



DOCUMENTATION

Portlet Integration Guide

Jahia v6.5

Jahia's next-generation, open source CMS stems from a widely acknowledged vision of enterprise application convergence – web, document, search, social and portal – unified by the simplicity of web content management.

Jahia Solutions Group SA

*9 route des Jeunes,
CH-1227 Les acacias
Geneva, Switzerland*

<http://www.jahia.com>

Summary

1	Introduction.....	4
1.1	What's in this documentation?	4
1.2	What this guide is not.....	4
1.3	Portlets versus modules	4
2	Quick tutorial: deploying the portlet test suite.....	7
3	Deployment	13
3.1	What is portlet preparation? Why do I need it?	13
3.2	Deploying through the administration user interface	14
3.3	Deploying directly through the file system	14
4	Using in a Jahia page.....	15
4.1	Portlet instances	15
4.2	Portlet roles and Jahia permissions.....	15
5	Hello world portlet	16
5.1	Requirements	16
5.2	Project structure.....	17
5.3	Maven project descriptor	17
5.4	Portlet Java code	19
5.5	Portlet descriptor.....	20
5.6	Web application descriptor	21
5.7	The Tomcat deployment descriptor	21
5.8	Building and deploying	21
5.9	Redeployment	22
5.10	Troubleshooting deployment.....	23
5.11	Learning more	23
6	Support for Bridges.....	23
7	How to share portlet objects?	23
7.1	Public Render Parameters	23
7.2	Portlet Events	24

7.3	Jahia Shared Map	24
7.3.1	How to share a simple object?	26
7.3.2	How to read a simple object?	26
7.3.3	How to share a bean?	26
8	Performance	27
9	Using portlets on other application servers	28
10	Additional resources	28

1 Introduction

Jahia includes an embedded portal server, which is based on the Apache Pluto reference implementation of the JCR Portlet API specification. The goal of this implementation is to offer support for integrators who need to embed portlets on content pages. This means that any portlet API compliant application may be integrated with Jahia in a few simple steps.

1.1 What's in this documentation?

Jahia is quite unique in its ability to integrate with both traditional CMS content and portlets on the same page. This document aims to explain how to work with portlets.

Should you have questions, please do not hesitate to contact us as mentioned on our website (<http://www.jahia.com>) or Community website (<http://www.jahia.org>).

1.2 What this guide is not

This guide is not a tutorial on the full portlet API, it is merely an introduction to portlet deployment and development with Jahia. In the "Additional resources" section at the end of this guide we have listed a few resources you might find interesting if you would like to learn more about the portlet standard.

1.3 Portlets versus modules

In order to differentiate portlets from modules, we offer the following table that summarizes the differences:

	Portlet	Module
Classification	Older technology, extension to traditional Web server model using well defined approach based on strict isolation mode	Using newer, loosely defined "Web 2.0" techniques, integration at both server or client level

Philosophy/Approach	Approaches aggregation by splitting role of Web server into two phases: markup generation and aggregation of markup fragments	Uses APIs provided by different modules as well as content server to aggregate and reuse the content
Content dependencies	Aggregates presentation-oriented markup fragments (HTML, WML, VoiceXML, etc.)	Can operate on pure content and also on presentation-oriented content (e.g., HTML, JSON, ...)
Location dependencies	Traditionally content aggregation takes place on the server	Content aggregation can take place either on the server or on the client, but usually happens on the server
Aggregation style	"Salad bar" style: Aggregated content is presented 'side-by-side' without overlaps	"Melting Pot" style - Individual content may be combined in any manner, resulting in arbitrarily structured hybrid rendering and editing
Event model	Read and update event models are defined through a specific portlet API	CRUD operations are based on JCR architectural principles, and on the client REST interfaces allow content interactions
Relevant standards	Portlet behavior is governed by standards JSR 168, JSR 286 and WSRP, although portal page layout and portal functionality are undefined and vendor-specific	Base standards are JCR API, REST, JSON and XML. Defacto standards include JQuery as a Javascript framework
Portability	Portlets developed with the portlet API are in theory portable to any portlet container	Modules are Jahia specific

Repositories	Portlet repositories have been a pipe dream for a long time, but despite multiple efforts they have never taken off and they stay usually specific to a portlet container implementation.	Modules are available on Jahia's forge, developers and integrators are encouraged and free to post them there, or anywhere else they wish.
Performance	A page will be fully dependent of the rendering speed of each portlet to achieve good performance, which may be difficult if closed source portlets are present in the system.	Modules have built-in support for page caching if they re-use Jahia-stored content, which is the general case.

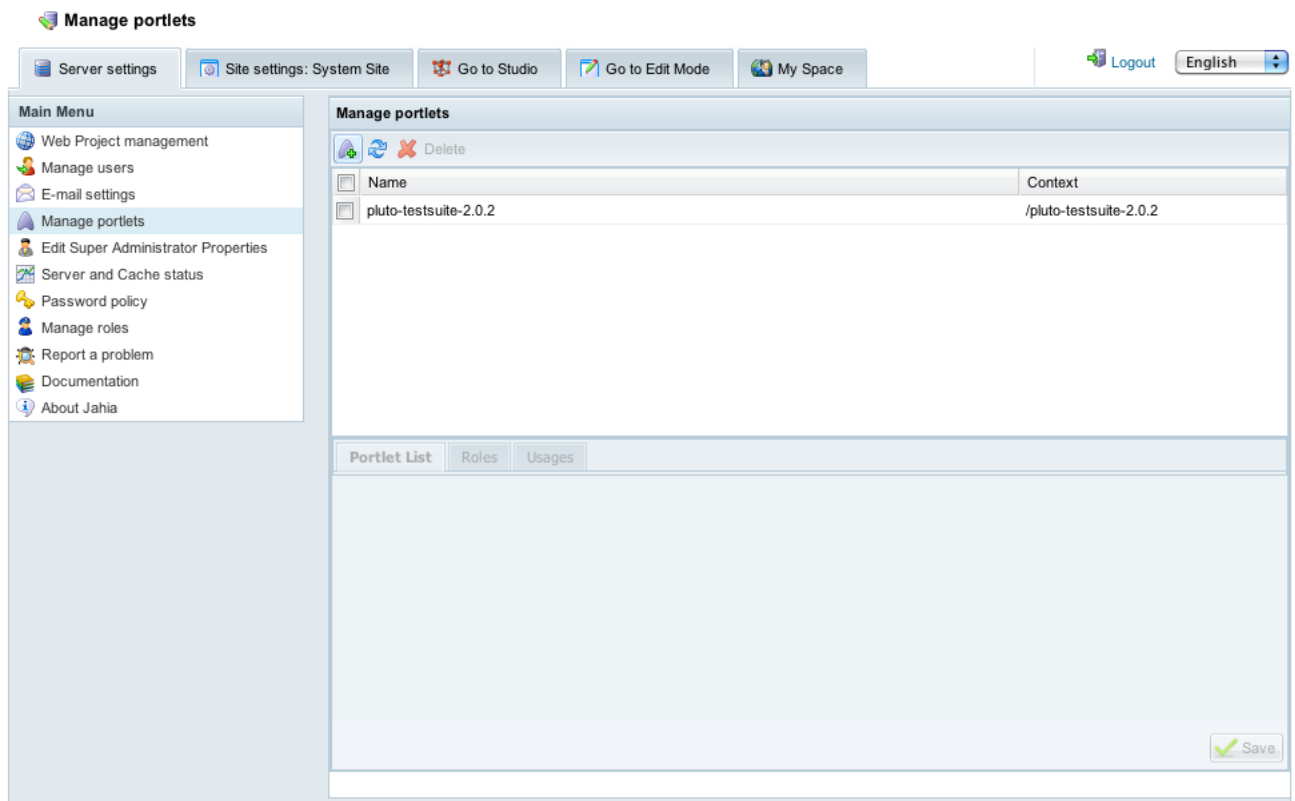
In general, integrators looking for a powerful and rapid integration solutions will probably want to use modules. The main use case for portlet usage is the integration of legacy applications that are only available as portlets. In the case of other systems (servlets, web services) it is preferred to use modules as the integration will be more robust, easier to re-use and deploy and of course to maintain.

