Hello World Portlet Rendered with JSP for WebSphere Portal Version 4.1

Table of Contents

Ron Lynn
IBM Hacker
IBM Santa Teresa Labs
June, 2002

Introduction

In our first portal based Hello World article, we've seen how to create a portlet in Java that spits out "Hello world!". While this is instructive, it's not a pragmatic approach to developing portlets. So, what is wrong with it? One problem is the subject of this article. In the simplest "Hello World!" program, the Java code contains both the text "Hello, world!" and the logic for displaying it. It would be far better if we could separate the logic of the portlet from how it gets rendered. This allows us to create portlets that support different meta languages and national languages. Today we might not have a requirement to create a portlet for Wireless Markup Language (WML) or in the German language. But what about tomorrow? We may also be working with a graphic designer who frequently changes the look and feel. If our rendering is done in Java code, it takes a developer and a recompilation to change it. The solution is to render your portlet with JSP. This allows you to create separate JSP files for each supported meta or national language. The designer can have full control over the JSP and change it without ever needing a developer. Let's look at what it takes to build a portlet that calls a JSP for rendering and then look at the JSP and a few of the tags from the portlet tag library. We'll create our deployment descriptors, package everything together and deploy it into the portal.

Creating the directory structure

To start off, you must create a directory structure in which you store your portlet. Here's what we'll use for this portlet:

- `helloWorld\com\ibm\portlets\sample` - where your source will go.
- `helloWorld\WEB-INF` - where the deployment descriptor is stored.
- `helloWorld\WEB-INF\lib` - where the JAR file will go.
- `helloWorld\jsp` - where the JSP files are stored.

Note: All directory and environment references are based on Windows convention. You will need to adjust these accordingly for Unix systems.

Creating the Java code

The sample directory is where we will put our Java source file. Create a file named `HelloWorldFromJSP.java` in the sample directory and open it in your favorite text editor. Here's the class that you will need to type (or cut and paste) into the `HelloWorldFromJSP.java` file:

```
package com.ibm.portlets.sample;

//portlet APIs
import org.apache.jetspeed.portlet.*;
import org.apache.jetspeed.portlets.*;

//Java stuff
```

```
import java.io.*;

public class HelloWorldFromJSP extends AbstractPortlet {

  public void service(PortletRequest request, PortletResponse response)
    throws PortletException, IOException {

    // Include the view jsp
    getPortletConfig().getContext().include("/jsp/view.jsp", request, response);
  }
}
```

Compiling the code

After we have created the source file we are ready to compile the Java code. The script below can be used in a batch file to compile the portlet. We start by defining some environment variables, such as environment variables that define where our Java is installed (JAVA_HOME). It's usually a good idea to compile this using the JDK provided with the WebSphere Application Server, since that is the environment we will be running under. We also need to set the PATH environment variable to include the location where the Java compiler is installed. Set the variable LIBPATH that will point to the directory where the WebSphere JAR files are found. Next, we set a variable called CP for our compilation classpath. The CP variable could be built on a single line, but we broke it up to show the various JAR files required. We then invoke the Java compiler and compile our code. This assumes we are in the helloWorld directory. Adjust your directory paths as appropriate for your installation.

```
set JAVA_HOME=C:\WebSphere\AppServer\java
set PATH=%JAVA_HOME%\bin
set LIBPATH=C:\WebSphere\AppServer\lib
set CP=.
set CP=%CP%;%LIBPATH%\j2ee.jar
set CP=%CP%;%LIBPATH%\websphere.jar
set CP=%CP%;%LIBPATH%\app\wpsportlets.jar
set CP=%CP%;%LIBPATH%\app\wps.jar
set CP=%CP%;%LIBPATH%\app\portlet-api.jar

javac -classpath %CP% com\ibm\portlets\sample\HelloWorldFromJSP.java
```

If this doesn't compile, you must fix the errors before moving on. Some common errors are:

- Typos - mistyped class names, paths, variable names.
- File name - the file name and the class name must match in our case the file name is `HelloWorldFromJSP.java.` and the class name it `HelloWorldFromJSP`. If you have changed one, change the other.
- Editor woes - make sure your editor really saves the file as text. An editor like WordPad does not save as text by default.

Creating the JAR file

Once the Java source is compiled, we need to create a JAR file in the `WEB-INF\lib` directory. Again, assume we are in the helloWorld directory. These statements could be included as part of a batch file if you created above. It relies on having the PATH set to get to the jar program.

```
set JAVA_HOME=C:\WebSphere\AppServer\java
set PATH=%JAVA_HOME%\bin

jar -cv0f .\WEB-INF\lib\HelloWorldFromJSP.jar com/ibm/portlets/sample/*.class
```

Creating the JSP file

Once the Java source is compiled and the JAR file is created, we can turn our attention to the JSP file. The JSP directory is where we will put our JSP source file. Create a file named `view.jsp` in the JSP directory and open it in your favorite text editor. Here's the JSP code that you will need to type (or cut and paste) into the `view.jsp` file:

```
Hello world from the JSP!
```

This is simple enough. Let's go ahead and get this packaged up and running. We'll come back to the JSP after that to see what more we can do to add support for different national and meta languages.

