Jetspeed 2 Enterprise Portal v.2.1.2

Project Documentation

Apache Software Foundation **15 July 2007**

Table of Contents

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

1 Essentials
1.1 Features . . . 1
1.2 Getting Started . . . 5
1.3 Jetspeed for Developers . . . 8

2 Building
2.1 From Source . . . 10
2.2 From Maven Plugin . . . 20
2.3 From Eclipse . . . 28

3 Get Jetspeed-2.1.2
3.1 Download . . . 29
3.2 Release Notes . . . 31

4 Documentation
4.1 Documentation Guides . . . 34
4.2 Jetspeed Tutorial . . . 36
4.3 Jetspeed-2 API . . . 37
4.4 Jetspeed-2 Plugin . . . 38

5 About Jetspeed-2
5.1 For Jetspeed-1 Users . . . 44
5.2 Jetspeed-1 Migration Guideline . . . 48
5.3 Supporting Projects . . . 68
5.4 Who Uses J2? . . . 69
5.5 Portlets Community . . . 71
5.6 How to Help? . . . 72

6 Support
6.1 Mailing List . . . 74
6.2 Bug Database . . . 75
6.3 Wiki . . . 76
6.4 Quality Testing . . . 77

7 Translation

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

7.1 Japanese 78

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

1.1 Features

Features

The Apache Portals Jetspeed Team is pleased to present the Jetspeed-2 Open Source Enterprise Portal. This open source project has matured past several releases, and has been fully-compliant with the Portlet Specification 1.0 (JSR-168) since version 2.0. Jetspeed-2 passes the TCK (Test Compatibility Kit) suite and is fully CERTIFIED to the Java Portlet Standard.

Standardized

- Fully compliant with Java Portlet API Standard 1.0 (JSR 168)
- Passed JSR-168 TCK Compatibility Test Suite
- J2EE Security based on JAAS Standard, JAAS DB Portal Security Policy
- LDAP Support for User Authentication

Portal Engine Features

- The Jetspeed Portal - Server-Side Parallel JSR-168 Portlet Rendering Engine
- The Jetspeed Desktop - Web 2.0 Client-Side JSR-168 Portlet Rendering Engine
- Full Support for JSR-168 Portlet Caching and per portlet cache configuration
- Portlet Timeout Tracking with mimimal render time limits
- Portlet Service Manager for automated removal of slow or dead portlets.

Customization Features

- Portal Page Customizer
- Desktop Page Customizer
- Nested Fragment Customization
- Drag and drop moving of portlets
- Resizing of desktop portlet windows
- Portlet Selector with Portlet Categorizations, Full-Text Search of Portlets
- Customizable Themes (Skins)

Security

- Portlet-level Security checks based on Security Constraints or Security Permissions
- Declarative Security Constraints and JAAS Database Security Policy

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

- Fully swappable Security Constraint or Security Permission support
- Jetspeed SSO (Single Sign-on)
- Delegation of Security

Distributed Cluster Support

- Jetspeed Distributed Cluster - support for distributed deployments of the portal on multiple application server platforms
- Distributed Cache for portal components including preferences, registry and portlet entities.
- Distributed invalidation of portlet cache

Foundation Component Architecture

- Spring-based Components and Scalable Architecture
- Configurable Pipeline Request Processor
- Auto Deployment of Portlet Applications
- Jetspeed Component Java API
- Jetspeed AJAX XML API
- PSML: Extended Portlet Site Markup Language
  - Database Persistent
  - Content Management Facilities
  - Security Constraints
- Full security maintenance using LDAP is now supported for many LDAP providers

Portal Core Features

- Runtime Portlet API Standard Role-based Security
- Portal Content Management and Navigations: Pages, Menus, Folders, Links
- PSML Folder CMS Navigations, Menus, Links
- Rules-based Profiler for page and resource location
- Role-based Aggregation of Visible Pages
- Integrates with most popular databases including Derby, MySQL, MS SQL, Postgres, Oracle, DB2
- Client independent capability engine (html, xhtml, wml,vml)
- Internationalization: Localized Portal Resources in 12 Languages
- Statistics Logging Engine
- Portlet Registry
- Full Text Search of Portlet Resources with Lucene
- User Registration
- Forgotten Password
- Rich Login and Password Configuration Management

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

- Custom Portlet Modes and Window States - a "print" PortletMode and "solo" WindowState are now standard supported

Administrative Portlets

- User, Role, Group, Password, and Profile Management
- Portal Site Manager
- Remote Portal Application Deployer (RPAD) - hot deploy portlet applications from remote locations on the Web
- JSR 168 Generic User Attributes Editor
- JSR 168 Preferences Editor
- SSO Manager
- Permission Management (JAAS Security)
- Security Constraints Management
- Portlet Application and Lifecycle Management
- Profiler Administration
- Statistics Reports
- Portlet Out of Service Manager

Web Framework Support and Sample Portlets

- Bridges to other Web Frameworks: JSF, Struts, PHP, Perl, Velocity
- Sample Portlets:
  - RSS, IFrame, Calendar XSLT, Struts Petstore, Bookmark, Database Browser
  - Integration with Display Tags, Spring MVC

Data Migration Features

- XML Import/Export Utility for all Jetspeed database data to support data migration over versions
- All initial portal data seeded with XML
- XML Schemas for all XML content

Portal Design Features

- Deployment Jetspeed Portlet and Page Skins (Decorators) CSS Components
- Configurable CSS Page Layouts
- Easy to Use Velocity Macro Language for Skin and Layout Components

Development Tools

- Automated Maven-1 Build
- Automated Maven-2 Build

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

- Jetspeed-2 Maven Plugin for Custom Portal Development
- Maven-2 Profiles and Archetypes
- AutoDeployment of Portlet Applications, Portal Resources
- Deployment Tools
- Plugin Goals integrated with Auto Deployment Feature
- XML Schemas for PSML, jetspeed-portlet.xml, and Jetspeed XML (seed data)

Other Features

- Installation choice of either Demo Portal or Minimal Starter Portal
- In-depth Jetspeed 2 Tutorial

Application Servers Supported

- Tomcat 5.0.x
- Tomcat 5.5.x
- Jetty
- Websphere 5.1, 6.0
- Geronimo
- JBoss
- Weblogic

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

1.2 Getting Started

Requirements

It is expected that the user is familiar with both the Apache Maven project management tool and the Apache Ant scripting utility.

- Ant 1.5 or higher
- Maven 1.0.2
- Java 1.4.2_02 or higher
- Servlet 2.4 Engine:
  - Tomcat 5.5.x
  - Jetty
  - Websphere 6.0
  - Geronimo
  - JBoss
  - Weblogic

1. Get Maven Ready

If you have not already done so, download and install Maven .

2. What Database do you want

Jetspeed's security model requires a database to authorize users and to retain the user information. Jetspeed security should work with any JDBC 2.0 compliant driver. The following databases are tested:

- Derby
- HSQLDB - Hypersonic SQL
- MySQL
- Oracle
- POstgres
- DB2
- Sybase
- SQL Server

Jetspeed is distributed with the Derby database configured as the default.

The database configuration will be setup during the installation process. If you are not going to use the

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

default Derby database, you need to select another database during installation.

3 Servlet Engines

In theory, Jetspeed 2 portals can be run under any servlet container supporting the 2.4 specification or greater. Successful Jetspeed 2 portal applications have been deployed using:

- Tomcat 5.5.8 or higher
- Jetty
- Websphere
- Geronimo
- JBoss

Tomcat Configuration

Jetspeed 2 can use the Tomcat Manager application for managing portlet applications with the Portlet Application Lifecycle Manager Portlet (PALM). To be able to do so it needs a configured Tomcat user with the predefined 'manager' role in the ${org.apache.jetspeed.server.home}/conf/tomcat-users.xml.

A minimal example tomcat-users.xml can look like:

```
<tomcat-users>
  <role rolename="manager"/>
    <user username="j2deployer" password="xxxxx" roles="manager"/>
</tomcat-users>
```

The attribute values for username and password must correspond to the specified values for `${org.apache.jetspeed.services.autodeployment.user}` *and* `${org.apache.jetspeed.services.autodeployment.user}` *as described above.*

Tomcat 5.5.9 on Windows

To have redeployment and undeployment working properly when using Tomcat 5.5.9 on Windows you have to set the global Context attribute "antiJARLocking" to true.

In ${org.apache.jetspeed.server.home}\conf\context.xml use:

```
<Context antiJARLocking="true">
...
</Context>
```

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Jetty - A Quick Test Environment or a Production Servlet Container

Jetty can be used for a production deployment but it is most commonly used to quickly test customizations without interfering with the production servlet container. It does not require any special configuration.

4. Installing Jetspeed from Source or Binary Distributions

Depending on what you want to do, you have the choice of installing Jetspeed from a binary release or from the sources. If you want to modify the core functionality of Jetspeed or contribute to the development of Jetspeed, you need to work with the sources. If you are only interested in building your own custom enterprise portal, you can start with a binary release of Jetspeed. Most people should start with the binary distribution.

5. Jetspeed build

Your installation instructions will depend on whether you are building from source or building from a binary distribution or installing with Jetspeed-2 installer .

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

1.3 Jetspeed for Developers

Jetspeed For Developers

When developing with Jetspeed, you may be creating portlet applications, or building and creating extensions to the Jetspeed portal. If you ar going to be creating portlet applications, check out this fine e-book for an overall guide to writing portlets:

- Portlets and Apache Portals Book

Here are a few links to get you started developing with the Jetspeed portal itself:

- Getting Started
- Building the Core Jetspeed from Source

Custom Building with Maven Plugins

Jetspeed 2.1 can be built with either Maven-1 or Maven-2. You can actually build your own portal without the Jetspeed source. You will want to customize your Jetspeed build, overriding the skins and themes, adding your own portlet applications and perhaps overriding key components of the portal. To do so, we provide two custom build frameworks: one with Maven-1, the second with Maven-2. With the custom build, you can easily build and create your own Jetspeed powered portal without ever building Jetspeed itself. Many developers still prefer Maven-1. If you are new to Maven, then maybe its best to go with the new version (2).

- Building a Custom Portal with the Maven-1 Plugin
- Jetspeed Tutorial - Building a Custom Portal with the Maven-2 Plugin
- Maven-1 Plugin Documentation
- Maven-2 Plugin Documentation

Jetspeed is built from the command line with Maven. However, you can still develop, compile, debug, remote debug, all from within Eclipse. Eclipse is a good tool for developing portlet applications as well as Jetspeed extensions.

- Developing with Eclipse

To get the binary installation of an official Jetspeed release, go here:

- Getting the Binary Installer

You can checkout from the SVN HEAD from here:

- Checking out the Source Code from Subversion

Get your Javadocs here:

- Portlet API Docs
- Jetspeed API Docs

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

2.1 From Source

1. Naming Conventions and Basic Assumptions

Source Basic Assumptions

- Unless otherwise specified, you should be running all maven build commands from within the Jetspeed directory (if you are just building Jetspeed) or from within your custom portal directory.
- You must use "/" as a file seperator on both *nix and windows, e.g. c:/windows, and /home.

Naming Conventions

Below is a listing of common conventions used within this document.

Variables are represented as ${ *some_variable* }. This may signify a setting in Jetspeed or may represent a setting within your environment. Properties files are also capable of specifying variables within them.

For example, ${org.apache.jetspeed.server.home} references either a property defined further up in the properties file, a variable that has been defined somehwere within the build process or defined in another build file within Jetspeed.

- **${USER_HOME}** : This is the user's home directory. For Windows systems, this generally c:\Documents and Settings\${userName} where ${userName} is the name you use to log into windows.
  ${USER.HOME} is synonymous with ${USER_HOME} within this document.
- **${CATALINA_HOME}** : This is the location of your tomcat installation, e.g. *c:/tomcat* .

Source Subversion (SVN)

Subversion (SVN) is used in the Jetspeed project to manage the source files. SVN is very similar to CVS. For those user's on Windows system who prefer non-command line access we suggest using TortoiseSVN which plugs into your Windows Explorer view. For those using the Eclipse IDE, the Subclipse plugin is available for SVN access.

Maven Setup

We will not go into the specifics of Maven as that is beyond the scope of this document. However, here are a few bits of standard maven jargon we feel is important for you to know.

You will see mention of a *maven repository* in this document. When you install Maven the .maven/ directory is created in your ${USER_HOME) directory.
Under .maven/ you will see a *repository* directory. This is were Maven stores all the jars that it downloads

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

when you run your builds. This is also were Maven puts your jars and wars that you build. They will be stored in a directory structure that has the format of ${groupId}/${projectId}/jars/${projectId}-{$version}.jar for jar files and ${groupId}/${projectId}/wars/${projectId}.war for war files. The ${groupId}, ${projectId} and ${version} variables are discussed later on in this document. Jar and war files will also be created in your project in the /*target* directory.

2. Jetspeed build and maven-plugin Properties

You need to set a few properties.

Creating your own custom portal is very easy with the maven plugin provided by Jetspeed 2. And, it is used when you build jetspeed from source as well. In fact, the jetspeed-2 build procedure is just one example of a custom portal configuration and setup.

The Jetspeed 2 maven-plugin defines default values for most of the properties you can set, but not all. When you download or checkout the jetspeed-2 source tree, you'll see it contains a project.properties file in the root folder overriding and setting some of these properties.

As said before: not all properties are provided with a default value: you *must* specify a few yourself. And you'll most likely want to override some others.

Set or override the build or maven-plugin properties in your `${USER_HOME}/build.properties` file.