2 Quick tutorial: deploying the portlet test suite

Before diving deeper into portlet integration, let's start with a quick example of portlet deployment and usage. In this tutorial we will deploy the Apache Pluto test suite portlet, which is used for compatibility testing with the specification.

Steps :

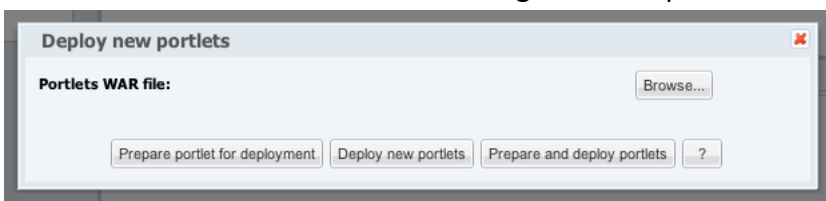
1. Download the Apache Pluto portlet testsuite from :
<http://search.maven.org/remotecontent?filepath=org/apache/portals/pluto/pluto-testsuite/2.0.2/pluto-testsuite-2.0.2.war>
2. Launch Jahia
3. Go into the administration screen, select Server settings and then Manage Portlet
4. Click on the "Deploy new portlets" button



[Back to menu](#)

© Copyright 2002-2011 Jahia Solutions Group SA - All rights reserved. Jahia 6.5.0.0 r38090.38090

5. Click on the "Browse.." button and navigate to the pluto-testsuite-2.0.2.war file and select it



- Then click on “Deploy new portlets”. If all goes well you should see the portlet application listed as “pluto-testsuite-2.0.2” in the Manage portlets screen. By selecting the application, you can then view the list of portlets embedded in the application :

Portlet Name	Portlet Description
Test Portlet #1	TestSuiteDescription
Test Portlet #2	TestSuiteDescription
JSR 286 Test Portlet	JSR 286 Compatibility Tests
286TestCompanionPortlet	JSR 286 Test Portlet Companion

- Go back to the server settings screen and create a web project if you haven’t done it yet. One quick way of doing that is to simply import the “ACME Website Demo 6.5” and click on “Proceed”:

Web Project List

Server settings | Site settings: System Site | Go to Studio | Go to Edit Mode | My Space | Logout | English

Main Menu

- Web Project management
- Manage users
- E-mail settings
- Manage portlets
- Edit Super Administrator Properties
- Server and Cache status
- Password policy
- Manage roles
- Report a problem
- Documentation
- About Jahia

Web Project management

Create a new Web Project

Use prepackaged demo project (will be used as default)

Please select the demo to be imported:

Import archive

Please select the file to import: no file selected

Or give the path of an import file on the server:

© Copyright 2002-2011 Jahia Solutions Group SA - All rights reserved. Jahia 6.5.0.0 r38090.38090

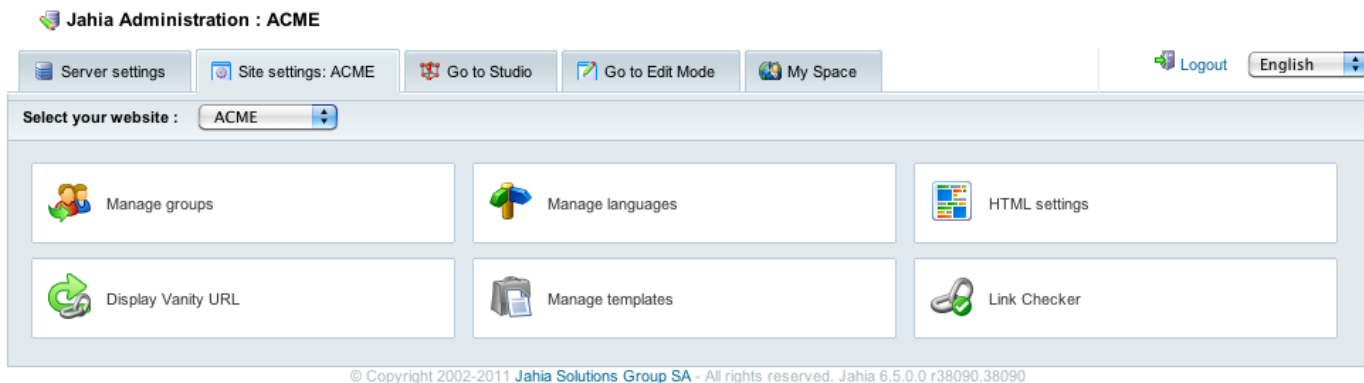
- Leave the defaults if they are ok and click “Import” :

Web Project variables

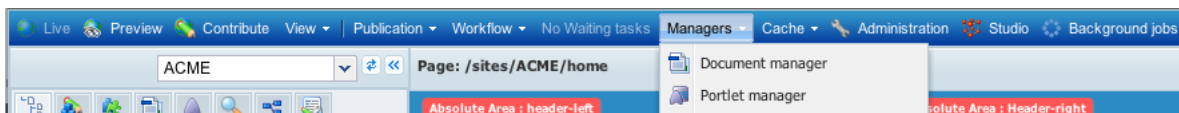
	Name								
<input checked="" type="checkbox"/>	Shared content: systemsite.zip								
<input checked="" type="checkbox"/>	<table> <tr> <td>Web project title*</td> <td><input type="text" value="ACME"/></td> </tr> <tr> <td>Web projekt name*</td> <td><input type="text" value="localhost"/></td> </tr> <tr> <td>Web Project key*</td> <td><input type="text" value="ACME"/></td> </tr> <tr> <td>Please choose a template set</td> <td><input type="text" value="templates-web-blue"/></td> </tr> </table>	Web project title*	<input type="text" value="ACME"/>	Web projekt name*	<input type="text" value="localhost"/>	Web Project key*	<input type="text" value="ACME"/>	Please choose a template set	<input type="text" value="templates-web-blue"/>
Web project title*	<input type="text" value="ACME"/>								
Web projekt name*	<input type="text" value="localhost"/>								
Web Project key*	<input type="text" value="ACME"/>								
Please choose a template set	<input type="text" value="templates-web-blue"/>								
<input checked="" type="checkbox"/>	Shared content: users.zip								