Creating the deployment descriptors

You will need to create two XML files:

- `helloWorld\WEB-INF\web.xml` - the web deployment descriptor.
- `helloWorld\WEB-INF\portlet.xml` - the portlet deployment descriptor.

web.xml - the web deployment descriptor

The web deployment descriptor is needed for WebSphere Portal. Portlets extend Servlets now, so the web deployment descriptor is needed to declare the Servlet (portlet) class.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
        Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">
<web-app id="HelloWorldFromJSPWebApp">
  <display-name>HelloWorldFromJSPPortlet</display-name>
  <servlet id="Servlet_1">
    <servlet-name>HelloWorldFromJSP</servlet-name>
    <servlet-class>com.ibm.portlets.sample.HelloWorldFromJSP</servlet-class>
  </servlet>
  <servlet-mapping id="ServletMapping_1">
    <servlet-name>HelloWorldFromJSP</servlet-name>
    <url-pattern>/HelloWorldFromJSP/*</url-pattern>
  </servlet-mapping>
</web-app>
```

portlet.xml - the portlet deployment descriptor.

The portlet deployment descriptor defines the portlet to the portal. Each portlet is mapped to a Servlet defined in the web deployment descriptor by way of the href attribute on the portlet element. These references are denoted in red. A portlet must define a unique id that each concrete portlet can reference. Each concrete portlet references a portlet using the given id in the href attribute of the concrete-portlet element. These references are denoted in purple.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE portlet-app-def PUBLIC "-//IBM//DTD Portlet Application
        1.1//EN" "portlet_1.1.dtd">
<portlet-app-def>
  <portlet-app uid="com.ibm.portlets.sample.HelloWorldFromJSP.1">
    <portlet-app-name>HelloWorldFromJSP0Portlet</portlet-app-name>
    <portlet href="WEB-INF/web.xml#Servlet_1" id="Portlet_1">
      <portlet-name>HelloWorldFromJSP</portlet-name>
      <cache>
        <expires>0</expires>
        <shared>no</shared>
      </cache>
      <allows>
        <maximized/>
        <minimized/>
      </allows>
      <supports>
        <markup name="html">
          <view/>
        </markup>
      </supports>
    </portlet>
  </portlet-app>
  <concrete-portlet-app uid="com.ibm.portlets.sample.HelloWorldFromJSP.1.2">
    <portlet-app-name>Concrete HelloWorldFromJSP</portlet-app-name>
    <context-param>
      <param-name>Author</param-name>
      <param-value>tcat@us.ibm.com</param-value>
    </context-param>
    <concrete-portlet href="Portlet_1">
      <portlet-name>HelloWorldFromJSP</portlet-name>
      <default-locale>en</default-locale>
      <language locale="en">
        <title>Hello World from JSP</title>
      </language>
    </concrete-portlet>
  </concrete-portlet-app>
</portlet-app-def>
```

Many things can go wrong when creating these XML files. You will not find out until you attempt to deploy the portlet into the portal. Here's some things to double check.

- Verify that for every XML element you have a closing element.
- Check for mistyped class names, especially for this example if you named your class something other than `HelloWorldFromJSP`
- Make sure your id and href attributes match.
- Make sure your editor really saves the file as text.

Creating the WAR file

Finally, we are ready to create the WAR file for distribution. We'll use the standard `jar` command to build the WAR file. We run this from the helloWorld directory.

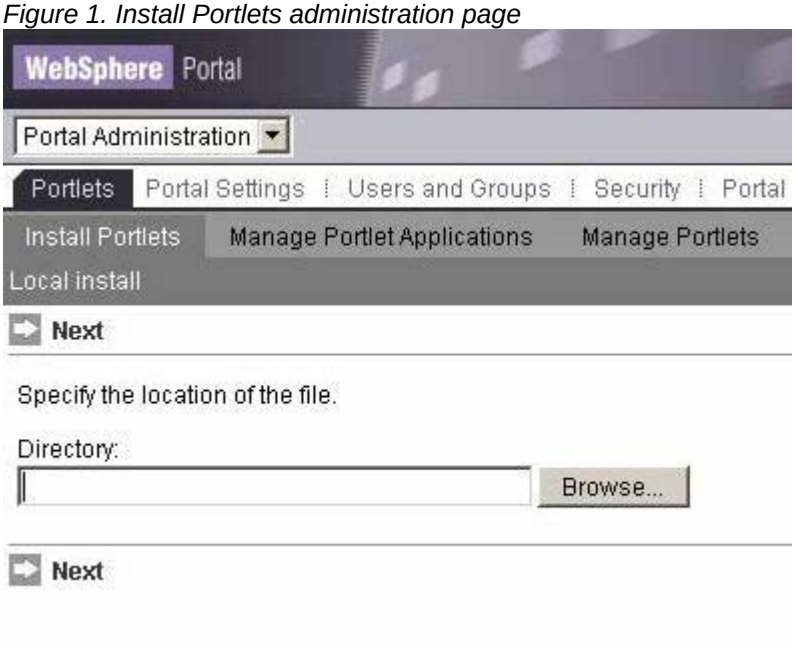
```
set JAVA_HOME=C:\WebSphere\AppServer\java
set PATH=%JAVA_HOME%\bin

jar -cf HelloWorldFromJSP.war WEB-INF jsp
```

Deploying the WAR file

The following steps walk us through the process of deploying the WAR file into the portal.

