed comes with several portlet decorations out of the box. The default page decorator for most pages is called *tigris*. It looks like this:

We are going to create a new decorator for this tutorial. This new decorator can be copied into our project from the */JetspeedTutorial/resources/decorations/portlet/express-portlet/* directory. This will save you the trouble of creating all the logo images and CSS definitions.

```
# Linux
cd /JetspeedTraining/workspace/jetexpress
mkdir portal/src/webapp/decorations/portlet/express-portlet
cp -r ../../resources/decorations/portlet/express-portlet/*
portal/src/webapp/decorations/portlet/express-portlet/

# Windows
cd \JetspeedTraining\workspace\jetexpress
mkdir portal\src\webapp\decorations\portlet\express-portlet
xcopy /s ..\..\resources\decorations\portlet\express-portlet\*
portal\src\webapp\decorations\portlet\express-portlet
```

We are going to skip over the intricate details of portlet decorators, as page decorators are much more important, as they set your organization's branding. Whereas the standard window decorators are good enough for getting you started. Go ahead and review the content of our custom portlet decorator. You will see that the directory layout is the same as for page decorators. The decorator simply creates a new color variation on an existing window decorator.

Previous Next

3.4 Desktop Themes

Desktop Page Themes

The Jetspeed Desktop has its own kinds of decorations. These decorations are called **Desktop Themes** Each Jetspeed Desktop page can be associated with a different theme. Themes control some important aspect of a desktop page:

- The colors, images, CSS styles that skin this page
- The header portion of the page
- The page margins
- The footer portion of the page
- Menus displayed on the page
- Action buttons displayed on the window

Themes do not control the placement of portlets. That is handled by the Jetspeed Desktop engine, which follows the layout plan provided by the stuctured page markup (PSML). This is the same layout instructions applied to a portal page. You will see that themes are much simpler content than decorators. That is because most of the content in a theme is populated by the Jetspeed Desktop engine at runtime. Jetspeed comes with a few desktop themes out of the box. The default desktop theme for most pages is called *blue*. It looks like this:

We are going to create a new desktop theme for this tutorial. This new theme can be copied into our project from the */JetspeedTutorial/resources/themes/express/* directory. This will save you the trouble of creating all the logo images and CSS definitions.

```
# Linux
cd /JetspeedTraining/workspace/jetexpress
mkdir portal/src/webapp/desktop-themes/express
cp -r ../../resources/desktop-themes/express/*
portal/src/webapp/desktop-themes/express/

# Windows
cd \JetspeedTraining\workspace\jetexpress
mkdir portal\src\webapp\desktop-themes\express
xcopy /s ..\..\resources\desktop-themes\express\*
portal\src\webapp\desktop-themes\express\
```

The Theme template

Have a look at the *express* theme directory. Notice that there are two theme files: *express.jsp* and *express.vm* Since there were so many complaints about no JSP support in templates, with the Desktop we decided to require support for both. The *theme.properties* determines which templates is active. Lets look at the Velocity template. We have macros to display-theme relative resources:

```
        <img src="$jetspeedDesktop.getDesktopThemeResourceUrl('images/logo.gif')"
alt="Logo" border="0"/>
```

Theme Variables

JSP and Velocity make several variables about the context of a theme available for dynamic substition of menus and content:

Variable	Desc	Usage
$jetspeedDesktop	Retrieve theme resources, and the name of the theme	${jetspeedDesktop.getDesktopTheme()

Content Divs

The remainder of the page is HTML DIV markup with special widget types and identifiers. The desktop will populate these Divs with various content such as the portlets and menus. *Jetspeed Menus* are build from a collection of portal resources known as the *Portal Site*. The portal site is a content tree (like a file system) of portal resources. The site can be stored in the file system or in a database. Resources can be a page, folder, or link. Lets look at some of the available macros for displaying menus on your page.

Widget	Type	Desc
jetspeed-menu-pages	jetspeed:PortalTabContainer	relative pages menu of pages in the current folder. Used to define the page tabs above the portal.
jetspeed-menu-breadcrumbs	jetspeed:PortalBreadcrumbContainer	paths to page used to provide history links below the page tabs
jetspeed-menu-navigations	jetspeed:PortalAccordionContainer	relative subfolders and root level links menu used to define the navigation pane beside the portal.