© Copyright 2002-2011 Jahia Solutions Group SA - All rights reserved. Jahia 6.5.0.0 r38090.38090

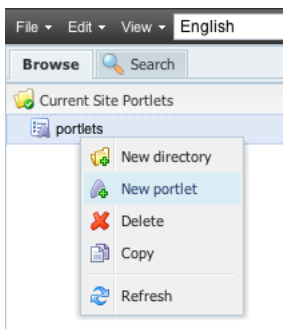
- The previous step will take some time to create the site. Once it has completed, it will display the list with an “ACME” site now present. Then go back to the main administration screen, click on Site settings and make sure the “ACME site” is selected in the site drop-down list:



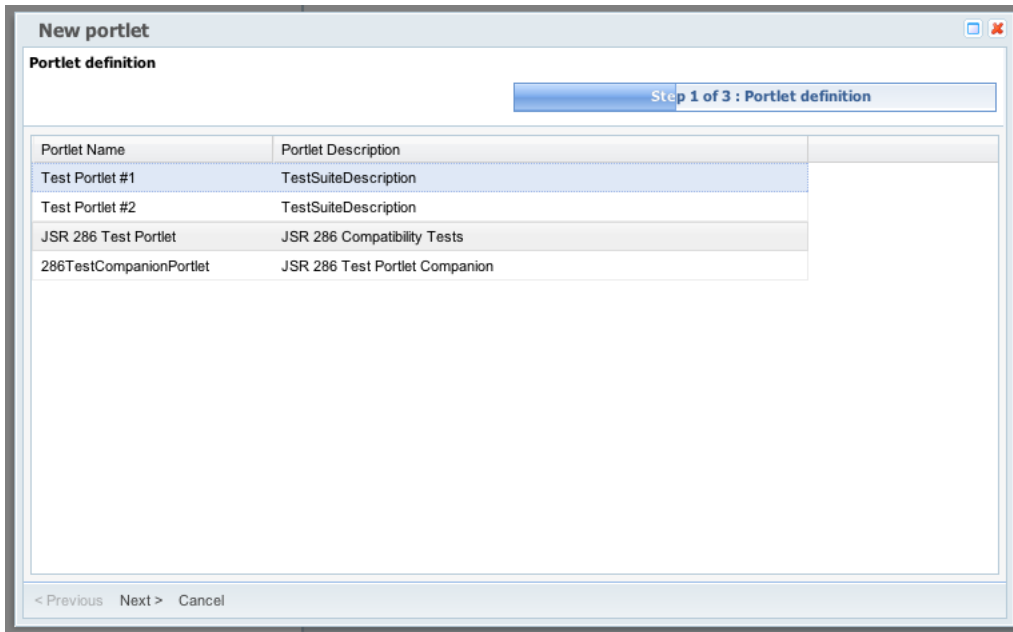
- Then click on “Go to Edit mode”
- Open the portlet manager from the “Managers” drop down menu in the top Edit mode toolbar:



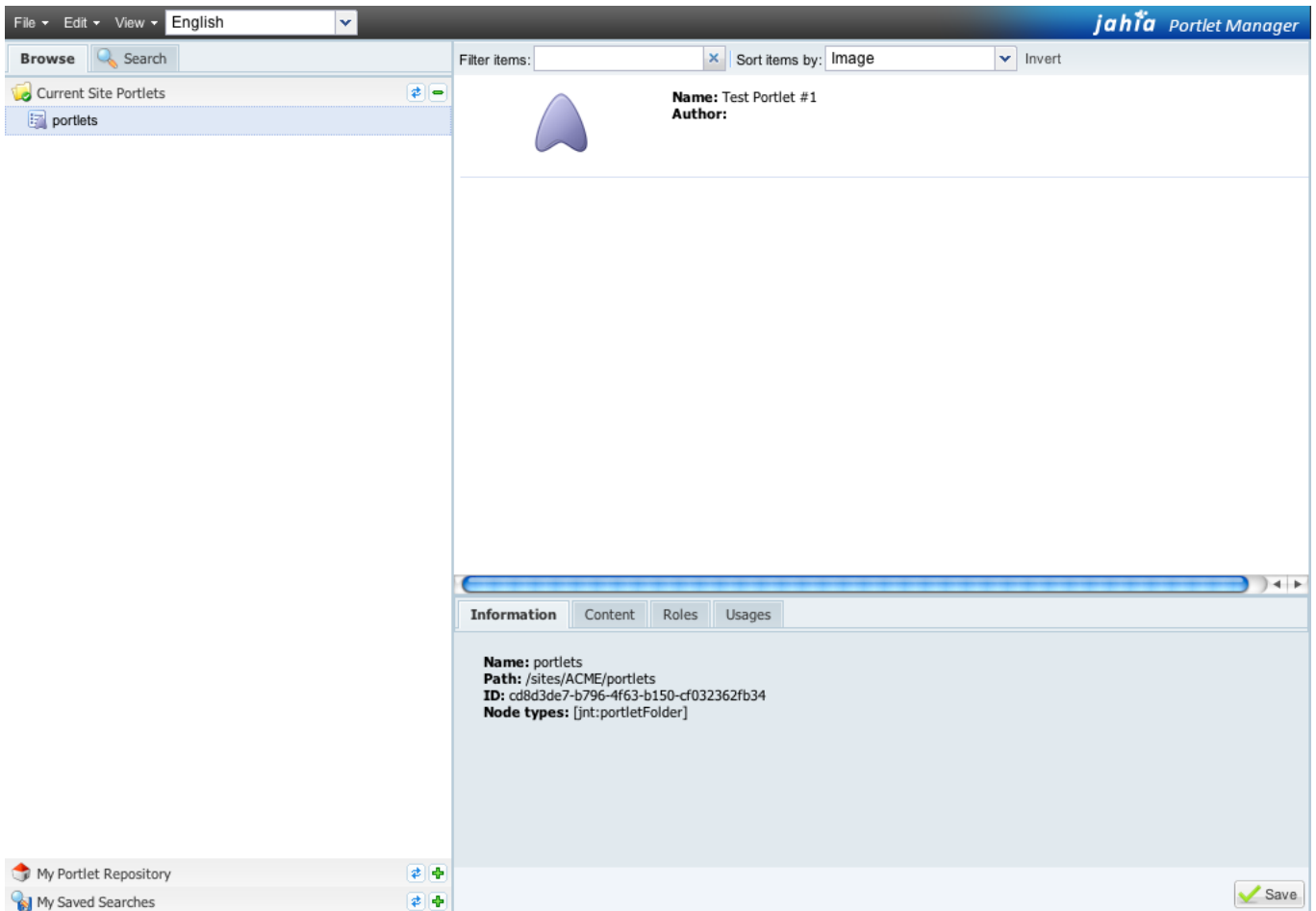
- You will see an empty manager, that is perfectly normal as we haven’t created any portlet instances yet. Select the “portlets” node under the “Current site portlets” node and right click :