1. Login to WebSphere Portal as the portal administrator -- `wpsadmin`.
2. Select the Portal Administration page group.
3. Select the Portlets page and the **Install Portlets** portlet. It should be selected for you.

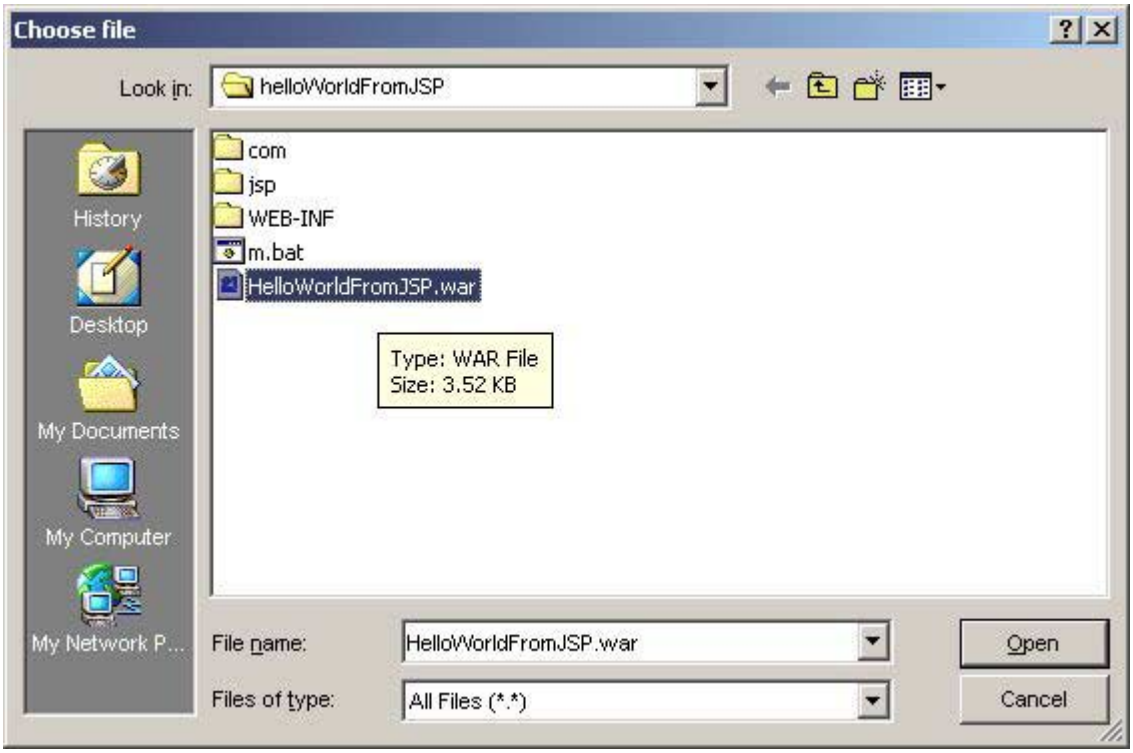
Figure 1. Install Portlets administration page



4. Click the  button.

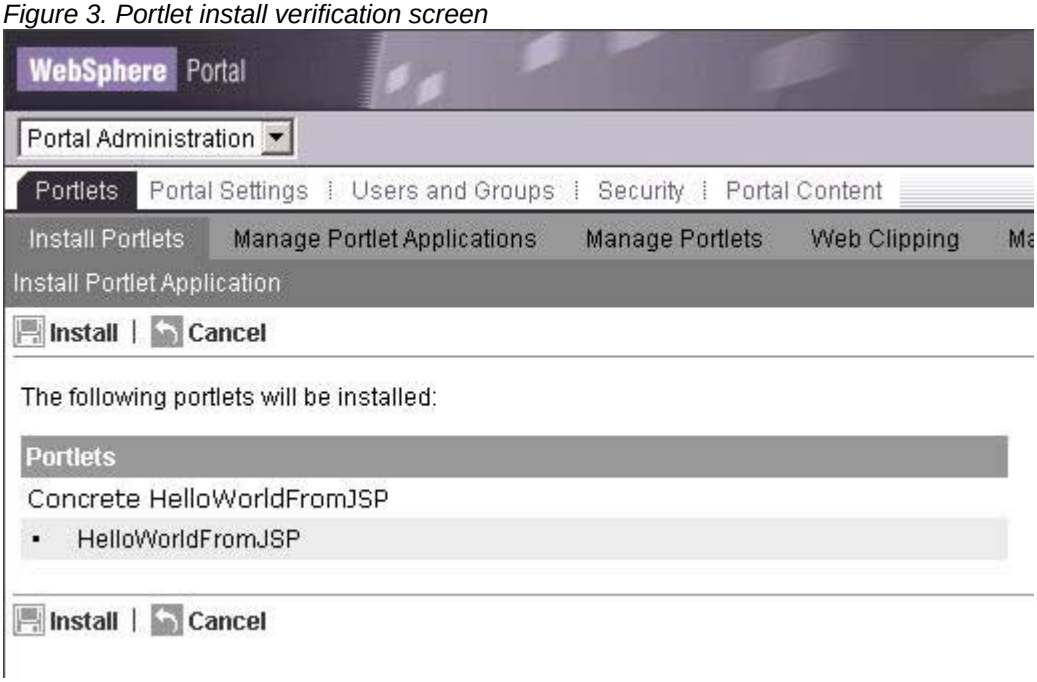
5. Find the location of the WAR file, select it, and click .

