WebSphere Application Server Version 6.1

Sales and Technical Enablement Workshop

Lab 03 – Portlet applications

Introduction

The *Application Server Toolkit (AST)* provides basic support for the creation of new applications targeting WebSphere Application Server V6.1. This includes wizards and tools for creating new Web applications, Web services, portlets, and EJBs, as well as annotation based programming support, new administration tools for the creation and maintenance of wsadmin Jython files, and tools to edit WebSphere-specific bindings and extensions.

Lab Requirements

This lab assumes that the following setup is complete prior to starting the lab:

- VMware Player 1.0.x or VMware Workstation v5.5.x installed on your machine. A free VMware player is available from http://www.vmware.com/products/player/
- A machine with 2 GB is RAM is preferred.

Overview

In this exercise you will explore the Application Server Toolkit by developing a simple portlet application.

Portlets are reusable Web modules that provide access to Web-based content, applications, and other resources. Portlets can run on WebSphere Application Server because it has an embedded JSR168 Portlet Container. You can assemble portlets into a larger portal page, with multiple instances of the same portlet displaying different data for each user.

From a user's perspective, a portlet is a window on a portal site that provides a specific service or information, for example, a calendar or news feed. From an application development perspective, portlets are pluggable Web modules that are designed to run inside a portlet container of any portal framework. You can either create your own portlets or select portlets from a catalog of third-party portlets.

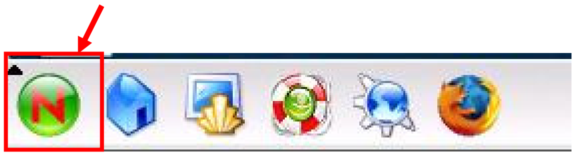
Each portlet on the page is responsible for providing its output in the form of markup fragments to be integrated into the portal page. The portal is responsible for providing the markup surrounding each portlet. In HTML, for example, the portal can provide markup that gives each portlet a title bar with minimize, maximize, help, and edit icons.

You can also include portlets as fragments into servlets or JavaServer Pages files. This provides better communication between portlets and the J2EE Web technologies provided by the application server.

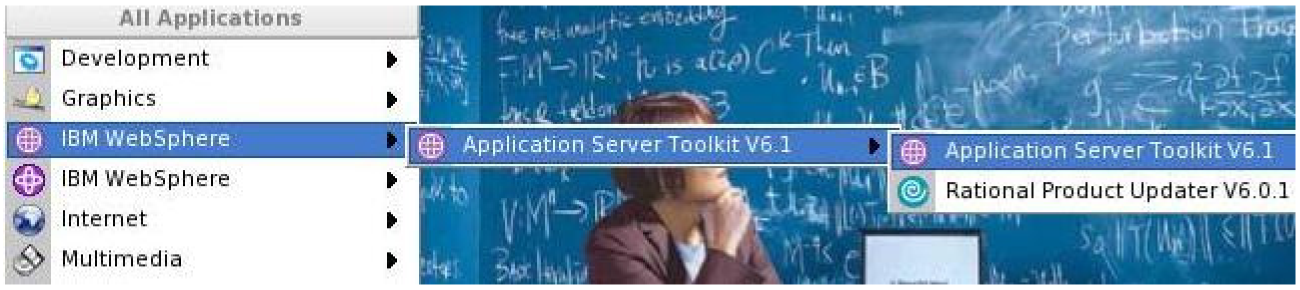
Part 1: Start the Application Server Toolkit

As an introduction to the Application Server Toolkit, start the Workbench and begin developing portlets.

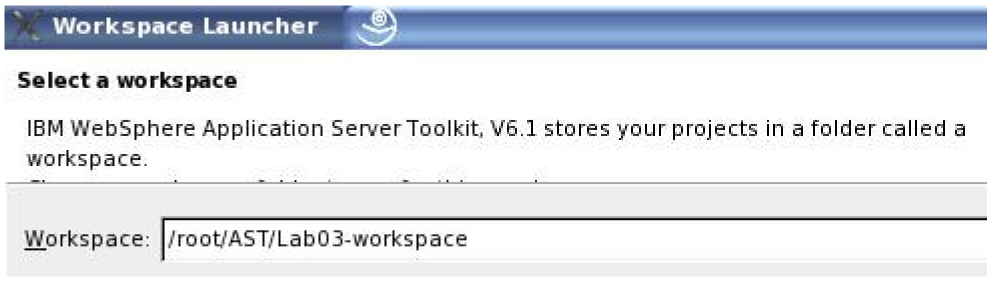
____ 1. From the SLES Desktop, locate the KDE Panel at the bottom of the workspace. Click on the '**N**' icon.