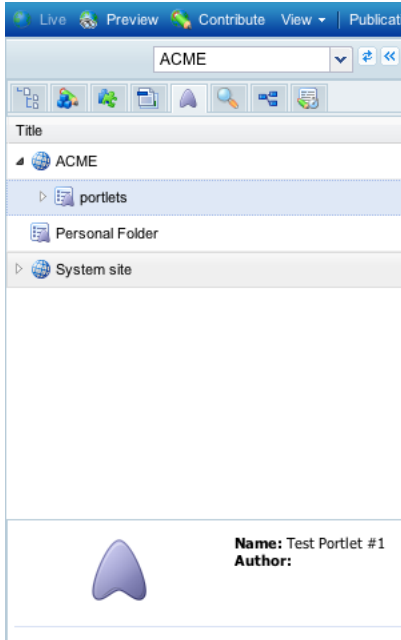
13. Click on “New portlet”. This will open up the portlet wizard :



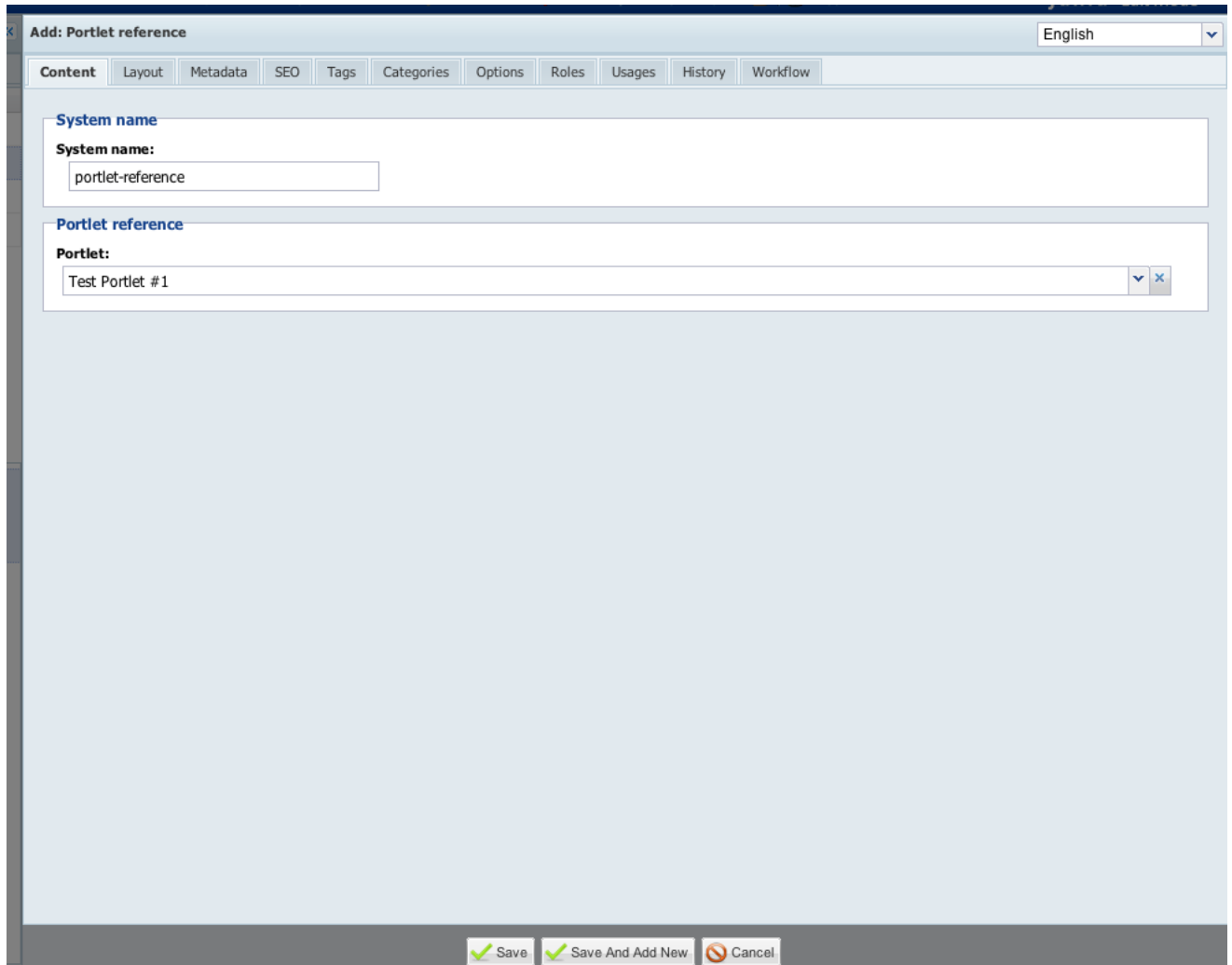
14. Select “Test portlet #1” and click Next on all the next screens and confirm by pressing “Finish”. You have now created a portlet instance for the portlet called “Test portlet #1”. The portlet manager will refresh and you will see the portlet instance displayed in the right panel:



15. You may now close the portlet manager and go back to the main Edit mode screen. In the left side panel, click on the portlet tab, open up the “ACME” site node and select the “portlets” node. You should then see in the lower half of the left side panel the newly created “Test portlet #1” as illustrated below :



16. You can then simply drag and drop the portlet instance into any page you wish, confirm the portlet reference in the content editing screen and click “Save” :



Add: Portlet reference English

Content Layout Metadata SEO Tags Categories Options Roles Usages History Workflow