Figure 2. File selection dialog



6. Click Next, this could take some time, so wait for it.
7. Verify Portlet is HelloWorldFromJSP, click , again wait for it.

Figure 3. Portlet install verification screen



8. Finally you should see confirmation that .

If you get some sort of error during deployment, this usually means that you've made a mistake in one of your xml files. Take a look in the latest log file:
`%WPS_HOME%\log\wps_YYYY.MM.DD-HH.MM.SS.log`

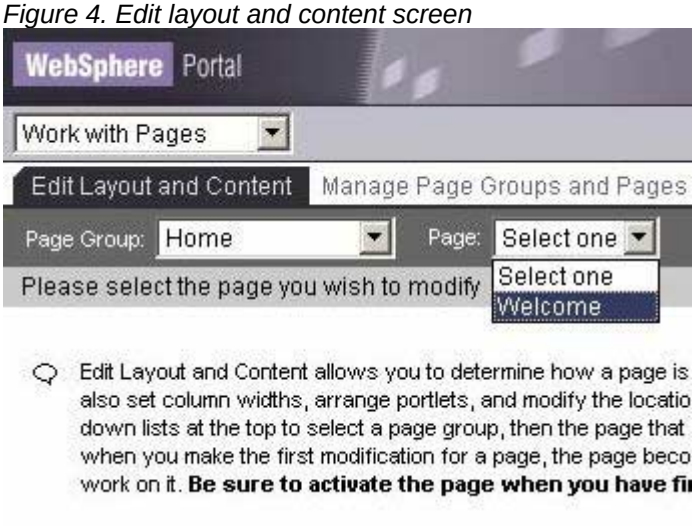
Some common mistakes made with the XML files are listed in section 5 you might also check the following:

- The XML declaration must be first line in file (no whitespace)
- The XML files have the correct names, in the correct case

Adding the portlet to the Welcome page

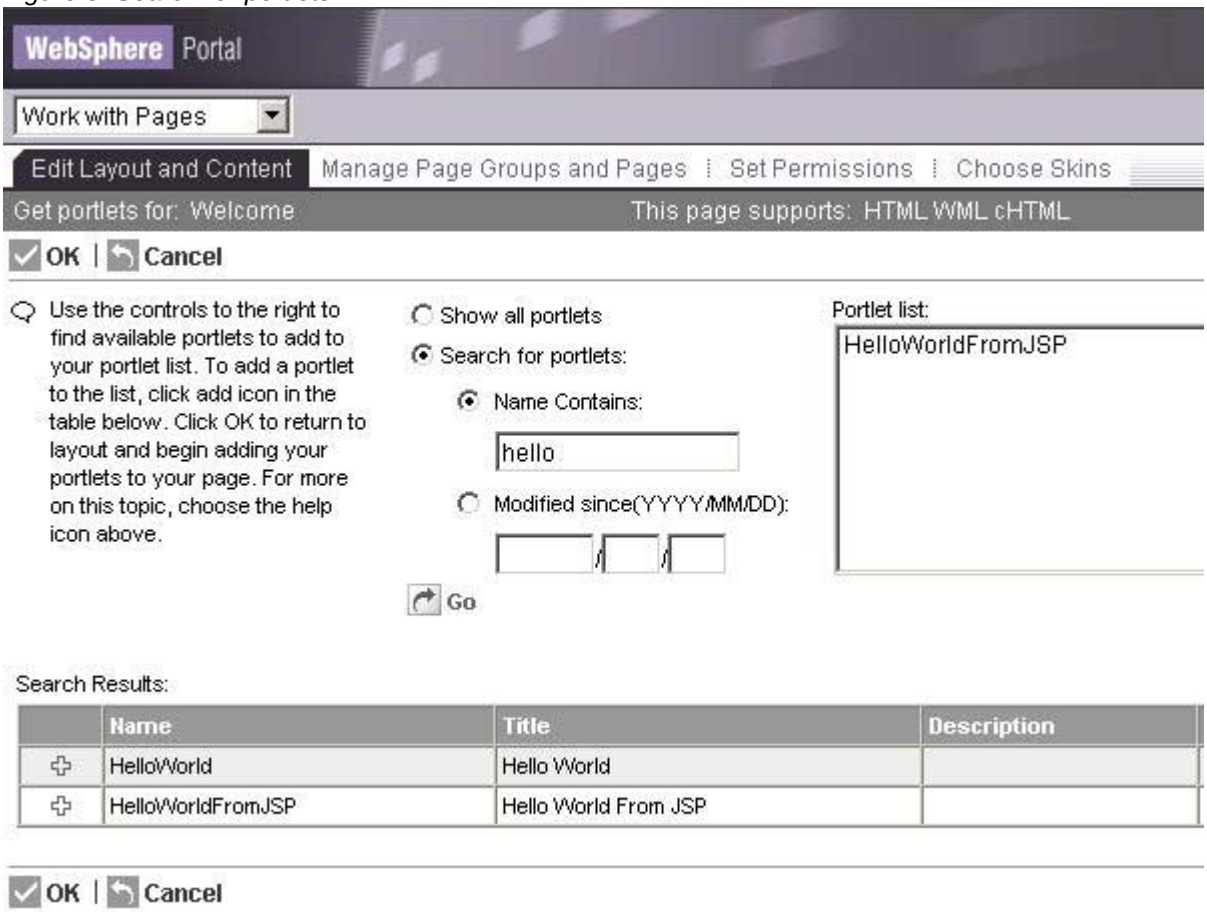
1. Select the Work with Pages page group.
2. On the Edit Layout and Content page, under Page Group, select the **Home** page group.
3. Then select the **Welcome** page.

