Jetspeed, Part 1: Developing portlets

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

Section 1. About this tutorial

Who should take this tutorial?

This tutorial teaches you how to develop a portal using Jetspeed. The course is intended for developers and technical managers who want to get an overview and understanding of portal and portlet development using Jetspeed.

About the authors

Vivek Malhotra is a subject matter expert on wireless technologies based in the Washington D.C. area. Vivek has several years of experience developing and implementing wireless applications and has spoken on expert panels focusing on the wireless industry. You can reach him at *vmalhot@yahoo.com* for any questions you might have about the content of this tutorial.

Roman Vichr is senior architect at DDLabs, an e-commerce and EAI consulting company. His latest interests include expanding databases into wireless technology, after focusing on database management for client/server and Web applications development over the past nine years. His background is in fiberoptics, culminating in a Ph.D. in the field from Prague's Institute of Chemical Technology in 1992. You can reach him at *rvichr@ddlabs.net*.

Introduction to the tutorial

Jetspeed, based on an open source implementation, can be used to implement both Web-based and wireless portals. This tutorial discusses Jetspeed and implementation of portlets, which are building blocks of a portal.

Prerequisites

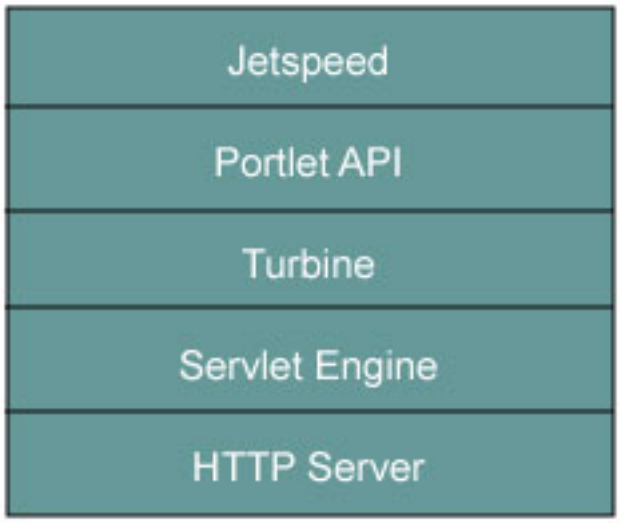
You should be familiar with basic Java programming and Wireless Markup Language (WML).

Section 2. Jetspeed overview and architecture

What is Jetspeed?

Jetspeed is an open source project from the Apache Software Foundation that allows developers to implement a portal. Written in Java language, Jetspeed offers a Portal API for developing portlets, small Java applications that are building blocks of a portal. Jetspeed makes connections to external data and content feeds to retrieve and display the data via a portal. You can implement the portal and access it from a Web browser or a wireless device (such as a WAP phone or Palm device). Jetspeed supports built-in services for user interface customization, caching, persistence, and user authentication, eliminating the need implement these services.

Jetspeed architecture model



This figure illustrates the Jetspeed architecture model. Jetspeed is built on top of Turbine, a servlet-based framework, which is also part of the Jakarta Apache Project. Turbine handles user authentication and page layout as well as scheduling. Jetspeed can run on a number of servlet engines and databases. This tutorial uses Tomcat 3.2 for the servlet engine and Web server.

Content formats supported by Jetspeed

Jetspeed supports the RSS (RDF Site Summary) and OCS (Open Content Syndication) formats. RSS is an XML format used for syndicating Web headlines. The OCS format describes multiple-content channels, including RSS headlines.

High-level Jetspeed features

Some of the high-level features of Jetspeed include:

- Both the RSS and OCS formats are supported

- Support for Wireless Markup Language (WML)
- A Web Application development infrastructure
- Portability across all platforms that support JDK 1.2 and Servlet 2.2
- User, group, role, and permission administration via security portlets
- Role-based security access to portlets

Installing and configuring Jetspeed

Follow the following steps to install and configure Jetspeed and get working:

- Jetspeed requires a servlet engine in order to run. This tutorial uses Tomcat, which you will need to download, install, and configure. You can download Tomcat from the Tomcat home page at *http://jakarta.apache.org/tomcat/*.
- Download, install, and configure Jetspeed. You can download Jetspeed from the Jetspeed home page at *http://jakarta.apache.org/jetspeed/*.
- Build Jetspeed. Go to the directory where Jetspeed is installed. Go to the build directory and execute "build war" at the DOS prompt.
- Copy the Jetspeed WAR file (jetspeed.war) from the \bin directory, which is under the directory where Jetspeed is installed, to the \webapps directory, which is under the directory where Tomcat is installed.
- Start Tomcat.
- Connect to Jetspeed. The URL to Jetspeed (on the local machine) is http://localhost:8080/jetspeed/. Two default login accounts are created: (a) Login: **turbine** and Password: **turbine**; (b) Login: **admin** and Password: **jetspeed**