Required Portal Configuration Properties

Property	Description	Default value
`org.apache.jetspeed.portal.home`	The folder where the maven-plugin will (re)create or update your custom portal maven project configuration (with goal `j2:portal.conf.project`). This will be where you will be performing all of your future custom portal development. Example: `/home/myportal/`	*no default*
`org.apache.jetspeed.portal.groupId`	The (maven) short name of your portal project group. This value is used for the maven repository folder in which the project artifacts (like the portal war file) is stored. Example: `myprojects`	*no default*
`org.apache.jetspeed.portal.artifactId`	The (maven) short name of your portal project. This value is used for the portal war file and the (portal) web application context name. Example: `myportal`	*no default*
`org.apache.jetspeed.portal.name`	The (maven) full name of your portal project. This value is used by maven for generating JavaDoc titles. Example: `My Test Portal`	*no default*

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Property	Description	Default value
`org.apache.jetspeed.portal.currentVersion`	The current version of your portal project. This value is used by maven as name postfix for the generated artifacts. Example: `1.0`	*no default*

Optional Portal Configuration Properties

The following properties all specify a subfolder of the `${org.apache.jetspeed.portal.home}` location as defined above.

Property	Description	Default value
`org.apache.jetspeed.portal.conf.dir`	The folder where the maven-plugin will generate and copy application server specific configuration files as a tomcat application context descriptor. This folder and its contents is created or updated by plugin goal `j2:portal.conf.tomcat` .	*target/portal-conf*
`org.apache.jetspeed.portal.sql.dir`	The folder where the maven-plugin will generate and copy portal and database server specific sql DDL and DML scripts. This folder and its contents is always (re)created by plugin goal `j2:portal.conf.sql` .	*target/portal-sql*
`org.apache.jetspeed.portal.db.dir`	The folder where the maven-plugin will create its build-in HSQLDB database(s). This folder and its contents is created or updated by plugin goal `j2:start.production.server` or `j2:start.test.server` .	*target/portal-db*
`org.apache.jetspeed.portal.webapp.dir`	The folder where the maven-plugin will copy the standard jetspeed web application resources. This folder and its contents is created or updated by plugin goal `j2:portal.copy.webapp` .	*target/${org.apache.jetspeed.portal.artifactId}*
`org.apache.jetspeed.portal.target.dir`	The folder where the maven-plugin will generate and copy runtime portal configuration files. These configuration files contain values derived from build/plugin properties for the portal and OJB. This folder and its contents is created or updated by plugin goal `j2:portal.conf.jetspeed` and goal `j2:portal.conf.ojb` .	*target/${org.apache.jetspeed.portal.artifactId}*

Required Application Server Properties

Note: The maven-plugin currently only supports the Tomcat Server 5.0.x or 5.5.x

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Property	Description	Default value
`org.apache.jetspeed.server.home`	The root folder of your Tomcat server installation. Example: `${CATALINA_HOME}/` .	*no default*
`org.apache.jetspeed.server.shared`	The location of the shared jars in your Tomcat installation. Example: `${org.apache.jetspeed.server.home}/shared/lib/`	*no default*
`org.apache.jetspeed.deploy.war.dir`	The location of web applications in your Tomcat installation. Example: `${org.apache.jetspeed.server.home}/webapps/`	*no default*
`org.apache.jetspeed.services.autodeploy.tomcat.user`	A Tomcat user with the manager role. Used to access the Tomcat Manager application from within the portal, explained below.	*no default*
`org.apache.jetspeed.services.autodeploy.tomcat.password`	The password of the Tomcat user above. Used to access the Tomcat Manager application from within the portal, explained below.	*no default*
`org.apache.jetspeed.catalina.version.major`	The major version of the Tomcat server you are using: 5 or 5.5 Example: `5.5`	*no default*

Optional Database Server Properties

Jetspeed-2 and its maven-plugin uses, as well as provides, by default a HSQLDB database.

If you want to use a different database you will need to override the following properties:

Property	Description	Default value
`org.apache.jetspeed.production.database.type`	The type of database you are using. Used for sql script generation with Torque. Currently supported databases (with corresponding Torque target database name): • hsql (hypersonic) • oracle (oracle) • mysql (mysql) • postgres (postresql) • mssql (mssql)	hsql
`org.apache.jetspeed.production.database.jdbc.1`	The jdbc connection url	jdbc:hsqldb:hsql://127.0.0.1:9001
`org.apache.jetspeed.production.database.user`	The database user name to connect with.	sa
`org.apache.jetspeed.production.database.password`	The database user its password to connect with.	*empty*
`org.apache.jetspeed.production.database.jdbc.driver`	The jdbc driver class name	org.hsqldb.jdbcDriver
`org.apache.jetspeed.production.jdbc.driver.classpath`	A java classpath style path to the jdbc driver classes or jar(s) needed for connecting to the database. Example: `/lib/ojdbc14.jar;/lib/nls_charset12.jar`	*empty*

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Example: A minimal custom portal configuration

Make sure you have define at least the required properties as described above in your `${USER_HOME}/build.properties` . A minimal custom portal configuration using the default HSQLDB database can be something like:

```
# required portal configuration properties
org.apache.jetspeed.portal.home           = /home/myportal/
org.apache.jetspeed.portal.groupId        = myprojects
org.apache.jetspeed.portal.artifactId     = myportal
org.apache.jetspeed.portal.name           = My Test Portal
org.apache.jetspeed.portal.currentVersion = 1.0

# required application server properties
org.apache.jetspeed.server.home                      = ${CATALINA_HOME}/
org.apache.jetspeed.server.shared                    =
${org.apache.jetspeed.server.home}/shared/lib/
org.apache.jetspeed.deploy.war.dir                   =
${org.apache.jetspeed.server.home}/webapps/
org.apache.jetspeed.services.autodeployment.user     = j2deployer
org.apache.jetspeed.services.autodeployment.password = xxxxx
org.apache.jetspeed.catalina.version.major           = 5.5
```

Note: If you're going to build the default Jetspeed 2 portal directly from the source only the `org.apache.jetspeed.portal.home` *property is required from the set of required portal configuration properties.*

4. Creating a new Portal Application

Now we're going to configure, setup and build a new custom portal application using the Jetspeed-2 maven-plugin.

4.1 Set the maven remote repository lookup configuration

To be able to setup a Jetspeed 2 based portal the maven remote repository lookup needs to be configured properly in your `${USER_HOME}/build.properties` :

```
maven.repo.remote = http://www.bluesunrise.com/maven/,
http://www.ibiblio.org/maven/, \
                  http://dist.codehaus.org/, http://cvs.apache.org/repository
```

Note: the order in which these repositories must be specified is significant!

4.2 Install the Jetspeed 2 maven-plugin

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

The first time, and when you want to upgrade to a newer version of Jetspeed 2, you need to install the maven-plugin as follows:

```
maven -DartifactId=maven-jetspeed2-plugin -DgroupId=org.apache.portals.jetspeed-2
-Dversion=2.1.2 plugin:download
```

Note: you can set the version flag to the specific version you want to install, 2.1.2 is just an example here.

4.3 Generate a new portal project

Once you have the maven-plugin installed and set properties as needed, generate a default portal configuration using the plugin as follows:

```
maven j2:portal.genapp
```

This maven goal actually executes several subgoals which are further described in the maven-plugin documentation itself.

4.4 Further customization of the portal

This section doesn't specify anything to do. After the portal project is generated you can adapt and customize it to your taste by overriding and merging your own configurations and extensions.

You can regenerate or update (part of) your portal project with the `j2:portal.genapp` goal as described in the previous section or use its subgoals directly.

4.5 Build the portal

Once your portal configuration and setup is ready, you can build and install the portal application in your local maven repository (as needed for deployment) using the following standard maven goal from your custom portal project directory (in ${org.apache.jetspeed.portal.home}):

```
maven war:install
```

You are now ready to deploy the new portal application. For this, skip the following section on building the Jetspeed 2 portal from source and continue with the deployment section.

5. Build Jetspeed 2 from source

Build the Jetspeed 2 portal directly from the source is somewhat easier to do but should only be done if

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

you don't want to create a new, customizable portal.

5.1 Setup the Jetspeed 2 source and build properties

The Jetspeed 2 source contains a `project.properties` file which provides all of the required portal configuration settings as described above .

You should ***not*** define any of those properties in your `${USER_HOME}/build.properties` . Instead, you *must* set a `org.apache.jetspeed.project.home` property, specifying the location where you expanded the downloaded source or checked out the source from subversion, like:

```
# required Jetspeed 2 portal configuration property for building from the source
org.apache.jetspeed.project.home = /home/apache/jetspeed-2/
```

Note: you still need to specify the required application server properties as described above .

The Jetspeed 2 project.properties uses this property to define the required `org.apache.portal.home` :

```
org.apache.jetspeed.portal.home = ${org.apache.jetspeed.project.home}
```

So, they are the same when you build the Jetspeed 2 portal from the source.

When you are going to deploy the portal as described further below, you'll see references to the `org.apache.jetspeed.portal.home` which you can translate with the root folder of your Jetspeed 2 source.

If you want to run the testcases when building the Jetspeed 2 sources *and* don't want to use the default HSQLDB test database, you need to override the default test database properties, similar to the production database properties as described above :

- `org.apache.jetspeed.test.database.default.name`
- `org.apache.jetspeed.test.database.url`
- `org.apache.jetspeed.test.database.user`
- `org.apache.jetspeed.test.database.password`
- `org.apache.jetspeed.test.database.driver`
- `org.apache.jetspeed.test.database.drivers.path`

Note: due to outstanding issue JS2-320 you currently ***must*** *use hard coded values for the test database properties.*

Initialize the maven-plugin

Instead of downloading and installing the Jetspeed 2 maven-plugin, you are going to build and install it directly from the source. You will need to repeat this every time you update to a newer version of

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Jetspeed 2 or change its project configuration, the plugin itself or the resources used by the plugin.

Build and install the maven-plugin as follows from the root directory of the Jetspeed-2 source:

```
cd ${org.apache.jetspeed.project.home}
maven initMavenPlugin
```

Optional: start the HSQLDB test database first

If you are going to run the testcases and are using the default HSQLDB database configuration, you will need to start the test database before building Jetspeed 2 in a *separate* console:

```
cd ${org.apache.jetspeed.project.home}
maven j2:start.test.server
```

After the build is finished you can stop the database and close this console with a `Ctrl-C` .

Build the Jetspeed 2 portal and demo portlet applications

For a full build and installation of the portal and the demo portlet applications in your local maven repository run:

```
cd ${org.apache.jetspeed.project.home}
maven allClean allBuild
```

But, if you also want to run the testcases during the build run the following instead:

```
cd ${org.apache.jetspeed.project.home}
maven -Dmaven.test.skip=false allClean allBuild
```

You are now ready to deploy the Jetspeed 2 Portal.

6. Deploy and Run

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Optional: start the HSQLDB production database first

If you are using the default HSQLDB database you need to start it before deploying the portal.

To start the HSQLDB production database run the following in a *separate* console:

```
cd ${org.apache.jetspeed.portal.home}
maven j2:start.production.server
```

You need to have this database running during the deployment and while running the application server. Afterwards you can stop the database and close this console with a `Ctrl-C` .

Note: this is required when using the `j2:quickStart` goal as described below. The Jetspeed 2 maven-plugin provides other (sub)goals which you can use without (re)creating a production database and/or inserting default portal configuration data. See the Plugin documentation for further information about the available goals.

Deploy

We currently only cover deploying to Tomcat 5 or Tomcat 5.5.

Information about deployment to other application servers can be found at the The Jetspeed 2 Wiki .

To deploy a default Jetspeed 2 portal, including the demo portlet applications, run the following in a *separate* console:

```
cd ${org.apache.jetspeed.portal.home}
maven j2:quickStart
```

Note: the maven-plugin documentation described other goals you can use to customize the deployment to your taste.

Run

The final step is starting up your Tomcat server and the portal will automatically install any deployed portlet applications.

Then you can access the portal with your browser at:

```
http://localhost:8080/jetspeed
```

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

or replace "jetspeed" in the above url with the name of you own portal application (`${org.apache.jetspeed.portal.artifactId}`).

Default installed user accounts

With the default Jetspeed 2 portal deployment, several example user accounts are inserted into the portal database with which you can logon to the portal:

username	password	roles
admin	admin	admin, manager, user
manager	manager	manager, user
jetspeed	jetspeed	manager
user	user	user
tomcat	tomcat	

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

2.2 From Maven Plugin

1. Some Basic Information

Basic Assumptions

- Unless otherwise specified, you should be running all maven build commands from within the Jetspeed directory (if you are just building Jetspeed) or from within your custom portal directory.
- You must use "/" as a file seperator on both *nix and windows, e.g. c:/windows, and /home.

Maven

We will not go into the specifics of Maven as that is beyond the scope of this document. If you have never used Maven, you need to read "What is Maven" just to get a sense of the role of Maven in a software development project. If your project involves more than one or two people, you may want to look into Maven in more detail since it does simplify and standardize many of the project management issues that are commonly encountered.

Here are a few bits of standard maven jargon we feel is important for you to know.

You will see mention of a *maven repository* in this document. When you install Maven the .maven/ directory is created in your ${USER_HOME) directory.
Under .maven/ you will see a *repository* directory. This is were Maven stores all of the jars that it downloads when you run your builds. This is also were Maven puts your jars and wars that you build. Jar files will be stored in a directory structure that has the format of ${groupId}/${projectId}/jars/${projectId}-{$version}.jar. The portal war file is stored as ${groupId}/${projectId}/wars/${projectId}.war. The ${groupId}, ${projectId} and ${version} variables are discussed later on in this document. Jar and war files will also be created in your project in the */target* directory.

Variables

Variables are represented as ${ *some_variable* }. Variable names are case sensitive. Variables are defined in several places in a Maven project and according to the Maven Setup section of the on-line Maven User's Guide, the properties files in Maven are processed in the following order:

- ${project.home}/project.properties - Properties that are common to the entire project
- ${project.home}/build.properties - Properties that describe the current release
- ${user.home}/build.properties - Properties that are particular to you

The usage of these files is quite different from the way they are used in a project that develops from the sources since the project in that case is the Jetspeed project rather than your own portal. In that case, the user's build.properties is used much more intensively used to override Jetspeed's parameters.

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Maven processes this sequence of properties files, overriding any previously defined properties with newer definitions. The last definition wins! In this sequence, your ${user.home}/build.properties has the final say in the list of properties files processed.

This list of properties files that Maven processes is called the "standard properties file set".