Figure 4. Edit layout and content screen



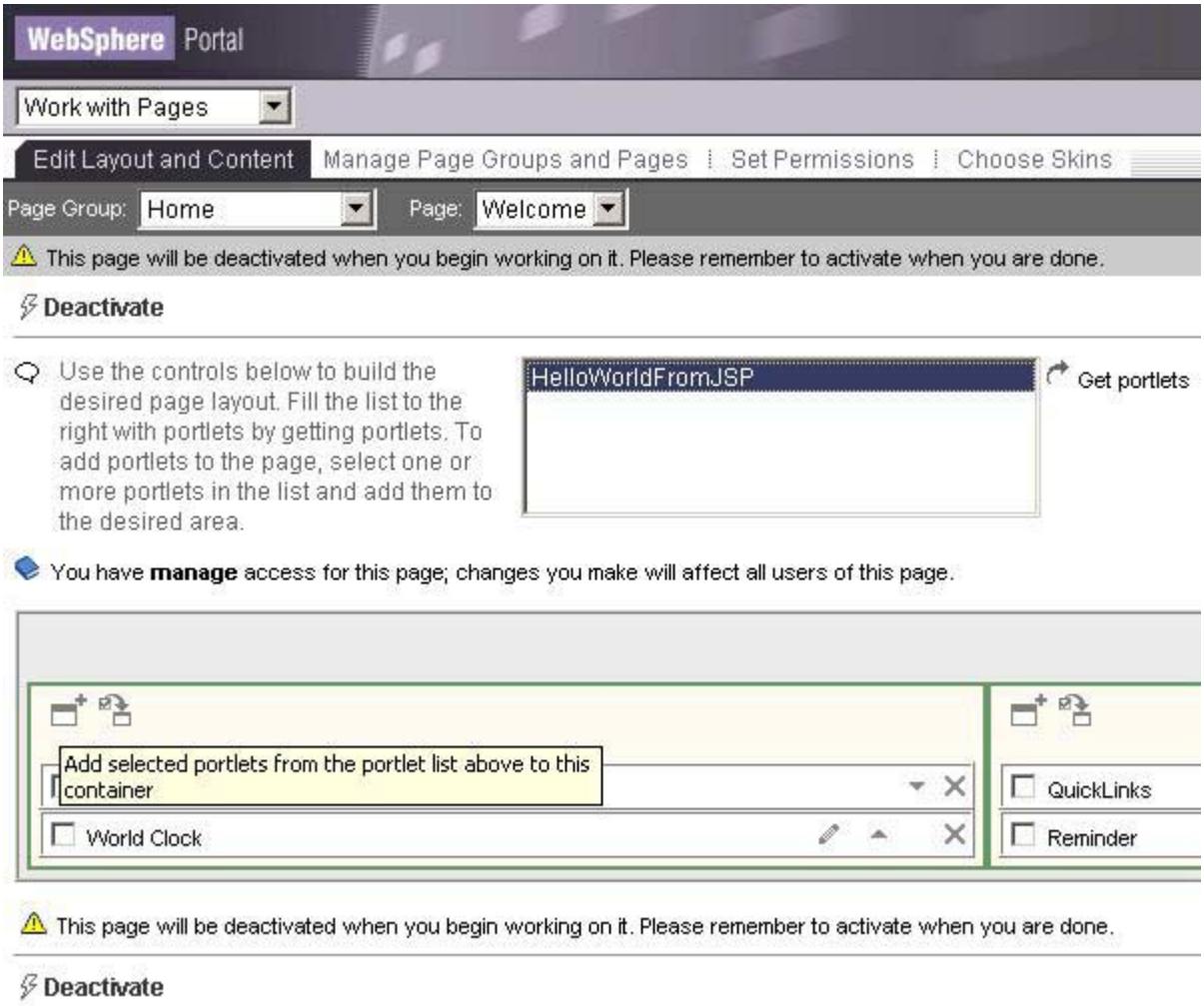
4. Click Get portlets
5. Search on Name Contains: `Hello`, click
6. Click beside `HelloWorldFromJSP` portlet, click