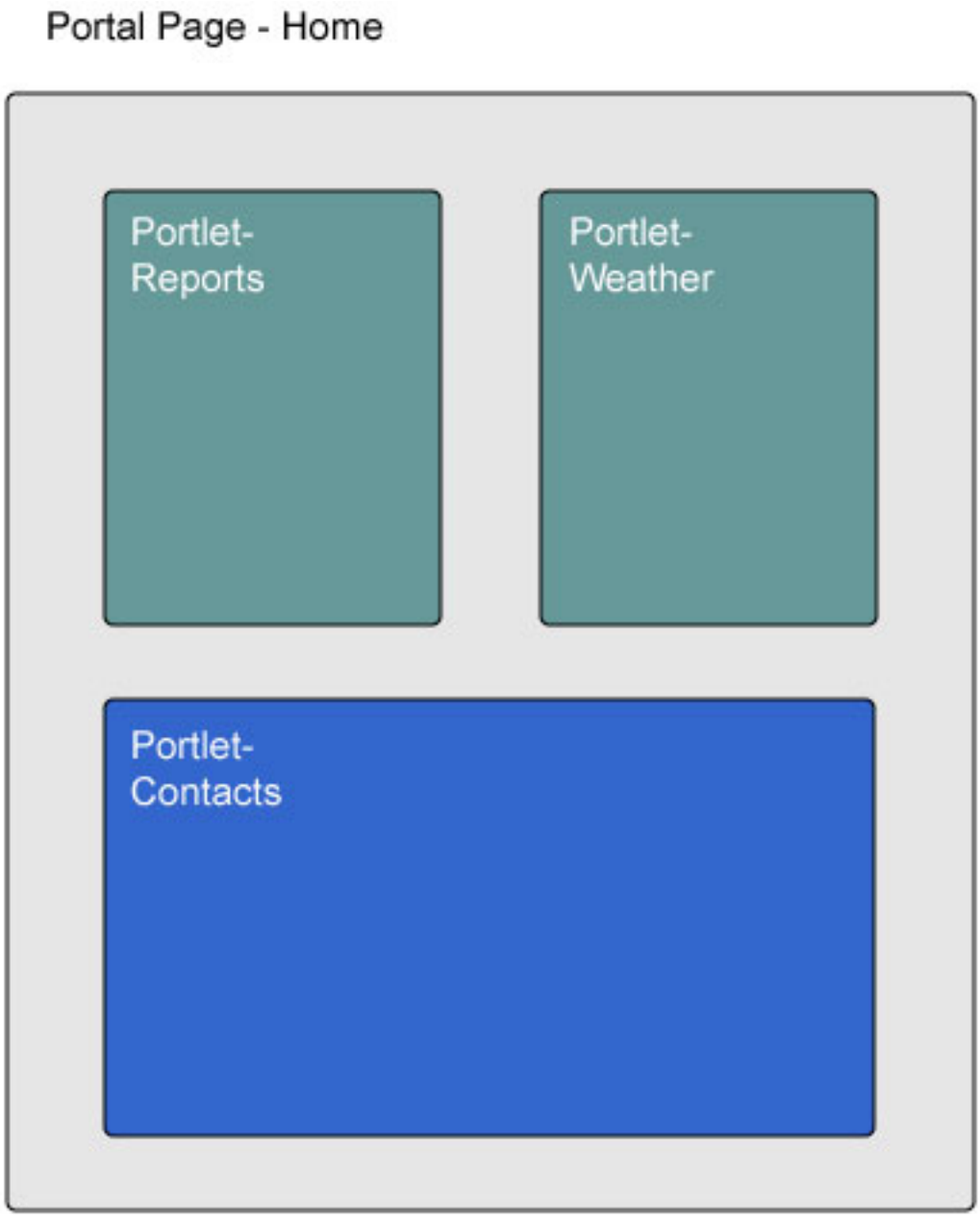
Resources

For additional information, resources, and tool kits refer to the following sites:

- *Jetspeed home page*
- *Turbine home page*
- *Tomcat home page*
- *Openwave SDK*
- *IBM Websphere Portal Server*
- *Java Technology Web site*