System name

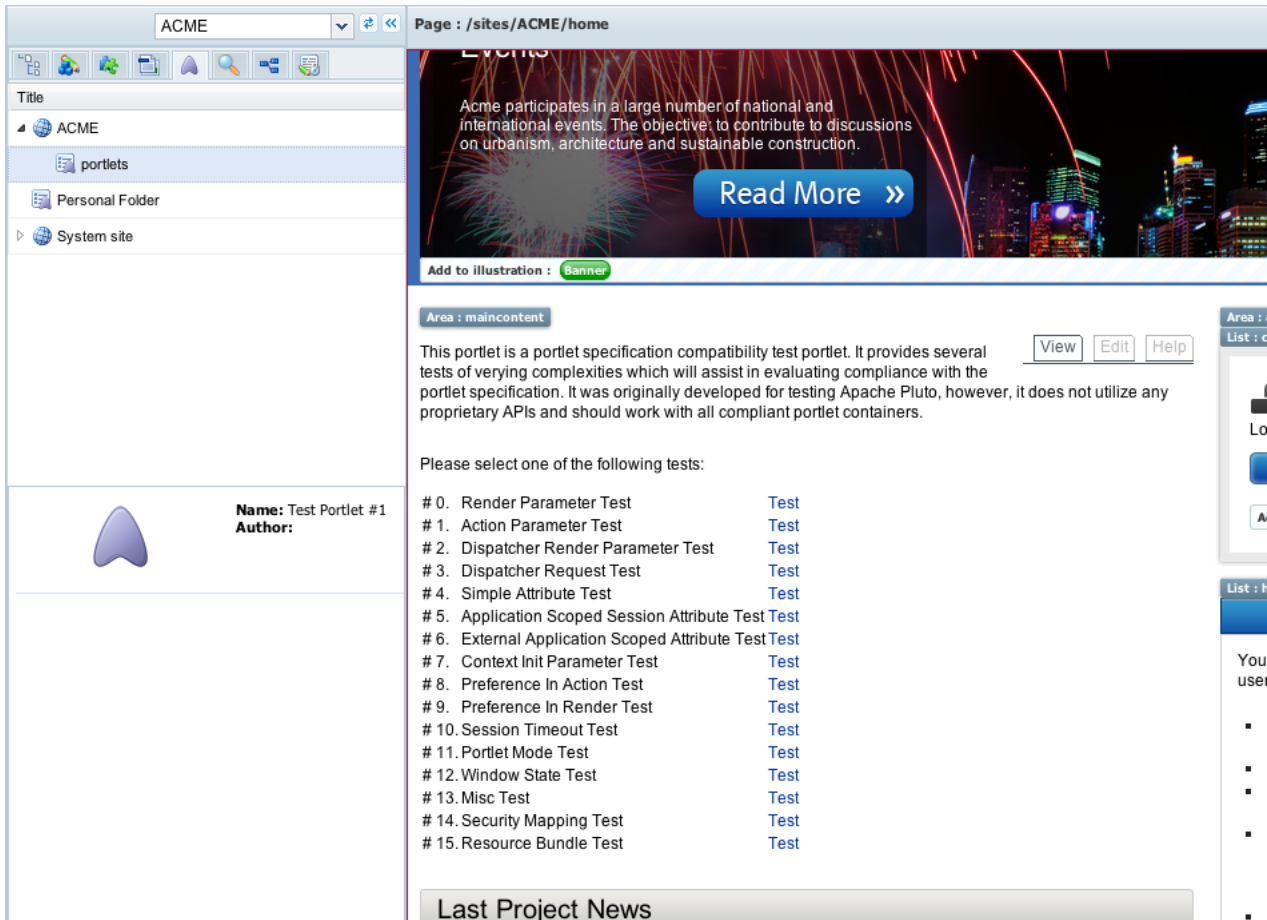
System name:
portlet-reference

Portlet reference

Portlet:
Test Portlet #1

Save Save And Add New Cancel

17. The page will then refresh and should look something like this, depending on where you have positioned the portlet:



18. You can then switch to preview mode to test your portlet (interacting with portlet in Edit mode is not supported nor recommended).

This concludes our quick tutorial on how to deploy, instantiate and position in a page a portlet.

3 Deployment

In this section we will talk about the overall process of portlet application preparation and deployment. As you will see there are different possibilities and user interfaces that may be used to accomplish these tasks.

3.1 What is portlet preparation? Why do I need it?

Each portlet container implementation has its own way of working internally with portlet application. In the case of the Apache Pluto portlet container embedded in Jahia, it works by inserting, upon deployment

or at packaging time, a servlet that will be used to dispatch to portlets. The portlet container then calls the servlet, which in turn will relay the request to the requested portlet.

Portlet preparation is a modification on the WEB-INF/web.xml file to inject the proper configuration for each portlet to be mapped through the PortletServlet servlet used internally by Apache Pluto. It is therefore a required step for proper deployment inside Jahia.

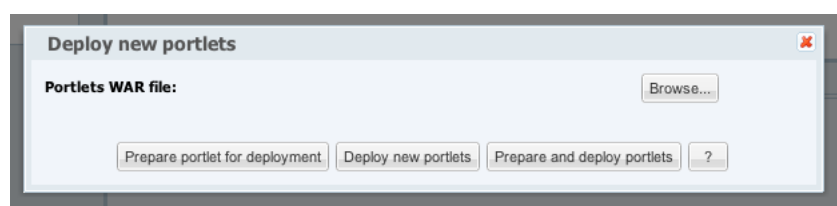
You can check if a portlet application has already been prepared by opening its WEB-INF/web.xml file. If you see lines similar to the following :

```
<servlet>
<servlet-name>TestPortlet1</servlet-name>
  <servlet-
class>org.apache.pluto.container.driver.PortletServlet</servlet-class>
  <init-param>
    <param-name>portlet-name</param-name>
    <param-value>TestPortlet1</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

This means the portlet has already been prepared, and you may deploy it without preparing it first.

3.2 Deploying through the administration user interface

As we have seen in the quick tutorial in section 2, it is possible to deploy a portlet application directly from the administration interface using the “Deploy new portlets” in the Manage Portlets administration screen. This brings up the following dialog box :



As you can see it offers preparation and deployment, depending on what you need.

3.3 Deploying directly through the file system

If a portlet application has already been prepared, you can deploy it simply by packaging it as a WAR file and copying the WAR package inside the tomcat/webapps directory. Tomcat will then explode the WAR

file and start the application, at which point Jahia will detect it and deploy it. You can check that it has been properly deployed if you see the following lines in the Jahia logs:

```
2011-05-25 12:05:27,318: INFO [PortletContextManager] - Portlet
Context '/pluto-testsuite-2.0.2' registered.
2011-05-25 12:05:27,319: INFO [PortletContextManager] - Registered
portlet application for context '/pluto-testsuite-2.0.2'
2011-05-25 12:05:27,319: INFO [PortletContextManager] - Registering 4
portlets for context //pluto-testsuite-2.0.2
```

4 Using in a Jahia page

As we have seen, using a portlet inside a Jahia requires understanding a few concepts. We will now detail these.

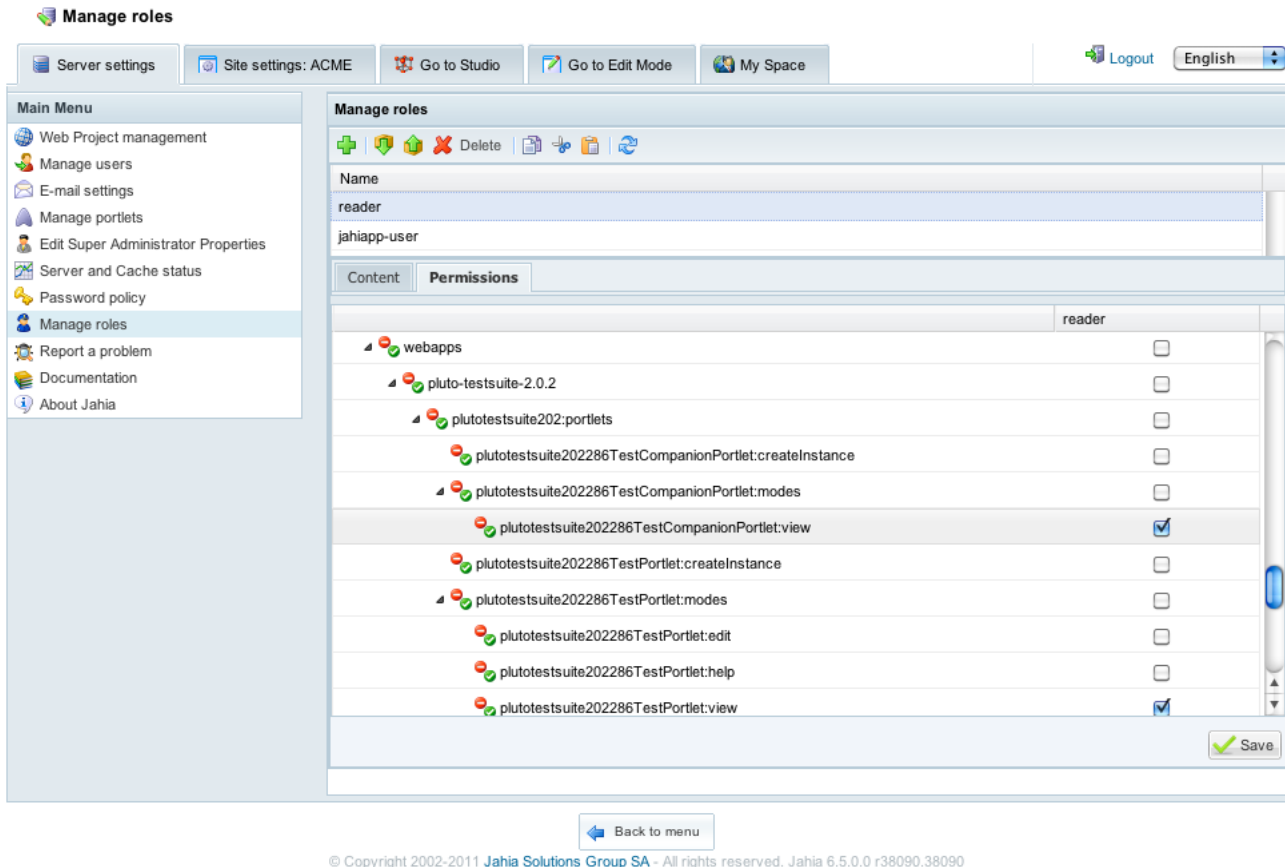
4.1 Portlet instances

Before you may use a portlet inside a Jahia page, you must first create a portlet instance. An instance may be seen as similar to what objects are to classes, meaning that it will have its own copy of preferences and configuration for the instance. An instance may of course be used on multiple pages on a Jahia site, making it easy to re-use the same portlet in multiple locations. If you want different instances of portlets on different pages, you must create different instances and put them on the pages. A good example of the difference between a portlet definition and a portlet instance is a Google Gadget portlet. Its definition offers the technology to render a Google Gadget, while the instance would be a specific Google Gadget, such as a calendar or a news gadget.

For an example on how to create an instance, please refer to our “Quick tutorial” that shows how this is done.

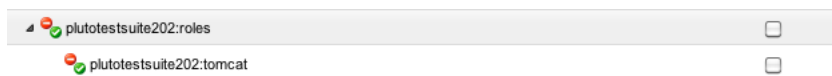
4.2 Portlet roles and Jahia permissions

Roles defined in a portlet web application will be mapped upon deployment as Jahia permissions. You can therefore see them appear in the “Manage roles” administration screen. Here is an example screenshot of permissions related to a portlet:



As

you can see there are more than the simple roles defined by a portlet. Jahia has also automatically created permissions for the various portlet modes, as well as a “createInstance” permission for each portlet definition. This last permission allows administrators to specify which roles may create portlet instances and therefore limit the feature to a subset of users. The list of roles mapped as permissions may be seen in the example below:



5 Hello world portlet

The Hello World portlet is based (with some modifications, notably in the pom.xml file) the example available at the Apache Pluto wiki: <http://wiki.apache.org/portals/Pluto/Pluto1.1ExamplePortlet> and updated based on the instructions available here: <http://portals.apache.org/pluto/v20/deploying.html>

5.1 Requirements

In order to work on this example, you will need the following tools installed.

- Jahia 6.5, <http://www.jahia.org>
- Oracle JDK 1.6 or more recent, <http://java.oracle.com>
- Apache Maven, <http://maven.apache.org>, make sure you follow all installation instructions

5.2 Project structure

Here is the Hello world directory structure.

```
HelloWorldPortlet (top level directory)
|- pom.xml (the pom file)
|- src (Subdir containing main subdirectory)
    |- main (Subdir containing java, resources and webapp subdirs)
        |- java (java source code goes under here)
            |           \- com
            |               \- mycompany
            |                   \- portlet
            |                       \- HelloWorldPortlet.java (portlet
source)
        |- webapp (webapp resources (jsp, css, images) go under
here)
            \- jsp
                \- HelloWorldPortletView.jsp (for view mode)
                    \- HelloWorldPortletEdit.jsp (for edit mode)
            \- META-INF
                \- HelloWorldPortlet.xml (Tomcat context deployment
descriptor)
            \- WEB-INF
                \- portlet.xml (JSR-168 deployment descriptor)
                \- web.xml (This will be modified by maven-pluto-
plugin)
```

5.3 Maven project descriptor

First let's setup the Maven project descriptor file, pom.xml in our project's root directory with the following content:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">

  <!-- Change this to something akin to your java package structure --
  >
```

```

<groupId>com.mycompany.portlet</groupId>
<modelVersion>4.0.0</modelVersion>
<!-- Version of this app -->
<version>0.1-alpha1</version>
<!-- Base name of the war file without .war ext -->
<artifactId>HelloWorldPortlet</artifactId>
<packaging>war</packaging>
<name>${pom.artifactId}</name>
<!-- Dependency Version Properties
===== -->
<properties>
  <pluto.version>2.0.2</pluto.version>
  <portlet-api.version>2.0</portlet-api.version>
  <servlet-api.version>2.4</servlet-api.version>
  <jsp-api.version>2.1</jsp-api.version>
  <junit.version>3.8.1</junit.version>
</properties>
<dependencies>
  <dependency>
    <groupId>javax.portlet</groupId>
    <artifactId>portlet-api</artifactId>
    <version>${portlet-api.version}</version>
    <scope>provided</scope><!-- Prevents addition to war file -->
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>${servlet-api.version}</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.portals.pluto</groupId>
    <artifactId>pluto-util</artifactId>
    <version>${pluto.version}</version>
    <scope>provided</scope>
  </dependency>
  <!-- Any other build or deployment dependancies go here -->
</dependencies>
<build>
  <finalName>${pom.name}</finalName>
  <plugins>
    <!-- configure to use Java 6 to compile (change to your JDK) -->
    <plugin>