Figure 5. Search for portlets



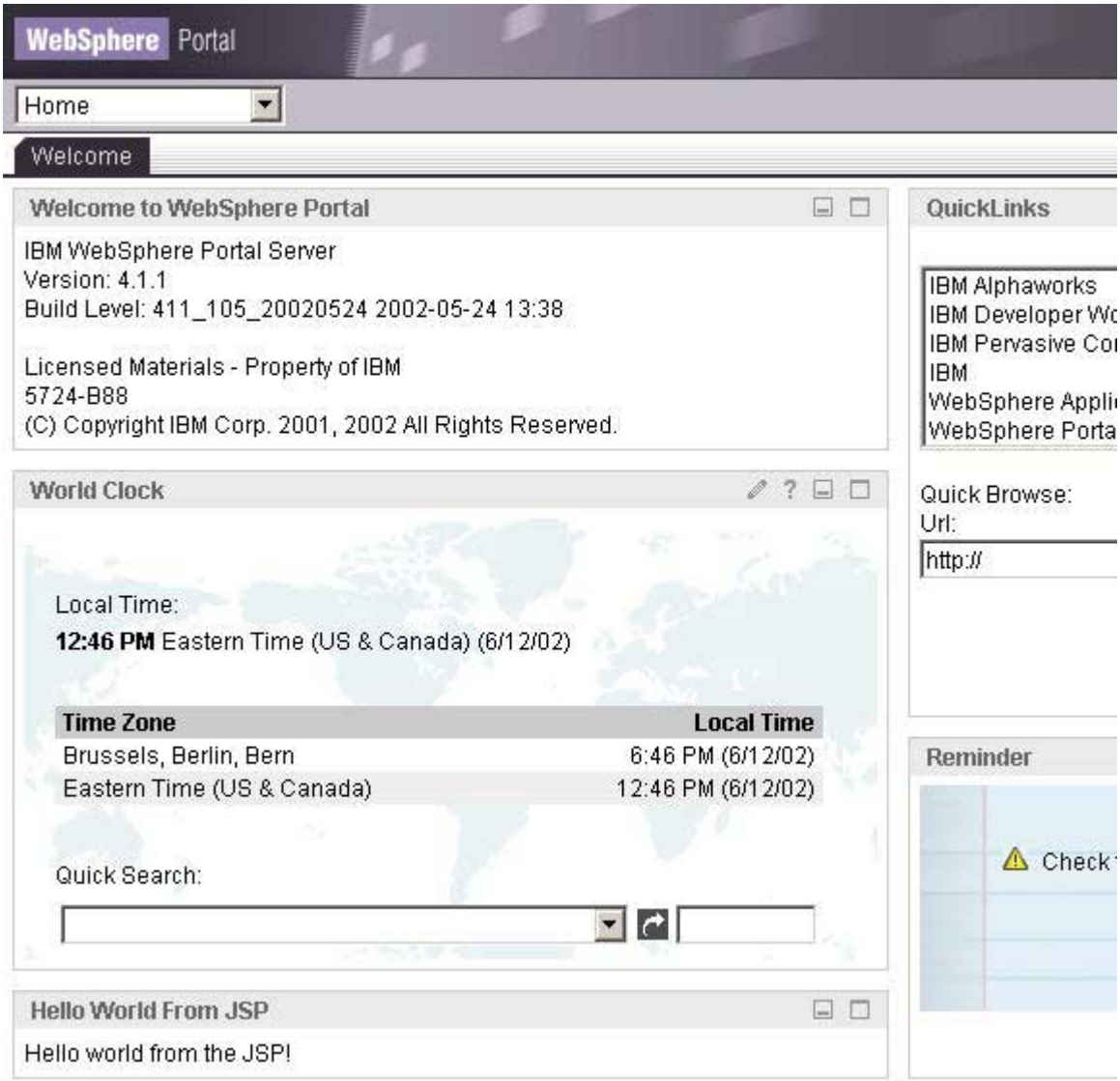
7. Highlight `HelloWorldFromJSP`, click , the add portlet icon

Figure 6. Add portlet to page screen



8. Click Activate to reactivate to page.
9. Select **Home** in drop down to view portlet. We've placed our portlet on the bottom left hand side of the Welcome page.