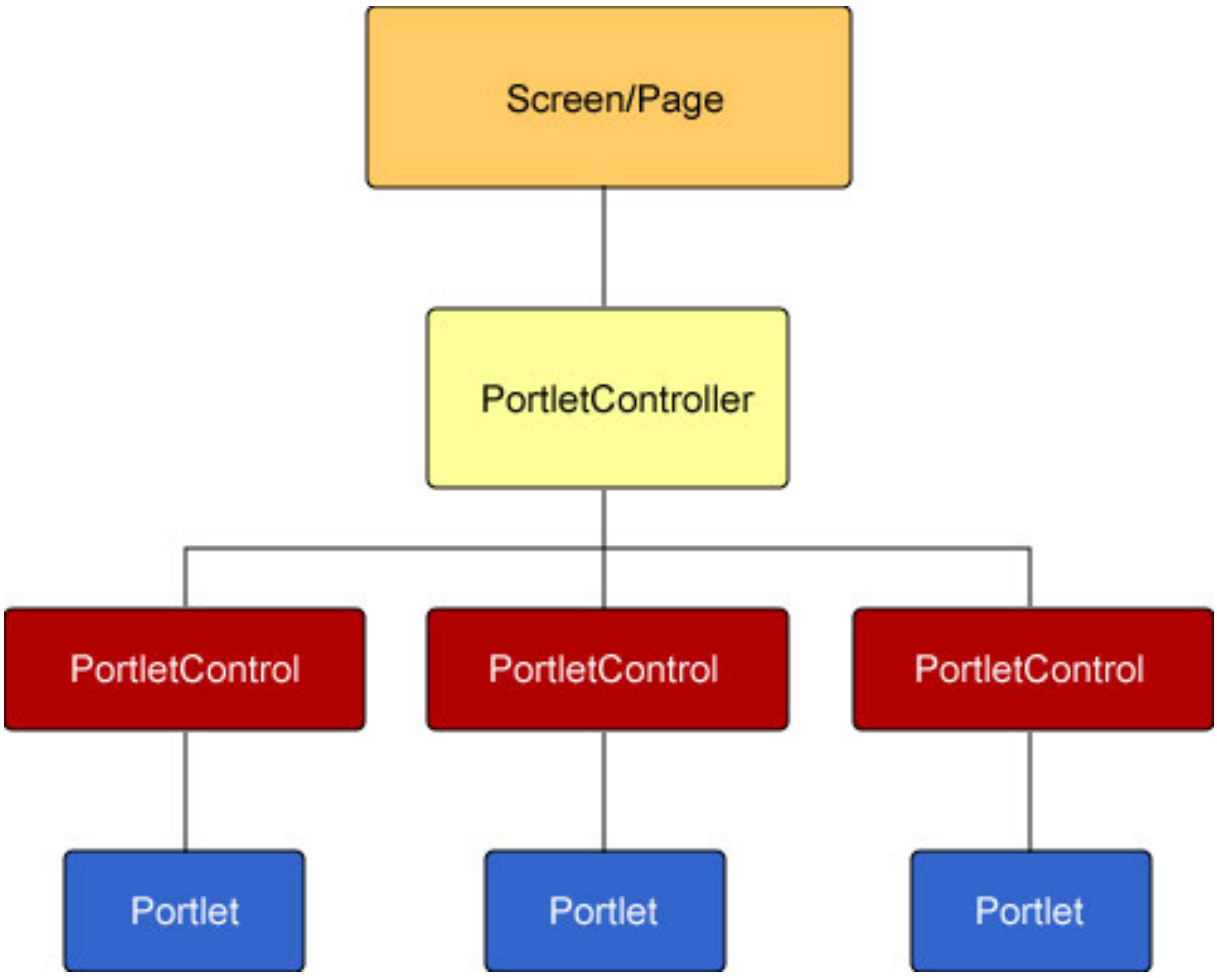
Section 3. Portlets overview

What are portlets?



Think of a portlet as a visibly active component displaying data within a portal page. The figure illustrates a portlet in relation to a portal. Three portlets called reports, weather, and contacts are displayed on a portal page called Home.

Page layout



This figure illustrates the anatomy of a page containing portlets. Within a page, portlet content gets rendered by a PortletControl and a PortletController.

- PortletControl: The function of the PortletControl within the Portlet page handles rendering the title and body of the Portlet.
- PortletController: The function of the PortletController is to handle multiple PortletControls (each controlling a Portlet) to provide an entire Page of information from all the Portlets.

How portlets display content

To display content, portlets use the Element Construction Set (ECS) API. This comes bundled with Jetspeed, which generates markup elements from Java objects. You can use a servlet template or JSP technology to generate content as an output, which is then captured by an ECS element that displays the content. ECS supports WML as well as HTML and XML.

Portlet caching

Jetspeed provides developers with an advanced caching mechanism. Portlets that need to

be cached are placed within the cache with relevant handle information like the classname and some portlet configuration information like the URL, etc. If there is a need to remove the portlet from the cache, it can be done by providing an expire() method, which determines when the portlet needs to expire itself.

Portal Structure Markup Language (PSML)

Portlets are registered manually with Jetspeed using Portal Structured Markup Language (PSML). PSML tells Jetspeed what portlets are available and registered with it. The configuration file for portlets is jetspeed-config.jcfg in the WEB-INF/conf directory. These default configuration files are called default.psml and defaultWML.psml and are in WEB-INF/psml. Once a user is created, each user has configuration files associated with them -- homeHTML.psml and homeWML.psml. These files are stored in WEB-INF/psml/<username> for each user. PSML is composed of two markups -- registry markup and site markup.

Registry markup

Registry markup describes all available portlets to the Jetspeed engine. All information about the portlet is stored within a portlet registry.

Site markup

Site markup describes what portlets available to the Jetspeed engine can be displayed for a given user. Information on how the portlets are organized on a screen/page and its presentation properties are described by the site markup.

Section 4. Standard portlets part of Jetspeed

What are the standard portlets?

Following are some of the standard and more commonly used portlets:

- HTML portlet
- JSP portlet
- RSS portlet
- WebPage portlet
- XSL portlet

Note that a description of the common elements used by the above portlets can be found on the *Jetspeed Web site*.

HTML portlet

The HTML portlet displays HTML content. Here is an example of its use:

```
<portlet-entry name="HelloWorld" hidden="false" type="ref" parent="HTML" application="fa
    <meta-info>
        <title>Hello World</title>
        <description>Example of HTML portlet</description>
    </meta-info>
    <url>hello.html</url>
</portlet-entry>
```

The url tag defines the location of the content to be displayed by the HTML portlet. Only HTML media type is supported by the HTML portlet.