In addition, System properties are processed after the standard properties files. So, a property specified on the command line using the -Dproperty=value convention will override any previous definition of that property.

For example, ${org.apache.jetspeed.server.home} references either:

- a property defined earlier in the standard properties file set,
- a property specified on the command line of the Maven invokation,
- a variable that has been defined somehwere within the build process or
- a variable defined in another build file within Jetspeed.

Installation dependent locations

The documentation refers to some common locations by the following names:

- **${USER_HOME}** : This is the user's home directory. For Windows systems, this generally c:\Documents and Settings\${userName} where ${userName} is the name you use to log into windows.
  ${user.home} is synonymous with ${USER_HOME} within this document.
- **${CATALINA_HOME}** : This refers to the location of your tomcat installation, e.g. *c:/tomcat* .

These are not used in the Jetspeed configuration but are merely shorthand notations to make the documentation more concise and precise.

Subversion (SVN)

Subversion (SVN) is used in the Jetspeed project to manage the source files. SVN is very similar to CVS. For those user's on Windows system who prefer non-command line access we suggest using TortoiseSVN which plugs into your Windows Explorer view. For those using the Eclipse IDE, the Subclipse plugin is available for SVN access.

2. Overview of the Jetspeed build Process

Setting up a custom portal development project using the binary distribution is a fairly simple process. At the end, you will you have a directory structure and set of files that will simplify building and deploying your own custom portal.

If you want to setup a Jetspeed portal application using Eclipse as your IDE, you should continue reading this page for background material but refer to Building a Jetspeed Enterprise Portal with Eclipse for the actual instructions on using Eclipse for Jetspeed 2 portal development.

Creating your own custom portal is very easy with the maven plugin provided by Jetspeed 2. The steps are:

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

- Download and install the Jetspeed plugin
- Use the plugin to download and generate the Jetspeed binary distribution.
- Customize the properties files to reflect your database installation and local environment.
- Prepare the Application Server
- Build the portal
- Start the Database Server(if required)
- Initialize the Database
- Deploy the default portal using your database
- Test the default portal
- Customize the default portal to include your logo and Portal name
- Generate, deploy and test your custom Portal

The Jetspeed 2 maven-plugin defines default values for most of the properties you can set, but not all. As you customize the portal, you will override others.

3. Installation Steps

3.1 Download the Jetspeed Plugin

3.1.1 Set the maven remote repository lookup configuration

Now we're going to configure your ${user.home}build.properties file to give Maven the information that it needs to download the Jetspeed-2 maven-plugin. The base directory where you are going to build your portal needs to be specified to Maven as well as the the maven remote repository need to be configured properly in your `${USER_HOME}/build.properties` :

```
basedir = c:/myportal
maven.repo.remote = http://www.bluesunrise.com/maven/,
http://www.ibiblio.org/maven/, \
http://dist.codehaus.org/, http://cvs.apache.org/repository
```

Note: the order in which the repositories are specified is significant!

3.1.2 Install the Jetspeed 2 maven-plugin

The first time, and when you want to upgrade to a newer version of Jetspeed 2, you need to install the maven-plugin as follows:

```
maven -DartifactId=maven-jetspeed2-plugin -DgroupId=org.apache.portals.jetspeed-2
-Dversion=2.1.2 plugin:download
```

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Note: you must set the version to the specific version you want to install, "2.1.2" is just an example.

3.2 Download the default Jetspeed portal project

Once you have the maven-plugin installed and set properties as needed, generate a default portal configuration using the plugin as follows:

```
maven j2:portal.genapp
```

This maven goal actually executes several subgoals which are further described in the maven-plugin documentation itself.

3.3 Customize the properties files

You can fill in as much of the project information in the project.xml file as you want. This will depend on how you intend to use Maven as a project management tool and is beyond the scope of this document. The information in the project.xml file distributed with Jetspeed reflects the Jetspeed development project.

You can now customize the properties files to reflect your database installation and local environment.

The `${basedir}project.properties` file provided by the Jetspeed developers includes all of the variables that are common to all portals based on Jetspeed. You should not have to change these.

The project properties are described in the Maven Properties Reference documentation. We have already filled in the value for basedir and maven.repo.remote in previous steps. You can fill in as much of the project information in the `${basedir}project.xml` file as you want. This will depend on how you intend to use Maven as a project management tool and is beyond the scope of this document. The Maven site has all of the information that you need to use Maven successfully.

The configuration of your specific properties needs to be done before we can build the portal. Review the definition of the configuration properties described in Basic Configuration Parameters .

In the case of a binary build, the basic configuration properties can be placed in `${basedir}build.properties` . A minimal custom portal configuration using the default Derby database can be something like:

```
# required portal configuration properties
org.apache.jetspeed.portal.home           = /home/myportal/
org.apache.jetspeed.portal.groupId        = myprojects
org.apache.jetspeed.portal.artifactId     = myportal
org.apache.jetspeed.portal.name           = My Test Portal
org.apache.jetspeed.portal.currentVersion = 1.0
```

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

If you are not using the Derby database that comes pre-configured in the `${basedir}project.properties` file, you also need to define the database parameters in the `${basedir}build.properties` file. Refer to the Database Configuration section for a description of the variables required.

3.4 Prepare the Application Server

Before running the portal, we need to prepare the Application server to run a Jetspeed portal. This consists of telling Jetspeed where the application server expects files to be placed and what authentication values are required to request service from the Application Server's management tools. There may also be modifications to the server configuration so be sure to read the Application server configuration documentation .

Verify that you made the Application Server changes suggested in the overview of the Getting Started documentation.

A minimal custom portal configuration using the Tomcat 5.5 Application Server could be something like:

```
# required portal configuration properties
org.apache.jetspeed.portal.home           = /home/myportal/
org.apache.jetspeed.portal.groupId        = myprojects
org.apache.jetspeed.portal.artifactId     = myportal
org.apache.jetspeed.portal.name           = My Test Portal
org.apache.jetspeed.portal.currentVersion = 1.0

# required application server properties
org.apache.jetspeed.server.home                      = ${CATALINA_HOME}/
org.apache.jetspeed.server.shared                    =
${org.apache.jetspeed.server.home}/shared/lib/
org.apache.jetspeed.deploy.war.dir                   =
${org.apache.jetspeed.server.home}/webapps/
org.apache.jetspeed.services.autodeployment.user     = j2deployer
org.apache.jetspeed.services.autodeployment.password = xxxxx
org.apache.jetspeed.catalina.version.major           = 5.5
```

3.5 Build the portal

Once your portal configuration and setup is ready, you can build and install the portal application in your local maven repository (as needed for deployment) using the following standard maven goal from your custom portal project directory (in ${org.apache.jetspeed.portal.home}):

```
maven war:install
```

You are now ready to deploy the new portal application. For this, skip the following section on building

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

the Jetspeed 2 portal from source and continue with the deployment section.

3.6 Start the Database Server

You need to make sure that your database server is running. If you are not using the default Derby database, you need to make sure that it is running and that the user that will own the Jetspeed tables is setup and ready for use. Refer to the Database Configuration section for more information.

You need to have this database running during the deployment and while running the application server.

Note: this is required when using the `j2:quickStart` goal as described below. The Jetspeed 2 maven-plugin provides other (sub)goals which you can use without (re)creating a production database and/or inserting default portal configuration data. See the Plugin documentation for further information about the available goals.

3.7 Initialize the Database

The database's tables and initial data needs to be loaded prior to Jetspeed being deployed. The maven-plugin includes a number of goals that can be used to manage the database. The easiest way to load the tables and deploy the application is to run the j2:quickstart goal.

```
cd ${org.apache.jetspeed.portal.home}
maven j2:quickStart
```

This can only be run once without a bit of a cleanup afterwards since it defines the tables and loads the data as part of starting the application. If you have an error and you want to run it again, you must make sure that the tables and data are cleaned out either by manually dropping the tables in the database or by using the j2:db.drop.production Maven goal.

The j2:quickstart currently only covers deploying to Tomcat 5 or Tomcat 5.5 application servers.

Information about deployment to other application servers can be found at the The Jetspeed 2 Wiki .

To deploy a default Jetspeed 2 portal, including the demo portlet applications, run the following in a *separate* console:

```
cd ${org.apache.jetspeed.portal.home}
maven j2:quickStart
```

Note: the maven-plugin documentation described other goals you can use to initialize the database.

3.8 Deploy the default portal using your database

The j2:quickstart task does this for you. If you have used another Maven goal to initialize the database,

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

then you can deploy the portal by using:

```
cd ${org.apache.jetspeed.portal.home}
maven j2:fullDeploy
```

The j2:fullDeploy goal currently only covers deploying to Tomcat 5 or Tomcat 5.5 application servers.

Information about deployment to other application servers can be found at the The Jetspeed 2 Wiki .

3.9 Test the default portal

The final step in getting the default portal running is starting up your Tomcat server. The portal will automatically install any deployed portlet applications.

Then you can access the portal with your browser at:

```
http://localhost:8080/jetspeed
```

or replace "jetspeed" in the above url with the name of your own portal application (`${org.apache.jetspeed.portal.artifactId}`).

If you see a running Jetspeed portal, we have succeeded in getting the software installed and working.

With the default Jetspeed 2 portal deployment, several example user accounts are inserted into the portal database with which you can logon to the portal:

username	password	roles
admin	admin	admin, manager, user
manager	manager	manager, user
jetspeed	jetspeed	manager
user	user	user
tomcat	tomcat	

3.10 Customize the default portal to include your logo and Portal name

Now that the default portal is working we can try a small customization to test out the customization process. We are going to change the logo and portal name.

3.10.1 Creating your customization area

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

The first step is to create a directory to hold your customized files. This will help you to preserve your changes when you install new versions of jetspeed. Create a directory in the top level of the portal home.

```
cd ${org.apache.jetspeed.portal.home}
mkdir customized
```

Make a copy of the build.properties file that you have already modified.

You may also want to make a directory to hold the original files before you modify them. This is not strictly requirred since you can always reload the distribution. However it might be handy to keep a copy for reference or to quickly get back to the state before you made changes.

3.10.2 Making some simple cuistomizations

We are going to change the logo and the portal name to test customization. You can make you own logo to replace the Jetspeed logo. Take a look at the ${basedir}/src/webapp/images/logo.png to get the size and to verify the background colour. Make your own logo or copy the testlogo.png file to your `${basedir}/customized` directory.

3.11 Generate, deploy and test your custom Portal

```
cd ${org.apache.jetspeed.portal.home}
maven j2:fullDeploy
```

You can access the revised portal with your browser at:

```
http://localhost:8080/jetspeed
```

or replace "jetspeed" in the above url with the name of your own portal application (`${org.apache.jetspeed.portal.artifactId}`).

You should see the new name and the new logo on the front page.

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

2.3 From Eclipse

Developing with Eclipse

The Eclipse Classpath

Compiling, debugging, external dependencies, source code completion, searching, auto imports, all rely on a properly configured classpath. When you first create a project, a .classpath file is created in the projects root directory. With the Jetspeed source, we provide you with a ready-to-use Eclipse .classpath file. We have already configured the relative source directories for you. Eclipse provides a .classpath GUI editor from the Project->Properties menu option.

JAR files and the Maven repository

Jetspeed requires quite a few JAR files to be able to compile. The .classpath file that comes with Jetspeed is setup to get its JAR files out of a local Maven repository. You can see all the JAR file dependencies from Eclipse. Go to Project->Properties->Java Build Path->Libraries. Notice all the JAR files are configured as VARIABLE library entries. Take one example:

```
MAVEN_REPO/commons-lang/jars/commons-lang-2.0.jar
```

The Variable is portion is MAVEN_REPO. The Extension portion is /commons-lang/jars/commons-lang-2.0.jar Eclipse locates the JAR dependency from a Variable location root. In order for this classpath to work correctly, the variable root is dependent on a Maven-1 local repository file structure.

To configure the MAVEN_REPO variable, go to Window->Preferences->Java->Build Path->Classpath Variables, click on New, and define a new variable named MAVEN_REPO, pointing it out the root of your local Maven-1 repository, usually someplace like your $$HOME/.maven/repository

Debugging with Eclipse

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

3.1 Download

Download Jetspeed-2 Distribution

Jetspeed-2 is distributed in several formats for your convenience and distributed under the Apache License, version 2.0 .

Jetspeed-2.1.2 Installer Distribution

Distribution	Mirrors	Checksum	Signature
Jetspeed-2 Standard with only the required Administrative Portlets	jetspeed-2.1.2-installer.jar	here	here
Jetspeed-2 Demo With lots of demo Portlets: RSS, JSF, JPetstore and many more	Jetspeed-2.1.2-demo-installer.jar	here	here

Both the installers support the following databases for Jetspeed: Derby (default), DB2, MySQL, MSSQL, Oracle, PostgreSQL, SapDB, as well as manual (do it yourself) configuration of other databases.

Furthermore, through a provided Ant script after installation, reinitializing or switching to another database is a simple one step operation.

Complete instructions for getting started using the installer is available here .

Jetspeed-2.1.2 Source Distribution

	Mirrors	Checksum	Signature
jetspeed-2.1.2-src.tar.bz2	here	here	here
jetspeed-2.1.2-src.tar.gz	here	here	here
jetspeed-2.1.2-src.zip	here	here	here

Jetspeed-2.1.2 Full Distribution (binaries, src and generated website)

	Mirrors	Checksum	Signature
jetspeed-2.1.2.tar.bz2	here	here	here

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

	Mirrors	Checksum	Signature
jetspeed-2.1.2.tar.gz	here	here	here
jetspeed-2.1.2.zip	here	here	here

System Requirements

The list of systems requirements for Jetspeed-2 is available here .

Migrating Guide

Important information for migrating existing Jetspeed-2.0 and Jetspeed-2.1 installations to Jetspeed-2.1.2 is provided in the migration guide .

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

3.2 Release Notes

Release Notes - Jetspeed 2 - Version 2.1.2

The list below outlines the issues that are addressed with release 2.1.2 For a full list of features, see the features list .