```

```

    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
      <source>1.6</source>
      <target>1.6</target>
    </configuration>
  </plugin>
  <!-- configure maven-war-plugin to use updated web.xml -->
  <plugin>
    <artifactId>maven-war-plugin</artifactId>
    <configuration>
      <webXml>${project.build.directory}/pluto-
resources/web.xml</webXml>
    </configuration>
  </plugin>
  <!-- bind 'pluto2:assemble' goal to 'generate-resources'
lifecycle -->
  <plugin>
    <groupId>org.apache.portals.pluto</groupId>
    <artifactId>maven-pluto-plugin</artifactId>
    <version>${pluto.version}</version>
    <executions>
      <execution>
        <phase>generate-resources</phase>
        <goals>
          <goal>assemble</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
</plugins>
</build>

</project>

```

5.4 Portlet Java code

The main portlet code is contained in the `src/main/java/com/mycompany/portlet/HelloWorldPortlet.java` class, which we show here :

```

package com.mycompany.portlet;

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

import javax.portlet.ActionRequest;
import javax.portlet.ActionResponse;

```

```

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

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**
 * Example Hello World portlet to demonstrate deployment using
 * Maven 2.
 *
 * @author <a href="mailto:cdoremus@apache.org">Craig Doremus</a>
 */
public class HelloWorldPortlet extends GenericPortlet {

    public void doView(RenderRequest req, RenderResponse res)
        throws IOException, PortletException {
        //Required call for use of getWriter() and
        getPortletOutputStream()
        res.setContentType("text/html;charset=UTF-8");
        PrintWriter out = res.getWriter();
        out.println("<h2>");
        out.println("Hello World!");
        out.println("</h2>");
    }

    public void processAction(ActionRequest req, ActionResponse res)
        throws IOException, PortletException {
    }
}

```

5.5 Portlet descriptor

The src/main/webapp/WEB-INF/portlet.xml file has the following content:

```

<portlet-app
  xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
  version="1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-
app_1_0.xsd
                        http://java.sun.com/xml/ns/portlet/portlet-
app_1_0.xsd">

```

```
<portlet>
  <description>Hello World as a portlet app</description>
  <portlet-name>HelloWorldPortlet</portlet-name>
  <display-name>Hello World Portlet</display-name>
  <portlet-
class>com.mycompany.portlet.HelloWorldPortlet</portlet-class>
  <supports>
    <mime-type>text/html</mime-type>
    <portlet-mode>VIEW</portlet-mode>
  </supports>
  <portlet-info>
    <title>Hello World Portlet</title>
  </portlet-info>
</portlet>
</portlet-app>
```

5.6 Web application descriptor

The src/main/webapp/WEB-INF/web.xml file :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
</web-app>
```

is empty, but it will be modified upon project building to prepare it for deployment by inserting the PortletServlet entries.

5.7 The Tomcat deployment descriptor

The file src/main/resources/HelloWorldPortlet.xml :

```
<Context path="/HelloWorldPortlet"
    docBase="HelloWorldPortlet"
    crossContext="true"/>
```

5.8 Building and deploying

You can package the portlet application by using the following command:

```
mvn package
```

This will generate a portlet application WAR file in the target/ directory. You can then either copy this WAR file to the tomcat/webapps directory of your Jahia installation, or add a section to plugins section of the Maven pom.xml file to automate the deployment as in the example below :

```
<plugin>
  <artifactId>maven-antrun-plugin</artifactId>
  <executions>
    <execution>
      <phase>integration-test</phase>
      <configuration>
        <tasks>
          <property environment="env"/>
            <!-- This assumes that you have set a CATALINA_HOME
environmental variable -->
          <property name="tomcat.home"
value="\${env.CATALINA_HOME}"/>
          <copy file="target/\${pom.name}.war"
todir="\${tomcat.home}/webapps"/>
        </tasks>
      </configuration>
      <goals>
        <goal>run</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

You can then deploy simply by launching the command:

```
mvn -Dtomcat.home=/path/to/jahia/tomcat integration-test
```

5.9 *Redeployment*

During development, you will often want to redeploy changes. This is possible by simply overwriting the deployed WAR file in the tomcat/webapps directory while it is running. This will undeploy and deploy the portlet application. This of course requires that your portlet application properly dispose of any resources upon undeployment.

5.10 Troubleshooting deployment

If something goes wrong during portlet deployment, try restarting the Jahia server. If that doesn't work, try removing the portlet WAR and exploded directory and try deployment again.

5.11 Learning more

See our "Additional resources" section for learning more about portlet application development, packaging and deployment.

6 Support for Bridges

Portlet bridges are layers that allow portlets to embed non-portlet application, such as Struts applications, JSF applications and the like. Although not commercially supported by Jahia, it is interesting to know that they are available. You may find more information about bridges in the Apache Portals Bridges project at the following URL: <http://portals.apache.org/bridges/>

7 How to share portlet objects?

This section assumes that you are familiar with portlet specification JSR286

7.1 Public Render Parameters

Public render parameters are intended for sharing view state across portlets. Using public render parameters instead of events avoids the additional process event call and enables the end-user using the browser navigation and bookmarking if the portal stores the render parameters in the URL.

An example where public render parameters are useful is the following: a weather portlet wants to display the weather of a selected city. It therefore uses the public render parameters for encoding the zip code.

The user now adds additional portlets on the page that also have zip code as one of their public render parameters, like a map portlet displaying the location of the city and a tourist information portlet displaying tourist information for the selected city.

For more details on public render parameters, see PLT.11.1.1.2.

7.2 Portlet Events

Portlet events are intended to allow portlets to react to actions or state changes not directly related to an interaction of the user with the portlet. Events could be the result of a user interaction with other portlets. The portlet event model is a loosely coupled, brokered model that allows creating portlets as stand-alone portlets that can be wired together with other portlets at runtime.

An example where a portlet may want to offer receiving events is for state changes triggered by simple user interactions, e.g. adding an item to a shopping cart. By offering this as an event to other portlets these can trigger adding items to the shopping cart based on the user interactions happening inside these portlets. In contrast to using the portlet application scope session this will work across portlet application boundaries.

For more details on events, see PLT.15.

7.3 Jahia Shared Map

Jahia shared map is a map that is accessible by any portlets and Jahia template.