JSP portlet

The JSP portlet displays the output of a JSP page. Here is an example of its use:

```
<portlet-entry name="HelloWorld" hidden="false" type="ref" parent="JSP" application="fal
    <meta-info>
        <title>Hello World</title>
        <description>Example of JSP Portlet</description>
    </meta-info>
    <parameter name="template" value="hello.jsp" hidden="false"/>
    <media-type ref="html"/>
</portlet-entry>
```

The content displayed by the portlet is the output of the hello.jsp page. Note that the JSP file needs to be located in <tomcat_home>/<jetspeed_directory>/WEB-INF/templates/jsp/portlets.

RSS portlet

The RSS portlet renders a RDF Site Summary format feed and presents it to the user as HTML.

```
<portlet-entry name="Apacheweek" hidden="false" type="ref" parent="RSS" application="fa
    <meta-info>
        <title>Apacheweek</title>
    </meta-info>
    <url>http://www.apacheweek.com/issues/apacheweek-headlines.xml</url>
</portlet-entry>
```

The URL tag is the location of the RSS feed, which needs to be an RSS-formatted XML file. WML and HTML media types are supported by the RSS portlet.

Web page portlet

The Web page portlet displays the content of a Web site. Here is an example of its use:

```
<portlet-entry name="JetspeedPage" hidden="false" type="ref" parent="WebPagePortlet" app
   <meta-info>
       <title>JetspeedPage</title>
       <description>Example of WebPage Portlet</description>
   </meta-info>
   <url>http://jakarta.apache.org/jetspeed</url>
</portlet-entry>
```

The URL tag defines the location of the Web page to be displayed by the portlet. HTML media type is supported by the Web page portlet.

XSL portlet

The XSL portlet is used to display an XML-transformed document. XSLT is used to convert the XML document into HTML, which is then displayed by the portlet.

Section 5. Developing portlets -- Hello World! portlet

Steps

You will need to complete the following steps before a portlet can be made usable by Jetspeed:

1. Implement and compile the portlet
2. Create the portlet registry fragment
3. Put the compiled portlet in the appropriate location
4. Register the portlet with the Jetspeed portlet registry

Hello World! portlet

In this section you will create a simple Hello World portlet. The following is the code for the Hello World portlet:

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

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

public class HelloWorldPortlet extends AbstractPortlet
{
    public ConcreteElement getContent (RunData runData)
    {
        return (new StringElement ("Hello World test!"));
    }
}
```

Every portlet that is to be part of the portal has to implement the Portlet interface org.apache.jetspeed.portal.Portlet. Jetspeed provides a number of classes that implement the Portlet interface with the most common functionality. The AbstractPortlet class is the simplest of these predefined classes. The RunData object is passed to the getContent() because it needs to be passed on to many other methods of the portal framework. To display content, portlets use the Element Construction Set (ECS) API.

Portlet location

Once the Java code is compiled, the class has to be placed in the classpath. For this example the classpath is <tomcat_home>/<jetspeed_directory>/WEB-INF/classes/com/bluesunrise/portal/portlets/.

Hello World! Portlet registry fragment

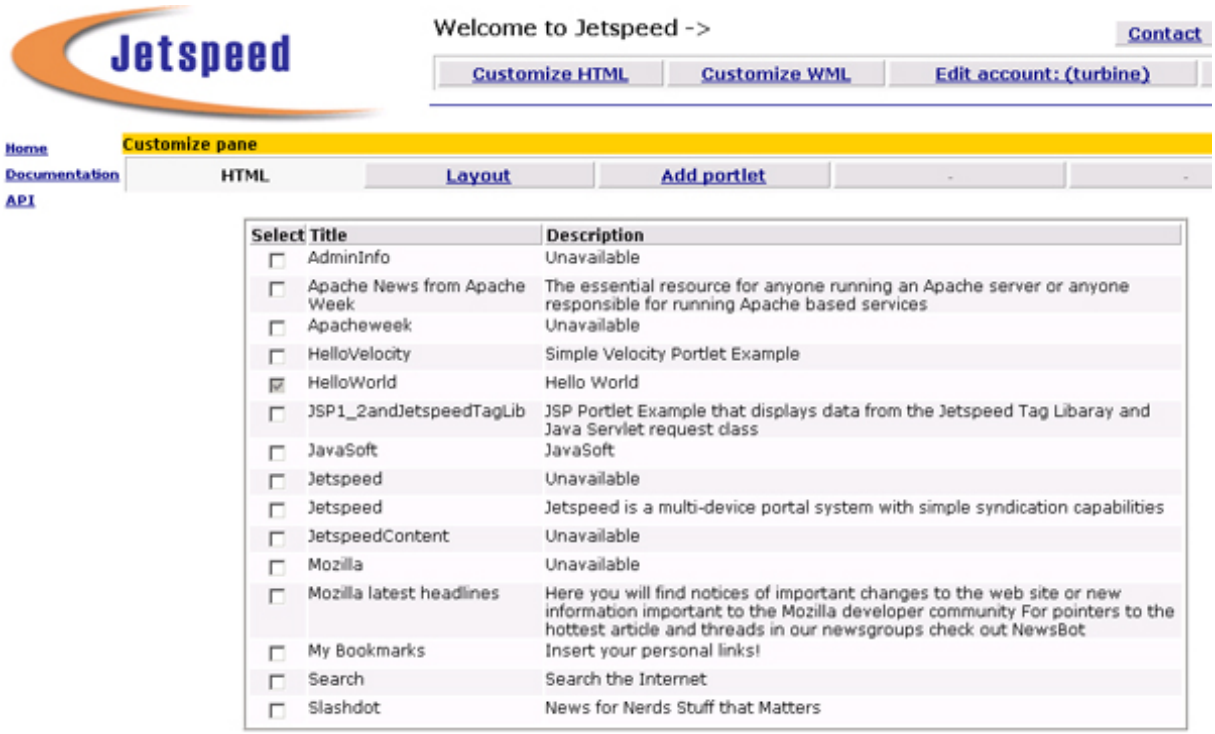
A registry fragment contains the definition of a portlet. Below is the registry fragment for the Hello World! portlet example:

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

The classname tag gives the location of the HelloWorldPortlet class.

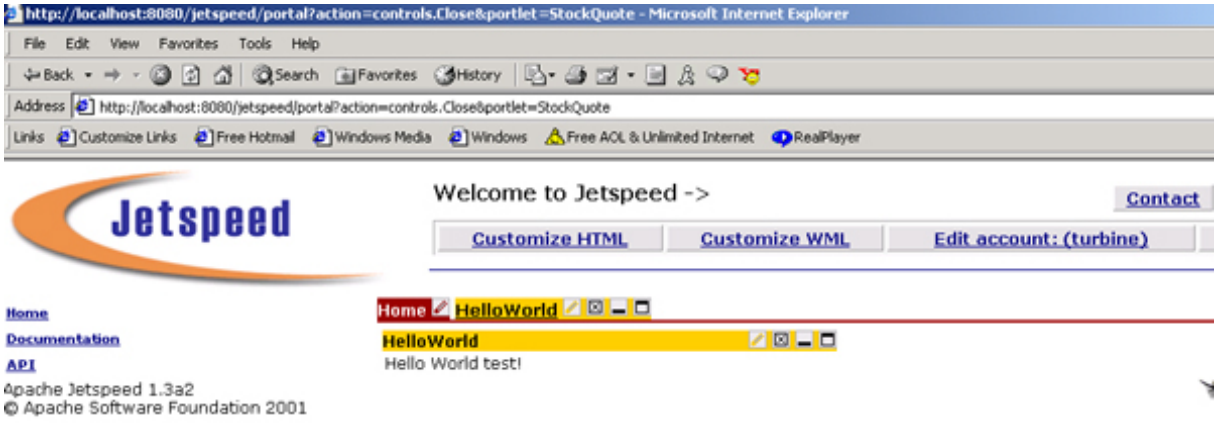
Portlet registry

After creating the registry fragment file, it needs to be deployed to Jetspeed under /WEB-INF/conf. Any file in the /WEB-INF/conf directory that has the .xreg extension will be included in the Jetspeed Registry. The following figure illustrates what the Jetspeed Registry page would look like:



View Hello World! from the portal

Once the portlet has been registered with Jetspeed, you can view the output of the portlet on the portal page. Following is an illustration of the output of the Hello World portlet from within the portal:


http://localhost:8080/jetspeed/portal?action=controls.Close&portlet=StockQuote - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost:8080/jetspeed/portal?action=controls.Close&portlet=StockQuote
Links Customize Links Free Hotmail Windows Media Windows Free AOL & Unlimited Internet RealPlayer
Jetspeed
Welcome to Jetspeed ->
Contact
Customize HTML
Customize WML
Edit account: (turbine)
Home
Documentation
API
Apache Jetspeed 1.3a2
© Apache Software Foundation 2001
Home
HelloWorld
HelloWorld
Hello World test!

Section 6. Jetspeed and wireless

Wireless support using Portlets

One of the features of Jetspeed is support for WAP phones. The WAP phone browser renders WML content. In this section you will create a simple Hello World portlet to be viewed on a WAP browser.

Hello World! portlet

The following is the code for the Hello World portlet:

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

import org.apache.ecs.ConcreteElement;
import org.apache.ecs.ElementContainer;
import org.apache.ecs.wml.Card;
import org.apache.jetspeed.capability.CapabilityMap;
import org.apache.jetspeed.capability.CapabilityMapFactory;
import org.apache.jetspeed.portal.portlets.AbstractPortlet;
import org.apache.jetspeed.util.MimeType;
import org.apache.turbine.util.RunData;

import java.io.*;

public class HelloWireless extends AbstractPortlet {


   public ConcreteElement getContent (RunData runData) {

      //create an ECS container for our content
      ElementContainer content = new ElementContainer();

      //get user's browser info from the Turbine runtime data.
      CapabilityMap capMap = CapabilityMapFactory.getCapabilityMap (runData);

      //WML code for wireless

          Card wmlCard = new Card();
          wmlCard.setCardId("_" + getPortletConfig().getName());
          wmlCard.addElement(new org.apache.ecs.wml.P().addElement("Hello Wireless"));
          content.addElement(wmlCard);
          return content;
   }
}
```

Again, like in the previous section, every portlet that is to be part of the portal has to implement the Portlet interface org.apache.jetspeed.portal.Portlet. The Element Construction Set (ECS) API displays the WML content to the WAP browser.