You can also define your own menus (not covered in this tutorial).

Finally, the Div to hold the portlet content must be defined. It is just a plain HTML DIV:

```
<div class="layout-${jetspeedDesktop.getDesktopTheme()}" id="jetspeedDesktop"></div>
```

Notice that while decorators require two templates, desktops only require one template. This makes for a much simpler page.

Previous Next

3.5 Changing the Default Page

Modifying the Default Page

Now that we have created our decorators and themes, lets put them to use on the default page. Lets get started with customization of the default portal page decoration. Copy in our sample default-page:

```
# Linux
cd /JetspeedTraining/workspace/jetexpress
cp ../../resources/pages/default-page.psml portal/src/webapp/WEB-INF/pages/

# Windows
cd \JetspeedTraining\workspace\jetexpress
copy  ..\..\resources\pages\default-page.psml portal\src\webapp\WEB-INF\pages
```

Refresh your Eclipse project and edit the default-page.psml. Make the following modifications

- for the layout-decorator, replace tigris with **express-page**
- for the portlet-decorator, replace tigris with **express-portlet**
- add a skin default attribute as **skin="express"** for the desktop theme

```
<page>
  <defaults layout-decorator="express-page"
            portlet-decorator="express-portlet"
            skin="express"/>
```

Go ahead and save that file. We are now ready to deploy your changes to the portal

Previous Next

3.6 Deploying your Customizations

Deploying Decorations, Themes and Pages

To deploy the new resources (decorations, themes, and page) that we created in this section of the tutorial, you could simply rebuild the portal by typing **mvn -P tomcat,min**. Or, if your portal is aleady up and running, a simpler way is to automate the easy deployment tasks with Ant. We have created a simple build.xml to copy over changed portal resources:

```
# Linux
cd /JetspeedTraining/workspace/jetexpress
cp ../../resources/build.xml .

# Windows
cd \JetspeedTraining\workspace\jetexpress
copy  ..\..\resources\build.xml
```

To copy over the resources to the running portal, type:

```
ant
```

Refresh the portal home page, http://localhost:8080/jetexpress/portal You should now see a portal with our new custom decorators:

and desktop theme:

Previous Next

4.1 Generating a Portlet Application

Generating a Portlet Application

Portlet applications should be stored in a separate sub-project of your main portal project. Our custom build is setup to put one or more portlet application projects under the *applications* directory. For this tutorial, we will create one portlet application. The portlet application will be automatically built and deployed when you run **maven -P tomcat,min**.

To create a new portal application named **express-demo**, enter the following commands:

```
# Linux
cd /JetspeedTraining/workspace/jetexpress/applications

# Windows
cd \JetspeedTraining\workspace\jetexpress\applications

mvn archetype:create -DarchetypeGroupId=org.apache.portals.jetspeed-2
                     -DarchetypeArtifactId=application-archetype
                     -DarchetypeVersion=2.1
                     -DgroupId=org.apache.portals.tutorials
                     -DartifactId=express-demo
                     -Dversion=1.0
```

Paste into Command Line:

A directory named **express-demo** under */JetspeedTraining/workspace/applications* should have been created. Notice that **express-demo** will be the name of your portlet application.

Lets have a closer look at what was created. There is a **src** directory, and underneath it three subdirectories **java, test, webapp**. Under the **java** directory, there is one sample portlet. Under the **test** directory, you will find one unit test. Under the **webapp** directory, you will find a number of files that are the basic template for any useful portlet application:

Here you will find the portlet and servlet deployment descriptors: **portlet.xml** and **web.xml**. Review the files in this directory. We are going to copy over some more sample portlets for the tutorial. Lets let an ant task to do the work for us:

```
ant copy-portlet-resources
```

Previous Next

4.2 Building and Deploying

Building and Deploying the Portlet Application

Now that you have generated a portlet app named **express-demo**, we are ready to build the portal. We are going to build the portal from the command line:

```
# Linux
cd /JetspeedTraining/workspace/jetexpress/applications/express-demo

# Windows
cd \JetspeedTraining\workspace\jetexpress\applications\express-demo

mvn
```

This builds a portlet application war file name *express-demo-1.0.war*. This war file is found under *target* directory. It is a standard JSR-168 portlet application distribution.