And the release notes of the previous major release 2.1 are available here: release notes 2.1 .

Bug

- [JS2-282] - Error when session expires and portlet window is maximized
- [JS2-471] - Document Derby as the default database
- [JS2-484] - UserDetailsPortlet doesn't "see" newly added roles until after logging out and in again
- [JS2-502] - cannot disable user
- [JS2-512] - Profiler admin portlet can not display other language except English.
- [JS2-525] - Roles and Groups appear in the User Details portlet when having been deleted
- [JS2-566] - Tapestry portlet's header doesn't display in jetspeed
- [JS2-580] - localization SSO Details portlet
- [JS2-657] - Installer fails with firewall
- [JS2-660] - Request attribute not available in jsp when using the JetspeedPowerTool
- [JS2-661] - Error in the antinstall-config script
- [JS2-665] - Duplicate Objects creation
- [JS2-667] - Portlet Selector is not returning to the correct page after navigation
- [JS2-668] - Adding Portlets to multiple layouts always adds to the top level layout
- [JS2-669] - Site Manager Admin portlet does not allow copying of a resource into the same folder where it exists
- [JS2-670] - Fragment Security Constraints only check View Mode
- [JS2-674] - Site component fails on profile navigations for subsites
- [JS2-675] - Site Manager Admin portlet: cannot view pages located via profiling rules with navigations or controls
- [JS2-681] - Login portlet doesn't run in Tomcat ROOT context
- [JS2-682] - Jetspeed thread waiting to lock for infinity time
- [JS2-687] - jetspeed deployment engine removes WEB-INF/tld/portlet.tld from portlet app web archive
- [JS2-688] - Unable to use Jetspeed services

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

- [JS2-689] - Spring Bean Factory creation of Prototype (non-singleton) beans causes serious performance degradation under load
- [JS2-690] - Caching issue with Print Mode
- [JS2-692] - Fragment ids are not automatically created, causing runtime errors
- [JS2-693] - Portal Site Manager error in java script for button "view"
- [JS2-694] - actionResponse.sendRedirect("some psml page.psml") fails on the desktop
- [JS2-696] - Creating actionURLs on the desktop with javascript: tags in them fails
- [JS2-697] - Maximized mode overlaps as popup on desktop
- [JS2-698] - Minimized mode functions only in un-tiled state on desktop
- [JS2-699] - When going back from view mode to edit mode, icon is not always updated on desktop
- [JS2-703] - Remove Print Mode Window Decoration on Desktop
- [JS2-705] - Desktop window dragging mouse position offset bug
- [JS2-714] - Filter Admin users from delegated-security portlets
- [JS2-717] - MenuElement interface does not support getUrl method
- [JS2-718] - forgot pass portlet: password reset link does not work
- [JS2-719] - Default ehcache configuration is setup for distributed operation which will fail the portal to startup when no network is available
- [JS2-725] - Dociumentation mismatch
- [JS2-727] - Appy Findbugs patches
- [JS2-731] - DESKTOP_ATTRIBUTE should be DESKTOP_CONTEXT_ATTRIBUTE
- [JS2-733] - Desktop: non-movable portlets are deletable
- [JS2-735] - Jetty-6 ConcurrentModificationException on logout
- [JS2-737] - Desktop doesn't work on IE 6.0
- [JS2-745] - File System Page Manager does not accept folders with dots

Improvement

- [JS2-584] - enable adding Velocity context objects without recompilation
- [JS2-672] - Add createProperty API to the Preferences Provider component
- [JS2-673] - Set sql.src.path value in the project.properties
- [JS2-680] - Folder configuration form at a page edit area
- [JS2-683] - Folder/Page customizer improvement
- [JS2-685] - Add functionality to AJAX API to information about users
- [JS2-695] - The Desktop does NOT support the no-action layouts
- [JS2-700] - Display loading in progress message from desktop
- [JS2-701] - Package and compress the Jetspeed Desktop Javascript
- [JS2-702] - Optimize desktop menu loading
- [JS2-707] - When creating a new user, give option to create inside a subsite

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

- [JS2-709] - Update to latest dependencies
- [JS2-711] - Support JSP decorators as well as Velocity
- [JS2-712] - Create new servlet session upon login (configurable)
- [JS2-713] - Put a hard-limit on session time-out for portal sessions
- [JS2-721] - Ability to determine if a Menu Option has a default page or not
- [JS2-723] - Option to configure DB PSML from the installer
- [JS2-739] - Improve Algorithms for Resource Validation and Template Localization
- [JS2-740] - Allow overriding the default Spring assembly without having to modify it
- [JS2-741] - More GroovyPortlet demos using the new Portals Bridges GroovyPortlet
- [JS2-747] - A valve creating template pages when a user logs on first.

New Feature

- [JS2-317] - Virtual Portal Implementation
- [JS2-691] - Allow user to customize all his pages and portlets (in user home folder) at once
- [JS2-716] - PSML and XML Import / Export Admin Portlet
- [JS2-724] - Audit logs for administrative actions
- [JS2-728] - Provide a Portals Bridges common.PortletResourceURLFactory implementation for Jetspeed
- [JS2-729] - Preliminary Portlet API 2.0 ResourceURL support allowing full response control like for cookies and compressed output streams
- [JS2-732] - A GroovyPortlet demo using the new Portals Bridges GroovyPortlet
- [JS2-743] - Maximize on Edit

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

4.1 Documentation Guides

Documentation Guides

Getting Started

- Getting started guide
- Getting started with Jetspeed-2 source guide
- Getting started with Jetspeed-2 binaries guide
- Getting started with Jetspeed-2 installer

Guides to Portal Concepts

- Guide to Jetspeed-2 pipeline
- Guide to decorators
- Guide to layouts

Configuration Guides

- Guide to configuration properties
- Guide to database configuration
- Guide to application servers configuration
- Guide to configuring Jetspeed-2 security
- Guide to using Jetspeed-2 single sign-on
- Guide to using NTLM Authentication
- Guide to defining user attributes (PLT.17 user information configuration)
- Guide to using profilers
- Guide to Aggregation
- Guide to Subsites
- Guide to Migration to 2.1.2

Portal Development Guides

- Guide to portal design
- Guide to PSML
- Guide to declarative security through PSML

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

- Guide to declarative menus in PSML
- Guide to working with a Jetspeed-2 site
- Guide to localization with Jetspeed-2
- Guide to Jetspeed-2 AJAX API
- Guide to a very simple portlet with Jetspeed-2
- Guide to a very simple velocity portlet with Jetspeed-2
- Guide to a very simple jsf portlet with Jetspeed-2
- Guide to portlet bridges
- Guide to Profiling IP Addresses

Jetspeed-2 Development Guide

- Guide to Jetspeed-2 development
- Guide to helping with Jetpseed-2
- Tomcat SSO and Cross Context Webapps Guide

Guides to Jetspeed-2 Tools

- Guide to Jetspeed-2 Power Tools
- Guide to Jetspeed-2 Portlet Application Deployment
- Guide to Jetspeed-2 Maven Plugin

Components Guides

- Guide to Jetspeed-2 component architecture
- Guide to Jetspeed-2 directory structure

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

4.2 Jetspeed Tutorial

http://portals.apache.org/tutorials/jetspeed-2/

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

4.3 Jetspeed-2 API

http://portals.apache.org/jetspeed-2/multiproject/jetspeed-api/apidocs/index.html

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

4.4 Jetspeed-2 Plugin

Plugin Overview

Adding functionality to Maven is done through the Maven plugin mechanism. Maven comes shipped with numerous plugins and provides an extensible framework for writing custom plugins. Details on custom Maven plugins can be found in the Writing a Plugin section of Maven's web site.

Jetspeed 2 has developed a custom Maven plugin that centralizes most common build goals required to build a Jetspeed 2 based portal application. This provides many benefits:

- Better reusability of common build goals. Developers creating a new portal application can leverage the Jetspeed 2 Maven plugin for common build operations.
- The ability to quickly get started with a portal application. With the goal `j2:portal.genapp` a new portal application can be created. The developer of the new application can reuse the Jetspeed 2 Maven plugin goals for common build operations for quickStart, portlet deployment, etc.
- Preparation for future migration to Maven 2 (M2). With M2, custom goals are encapsulated in plugins, maven.xml is deprecated. By centralizing most of the Jetspeed 2 build goals to the Jetspeed 2 Maven plugin, migration to M2 should be much easier.

Portal Application creation and configuration Goals

Creating a new Portal Application

Goal	Description
`j2:portal.genapp`	Generates or updates a custom portal application. Checkout the Getting Started document for basic usage of this goal.
`j2:portal.genapp.minimal`	Works similar to `j2:portal.genapp`. However, it will only copy the following directories/files from `WEB-INF/pages`. • /WEB-INF/pages/Administrative/** (all contents) • /WEB-INF/pages/page.security

Configuring and updating a Portal Application

The `j2:portal.genapp` goal above is actually no more than a wrapper around several (sub)goals which can also be used individually to update and configure your portal application:

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Goal	Description
`j2:portal.copy.webapp`	Copies the (static) Jetspeed 2 portal web resource from the plugin to the `${org.apache.jetspeed.portal.webapp.dir}` folder. Running this goal again will not clear out previous resources but *will* overwrite existing resources. If you need to upgrade to a newer version of the Jetspeed 2 portal you should clear out these resources yourself first. By default, the target folder is configured within the maven default target folder and running the `clear` goal will do exactly that.
`j2:portal.copy.webapp.minimal`	Similar to `j2:portal.copy.webapp` However, it will only copy the following directories/files from `WEB-INF/pages`. • /WEB-INF/pages/Administrative/** (all contents) • /WEB-INF/pages/page.security
`j2:portal.conf.project`	Creates a new maven or updates an existing portal project configuration in the `${org.apache.jetspeed.portal.home.dir}` folder. It creates a hiearchy of 5 maven project files: • `project-info.xml` • `core-build.xml extends project-info.xml` • `jetspeed-components.xml extends core-build.xml` • `full-portal.xml extends jetspeed-components.xml` • `project.xml extends full-portal.xml` Of the above files, the first and the last (project-info.xml and project.xml) may be modified to provide additional project information and configuration and will not be updated by this goal again. Only the other 3 files will be rewritten by this goal. If you need to upgrade to a newer version of the Jetspeed 2 portal your own customization will be preserved. Additionally, you can add your own maven.xml and/or project.properties or build.properties to further customize your portal. These also will be preserverd when you run this goal again.
`j2:portal.conf.sql`	Generates the portal sql schema DDL for the configured database(s) under the `${org.apache.jetspeed.portal.sql.dir}`, as well as copies over statically defined common and selected database specific sql DML and DDL (possibly overriding generated DDL). The content of the sql target folder is cleared out first when this goal is run.
`j2:portal.conf.ojb`	Copies the OJB configuration, filtered for the currently selected production database to the `${org.apache.jetspeed.portal.target.dir}` folder. As default, the above target folder is configured under the default maven war target folder. The maven `clear` goal will also remove this filtered OJB configuration.
`j2:portal.conf.ldap`	Copies the LDAP configuration, to the `${org.apache.jetspeed.portal.target.dir}` folder.
`j2:portal.conf.jetspeed`	Copies the filtered `jetspeed.properties` portal configuration to the `${org.apache.jetspeed.portal.target.dir}` folder. As default, the above target folder is configured under the default maven war target folder. The maven `clear` goal will also remove this filtered `jetspeed.properties` file.
`j2:portal.conf.tomcat`	Copies a filtered Tomcat context descriptor, containing current database connection configuration to the `${org.apache.jetspeed.portal.conf.dir}` folder. Based on the `${org.apache.jetspeed.catalina.major.version}` setting, a Tomcat 5.0.x or 5.5.x type template context descriptor will be used. The filtered Tomcat context descriptor will be copied to the Tomcat server by the `j2:portal.deploy` goal. If you need to change the Tomcat major version and/or database connection configuration, you need to run this goal again before (re)deploying your portal. As default, the above target folder is configured under the default maven target folder. The maven `clear` goal will remove this filtered context descriptor file.

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Portal Application Deployment Goals

Quickstart deployment goals

Several goals are available for quickly deploying the Portal Application together with a predefined set of Portlet Applications and optionally with creating and seeding the portal database.

Goal	Description
`j2:doStart`	A generic goal for deploying the portal application and setting up the required dependencies for the configured Tomcat Server (shared libraries, portal application context, etc.). This goal requires the plugin property `deployType` to be set. The default value is `"j2:fullDeploy"` (see below). If plugin property `recreateDB` is set, goal `j2:db.recreate` is invoked. All existing Jetspeed 2 standard and demo portlet applications are removed through goal `j2:remove.wars`. The shared dependecies are copied to the Tomcat Server with goal `j2:copy.shared.deps`. And finally, the set `deployType` plugin property value is used to run a specific deploy goal.
`j2:quickStart`	Invokes `j2:doStart` with `deployType="j2:fullDeploy"` and `recreateDB=true`.
`j2:nodbQuickStart`	Invokes `j2:doStart` with `deployType="j2:nodbfullDeploy"` and `recreateDB=false`.
`j2:basicStart`	Invokes `j2:doStart` with `deployType="j2:basicDeploy"` and `recreateDB=true`.
`j2:nodbBasicStart`	Invokes `j2:doStart` with `deployType="j2:nodbBasicDeploy"` and `recreateDB=false`.
`j2:minStart`	Invokes `j2:doStart` with `deployType="j2:minDeploy"` and `recreateDB=true`.
`j2:nodbMinStart`	Invokes `j2:doStart` with `deployType="j2:nodbMinDeploy"` and `recreateDB=false`.

Deployment supporting Goals

Goal	Description
j2:remove.wars	Removes the portal, all standard and demo portlet applications (see `j2:fullDeploy` below) and their context descriptors (if any) from the Tomcat Server.
`j2:catalina.base.shared`	Copies all base jars necessary for the common portlet container.
`j2:catalina.shared`	Copies all jars necessary for common container
`j2:copy.shared.deps`	Wrapper goal invoking `j2:catalina.base.shared` and `j2:catalina.shared`.