Portlet Location

Once the Java code is compiled, the class needs to be placed in the classpath. For this example the classpath is <tomcat_home>/<jetspeed_directory>/WEB-INF/classes/com/bluesunrise/portal/portlets/.

Hello World! portlet registry fragment

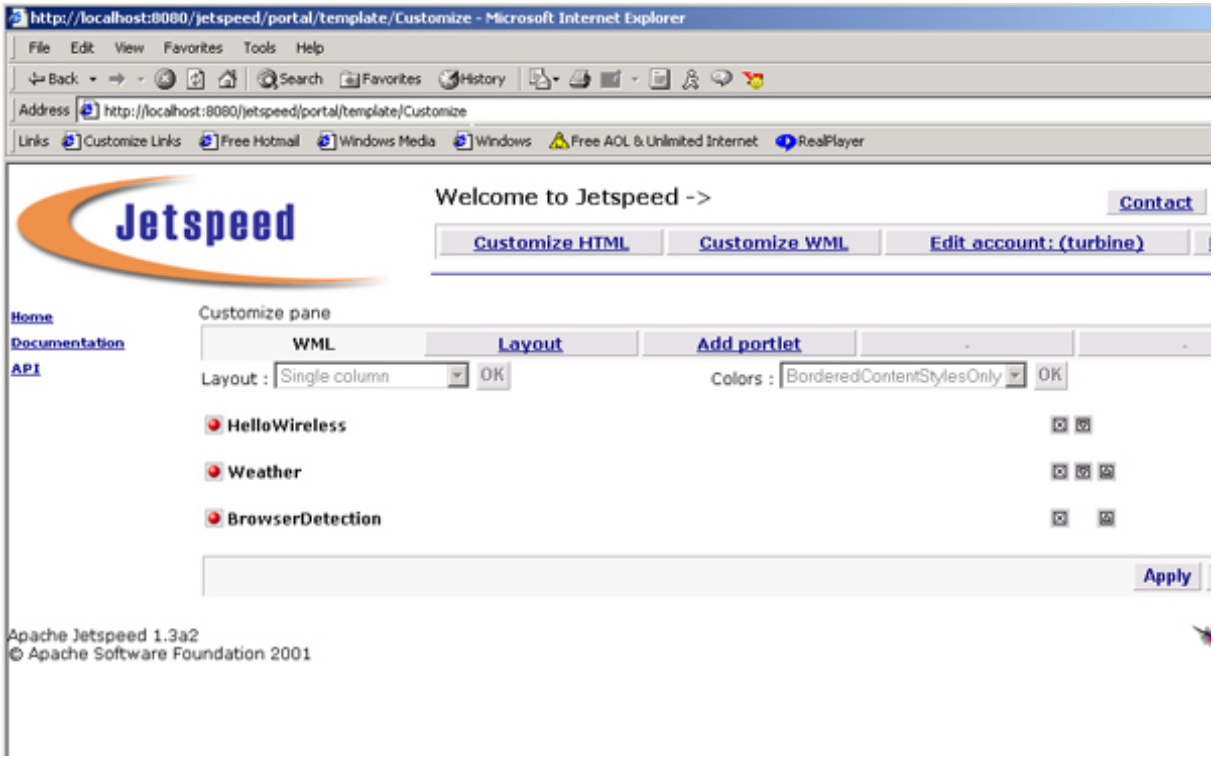
Following is the registry fragment for the wireless Hello World! portlet example:

```
<?xml version="1.0" encoding="UTF-8"?>
<registry>
    <portlet-entry name="HelloWireless" hidden="false" type="instance" application="fals
        <meta-info>
            <title>HelloWireless</title>
            <description>HelloWireless</description>
        </meta-info>
        <classname>com.bluesunrise.portal.portlets.HelloWireless</classname>
        <media-type ref="wml"/>
    </portlet-entry>
</registry>
```

The classname tag gives the location of the HelloWorldPortlet class. Note, that the media type is "wml".

Portlet registry

The "Customize WML" section contains all WML-type registered portlets with Jetspeed. The figure below illustrates what the Jetspeed Registry page would look like containing the registered Hello World portlet:



Viewing Hello World! from WAP phone

Following is an illustration of the output of the Hello World portlet from within the wireless portal:

The first figure is the portal view and the second figure illustrates the output of the Hello World! portlet.

Section 7. Portlet example

Example overview

In this portlet example, we will demonstrate implementing different portlets that display the results on a WAP-enabled phone. This example consists of three portlets. The first one, the "Hello World!" portlet, was discussed in the previous section. The second portlet implementation illustrates content being retrieved from a .wml file. The third portlet implementation illustrates the rendering of content based on the browser making the request, for example HTML or WML. The third example takes advantage of the browser characteristics. The application detects the requests and, based on that, delivers content that the particular browser can render.