Now that we've built and deployed the portal, lets start up the application server (if its not already started). While the app server is running, we can drop in the portlet application.

```
# Linux
cd /JetspeedTraining/workspace/jetexpress/applications/express-demo
cp target/express-demo-1.0.war
/JetspeedTraining/tomcat-express/webapps/express-demo.war

# Windows
cd \JetspeedTraining\workspace\jetexpress\applications\express-demo
copy target\express-demo-1.0.war
\JetspeedTraining\tomcat-express\webapps\express-demo.war
```

This will deploy your portlet application. Next, lets see how to add your new portlets to a page.

Previous Next

4.3 Configuring Portal Pages

Configuring Portal Pages

Now that you have generated a portlet app named **express-demo**, we are ready to integrate the new portlets into our system. This is done by adding the portlets to the portal pages. These pages are also known as PSML files. Your pages makeup the navigational site of your system. The new pages we are going to add will have references to the new portlets added in the express-demo portlet applications. Of course users can use the Portlet Selector and Jetspeed Customizer to select portlets interactively. Here we are setting up the collection of pages that will always be a part of your portal. The file-system tree-like collection of pages is known as your **portal site**

We provide a quick Ant task to copy in the new, preconfigured pages:

```
# Linux
cd /JetspeedTraining/workspace/jetexpress/

# Windows
cd \JetspeedTraining\workspace\jetexpress\

ant copy-page-resources
```

Open up some of the pages found under *portal/src/webapp/WEB-INF/pages/* . You will see that we've added a new *Contact Us* page, and added the some new demo portlets the the default page. We have also changed the default themes and decorators for all administrative pages.

Go ahead and deploy those changes:

```
ant
```

This will deploy your new pages. Refresh the portal home page to see the new portlets:

Likewise for the desktop:

Previous Next

5.1 Overview

Portlet Development with the Portlet API

This section of the tutorial covers Portlet Development with the Java Portlet API.

- Getting started with Eclipse
- Bonjour Monde Portlet: the staple Hello World demo to get things rolling introduces the portlet.xml, as well as basic Portlet API concepts.
- The Weather Portlet: using init parameters, preferences, and request parameters
- The Stock Quote Portlet: actions, JSP, Portlet API Tag Library
- Interportlet communication, more advanced Portlet programming.

Previous Next

5.2 Eclipse

Eclipse Integration

Before getting started, lets get Eclipse setup. Lets create a Eclipse project for the jetexpress project. Maven-2 can create Eclipse project and classpath files from your Maven-2 project POM, automatically bringing in all dependencies:

```
# Linux
cd /JetspeedTraining/workspace/jetexpress
mvn eclipse:eclipse
cp portal/.classpath .
cp portal/.project .
# Windows
cd \JetspeedTraining\workspace\jetexpress
mvn eclipse:eclipse
copy portal\.classpath .
copy portal\.project .
```

Lets do the same for the *express-demo* portlet application. You may prefer to actually create a new Eclipse project to represent the portlet app subproject. We recommend this, since Eclipse doesn't work very well multi-Maven projects. Best to break them out into one Maven project == one Eclipse project:

```
# Linux
cd /JetspeedTraining/workspace/jetexpress/applications/express-demo
mvn eclipse:eclipse
# Windows
cd \JetspeedTraining\workspace\jetexpress\applications\express-demo
mvn eclipse:eclipse
```

Now lets import these new projects. From the eclipse menu, select File->Import:

Repeat for both the applications/express-demo project.

Next, lets setup a classpath variable to point at the Maven Repo, effectively resolving the jars for Eclipse. In Eclipse, go to Window->Preferences->Java->Build Path->Classpath Variables->New...:

If you would like to work with the Jetspeed source from the SVN Trunk, you will need to install the Subversion Plugin for Eclipse. After doing so, simply check out the project using the Subversion plugin:

We are now ready to start developing with Eclipse.