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Goal	Description
`j2:portal.deploy`	Deploys the portal application only and its dependencies to the Tomcat Server, but no portlet applications. First, it removes the current portal installation with goal `j2:remove.wars`. Then it copies and expands the build portal war from the local maven repository to the application server. And it copies a Tomcat context descriptor for the portal (see also goal `j2:portal.conf.tomcat`). Finally, it installs the shared dependencies with goal `j2:copy.shared.deps`.

Standard and Demo Portlet Application deployment goals

Goal	Description
`j2:deploy`	Generic goal to deploy a portlet application identified by property `${maven.war.final.name}`. The portlet application is searched for in the Jetspeed2 group of the local maven repository. When searching portlets in the repository, this goal searches for the portlet application given the Jetspeed 2 version number configured in the plugin. The name of the file searched in the repository following the convention `${maven.war.final.name}-${jetspeed.version}.war` and is deployed as `${maven.war.final.name}.war`.
`j2:pam.layoutdeploy`	Deploys Jetspeed local layout portlet application.
`j2:pam.admindeploy`	Deploys Jetspeed Administration portlet application.
`j2:pam.tsdeploy`	Deploys Pluto Test Suite portlet application.
`j2:pam.strutsdeploy`	Deploys the struts mailreader demo portlet application.
`j2:pam.jpetstoredeploy`	Deploys the iBatis JPetstore based demo portlet application.
`j2:pam.jsfdeploy`	Deploys the JSF demo portlet application which uses Jetspeed generic JSF portlet bridge.
`j2:pam.jsfmyfacesdeploy`	Deploys the JSF demo portlet application which uses MyFaces native JSF portlet bridge.
`j2:pam.phpdeploy`	Deploys the Jetspeed PHP bridge demo portlet application.
`j2:pam.perldeploy`	Deploys the Jetspeed Perl bridge demo portlet application.
`j2:pam.rssdeploy`	Deploys the RSS feed demo portlet application.
`j2:nodbMinDeploy`	Deploys the portal using the `j2:portal.deploy` goal and only the layout and admin portlets using `j2:pam.layoutdeploy` and `j2:pam.admindeploy`.
`j2:minDeploy`	The same functionality as `j2:nodbMinDeploy` and additionaly seeds the portal database using the `j2:db.entities` goal.
`j2:nodbfullDeploy`	The same functionality as `j2:nodbMinDeploy` but additional deploys all other demo portlet applications (see full list above).
`j2:fullDeploy`	The same functionality as `j2:nodbFullDeploy` and additionaly seeds the portal database using the `j2:db.entities` goal.

Database Management Goals

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Goal	Description
`j2:start.production.server`	Starts the HSQLDB production database for usage by the portal. This goal is optional to those who use the default embedded Derby database.
`j2:start.test.server`	Starts the HSQLDB test database to be used for the testcases during the build of Jetspeed 2. This goal is optional for those who use the default embedded Derby database for testing Jetspeed 2.
`j2:db.create.test`	Creates the test database tables. If using the HSQLDB database, it should be started first with goal `j2:start.test.server`. Existing portal tables are dropped first. The first time, this will lead to "table does not exist" error messages but they can be (and are) ignored.
`j2:db.create.production`	Creates the production database tables. If using the HSQLDB database, it should be started first with goal `j2:start.production.server`. Existing portal tables are dropped first. The first time, this will lead to "table does not exist" kind of error messages but they can (and are) ignored.
`j2:db.recreate`	Recreates the production database using the `j2:db.create.production` goal but first (re)generates the sql scripts using `j2:portal.conf.sql`.
`j2:db.drop.test`	Drops the test database portal tables.
`j2:db.drop.production`	Drops the production database portal tables.
`j2:db.entities`	Populates the users information for the default PSML configuration configured with Jetspeed 2.

LDAP Management Goals

Goal	Description
`j2:start.ldap.server`	Starts the default Apache Directory Server and load the default *apacheds-server.xml* configuration.

Auxillary Jetspeed Components Deployment Goals

Generic set of goals for redeploying a specific Jetspeed component.

Goal	Description
`j2:jar.deploy`	Deploys a Jetspeed core component from the local maven repository to the deployment directory.
`j2:jar.deploy.shared`	Deploys a Jetspeed core component shared library from the local maven repository to the Tomcat shared library directory.
`j2:deployDecorations`	Deploys all decoration files from the WEB-INF/decorations directory to the deployed portal.
`j2:deployTemplates`	Deploys all template files from the WEB-INF/templates directory to the deployed portal.

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

5.1 For Jetspeed-1 Users

For Jetspeed-1 Users

Jetspeed-2 is a new project, written from groundup and does not have any dependencies on Jetspeed-1. Jetspeed-2 is based on industry standards, designed for high-volume enterprise portals applications. The foremost difference is Jetspeeds Component Oriented Architecture, all assembled together with Spring. Components replace Turbine services with a standardized component model. Deployment of new portlet applications, which was completely missing in Jetspeed-1, is implemented to the Portlet API specification. Turbines file-based configuration for properties are replaced managed components. Jetspeed-2 is fully decoupled from the legacy projects that were intertwined in the Jetspeed-1 architecture.

Whats New in Jetspeed-2

- 1. Fully Compliant with Java Portlet API Standard
- 2. Separation of Portlet Applications From Portal
- 3. Live Deployment Model for Portlet Applications and Portal Layouts
- 4. Spring Component Based Architecture
- 5. Multi-threaded Portlet Aggregation Engine
- 6. Scalable Architecture
- 7. Pipeline-based Request Processing
- 8. JAAS Security
- 9. Bridges Integration with Struts, JSF, PHP, Perl, Velocity
- 19. CMS-based Site Navigations

Whats the same in Jetspeed-2

Not much.

In fact Jetspeed-2 does not re-use any of the code in Jetspeed-1. Some concepts are continued in Jetspeed-2, but with new design and implementations. The table below shows some of the concepts continued in Jetspeed-2 from Jetspeed-1. Note that even though the concepts are continued, they are have changed, in some cases significantly:

- 1. PSML - Portlet Structured Markup Language. Defines the layout of portlets on a page. While the purpose is still the same, the XML format has changed. Porting is possible, requires a migration tool. PSML now fits into an overall Jetspeed Navigation Site as a page-type resource. No PSML porting tool is currently available. However, an XSLT transform could be a good choice.
- 2. Portal Wide Security Policy and Constraints - Jetspeed-2 has two kinds of security mechanisms:

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

JAAS-based security policies, and declarative security constraints much like Jetspeed-1 constraints. Where as Jetspeed-1 constraints were limited to PSML, Jetspeed-2 declarative security constraints are also applied to folders and links.

- 3. Portlets - Portlets now must adhere to the new Portlet API. No porting tool is currently available. The Jetspeed-1 Portlet API will not be continued in Jetspeed-2.
- 4. Turbine Services are out (Fulcrum). Jetspeed-2 is based on Spring components.
- 5. Registries - The Jetspeed-1 Registries are discontinued in Jetspeed-2. All portlets are now stored in a Registry database in Jetspeed-2. No porting tool is available. Recommend converting your portlets to JSR-168 portlets, packaging all portlets in a portlet application, and deploying as standard WAR file. Other registries are all deprecated.
- 6. JSP and Velocity Templates - Templates can be re-used to some extend. Any references to Rundata or any other Turbine or Jetspeed-1 tools or tags must be converted.
- 7. Controls and Controllers - These concepts have changed, and are now called decorators and layouts. The Turbine module concept, which backed controls and controllers, is no longer supported. Layouts and decorators are now only implemented as portlets, or as just plain markup. Layouts and templates can be deployed to the portal as a deployable unit.
- 8. Jetspeed Configurations and Jetspeed Component Assemblies replace Property Files. Component (services) should be assembled, not defined in property files. Many of the features in Jetspeed-1 were represented as read-only properties in the Jetspeed-1 static property files. Jetspeed-2 components can be configured with JMX.

Turbine Gone

Jetspeed-1 is tightly coupled to the Turbine MVC-2 framework, and this coupling permeates many areas of the Jetspeed API. Jetspeed-2 does not rely on Turbine as the MVC-2 controller. Instead, we follow the separation of concerns pattern, and concentrates on doing one thing and doing it well. That is, implementing a portal. Where as Jetspeed-1 coupled MVC Controller, portal engine, and portlet container all into one deeply coupled servlet, Jetspeed-2 separates these concerns clearly in its architecture. The portal engine is Jetspeed-2. It is the MVC for page aggregation, leveraging the dispatching nature of the servlet architecture, and delegating the actual rendering of portlets to portlet application frameworks. These portlet applications can in turn have their own MVC frameworks, such as Struts portlet applications, JSF portlet applications, or Turbine portlet application frameworks.

RunData No More

Most notably missing from Portlet API portlets is the RunData class. The Jetspeed-1 API uses the RunData class ubiquitously, serving as a wrapper for both the servlet request and response. Other dependencies on Turbine include Portlet Actions, Portlet Aggregation Engine (ECS), the Service Architecture, Configuration and Turbine Modules. None of these exist in the newer version.

Jetspeed-1	Jetspeed-2
Run Data	Portlet API: Portlet Request and Portlet Response
Portlet Aggregation Engine (ECS)	Jetspeed-2 Multi-threaded Portlet Container Engine

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Jetspeed-1	Jetspeed-2
Turbine Service Architecture	Jetspeed-2 Components
Property Configuration Files	Spring Configurations, JMX
Turbine Modules (Actions)	Portlet API Actions

Pluto is the Portlet Container

The Jetspeed-2 portal does not implement the Portlet container. Pluto implements the JSR 168 interface contract for portlets running inside our portal. The Pluto container handles all communication with portlets for the portal.

Aggregating, Isnt It?

The aggregation engine and the Jetspeed-1 Portlet API are both coupled to a deprecated Jakarta package ECS (Element Construction Set). ECS generates HTML with Java code, storing the content in temporary Java objects before sending the HTML out to the servlet output stream. This wasteful use of Java objects leads to fragmentation on memory, accelerated garbage collection, and paging in high volume sites. The servlet API clearly provides a content stream for streaming out portlet content. Jetspeed-2 models its aggregation engine upon the Portlet APIs streams and readers, analogous to the stream-based Servlet API for rendering content.

State and Life Cycle

The Portlet API clearly defines the lifecycle of a portlet, the event sequences for actions, and how the container can cache content from a portlet. The Portlet Lifecycle was not clearly defined in Jetspeed-1. The portlet API clearly states that only one instance of a portlet will reside in memory inside a container. The state of the portlet is directly related to the servlet state for the current user session. While this may seem obvious, portlet state and lifetime was not clearly defined in Jetspeed-1.

Actions

In version 1, actions were coupled to Turbine and not properly integrated into the Portlet class. In fact, actions were separate objects from portlets. In the Portlet API, actions are methods on the portlet. Action event handling and sequencing is clearly defined in the specification.

Standard Deployment

Jetspeed-1 does not have a standardized method of deploying portlets and their supporting files, commonly referred to as portlet applications. In order to import an application, one must package registry files, class and jar files, PSML and templates so that they match the Jetspeed web application format.

In Jetspeed-2, the Portlet API defines a standard deployment descriptor for deploying Portal Applications

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

into Jetspeed. Portal applications must be deployed to the portal. Analogous to the servlets packaged in a web-application (WAR) deployment model, portals support portlets packaged in a portal-application deployment model. The Portal Application archive follows the same format as the WAR format defined in the Servlet specification with an additional Portlet deployment descriptor file. The clear advantage in Jetspeed-2 is the ability to deploy live portlet applications to the server in a standardized format.

Resources and Deployment

Jetspeed-1 resources such as portal templates, images, skins, controllers, controls, are all merged into the single Jetspeed web application with no deployment model. For example, to override the default skin or top banner, the resource files are copied into the portal directory, property files updated to point to the new resources, and the server must be restarted. This made for the process of tailoring Jetspeed-1 portals to real production portals a process of property and file merging. In fact Jetspeed-1 now has a Maven plug-in to manage production portals separately from the core Jetspeed-1 portal. The need for this kind of tool covers up the fact that Jetspeed-1 is missing a good deployment model for portal resources, requiring difficult portal maintenance procedures.

For a Jetspeed-2 production portal, portal resources are packaged in a Jetspeed-specific archive format. Thus portal resources (top banners, skins, images, style sheets) can all be deployed to dynamically tailor the portal at runtime.

the Standard

JSR168 is the Portlet specification enables interoperability between Portlets and Portals. The specification defines a set of APIs that addresses standardization of portlet aggregation, personalization, presentation and security. The goals of JSR168 are to:

- Define common Portal metaphor
- Define a standard Portlet Java API
- Ensure interoperability and portability
- Enable multiple markups support
- Ensure compatibility with other technologies

The Jetspeed-2 Portlet Server supports the JSR 168 standard. This is an important initiative, introducing true portlet portability.

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

5.2 Jetspeed-1 Migration Guideline

Overview

This migration guide will help you migrate from Jetspeed version 1 to Jetspeed version 2. Note that there are currently no migration tools, nor are the plans to create a migration tool to convert portal resources from version 1 to version 2. This document provides only guidelines for migration.

With the development of the new portal standard (The Portlet API), the common portlet metaphor has changed quite drastically from the Turbine-based origins in version 1. The programming API is completely changed. There are no longer XREG files, but instead standard deployment descriptors. The are also new concepts introduced by the portlet standard such as portlet applications, portlet preferences, user attributes and init parameters that have no direct mapping from version 1. Creating a migration tool would be a large undertaking. The Jetspeed development team is not prepared to make this investment. By following the guidelines provided here, you can easily migrate your Jetspeed 1.x applications to Jetspeed 2. For an overview of architectural differences, see the document For Jetspeed-1 Users

Migration Table

The table below gives you an idea of how to migrate. We will cover each subject in more detail further on in this document.