Creating the Hello World portlet

Copy and save the following code as HelloWireless.java and compile it.

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

import org.apache.ecs.ConcreteElement;
import org.apache.ecs.ElementContainer;
import org.apache.ecs.wml.Card;
import org.apache.jetspeed.capability.CapabilityMap;
import org.apache.jetspeed.capability.CapabilityMapFactory;
import org.apache.jetspeed.portal.portlets.AbstractPortlet;
import org.apache.jetspeed.util.MimeType;
import org.apache.turbine.util.RunData;

import java.io.*;

public class HelloWireless extends AbstractPortlet {


   public ConcreteElement getContent (RunData runData) {

      //create an ECS container for our content
      ElementContainer content = new ElementContainer();

      //get user's browser info from the Turbine runtime data.
      CapabilityMap capMap = CapabilityMapFactory.getCapabilityMap (runData);

      //WML code for wireless

          Card wmlCard = new Card();
          wmlCard.setCardId("_" + getPortletConfig().getName());
          wmlCard.addElement(new org.apache.ecs.wml.P().addElement("Hello Wireless"));
          content.addElement(wmlCard);
          return content;
   }
}
```

Once the class file is created, the class needs to be placed in the classpath. For this example the classpath is <tomcat_home>/<jetspeed_directory>/WEB-INF/classes/com/bluesunrise/portal/portlets/.

Hello World! portlet registry fragment

Copy the following code as hellowireless.xreg and save it to the <tomcat_home>/<jetspeed_directory>/WEB-INF/conf directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<registry>
    <portlet-entry name="HelloWireless" hidden="false" type="instance" application="fals
        <meta-info>
            <title>HelloWireless</title>
            <description>HelloWireless</description>
        </meta-info>
        <classname>com.bluesunrise.portal.portlets.HelloWireless</classname>
        <media-type ref="wml"/>
    </portlet-entry>
</registry>
```

WML document

In this example a WML portlet renders the content of a WML document. Copy and save the following code as weather.wml:

```
<card id="_MyWeather">
  <p>
    <a href="#dc">Washington DC</a><br/>
    <a href="#chicago">Chicago</a><br/>
    <a href="#austin">Austin</a>
  </p>
</card>
<card id="dc">
<p>
  Washington DC<br/>
  82 F<br/>
  Sunny<br/>
</p>
</card>
<card id="chicago">
<p>
  Chicago<br/>
  70 F<br/>
  Rainy<br/>
</p>
</card>
<card id="austin">
<p>
  Austin<br/>
  100 F<br/>
  Cloudy<br/>
</p>
</card>
```

Portlet registry fragment

Copy the following code as weather.xreg and save it to the <tomcat_home>/<jetspeed_directory>/WEB-INF/conf directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<registry>
    <portlet-entry name="Weather" hidden="false" type="ref"
        parent="WML" application="false">
        <meta-info>
            <title>Weather</title>
        </meta-info>
        <url>weather.wml</url>
    </portlet-entry>
    <portlet-controller-entry name="WAPPortletController" hidden="false">
        <classname>org.apache.jetspeed.portal.controllers.WAPPortletController</classnar
        <media-type ref="wml"/>
    </portlet-controller-entry>
</registry>
```

Note, the parent is of type WML, indicating that a WML portlet will be rendering the content of the specified WML file within the url tags. Make sure that the location of the wml file location within the url tags is correct.

Creating the browser detection portlet

The following code detects the mime type of the client browser. Based on the mime type, the browser of the appropriate content gets rendered. In the following example, either HTML or WML content is rendered. Copy and save the following code as WMLPortlet.java and compile it.

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

import org.apache.ecs.*;
import org.apache.ecs.html.*;

import org.apache.ecs.wml.Card;
import org.apache.jetspeed.capability.CapabilityMap;
import org.apache.jetspeed.capability.CapabilityMapFactory;
import org.apache.jetspeed.portal.portlets.AbstractPortlet;
import org.apache.jetspeed.util.MimeType;
import org.apache.turbine.util.RunData;

import java.io.*;

public class WMLPortlet extends AbstractPortlet {

   //detect the MIME type
   public boolean supportsType (MimeType browserMimeType) {

      if (MimeType.HTML.equals(browserMimeType)) {
         return true;
      }
      if (MimeType.WML.equals(browserMimeType)) {
         return true;
      }
      return false;
   }

   public ConcreteElement getContent (RunData runData) {

      //create an ECS container for our content
      ElementContainer content = new ElementContainer();
```

```
      //get user's browser info from the Turbine runtime data.
      CapabilityMap capMap = CapabilityMapFactory.getCapabilityMap (runData);

      //show HTML code for the web
      if (capMap.getPreferredType().equals (MimeType.HTML)) {

         return (new StringElement ("Hello World from HTML Browser!"));
      }
      //show WML code for wireless
      else if (capMap.getPreferredType().equals (MimeType.WML)) {

             Card wmlCard = new Card();
             wmlCard.setCardId("_" + getPortletConfig().getName()); // To match Jetspeed
             wmlCard.addElement(new org.apache.ecs.wml.P().addElement("Hello World from
             content.addElement(wmlCard);
            return content;
      }

      return content;
   }
}
```

Once the class file is created, the class needs to be placed in the classpath. For this example the classpath is <tomcat_home>/<jetspeed_directory>/WEB-INF/classes/com/bluesunrise/portal/portlets/.