Previous Next

5.3 Hello World

Hello World Portlet

We are going to create a portlet using the Eclipse Java perspective. Go to the *express-demo* project, click on the *com.bluesunrise.portal.portlets.tutorial* package, and create a new Java class:

You will see a new portlet in Eclipse named BonjourWorld. Go ahead Override and Implement the following methods:

Each one of these methods is associated with a portlet mode. Lets make these methods actually do something. Since we are in the render phase when doView/doEdit/doHelp are called, its probably best to render something. The RenderResponse renders content to the output stream of the portlet. Set the content type on the response, and then print a *hello world* message using a Java Writer:

```
    protected void doView(RenderRequest request, RenderResponse response) throws
PortletException, IOException
    {
        response.setContentType("text/html");
        response.getWriter().println("<b>Bonjour: View Mode</b>");
    }
```

Repeat the same process for Edit and Help modes. Now lets edit the portlet.xml, and create a portlet descriptor entry for our portlet. Notice that the <supports> element contains that same portlet modes that we support in our *do* methods.

```
    <portlet>
        <description>Bonjour Monde Portlet</description>
        <portlet-name>BonjourMonde</portlet-name>
        <display-name>Bonjour Monde</display-name>
<portlet-class>com.bluesunrise.portal.portlets.tutorial.BonjourWorld</portlet-class>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>VIEW</portlet-mode>
            <portlet-mode>EDIT</portlet-mode>
            <portlet-mode>HELP</portlet-mode>
        </supports>
        <supported-locale>en</supported-locale>
            <portlet-info>
        <title>Bonjour Monde</title>
          <short-title>Bonjour</short-title>
          <keywords>tutorial,bonjour,hello</keywords>
```

```
      </portlet-info>
       </portlet>
```

Previous Next

5.4 Adding a Page

Adding a Page for the Tutorial Portlets

Now that we've created a new portlet, lets add a page to hold that portlet. In the **jetexpress** project, lets add a folder to the root of our site named *portal/src/webapp/WEB-INF/pages/tutorial/* . In addition to creating the folder, you will need to create a *folder.metadata* file:

```
<?xml version="1.0" encoding="UTF-8"?>
<folder>
  <title >Tutorial</title>
  <metadata name="title" xml:lang="fr">Autodidacte</metadata>

  <security-constraints>
    <security-constraints-ref>public-edit</security-constraints-ref>
  </security-constraints>
</folder>
```

Then lets add a new page named **default-page.psml** under the tutorial directory. Add a portlet window to reference our new portlet:

```
<page>
  <defaults layout-decorator="express-page"
            portlet-decorator="express-portlet"
            skin="express"/>
  <title>JetExpress Tutorials</title>
  <short-title>Tutorials</short-title>
  <fragment id="tutorial-100" type="layout"
name="jetspeed-layouts::VelocityTwoColumns">
          <fragment id="express-101" type="portlet"
name="express-demo::BonjourMonde"/>
  </fragment>
</page>
```

Lets deploy our portlet and the new pages:

```
# Linux
cd /JetspeedTraining/workspace/jetexpress
```

```
ant
cd applications/express-demo
mvn
cp target/express-demo-1.0.war
/JetspeedTraining/tomcat-express/webapps/express-demo.war

# Windows
cd \JetspeedTraining\workspace\jetexpress
ant
cd applications\express-demo
mvn
copy target\express-demo-1.0.war
\JetspeedTraining\tomcat-express\webapps\express-demo.war
```

Previous Next

5.5 JSP and Portlet API Taglib

JSP and Portlet API Taglib

Lets create another portlet. This portlet will not have a Java class. Instead it will be written entirely in JSP. Note that you can mix JSP and a Java class for the implementation of your Java class as you will see in the Stock Quote portlet example. Go to the *express-demo* project, click on the *src/webapp/WEB-INF/view/* directory, and create a JSP file named **tutorial.jsp**. Enter the following JSP code:

```
<%@ page session="true" contentType="text/html;charset=utf-8"%>
<%@ taglib uri='/WEB-INF/portlet.tld' prefix='portlet'%>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix='c' %>

<portlet:defineObjects/>

<portlet:renderURL var="max" windowState='maximized'/>
<portlet:renderURL var="normal" windowState='normal'/>
<c:out value="${renderRequest.windowState}"/>
<c:if test="${renderRequest.windowState == 'maximized'}">
<a href='<%=normal%>'>Normal</a>
</c:if>
<c:if test="${renderRequest.windowState == 'normal'}">
<a href='<%=max%>'>Max</a>
</c:if>
```