1.x Feature	2.x Feature	Description
J1 Portlet Java Code	Portlet API Standard Code	Rewrite the java code to the new specification. Involves replacing Turbine action with standard processAction, and replacing Rundata with PortletRequest/Response
XREG Portlet Registry	portlet.xml deployment descriptor	There are pretty big differences here. Migrate <portlet-entry> to <portlet> entries, <parameter> to <preference> or <init-param>
J1 PSML	J2 PSML	Migrate Tabs to Folders and Pages, migrate to new tag syntax
XREG Security Registry	Security Constraints	Migrate J1 security constraint format to J2 security constraint format
J1 Controllers	J2 Layouts	Controllers are deprecated. Recommend using the new Jetspeed-2 Layout portlets. If porting necessary, HTML portions of VM code may port, but not context model variables
J1 Controls	J2 Portlet Decorators	Controls are deprecated. Recommend using the new Jetspeed-2 Portlet Decorators. If porting necessary, HTML portions of VM code will port, but not context model variables

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

1.x Feature	2.x Feature	Description
J1 Layouts, Screens, Navigations	J2 Page (Layout) Decorators	All deprecated. Recommend using the new Jetspeed-2 Page (Layout) Decorators as a starting point to writing your own page decorators. HTML portions of VM code will port, but not context model variables

Portlet Applications

One of the most important differences in writing Jetspeed-2/Portlet API portlets is that you must package your portlet code separate from the Jetspeed portal. In Jetspeed-1, all the user code, the portlet business logic, is packaged in one big war file mixed in with the Jetspeed-1 implementation. The Portlet API clearly abolishes this practice of mixing the portal implementation with your portlets. Jetspeed-2 is packaged as a single web application itself. When you write your portlets for Jetspeed-2, you will need to write and package your own portlets. The portlet classes and deployment descriptors must all be packaged into a single war file, known as a portlet application. A portlet application contains one or more portlets, along with a deployment descriptor, the portlet.xml. A portlet application is an extension of a web application. The portlet.xml holds the definitions of one or more portlets and is analogous to the xreg files used in Jetspeed-1.

Java Code

In this section we demonstrate how to convert a Jetspeed-1 portlet to a JSR-168 Java Standard Portlet. This involves the following steps:

- Converting the Portlet Init Java Code
- Converting the Portlet getContent Java Code
- Converting a Turbine Action

Jetspeed-1 portlet implementations are normally separated between two different Java source files.

- The Portlet Source Code
- The Turbine Action Source Code

The Portlet Source Code handles the **View** part of the MVC pattern. The **getContent** method is the standard method in Jetspeed-1 to call to render the content of a portlet. The corresponding methods in Jetspeed-2 and in the Portlet API, the **doView, doEdit, doHelp**. In the Portlet API terminology, this phase of portlet processing is known as the **render phase**. During the render phase, the portlet should not perform any business logic or other manipulation on the **Model**. All model manipulation should be left to the **action phase**

The Turbine Action performs the **action phase** of the portlet processing. During the action phase of the Portlet API standard, rendering of all other portlets is blocked until the action completes. This is also true in the Jetspeed-1/Turbine model.

Creating a new Portlet Class

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

The best place to get started in migrated your portlet is to create a new JSR-168 standard portlet. Simply create a new Java class inheriting from the GenericPortlet interface provided by the Portlet API. You can also use one of the frameworks or bridges available from Apache Portals or Spring MVC. The example below writes directly to the Portlet API. The code below can be used a skeleton for writing a portlet.

```
import java.io.IOException;
import javax.portlet.GenericPortlet;
import javax.portlet.PortletConfig;
import javax.portlet.PortletException;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
import javax.portlet.ActionRequest;
import javax.portlet.ActionResponse;

public class HelloWorld extends GenericPortlet
{
    public void init(PortletConfig config)
    throws PortletException
    {
    }
    public void doEdit(RenderRequest request, RenderResponse response)
    throws PortletException, IOException
    {
    }
    public void doHelp(RenderRequest request, RenderResponse response)
    throws PortletException, IOException
    {
    }
    public void doView(RenderRequest request, RenderResponse response)
    throws PortletException, IOException
    {
    }
    public void processAction(ActionRequest request, ActionResponse actionResponse)
    throws PortletException, IOException
        {
        }
}
```

To find out more about Portals Bridges and other Frameworks, explore these links:

- Portals Bridges
- JSF Bridge
- Struts Bridge
- Velocity Bridge
- Spring Portlet MVC

Converting the Portlet Init Java Code

The Portlet Source code handles the **Init** phase of a portlet lifecycle. The init phase is very similar in both the Java Portlet API and in Jetspeed 1. Here we have an example of the init method of a Jetspeed-1 portlet:

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

```
    public void init() throws PortletException
    {
    }
```

The equivalent method in the Portlet API (Jetspeed-2) would be, note the difference being the PortletConfig parameter (although the exception classes are named the same, they are entirely different classes, one from Jetspeed-1, the other from the Portlet API):

```
    public void init(PortletConfig config)
    throws PortletException
    {
    }
```

In Jetspeed-1, you would normally access Turbine Services with static acccessors, for example:

```
            JetspeedSecurity.addUser(user);
```

In Jetspeed-2, Jetspeed Services the standard way to access Jetspeed Services is to get a handle in the init phase, for example:

```
    private UserManager userManager;
...
    public void init(PortletConfig config)
    throws PortletException
    {
        userManager =
(UserManager)getPortletContext().getAttribute(CommonPortletServices.CPS_USER_MANAGER_COMPONENT);
        if (null == userManager)
        {
            throw new PortletException("Failed to find the User Manager on portlet
initialization");
        }
    }
```

Converting the Portlet getContent Java Code

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

In Jetspeed-1, the **getContent** method renders the content of your portlet. The render phase of Jetspeed-1 is implemented by the getContent method of your Portlet as defined by the Jetspeed-1 Portlet interface.

```
public ConcreteElement getContent(RunData rundata);
```

The only parameter passed in to the getContent method is a **RunData** parameter. RunData is a part of the Turbine web framework . RunData is basically a wrapper around the Servlet request and response, along with other Turbine-specific information. When writing portlets for Jetspeed-2, you write to the Portlet API.

```
public void doView(RenderRequest request, RenderResponse response)
throws PortletException, IOException
{
    response.setContentType("text/html");
    ...
```

The **doView** method is the Portlet API equivalent of the **getContent** method of the Jetspeed-1 API. The Portlet API has the concept of **portlet modes**. There are three default portlet modes **view, edit, and help**. For each of these modes, there are three methods you can override in your portlet: **doView, doEdit and doHelp**. Notice that where the Jetspeed-1 API has one RunData parameter, the Portlet API is more like the Servlet API, with two parameters, the **RenderRequest** and **RenderResponse**. One of the biggest parts of migrating your app will be to convert RunData references to RenderRequests and RenderResponses. Before starting, we recommend taking a training course on the Portlet API, or learning the API yourself by reading the Portlet specification as well as any articles or books on the subject. A good book to get started on the Portlet API is Portlets and Apache Portals .

When rendering content, Jetspeed 1 makes use of a HTML construction kit called **ECS**. All rendering goes through Turbine and ECS. The return type of the getContent method is a **ConcreteElement**, which is defined in the ECS API. Here is the typical way to generate output from a portlet in Jetspeed-1:

```
...
String helloString = "Hello World. This is the portlet output in view mode.";
return new org.apache.jetspeed.util.JetspeedClearElement(helloString);
```

When rendering content in Jetspeed-2, the Portlet API uses a streaming interface:

```
response.setContentType("text/html");
String helloString = "Hello World. This is the portlet output in view mode.";
```

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

```
// using Java writers
response.getWriter().println(helloString);

.. OR ...

// using Java streaming
response.getPortletOutputStream().write(helloString.getBytes());
```

Of course you can use JSPs or Velocity with either Jetspeed-1 or Jetspeed-2. With Jetspeed-1, the common practice is to make use of the Jetspeed-1 **GenericMVCPortlet** or one of its derivitives, the **VelocityPortlet** or the **JspPortlet**. Both the VelocityPortlet and JspPortlet are really just GenericMVCPortlets. Here is the xreg example of a WeatherPortlet which extends the GenericMVCPortlet by setting its parent to Velocity

```
    <portlet-entry name="WeatherPortlet" hidden="false" type="ref" parent="Velocity"
application="false">
        <parameter name="template" value="weather" hidden="true"/>
    </portlet-entry>
```

The template parameter is named **weather**. Since this is a Velocity MVC portlet, Jetspeed-1 knows to look under the **WEB-INF/templates/vm/portlets/html** directory to find **weather.vm**. The MVC portlet will automatically handle the details of dispatching to this Velocity template to render your portlet. Here is the actual contents of the velocity template. Note that we don't have to write any portlet Java code in this case, but only the actual template.

```
#if (!$weather_city_info)
<BR>${l10n.WEATHER_PLEASE_CUSTOMIZE_YO_VM}<br><BR>
#else
<a href="http://www.wunderground.com/${weather_city_info}.html"
target="_blank"><img
src="http://banners.wunderground.com/banner/${weather_style}/language/www/${weather_city_info}.gif"
alt="Click for ${weather_city_info} Forecast" border="0"></a>
#end
```

With Jetspeed-2 and the Portlet API, we can make use of the Velocity Bridge or the JSP Bridge to delegate to portlets. The simplest case is just dispatching the call yourself to the JSP or Velocity servlet. Here is an example of dispatching to a JSP from the doView:

```
protected void doView(RenderRequest request, RenderResponse response) throws
PortletException, IOException
{
```

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

```
    PortletContext context = getPortletContext();
    ResourceBundle resource =
getPortletConfig().getResourceBundle(request.getLocale());
    request.setAttribute("viewMessage",
resource.getString("preference.label.MyModeIsView"));
    PortletRequestDispatcher rd =
context.getRequestDispatcher("/WEB-INF/demo/preference/pref-view.jsp");
    rd.include(request, response);
}
```

And here is an example of the WeatherPortlet extending the Velocity Bridge, and making use of the Portlet API User Preferences feature, note that we do not directly create a dispatcher here, but the framework will do that automatically:

```
import org.apache.portals.bridges.velocity.GenericVelocityPortlet;
...

public class WeatherPortlet extends GenericVelocityPortlet
{
...

public void doView(RenderRequest request, RenderResponse response)
        throws PortletException, IOException
{
    Context context = super.getContext(request);

    String cityInfo = (String) request.getPortletSession().getAttribute(
            WEATHER_CITY_INFO);

    PortletPreferences prefs = request.getPreferences();
    String city = prefs.getValue(WEATHER_CITY, "Bakersfield");
    String state = prefs.getValue(WEATHER_STATE, "CA");
    String station = prefs.getValue(WEATHER_STATION, null);
    cityInfo = getCityInfo(city, state, station);
    context.put(WEATHER_CITY_INFO, cityInfo);

    String style = prefs.getValue(WEATHER_STYLE, "infobox");
    context.put(WEATHER_STYLE, style);
    response.setProperty("david", "taylor");
    super.doView(request, response);
}
```

And here is the Velocity template to render the portlet content:

```
#if (!$weather_city_info)
Please configure your Weather settings.
#else
<a href="http://www.wunderground.com/${weather_city_info}.html"
target="_blank"><img
src="http://banners.wunderground.com/banner/$!weather_style/language/www/${weather_city_info}.gif"
```

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

```
alt="Click for $weather_city_info Forecast" border="0"></a>
#end
```

Converting a Turbine Action

The Portlet API defines several phases of execution during the processing of a portlet page. The action phase is designed to be executed before the render phase of a portlet. There can only be one action phase targeting only one portlet. Once the action phase completes, then the render phase for all portlets on a page can be executed. Thus the action phase is said to be a *blocking* phase, meaning that it must complete before the render phase for each portlet on the page can commence. Actions are usually some kind of user interaction that manipulates the *Model* of the MVC framework, such as a user submitting a form and updating the model, or adding or deleting a record. The concept of actions ports fairly well from Turbine and Jetspeed-1 to Jetspeed-2 and the Portlet API. Whereas Turbine has the concept of one class per action, the Portlet API has an entry point for all actions to come through as a method on your portlet. Frameworks such as the Spring MVC framework provide better abstractions for modeling one method per action.

Lets again look at the WeatherPortlet with Jetspeed-1. First the xreg defines the actions:

```
       <parameter name="action" value="portlets.WeatherAction" hidden="true"/>
```

We must then implement the action class which are usually placed in the Jetspeed-1 webapp class loader space. Here is the code for the WeatherAction, which extends a Jetspeed-1 framework class VelocityPortletAction:

```
public class WeatherAction extends VelocityPortletAction
{

    protected void buildNormalContext( VelocityPortlet portlet,
                                       Context context,
                                       RunData rundata )
    {

        String cityInfo = PortletConfigState.getParameter(portlet, rundata,
WEATHER_CITY_INFO, null);
        //if (cityInfo == null)
        //{
            String city = portlet.getPortletConfig().getInitParameter(WEATHER_CITY);
            String state =
portlet.getPortletConfig().getInitParameter(WEATHER_STATE);
            String station =
portlet.getPortletConfig().getInitParameter(WEATHER_STATION);
            cityInfo = getCityInfo(city, state, station);
        //}
        context.put(WEATHER_CITY_INFO, cityInfo);
        //PortletConfigState.setInstanceParameter(portlet, rundata,
```

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

```
WEATHER_CITY_INFO, cityInfo);

        String style = PortletConfigState.getParameter(portlet, rundata,
WEATHER_STYLE, "infobox");
        context.put(WEATHER_STYLE,style);
    }
```

In Jetspeed-1 there is some really bad architecture interfering with easily writing portlets. Here in our action, we are actually implementing the **View** portion of our code by populating the Velocity context with **context.put** statements. Please beware that all code implemented in the **buildNormalContext** method should be ported to the **doView** method of the Portlet API. Note how the actual portlet must be passed in as the first parameter to the buildNormalContext method.