Browser Detection Portlet Registry Fragment

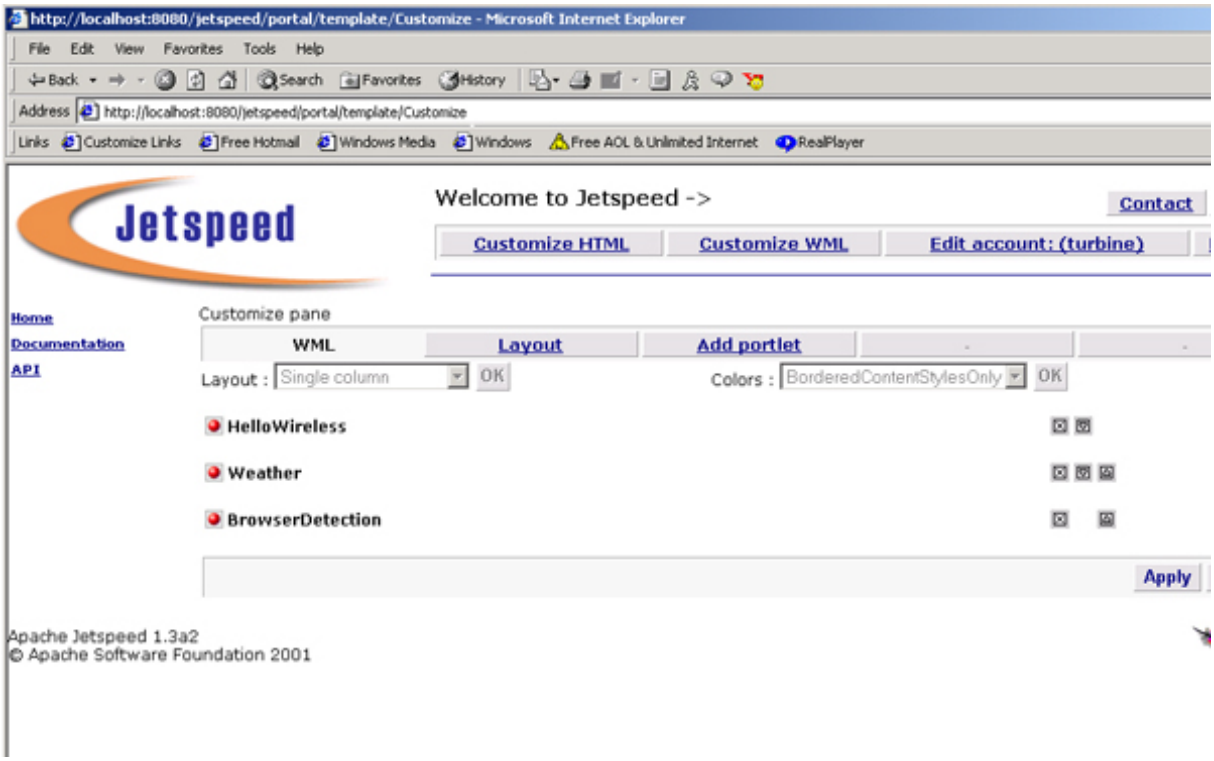
Copy the following code as browserdetection.xreg and save it to the <tomcat_home>/<jetspeed_directory>/WEB-INF/conf directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<registry>
    <portlet-entry name="BrowserDetection" hidden="false"
        type="instance" application="false">
        <meta-info>
            <title>BrowserDetection</title>
            <description>Browser Detection</description>
        </meta-info>
        <classname>com.bluesunrise.portal.portlets.WMLPortlet</classname>
        <media-type ref="html"/>
        <media-type ref="wml"/>
    </portlet-entry>
</registry>
```

Note, that the HTML and WML media types are specified.

Portlets registry

Once all the portlet registry files are created, stop and start Tomcat and connect to the Jetspeed portal. Click on the "Customize WML" link. This section should contain the three portlets created. Select the portlets and click on the add button. The figure below illustrates what the Jetspeed Registry page would look like containing the registered portlets:



View example from a WAP phone

Following is an illustration of the output of the portlets from within the wireless portal:


Go
http://localhost:8080/jetspeed/portal
OPENWAVE
Welcome to Jetspeed
Profile for Vivek
[HelloWireless]
[Weather]
[BrowserDetection]
Link
SEND
END


Go
http://localhost:8080/jetspeed/portal
OPENWAVE
Hello Wireless
Back
SEND
END


Go
http://localhost:8080/jetspeed/portal
OPENWAVE
[Washington DC]
[Chicago]
[Austin]
Link
SEND
END

The first figure is the portal view and the second, third, and fourth figures illustrate the output of each link.

Section 8. Feedback

Feedback

Please send us your feedback on this tutorial. We look forward to hearing from you!

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial Building tutorials with the Toot-O-Matic demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.