Every portlet JSP page is required to have the **defineObjects** tag at the top. Of course you also need the TLD reference. Portlets need to write their links to go back to the portal, not back to each individual servlet or JSP. That is the main difference between writing portlets and servlets. If you are using a framework like Struts or JSF correctly, these details should be hidden from you in the framework. The tag that we are using here is the **<portlet:renderURL>**. It allows you to create a render phase link back to this portlet, going through the portal. You can set window states, request parameters, and portlet mode changes on the URL. The other kind of link that you can create is an action URL: **<portlet:actionURL>**, which is usually used with a HTML form to post back parameters to the portlet and initial a blocking action phase event for the targeted portlet. The **<portlet:defineObjects>** tag declares three variables for your page:

JSP variable	Description
renderRequest	The RenderRequest object
renderResponse	The RenderResponse object

JSP variable	Description
portletConfig	The PortletConfig object

Here is the portlet.xml for our JSP portlet. It is based on the **GenericServletPortlet**, provided by Portals Bridges in a jar file dependency. Notice the init-param named **ViewPage**. This param defines which webapp-relative JSP to use for View Mode. Similiarly we have are **EditPage** for edit mode, and **HelpPage** for help mode.

```
 <portlet>
    <description>The 2nd Tutorial with JSP</description>
    <portlet-name>TutorialPortlet2</portlet-name>
    <display-name>Tutorial Portlet 2</display-name>
<portlet-class>org.apache.portals.bridges.common.GenericServletPortlet</portlet-class>
    <init-param>
        <name>ViewPage</name>
        <value>/WEB-INF/view/tutorial.jsp</value>
    </init-param>
    <init-param>
        <name>EditPage</name>
        <value>/WEB-INF/view/tutorial.jsp</value>
    </init-param>
    <init-param>
        <name>HelpPage</name>
        <value>/WEB-INF/view/tutorial.jsp</value>
    </init-param>
    <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>VIEW</portlet-mode>
        <portlet-mode>EDIT</portlet-mode>
        <portlet-mode>HELP</portlet-mode>
    </supports>
    <supported-locale>en</supported-locale>
        <portlet-info>
        <title>Tutorial Portlet</title>
        <short-title>tutorial</short-title>
                <keywords>tutorial,hello,JSP,taglib</keywords>
    </portlet-info>
    <portlet-preferences>
        <preference>
            <name>test</name>
            <value>hello</value>
        </preference>
    </portlet-preferences>
</portlet>
```

Add this portlet window fragment to the tutorial default page, underneath the BonjourMonde fragment:

```
        <fragment id="express-102" type="portlet"
```

```
name="express-demo::TutorialPortlet2"/>
```

And then deploy your changes:

```
# Linux
cd /JetspeedTraining/workspace/jetexpress/
ant
cd applications/express-demo
mvn
cp target/express-demo-1.0.war
/JetspeedTraining/tomcat-express/webapps/express-demo.war

# Windows
cd \JetspeedTraining\workspace\jetexpress
ant
cd applications\express-demo
mvn
copy target\express-demo-1.0.war
\JetspeedTraining\tomcat-express\webapps\express-demo.war
```

Previous Next

6.1 Services Tutorial Portlet

Jetspeed Services

This tutorial shows you how to use Jetspeed Services from the Express Demo Portlet Application. Please note that all edits, unless explicity specified otherwise, are applied to the express-demo PA source tree. We will learn how to:

- add new roles
- add new groups
- register new users
- manipulate pages
- get a filtered list of portlets

using the RoleManager, GroupManager, PortletAdministration, and Page Manager Jetspeed API interfaces.