The actual action code implemented as **do..** methods on your action class will need to be ported to the **processAction** method on the Portlet API.

```
    public void doInsert(RunData rundata, Context context)
        throws Exception
    {
```

The **doInsert** method is linked by Turbine to an action in the Velocity template with the **eventSubmit_** prefix:

```
<input type="submit" name="eventSubmit_doInsert"
value="${l10n.USER_FORM_ADD_USER_VM}"/>
```

Here is the equivalent in the Portlet API (Jetspeed-2):

```
    public void processAction(ActionRequest actionRequest, ActionResponse
actionResponse)
        throws PortletException, IOException
```

The Portlet API provides two parameters to the processAction method: the ActionRequest and ActionResponse.

Request Parameters, Portlet Modes, Window States

Request parameters are accessed via RunData in Jetspeed-1:

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

```
String name = rundata.getParameters().getString("username");
```

With the Portlet API, portlet request parameters are accessed via the ActionRequest:

```
String name = actionRequest.getParameter("username");
```

With the Portlet API, you can check the Portlet Mode or Window State:

```
if (actionRequest.getPortletMode() == PortletMode.EDIT)
{
        if ( !request.getWindowState().equals(WindowState.MINIMIZED))
        {
        ...
```

The basic Portlet API does not have a way to map actions to methods as in Jetspeed-1. If you would like this kind of behavior, we recommend using the Spring MVC Portlet framework Here we demonstrate using portlet request parameters per form to map to specific actions:

```
            String action =
actionRequest.getParameter(SecurityResources.PORTLET_ACTION);
        if (action != null && action.equals("remove.user"))
        {
            removeUser(actionRequest, actionResponse);
        }
        else if (action != null && action.equals("add.new.user"))
        {
            PortletMessaging.cancel(actionRequest, SecurityResources.TOPIC_USERS,
SecurityResources.MESSAGE_SELECTED);
        }
        else if (action != null && action.equals("add.user"))
        {
            addUser(actionRequest);
        }
            ...
```

Persisting State: The Portlet Session

The Portlet API provides built-in support for persistence of Portlet state in the session. The Portlet

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Session is similar to the **setTemp** methods in Turbine/Jetspeed-1, or the session support built into the Servlet API. The Session is for persisting state associated with the current user session. There are two kinds of session state supported by the Portlet API:

- Application Session State: the session variable is shared by all portlets in a portlet application
- Portlet Session State: the session variable is specific to the one portlet instance window

Here is how we would get and set session information in Jetspeed-1, using the Turbine RunData API. Note that for both Jetspeed-1 and Jetspeed-2, the object put in the session must be serializable:

```
            rundata.getUser().setTemp(ACCOUNT_INFO, accountInfo);
            ...
            AccountInfo accountInfo =
(AccountInfo)rundata.getUser().getTemp(ACCOUNT_INFO);
```

In here is the equivalent in Jetspeed-2 using the Portlet API:

```
       AccountInfo accountInfo = (AccountInfo)
               actionRequest.getPortletSession().getAttribute(ACCOUNT_INFO,
PortletSession.PORTLET_SCOPE);
       -- or --
       AccountInfo accountInfo = (AccountInfo)
               actionRequest.getPortletSession().getAttribute(ACCOUNT_INFO,
PortletSession.APPLICATION_SCOPE);

               -- the setters --
               PortletSession session = actionRequest.getPortletSession();
               session.setAttribute(ACCOUNT_INFO, accountInfo,
PortletSession.PORTLET_SCOPE);
               -- or --
               session.setAttribute(ACCOUNT_INFO, accountInfo,
PortletSession.APPLICATION_SCOPE);
```

Persisting State: User Preferences

The Portlet API provides a second persistence mechanism: User Preferences. User Preferences are fields of information stored on a per user/per portlet window basis. The equivalent in Jetspeed-1 is Portlet Instance data, which is stored in the Jetspeed-1 Portlet Registry as name/value pair **parameter** XML elements. Looking at the XREG file in Jetspeed-1, we have:

```
       <parameter name="weather_city_info" value="US/IN/Bloomington"
hidden="true"/>
```

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

The Portlet API allows you to define default values for preferences in the portlet.xml deployment descriptor. The user-specific values are stored in the Jetspeed Preferences database. Here is an example of the default value for a preference as it would be defined in the deployment descriptor:

```
            <preference>
                <name>weather_city</name>
                <value>Oakland</value>
            </preference>
```

Jetspeed-1 provides the **PortletInstance** interface on every portlet for accessing preference-like information. Whereas the preference information is per-user and per-instance in Jetspeed-2, in Jetspeed-1 preference information accessed via the PortletInstance interface is only per-instance(per PortletWindow) specific. These values are stored in the PSML file associated with the PortletWindow. Please note that the values can still be *user-specific* when you are using the default mechanism for locating pages, which is by user. This means that in Jetspeed-1 preferences (or parameters) are made user-specific by the nature of how pages are retrieved. Since a page is located under a user home directory, then the preference is naturally per user.

With Jetspeed-1, here we can retrieve PortletInstance data:

```
        // where "this" is a Jetspeed-1 Portlet object
            PortletInstance instance = this.getInstance(rundata);
            String value = instance.getAttribute("favoriteColor", "blue");
            -- or --
            this.getAttribute("favoriteColor", "blue", rundata);

            -- we can set preference data the same way in Jetspeed-1
            PortletInstance instance = this.getInstance(rundata);
            instance.setAttribute("favoriteColor", "red");
            -- or --
            this.setAttribute("favoriteColor", "red", rundata);
```

With the Portlet API in Jetspeed-2, we can use the Portlet Preferences in a more direct manner. Remember that the store() method must always be called after all modifications to the prefs during a request:

```
        PortletPreferences prefs = actionRequest.getPreferences();
        String color = prefs.getAttribute("favoriteColor", "blue");
        ...
        prefs.setAttribute("favoriteColor", "red");
        prefs.store();

        // note that you can also retrieve multivalues for prefs
        String values[] = actionRequest.getPreferences().getValues("stocks",
defaultValues);
```

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

```
// or retrieve all preferences as a Map
Map allPrefs = actionRequest.getPreferences().getMap();
```

Registries

The Jetspeed-1 Registries hold the following information:

- Portlet Definitions
- Security Definitions
- Web Clients and Media Type Registries
- Skins Definitions
- Controller Definitions
- Control Definitions

This section will guide you through how to migrate each of these registries from Jetspeed-1 to Jetspeed-2

Portlet Definitions

Jetpeed-1 requires that all portlets are defined in an XML file known as an XREG file (XML Registry). Jetspeed-2 stores its portlet registry in the database. In Jetspeed-1, the XML registry is on the file system under the jetspeed webapp under WEB-INF/conf. There can be one or more portlet registry entries. All portlets are defined with the element type **portlet-entry**.

Migrating your Jetspeed-1 portlet registries to Jetspeed-2 registries requires writing a new Portlet API standard **portlet.xml** definition file. We do not provide an XSLT transform to do this for you. Whereas the portlet.xml is defined by the Java Standard Portlet API, Jetspeed allows for additional information to be defined specific to the Jetspeed portal: the **jetspeed-portlet.xml** can hold Jetspeed-specific deployment configurations. Some of the XREG elements map to the portlet.xml, whereas others will map to the jetspeed-portlet.xml as noted in the tables below. The table below describes how to map each XML attribute of the **portlet-entry** element to its equivalent in the Portlet API portlet.xml or jetspeed-portlet.xml. Note that we are mapping in this table from XML attributes to XML elements in the portlet.xml or jetspeed-portlet.xml:

J1 Attribute	J2 Element	
name	portlet-name	The name of the portlet. This name is unique to each portlet application, but not unique system-wide.
hidden		No equivalent in the Portlet API, not applicable.
type		No equivalent in the Portlet API, not applicable.
parent		No equivalent in the Portlet API, not applicable.

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

J1 Attribute	J2 Element	
application		No equivalent in the Portlet API, not applicable.

Continuing with the Portlet XREG conversion, lets now look at how to convert the XML elements of the **portlet-entry** element. The table below describes how to map each XML element to its equivalent in the Portlet API portlet.xml:

J1 Element	J2 Element	
classname	portlet-class	The implementing Java class. This class will need to be ported at the source level.
media-type	supports, supports/mime-type, supports/portlet-mode	Media types supported by the portlet must be mapped to one or more *supports* elements, with subelements of *mime-type* and *portlet-mode* pairs.
meta-info/title	title	The title of the Portlet.
meta-info/description	description	The description of the portlet
category	portlet-info/keywords	Where there are multiple categories elements, keywords are comma-separated. In Jetspeed-2, you can configure categories in the Portlet-Selector administrative portlet based on keywords.
security-ref	jetspeed-portlet.xml: js:security-constraint-ref	If you port your Security constraints definitions, you can keep the same security definition names. Just note that security constraint definitions are referenced from the jetspeed-portlet.xml, not portlet.xml
parameter	init-param	Parameters in Jetspeed-1 should normally map to *init-param*s in the Portlet API. These are read only values that can only be changed by the administrator
parameter@name	init-param/name	The name of the init parameter
parameter@value	init-param/value	The value of the init parameter
parameter/meta-info/description	init-param/description	The description of the init parameter
parameter	portlet-preferences/preference	As well as migrating to init-params, parameters may also be migrated as default preferences. Note that preferences can optionally be read-only.
parameter@name	portlet-preferences/preference/name	The name of the preference
parameter@value	portlet-preferences/preference/value	The value of the preference
parameter@hidden	portlet-preferences/preference/read-only	Optionally you map want to map hidden values to read-only (true/false)

Security Definitions

Jetspeed-1 supports a Security Constraint XML definition language that is very similiar to the XML security constraint definitions in Jetspeed-2. Jetpeed-1 requires that all security definitions are defined in

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

an XML file known as an XREG file (XML Registry). Jetspeed-2 stores its security registry either in an XML file or in the database. In Jetspeed-1, the XML registry is on the file system under the jetspeed webapp under WEB-INF/conf. There can be one or more security registry entries. All security constraints are defined with the element type **security-entry**.

Migrating your Jetspeed-1 security constraints registries to Jetspeed-2 registries requires writing a new **page.security** XML definition file. We do not provide an XSLT transform to do this for you. The table below describes how to map each XML attribute of the **security-entry** element to its equivalent in the Portlet API portlet.xml or jetspeed-portlet.xml. Note that we are mapping in this table from XML attributes to XML elements in the portlet.xml or jetspeed-portlet.xml:

J1 Attribute	J2 Attribute	
security-entry@name	security-constraints-def@name	The name of the security constraint definition. This name is unique to the entire page.security file.
meta-info/title		No equivalent in Jetspeed-2, not applicable.
meta-info/description		No equivalent in Jetspeed-2, not applicable.
access	security-constraint	Jetspeed-1 security-entries contain 0..n *access* elements, Jetspeed-2 security-constraint-defs contain 0..n *security-constraint* elements.
access@action	security-constraint/permissions	Actions in Jetspeed-1 are called Permissions in Jetspeed-2. Both versions support wildcarding with the * character. • Jetspeed-1 default actions are view, customize, maximize, minimize, info, close. • Jetspeed-2 default permissions are view, edit, help, print
access/allow-if@role	security-constraint/roles	Jetspeed-1 constrains by role through *allow-if* elements with a *role* attribute. Jetspeed-2 constrains by role with the *roles* element and a comma-separated list of one or more roles
access/allow-if@group	security-constraint/groups	Jetspeed-1 constrains by group through *allow-if* elements with a *group* attribute. Jetspeed-2 constrains by group with the *groups* element and a comma-separated list of one or more groups
access/allow-if@user	security-constraint/users	Jetspeed-1 constrains by user through *allow-if* elements with a *user* attribute. Jetspeed-2 constrains by user with the *users* element and a comma-separated list of one or more users, or the wildcard * to specify all users.
access/allow-if-owner	security-constraints/owner	You can set the constraint to be only accessible by the owner of the page. In Jetspeed-1, this is implied by the location of the page. With Jetspeed-2 you must explicity name the owner in the element text of the owner element.

Web Clients and Media Type Registries

The Web Clients and Media Type registries are already ported to Jetspeed-2 and a part of the core system.

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Jetspeed-2 stores these registries in the database. However these tables can be populated using seed data as described in the section below on seed data.

Skins

The Skin registries are not directly portable to Jetspeed-2. Jetspeed-2 has moved towards a more standard CSS based skinning approach. There are two basic skinning techniques which can be combined:

- 1. Portlet API Standard Skins - see PLT.C of the portlet specification. A standard set of CSS styles are defined for global skinning of portlet content.
- 2. Jetspeed Decorators - Decorators can define their own skins which can then be leveraged by portlets by accessing these styles. The default decorators in Jetspeed also define the PLT.C styles as well

Controllers

Controllers are deprecated in Jetspeed-2. There is no direct mapping for converting the Java code. Instead you will need to rewrite a new Layout portlet, or more likely simply use one of the existing Layout Portlets that come with Jetspeed, which are quite flexible. The default layout portlets in Jetspeed support multi-column grids, nesting portlets, and complete customization using the Portlet Customizer.

Controls

Controls are deprecated in Jetspeed-2. There is no direct mapping for converting the Java code. Instead you will need to rewrite a new Portlet decorator, or more likely simply use one of the existing Portlet decorators that come with Jetspeed, which are quite flexible.

PSML

The Jetspeed Sitemap

The Jetspeed Sitemap defines the navigational space of all pages in the portal. Both versions 1 and 2 have similiar hiearchical file system-like site maps. Both contain a root folder /, which in turn contains a tree of subfolders, where each subfolder can contain pages or more subfolders.

Site Resources

In Jetspeed-2, there is a well-defined portal resources that do not always have equivalents in Jetspeed-1:

2.x	1.x	File
Page	Page	A **.psml** file.
Folder	--	A **folder.metadata** file, one per folder, N/A in Jetspeed-1
Link	--	A **.link** file, N/A in Jetspeed-1

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

2.x	1.x	File
Menu	--	Menus are defined in folder.metadata, N/A in Jetspeed-1

Reserved Directories

There are reserved directories available in both versions. The naming is a little different. Any directory starting with an underscore (_) in Jetspeed-2 is considered a control directory and can be used by the profiler (see below) to locate special directories based on runtime criteria such as the user name or the roles of the user. Jetspeed-1 has a hard-coded set of reserved (control) directories that are hard-coded into the profiling rules.