Within Jahia, it's stored as user session attribute named "jahiaSharedMap" and can be retrieved thanks to the following code:

```
Map sharedMap = (Map) session.getAttribute("jahiaSharedMap");
```

Within a portlet, it's stored as a portlet request attribute named "jahiaSharedMap". It can be retrieved thanks to the following code:

```
Map map = (Map) portletRequest.getAttribute("jahiaSharedMap");
```

Jahia shared map can be updated only during the processAction(...) or from a template.

```
Map map = (Map) actionRequest.getAttribute("jahiaSharedMap");  
public void processAction(ActionRequest actionRequest, ActionResponse  
actionResponse) throws PortletException, IOException {  
    Map map = (Map) actionRequest.getAttribute("jahiaSharedMap");
```

```

if (map == null) {
    map = new HashMap<Object, Object>();
}

map.put("date", new Date());
actionRequest.setAttribute("jahiaSharedMap", map);
}

```

If the portlet tries to modify the map during the render phase of the portlet, an exception is thrown.

```

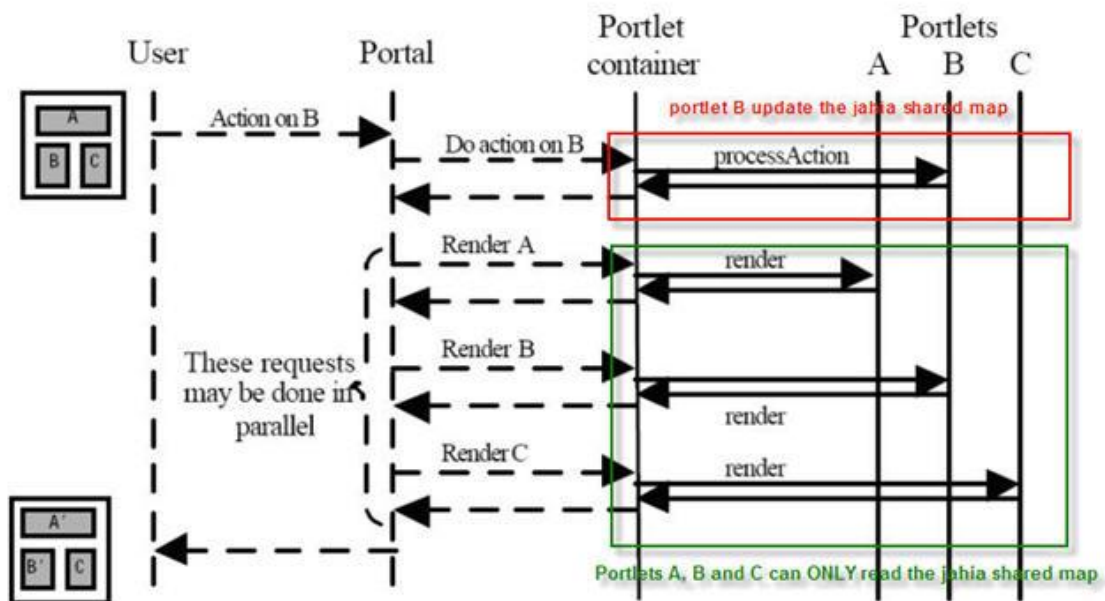
public void render(RenderRequest renderRequest, RenderResponse
renderResponse) throws PortletException, IOException {
    Map map = (Map) renderRequest.getAttribute("jahiaSharedMap");
    if (map == null) {
        map = new HashMap<Object, Object>();
    }

    map.put("date", new Date());

    // an exception will be thrown
    renderResponse.setAttribute("jahiaSharedMap", map);
}

```

Here is an example of execution:



Portlet A, B and C could be in different application (context).

7.3.1 *How to share a simple object?*

1. Get the shared map from the action portlet request or Jahia session

```
Map map = (Map) actionRequest.getAttribute("jahiaSharedMap");
```

2. Put the object in the shared map during the portlet action phase or from a Jahia template

```
map.put("date", new Date());
```

3. Update the shared map

```
actionRequest.setAttribute("jahiaSharedMap", map);
```

7.3.2 *How to read a simple object?*

1. Get the shared map from the portlet request or Jahia session

```
Map map = (Map) portletRequest.getAttribute("jahiaSharedMap");
```

2. Get the object in the shared map during the portlet action phase or from a Jahia template

```
Date date = (Date)map.get("date");
```

7.3.3 *How to share a bean?*

To share a bean, you have to:

1. Create a jar that contains the class of the bean
2. Put the jar in the shared lib of your application server

3. Share the bean as any simple object

```
/**
 * SimpleEvent class is in a shared lib.
 *
 * Example of processAction(...) of any portlet
 */
public void processAction(ActionRequest actionRequest, ActionResponse
actionResponse) throws PortletException, IOException {
    Map map = (Map) actionRequest.getAttribute("jahiaSharedMap");

    map.put("simpleEvent",
        new SimpleEvent(new Date(), "simple event test"));
    actionRequest.setAttribute("jahiaSharedMap", map);
}

/**
 * SimpleEvent class is in a shared lib.
 *
 * Example of render(...) of any other portlet
 */
public void render(RenderRequest renderRequest, RenderResponse
renderResponse) throws PortletException, IOException {
    Map map = (Map) renderRequest.getAttribute("jahiaSharedMap");

    SimpleEvent event = ((SimpleEvent)map.get("simpleEvent"));
}
```

8 Performance

Portlet performance is very important, as it may often be the weakest performance link in a page. As portlets are very dynamic by nature, it is important to make sure that their rendering performance is optimal. When designing your portlet, make sure that you will never calculate data systematically on render. Also, it might be a good idea to either use the Portlet API caches or integrate your own caches to make sure you do not unnecessarily regenerate the same HTML output over and over.

Finally it is always recommended to load test the pages on which the portlet will be used, to make sure the performance impact is minimal.

9 Using portlets on other application servers

Using portlets on other applications servers than Tomcat may require additional or different deployment steps. Please refer the Jahia Configuration and Fine Tuning guide that contains sections for the various supported applications servers and instructions on how to deploy portlets on those platforms.

10 Additional resources

We list here below some additional resources that are very useful to learn more about the portlet specification as well as learning how to write portlets.

- Deploying portlet to Pluto Portal: <http://portals.apache.org/pluto/v20/deploying.html>
- Introducing Java Portlet Specifications:
<http://developers.sun.com/portalserver/reference/techart/jsr168/>
- Portlet API v1: <http://jcp.org/aboutJava/communityprocess/final/jsr168/>
- Portlet API v2: <http://jcp.org/aboutJava/communityprocess/edr/jsr286/>
- Understanding the Portlet Container 2.0 Beta Software:
http://java.sun.com/developer/technicalArticles/J2EE/sdk_portletcontainer2/
- Hello World Pluto portlet: <http://wiki.apache.org/portals/Pluto/Pluto1.1ExamplePortlet>
- Apache Portals Applications: <http://portals.apache.org/applications/>
- Apache Portals Bridges: <http://portals.apache.org/bridges/>



Jahia Solutions Group SA

*9 route des Jeunes,
CH-1227 Les acacias
Geneva, Switzerland*

<http://www.jahia.com>