Figure 7. Portal displaying our portlet



I18N and multiple meta languages

I18N is the common abbreviation for internationalization. Internationalization refers to designing an application, our portlet, in such a way that it can be adapted to different languages and regions without code changes. With our current implementation, we have no way to support multiple national languages or multiple meta languages. Portlets that use JSP for rendering are given two types of support for I18N, one of which also helps with multiple meta languages.

This first support is for multiple JSPs to render our portlet. This allows a portet to have a different layout, colors, images, text, et cetera for each supported language, region and meta language.

The directory structure we chose for our JSP to live in is `\jsp\view.jsp`. The portal will use this directory as a base to search for a best fit JSP. Here's the search order that will be searched for a web browser set to the US english local:

```
\jsp\html\en_US\view.jsp
\jsp\html\en\view.jsp
\jsp\html\view.jsp
\jsp\view.jsp
```

The first `view.jsp` found will be used for rendering the portlet. This allows us to specify different JSPs for different national languages. For example, we could have a `view.jsp` in the `\jsp\html\de\` directory. It would read, "Hallo Welt" (which I hope is "Hello World" in German). The general idea is that each language and region could have it's own JSP providing specialized layout and verbage. What about differnt meta languages? you can see from the search path above that `\html\` is specified. We could have JSPs for our portlet in the following paths:

```
\jsp\html\en_US\view.jsp
\jsp\html\en\view.jsp
\jsp\html\de\view.jsp
```

```
\jsp\html\view.jsp
\jsp\wml\en_US\view.jsp
\jsp\wml\en\view.jsp
\jsp\wml\de\view.jsp
\jsp\wml\view.jsp
\jsp\view.jsp
```

This would allow us to come in from a WML browser and the portlet would use the JSPs in the `\wml\` directory structure for searching.

So, that's the first level of support for I18N. The second level of help is in the form of Java resource bundles. For the purposes of our portlet, Java resouce bundles give us the ability to put strings into properties files that are easily translated apart from the JSP code. This allows one to have the same layout, but have the text in the correct national language. To do this, we need to take a peek at the portlet tag library provided by WebSphere Portal

An introduction to the portlet tag library

The portlet tag library provides JSP tags to access a variety of portlet data, tags for conditional branching in the JSP, and tags for other utility methods. To utilize the portlet tag library the following directive is required at the beginning of the JSP:

```
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
```

This gives us access to the portlet API using the JSP tag prefix `portletAPI`. This prefix is arbitrary, but appropriate for our purposes. Following this definition, you would usually call the following command to initialize a few variables that are frequently used in a portlet's JSP:

```
<portletAPI:init/>
```

The `init` command initializes the following variables:

1. portletRequest
2. portletResponse
3. portletConfig

To get access to a text string in a resource bundle, we use the `text` method of the tag library:

```
<portletAPI:text bundle="nls.labels" key="helloWorldLabel">
  Hello world from the JSP!
</portletAPI:text>
```

This attempts to get the "helloWorldLabel" from the labels properties file. If no suitable properties file is found, then the default is to render what is between the opening and closing text tag, in our case, "Hello world from the JSP!". The properties files are searched the in the same fashion as standard ResourceBundles are searched in Java:

```
<basename>_<lang>_<country>_<variant>
<basename>_<lang>_<country>
<basename>_<lang>
<basename>
```

For our purposes, we will create a couple of properties files:

```
/WEB-INF/classes/nls/labels_en.properties
/WEB-INF/classes/nls/labels_de.properties
```

They will have the following content:

```
/WEB-INF/classes/nls/labels_en.properties:
helloWorldLabel=Hello World!

/WEB-INF/classes/nls/labels_de.properties:
helloWorldLabel=Hallo Welt!
```

To sum up, our JSP file, `/jsp/view.jsp`, has the following content:

```
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>

<portletAPI:init/>
<portletAPI:text bundle="nls.labels" key="helloWorldLabel">
  Hello world from the JSP!
</portletAPI:text>
```

This along with the two properties files give us the following when our language is set to English in our html browser:

Figure 8. Hello World portlet in English

Then when our language is set to German in our html browser, the portlet will produce:

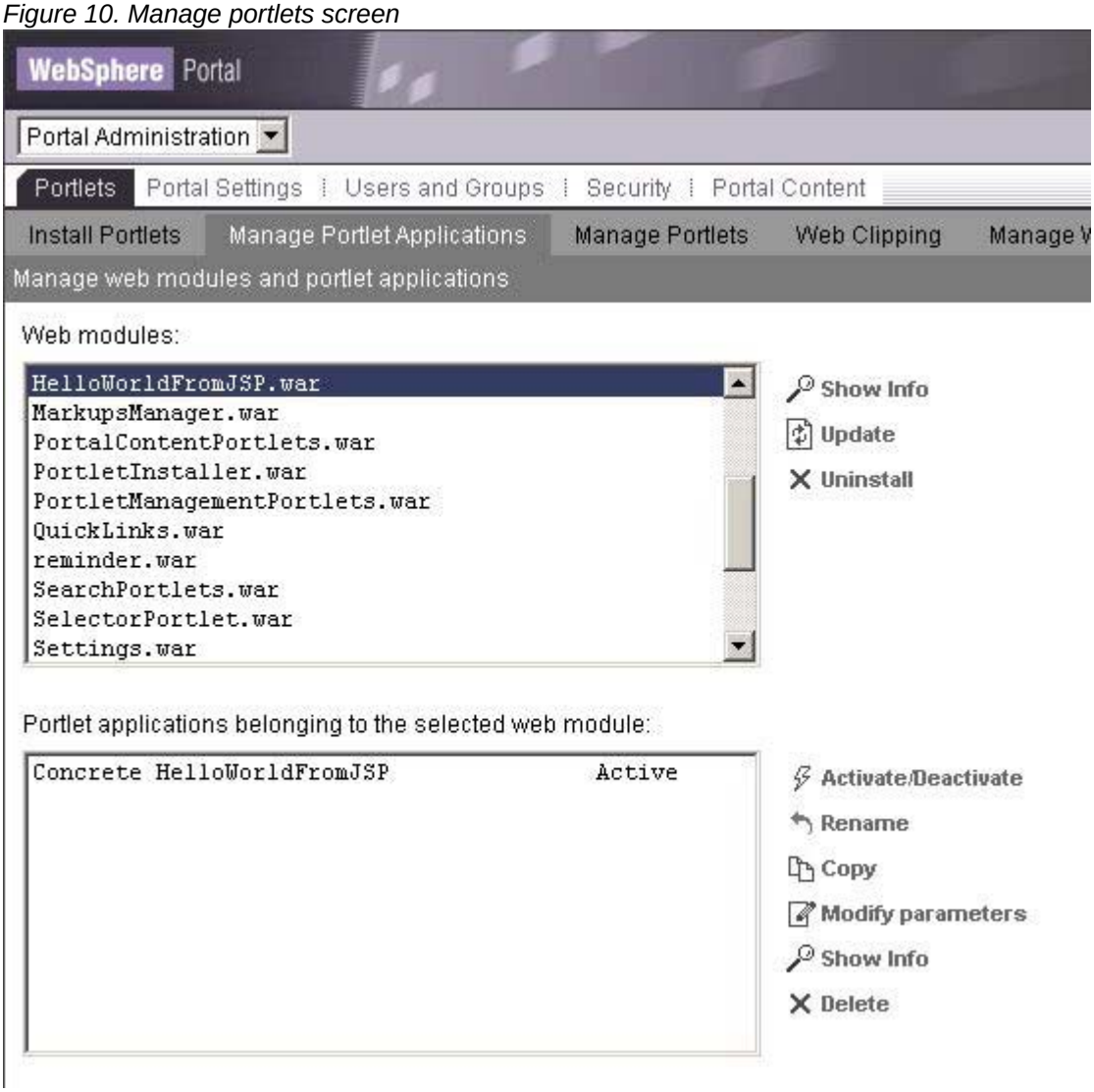
Figure 9. Hello World portlet in German

Pretty nifty. Ah, but you ask how I updated the portlet that we had already deployed into the portal. One could just delete the portlet and redeploy. But in a real portal environment, you'd lose user data and deployment information. The last section of this article will briefly cover how to update a deployed portlet.

Updating a deployed portlet

1. Login to the portal as the portal administrator -- wpsadmin.
2. Select the **Portal Administration** page group.
3. Select the **Portlets** page and the **Manage Portlet Applications** portlet

Figure 10. Manage portlets screen



4. Click the Update button.
5. Click the Browse... button.

6. Find the location of WAR file, select it, and click .
7. Click Next, this could take some time, so wait for it.
8. Verify Portlet is HelloWorldFromJSP, click , again wait for it to complete.
9. Finally you should see confirmation that says,

Conclusion

There it is. That's how you render your portlet using a JSP. If you worked along with this article, you wrote, compiled and packaged Java code for a portlet. You then created the deployment descriptors and packaged the portlet for distribution and deployment. You then deployed the portlet into your portal. Finally you rewrote the JSP, internationalized and updated the deployed portlet. In the course of portlet development you'll do this time and again. Good luck and happy hacking!

References

- Portlet Developer's Guide

Related information

- WebSphere Portal
- WebSphere Developer Domain Portal Zone

Top of page

About the author

Ron Lynn (a.k.a. tcat) is an Advisory software engineer at IBM's Santa Teresa Lab. Ron has worked on both sides of knowledge management. He enjoys playing with his 6 month old son Rowan, making wooden toys for said son, and futzing with Linux on his PlayStation 2. You can contact Ron at tcat@us.ibm.com.

Do you want to discuss this article with others? Go to the our WebSphere Developer Domain newsgroup.

Printing this page: *Microsoft Internet Explorer versions prior to 5.5 may split graphics and tables across printed pages. This is a known problem with Internet Explorer that has been fixed in version 5.5 SP1.*