1.x	2.x	
user	_user	Holds all user folders
role	_role	Holds all role folders
group	_group	Holds all group folders
{language}	_lanaguage	Content per language
{country}	_country	Content per country code

Where the J1 directory names are actually the names of the reserved directory, such as {mediatype} would be actually **html** or {language} would be **en**. J2 requires specifing control directories (_) such as **_mediatype/html**, or **_language/en**

Profiling

The Profiling algorithm discovers the correct page to display during a request. J1 has only two hard-coded algorithm for finding pages:

- J1 user/mediatype/language/country fallback
- J1 rollback

Note that these settings are system wide and must be changed on a per portal basis. J1 expects an explicit container order of mediatype / language / country

J2 has a profiling rules engine that takes dynamic runtime user information, and using profiling rules discovers the rules based on the algorithm defined in the rules. In J2 profiling rules are defined on a per user basis, although there is a system-wide default profiling rule.

Differences in PSML Page

Jetpeed-1 requires that all portlets are defined in an XML file known as an XREG file (XML Registry). Jetspeed-2 stores its portlet registry in the database. In Jetspeed-1, PSML files can be stored under the jetspeed webapp under WEB-INF/psml. Or, Jetspeed-1 supports storing PSML files in the database. In

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Jetspeed-2, PSML files can be stored under the jetspeed webapp under WEB-INF/pages or WEB-INF/min-pages. Or, Jetspeed-2 supports storing PSML files in the database.

Migrating your Jetspeed-1 PSML files to Jetspeed-2 PSML files requires porting the files manually, or writing a database conversion utility or XSLT transform. We do not provide an XSLT transform to do this for you. The table below describes how to map each XML element or attribute from Jetspeed-1 to Jetspeed-2:

J1 Element	J2 Element	
portlets	page	The outermost container of all content found on a PSML page.
portlets@id	page@id	System wide unique identifier for this page.
metainfo/title	title	The Page Title.
security-ref	security-constraints/security-constraints-ref	The security constraint reference (0..1 in Jetspeed-1, 0..n in Jetspeed-2)
control	defaults/portlet-decorator	Requires porting your controls to J2 portlet decorators, or at least mapping the names to existing decorators in Jetspeed-2. Or you can use a global portlet decorator and ignore this optional setting.
controller	defaults/layout-decorator	Requires porting your Turbine controllers, screens navigations to J2 layout(page) decorators, or at least mapping the names to existing page decorators in Jetspeed-2. Or you can use a global portlet decorator and ignore this optional setting.
portlets/portlets/...	page/fragment/..., type="layout"	Sub-containers of fragments or portlets. In Jetspeed-2, fragments can be either containers or portlet definitions. Only fragments with the type of **layout** can be a container holding more fragments and containers.
portlets/portlets/controller	page/fragment@type=layout@name={layout-name}	Controllers roughly map to fragments of type = layout, named by the name attribute. Note that layouts are implemented as portlets and must be specified as PA::portlet-name.
portlets/entry	page/fragment/fragment@type="portlet"	A portlet window on a page.
entry@id	fragment@id	The system-wide unique ID of the portlet window.
entry@parent	fragment@name	The portlet registry reference. In Jetspeed-2 the name of the portlet must be specified as PA::portlet-name
entry/layout/property@name="column"@value={column}	fragment/property@name="column"@value={column}	The property containing the column position
entry/layout/property@name="row"@value={row}	fragment/property@name="row"@value={row}	The property containing the row position

Menus vs Tabs

There is a big difference with the navigational aspects, or menus, between Jetspeed-1 and Jetspeed-2. Jetspeed-1 restricts menus navigation to navigation amongst *tabs*. Tabs are defined within a PSML page. Tabs are simply subcontainers in the PSML page, defined by the **portlets** element. Whereas Jetspeed-1

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

does support navigation to other pages, the Tabbing Menus do not directly support it without writing a specific portlet to act as an external link.

Jetspeed-2 menu navigations map directly onto the Portal Site. Thus menu tabs represent portal resources. Menus in Jetspeed-2 can point to folders, pages or links. This more naturally allows the user to navigate over the entire portal site.

When migrating PSML files from Jetspeed-1 to Jetspeed-2, depending on whether you use advanced Jetspeed-1 controllers such as Card or Tab controllers, you may find that the pages do not port to Jetspeed-2 very well. In consideration of the lack of migration tools, this leaves two immediate options:

- Rewrite your PSML files to better map to the Jetspeed-2 site constructs, folders and multiple pages.
- Enhance Jetspeed-2 to support card and tab controller behavior

XML API - Seed Data

Jetspeed-2 defines an XML API for populating the initial "Seed" data for your portal. Populating your seed data via the XML API provides an alternative to populating database data with database-specific and hard to read SQL scripts. Additionally, the XML API can be used for importing and exporting data, or backing up and restoring from your Jetspeed-2 database.

The XML API also provides a migration path over the maintenance cycle of your Jetspeed portal. The XML API was first implemented in version 2.1. To migrate your data from version 2.1 to 2.2, (if there are any database schema changes), the XML API can be used to migrate (by exporting and importing) across versions.

As of 2.1, the Jetspeed API supports the following elements:

Element	Description
MimeTypes	Mime Types supported by the portal such as text/html, text/xhtml....
MediaTypes	Mediat Types supported by the portal such as html, xml, wml...
Capabilities	General capabilities of web clients that access the portal
Clients	Supported Web Clients by the portal
Roles	Define all roles defined to the initial configuration of the portal
Groups	Define all groups defined to the initial configuration of the portal
Users	Define all initial users defined to the initial configuration of the portal, minimally admin and guest(anon) users
Permissions	Define initial J2EE security policy for this portal. Note that permissions are turned off by default.
ProfilingRules	Define all the profiling rules in the initial portal such as role fallback, user-role-fallback, j1-emulation, default-j2, subsites and more

XML Schemas

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Reference for Jetspeed-2 XML schemas:

Jetspeed-2 Folder Metadata	http://portals.apache.org/jetspeed-2/2.1/schemas/folder-metadata.xsd
Jetspeed-2 Seed Data	http://portals.apache.org/jetspeed-2/2.1/schemas/j2-seed.xsd
Jetspeed-2 Security Constraints	http://portals.apache.org/jetspeed-2/2.1/schemas/page-security.xsd
Jetspeed-2 Links	http://portals.apache.org/jetspeed-2/2.1/schemas/link.xsd
Jetspeed-2 Extended Portlet Descriptor	http://portals.apache.org/jetspeed-2/2.1/schemas/jetspeed-portlet.xsd

Where to Get Started?

The best place to get started is to create your own custom portal. This process is defined online at Apache. The Jetspeed Tutorial will take you through the initial steps of setting up your own (custom) Jetspeed portal, including setting up XML seed data, PSML, custom decorations and portlet applications.

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

5.3 Supporting Projects

Supporting Projects

Project	Description
	Derby provides Jetspeed-2 default embedded database engine.
	Apache Directory Server provides Jetspeed-2 default embedded LDAP engine.
	Lucene provides Jetspeed-2 embedded search engine.
	OJB provides Jetspeed-2 default persistence layer. Jetspeed-2 uses OJB's PersistenceBroker API.
	Pluto provides Jetspeed-2 portlet container. Pluto is the Reference Implementation of the Java Portlet Specfication.
	The Spring Framework provides Jetspeed-2 default component framework.

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

5.4 Who Uses J2?

Who Uses Jetspeed-2?

Company/Project	Description
	APPLIED Co.,Ltd. is a Japanese company providing services to develop IT business applications and Web sites, having many customized J1 and J2 based actual results.
	BlueSunrise provides services to help companies implement Jetspeed-2 solutions.
	CARDINIS Solutions S.p.A. is leader in providing solutions for the governance of innovation and companies, leveraging on enterprise project portofilo management and strategy management. Cardinis Suite, the flagship product of CARDINIS Solutions, is based on international standards and methodologies, and on from-the-field experiences from our consulting activities and a continuous synergy with leading analysts and universities. It deploys technologies that enable communication and information sharing, through a collaborative platform for project, program, portfolio and demand management. To satisfy our clients' requests about capability of accessing critical information about projects and other business initiatives, we decided to adopt Jetspeed2 as the portal reference, providing custom portlets that respond to specific needs.
	Chikpea provides a self-service portal, where Business can register as a Service Provider to serve their own customers to manage Sales and Services. It allows the Business to represent its own self-service website to capture sales and service requests, as well as provides complete solution to manage those requests.
	Convergys uses Jetspeed-2 as a B2B Business Intelligence Portal to expose reports from third party enterprise OLAP engines such as Cognos and MicroStrategy.
	GroundWork 's open source IT infrastructure monitoring solution delivers enterprise-class availability and performance for a fraction of the cost of commercial alternatives.
	Hippo is a Dutch open-source Content Management Software provider developing the Hippo Portal which integrates Hippo CMS with Jetspeed-2. Hippo Portal will provide a complete Content Repository based Portal Site Management and Delivery solution available under the ASF 2.0 license.

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

Company/Project	Description
	The Jahia 5.0 line of products includes a Corporate Portal Server based on Jetspeed-2. 100% Java based, the full Jahia source code is available under a collaborative and community source license (contribute or pay paradigm).
	N2SM provides solutions and services to construct IT business, such as Company Internal Portal and EC site.
	OpenXava generates JSR-168 portlets deployables in Jetspeed-2. It also generates all .psml, .ds, page.metadata to deploy an OpenXava application automatically in Jetspeed-2. OpenXava distribution is bundled with Jetspeed-2 installation and its web site is powered by Jetspeed-2.
	PortalU is the German Environmental Information Portal! It offers a comfortable and central access to over 1.000.000 web-pages and database entries from public agencies in Germany. We also guide you directly to up-to-date environmental news, upcoming and past environmental events, environmental monitoring data, and interesting background information on many environmental topics.
	R.O.S.A. Creation. Technology. Intelligence. AG deploys collaborative portals using Jetspeed-2 and provides services in portal development.
	UGS provides portals for its global sales partners using Jetspeed 2.
	The german company wemove digital solutions creates portal solutions for various customers. The current project "PortalU" uses Jetspeed2 to provide an interface to a powerful search-engine for enviromental data.
WfMOpen	In its 1.4 version, WfMOpen provides resource management for the workflow engines (BPE) and uses Jetspeed-2 as a container for the engine's administrative portlets.

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

5.5 Portlets Community

Portlets Community

Project	Description
	Gems provides a collection of JSR-168 portlets. The list of available portlets include: • E-Mail Portlet • Calendar Portlet • Blog Portlet • RSS Feed Portlet • Calculator Portlet • Image Viewer Portlet • Horoscope Portlet
	PAL provides a useful JSR-168 portlets, such as File Manager, Blog, Yahoo! Japan Search portlets.

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

5.6 How to Help?

How to Help?

Simple Things I Can Do to Help

There are many ways to help with Jetspeed-2 as with most open source projects:

- Subscribe to the user mailing list and help answer questions from the community. In open source a thriving community makes the project successful. Don't be shy to ask basic questions, we have all been there.
- Report bugs and issues that you encounter in Jetspeed-2 bug tracking system . Prior to reporting a bug, make sure to discuss the issue on the user mailing list or even the developer mailing list .
- When you encounter an issue, you may be compelled to fix it. We encourage this as this makes for a vibrant community. Once you have a fix, submit a patch on the reporting issue.

Why Should I Get Involved?

There are many reasons why you want to help, just a few strong points:

- If you help others solve there issues, they will most likely help you when you run into some problems.
- By contributing patches, you can influence the prioritization of functionality and get your changes incorporated.
- By reporting issues, you help Jetspeed-2 become a stronger project and improve its quality overall.
- You will meet and get to know great people as well as share and learn best practices which will help you on your project.

We are looking forward to have you part of our community!

How do I Join the Project?

Projects at Apache operate under a meritocracy . To become a committer, you first need to demonstrate your committment. The best way to do so is to start contributing patch, participate in the community and make your interest known. It takes time and willingness to help and contribute! This may seem a bit intimidating at first, but the community will always help people who show interest and committment.

Here are some useful links for other resources for help.

- Portals project coding standards .
- Portals project documentation standards .
- How does the Apache Software Foundation work ?
- Portals mailing lists statistics . This can be helpful to help you decide which mailing list to subscribe

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

to.

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

6.1 Mailing List

Mailing Lists

These are the mailing lists that have been established for this project. For each list, there is a subscribe, unsubscribe, and an archive link.

List Name	Subscribe	Unsubscribe	Archive
Jetspeed 2 User List	Subscribe	Unsubscribe	Archive
Jetspeed 2 Developer List	Subscribe	Unsubscribe	Archive

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

6.2 Bug Database

Issue Tracking

http://issues.apache.org/jira/secure/BrowseProject.jspa?id=10492

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

6.3 Wiki

http://wiki.apache.org/portals/Jetspeed2

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

6.4 Quality Testing

Jetspeed Build and Quality Testing

SpikeSource runs nightly builds of Jetspeed-2 on a set of Linux platforms (Suse, Fedora, Redhat...). The builds run an entire Jetspeed-2 build and all unit tests. Additionally, code coverage tests are run. The results of these tests are available at the SpikeSource website.

Jetspeed-2 Build Results

Jetspeed-2 is built nightly. Unit tests and code coverage results are found here:

Jetspeed-2 Nightly Build Results

Jetspeed-2 Build Results: REST

Jetspeed-2 is built nightly. Unit tests and code coverage results in REST format are found here:

Jetspeed-2 Nightly Build Results - REST

Open Source Build Results

Find the results of Jetspeed-2 and other open source builds here:

Open Source Build Results

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED

7.1 Japanese

http://jetspeed-japan.sourceforge.jp/jetspeed-2-trans/ja/index.html

©1999 APACHE SOFTWARE FOUNDATION • ALL RIGHTS RESERVED