Lets get started by entering a new portlet in the portlet.xml:

```
<portlet id="ServicesTutorialPortlet">
    <description>Tutorial for using Jetspeed Services, such as PortalAdministration,
PageManager, Registry.</description>
    <portlet-name>ServicesTutorialPortlet</portlet-name>
    <display-name>Jetspeed Services Tutorial Portlet</display-name>
<portlet-class>com.bluesunrise.portal.portlets.services.ServicesTutorialPortlet</portlet-class>
    <init-param>
        <description>This parameter sets the template used in view
mode.</description>
        <name>ViewPage</name>
        <value>/WEB-INF/view/services-tutorial.jsp</value>
    </init-param>
    <init-param>
        <description>Comma-separated list of roles to create via Role
Manager</description>
        <name>roles</name>
        <value>role1,role2,role3</value>
    </init-param>
    <init-param>
        <description>Comma-separated list of groups to create via Group
Manager</description>
        <name>groups</name>
        <value>group1,group2,group3</value>
    </init-param>
    <init-param>
        <description>Comma-separated list of Users to create and Register via
PortalAdminstration service</description>
        <name>users</name>
        <value>user1,user2,user3</value>
```

```
    </init-param>
    <init-param>
        <description>Comma-separated list of roles to assign to a new
user</description>
        <name>registration-roles</name>
        <value>user,role1,role2</value>
    </init-param>
    <init-param>
        <description>Comma-separated list of groups to assign to a new
user</description>
        <name>registration-groups</name>
        <value>group1,group2</value>
    </init-param>
    <init-param>
        <name>portlet-icon</name>
        <value>start-here.png</value>
    </init-param>
    <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>VIEW</portlet-mode>
    </supports>
    <supported-locale>en</supported-locale>
    <portlet-info>
        <title>Services Tutorial</title>
        <short-title>Services</short-title>
        <keywords>tutorial,services,jetspeed-services</keywords>
    </portlet-info>
 </portlet>
```

Jetspeed has an extended descriptor for defining extended portal features and services. Edit the **jetspeed-portlet.xml** found in *src/webapp/WEB-INF/*, and add the following services under the <js:services> element. This tells Jetspeed what services you require:

```
        <js:service name='GroupManager'/>
        <js:service name='PageManager'/>
        <js:service name='PortalAdministration'/>
        <js:service name='PortletRegistryComponent'/>
        <js:service name='RoleManager'/>
        <js:service name='UserManager'/>
```

Create a new JSP page named **services-tutorial.jsp** in the *src/webapp/WEB-INF/view/* directory. Enter the following code:

```
<%@ page language="java" session="true" %>
<%@ page import="javax.portlet.*" %>

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

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

<portlet:defineObjects/>

<portlet:actionURL var="newRolesAction"/>
<br/>
<div class='portlet-section-header'>Services Tutorial Portlet</div>

<form name="servicesTutorialForm" action="<c:out value="${newRolesAction}"/>"
method="post">
<input type="submit" name='action' value="createRoles" class="portlet-form-button"
/>
<input type="submit" name='action' value="createGroups" class="portlet-form-button"
/>
<input type="submit" name='action' value="registerUsers" class="portlet-form-button"
/>
<input type="submit" name='action' value="modifyPages" class="portlet-form-button"
/>
<input type="submit" name='action' value="createSharedPages"
class="portlet-form-button" />
</form>
<c:if test="${message != null}">
<div class='portlet-msg-info'><c:out value="${message}"/></div>
</c:if>
<c:if test="${errorMessage != null}">
<div class='portlet-msg-error'><c:out value="${errorMessage}"/></div>
</c:if>
```

- Create a new package using Eclipse: **com.bluesunrise.portal.portlets.services**
- Create a portlet in the above package named **ServicesTutorialPortlet.java** extending **GenericServletPortlet**.
- Override and implement the **init**, **doView** and **processAction** methods

Add the following data members to the portlet class:

```
    private PortalAdministration admin;
    private PageManager pageManager;
    private RoleManager roleManager;
    private UserManager userManager;
    private GroupManager groupManager;
    protected PortletRegistry registry;

    private List registrationRoles;
    private List registrationGroups;
    private List newRoles;
    private List newGroups;
    private List newUsers;
```

Press **Ctrl-Shift-O** to resolve the two above class imports.

Enter the following code into the init(PortletConfig config) method, replacing whats there:

```
            super.init();
        admin = (PortalAdministration) getPortletContext().getAttribute(
                                CommonPortletServices.CPS_PORTAL_ADMINISTRATION);
                if (null == admin) {
                        throw new PortletException(
                                        "Failed to find the Portal Administration on
portlet initialization");
                }
                userManager = (UserManager) getPortletContext().getAttribute(
                                CommonPortletServices.CPS_USER_MANAGER_COMPONENT);
                if (null == userManager) {
                        throw new PortletException(
                                        "Failed to find the User Manager on portlet
initialization");
                }
                roleManager = (RoleManager) getPortletContext().getAttribute(
                                CommonPortletServices.CPS_ROLE_MANAGER_COMPONENT);
                if (null == roleManager) {
                        throw new PortletException(
                                        "Failed to find the Role Manager on portlet
initialization");
                }
                groupManager = (GroupManager) getPortletContext().getAttribute(
                                CommonPortletServices.CPS_GROUP_MANAGER_COMPONENT);
                if (null == groupManager) {
                        throw new PortletException(
                                        "Failed to find the Group Manager on portlet
initialization");
                }
                pageManager = (PageManager) getPortletContext().getAttribute(
                                CommonPortletServices.CPS_PAGE_MANAGER_COMPONENT);
                if (null == pageManager) {
                        throw new PortletException(
                                        "Failed to find the Page Manager on portlet
initialization");
                }
        registry =
(PortletRegistry)getPortletContext().getAttribute(CommonPortletServices.CPS_REGISTRY_COMPONENT);
        if (null == registry) {
                        throw new PortletException(
                                        "Failed to find the Portlet Registry on
portlet initialization");
                }
        this.newRoles = getInitParameterList(config, "roles");
        this.newGroups = getInitParameterList(config, "groups");
        this.newUsers = getInitParameterList(config, "users");
        this.registrationRoles = getInitParameterList(config, "registration-roles");
        this.registrationGroups = getInitParameterList(config,
"registration-groups");
```

Add this helper function to the class:

```
    protected List getInitParameterList(PortletConfig config, String ipName)
    {
        String temp = config.getInitParameter(ipName);
        if (temp == null) return new ArrayList();

        String[] temps = temp.split("\\,");
        for (int ix = 0; ix < temps.length; ix++)
            temps[ix] = temps[ix].trim();

        return Arrays.asList(temps);
    }
```

Write the doView method:

```
        public void doView(RenderRequest request, RenderResponse response) throws
PortletException, IOException
        {
                request.setAttribute("message", request.getParameter("message"));
                request.setAttribute("errorMessage",
request.getParameter("errorMessage"));
                super.doView(request, response);
        }
```

Write the portletAction method:

```
        public void processAction(ActionRequest request, ActionResponse response)
throws PortletException, IOException
        {
                String action = request.getParameter("action");
                try
                {
                        if (action != null)
                        {
                                if (action.equals("createRoles"))
                                {
                                        String message = "Created " + createRoles()
+ " roles";
                                        response.setRenderParameter("message",
message);
                                }
                                else if (action.equals("createGroups"))
                                {
                                        String message = "Created " + createGroups()
+ " groups";
                                        response.setRenderParameter("message",
message);
                                }
```

```
                                else if (action.equals("registerUsers"))
                                {
                                        String message = "Registered " +
registerUsers() + " users";
                                        response.setRenderParameter("message",
message);
                                }
                                else if (action.equals("modifyPages"))
                                {
                                        String message = "Modified " + modifyPages()
+ " pages";
                                        response.setRenderParameter("message",
message);
                                }
                                else if (action.equals("createSharedPages"))
                                {
                                        String message = "Created " +
createSharedPages() + " pages";
                                        response.setRenderParameter("message",
message);
                                }
                        }
                }
                catch (Exception e)
                {
                        response.setRenderParameter("serviceError", e.getMessage());
                        // TODO: proper logging
                        e.printStackTrace();
                }
        }
```

Implement the undefined methods using the Jetspeed Services...

method	purpose
createRoles	using the roles init param, create new roles with the RoleManager service. If the role already exists, skip it.
createGroups	using the groups init param, create new groups with the GroupManager service. If the group already exists, skip it.
registerUsers	using the users init param, register new users with the PortalAdministration service. If the user already exists, skip it.
modifyPages	using the users init param, modify pages with the PageManager service. If the page doesnt exist, dont create it. Modifications: for user1, create a 1 column collection of 1 portlet, for user2, create a 2 column collection of 2 portlets, for user3 create a 3 column collection of 3 portets
createSharedPages	create a folder named /shared, create a page name /friends.psml. add some portlets to the page. grant public-view security constraint to the folder

Previous