____ 2. In the menu, select **IBM WebSphere→ Application Server Toolkit V6.1 → Application Server Toolkit V6.1.** This will start the AST**.** Alternatively, you can start the AST from the command line using **`/opt/IBM/AST61/ast`**


All Applications
Development
Graphics
IBM WebSphere
IBM WebSphere
Internet
Multimedia
Application Server Toolkit V6.1
Application Server Toolkit V6.1
Rational Product Updater V6.0.1

____ 3. When the Workbench is launched the first thing you see is a dialog that allows you to select where the workspace should be located. The workspace is the directory where your work will be stored. Enter a workspace of **`/root/AST/Lab03-workspace`** and click **OK** to continue**.**

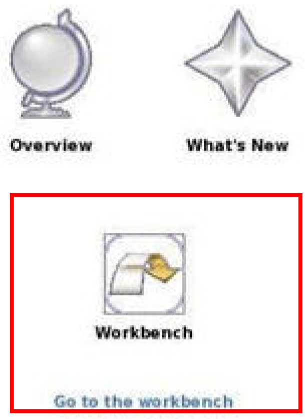


Select a workspace

IBM WebSphere Application Server Toolkit, V6.1 stores your projects in a folder called a workspace.

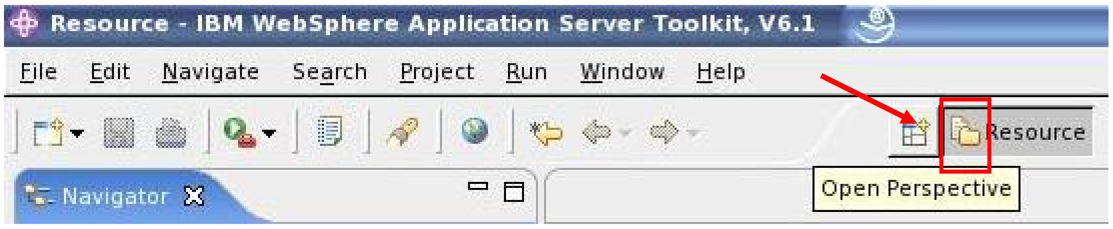
Workspace: /root/AST/Lab03-workspace

____ 4. From the Application Server Toolkit Welcome page, click on **Workbench – Go to the workbench**

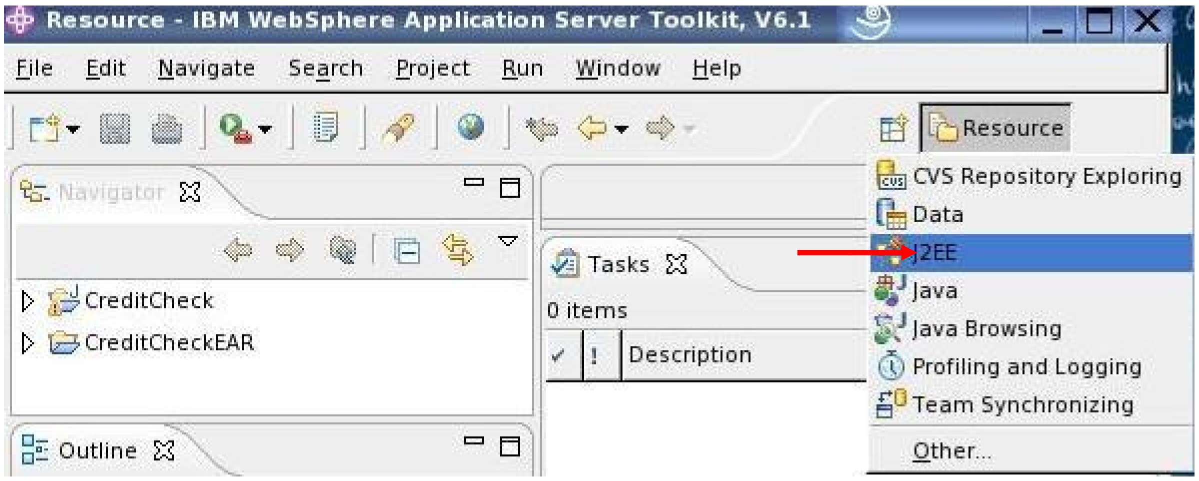


____ 5. Initially, in the first Workbench window that is opened, the Resource perspective is displayed. A shortcut bar appears in the top right corner of the window that allows the user to open new perspectives and switch between ones already open. Open a Data perspective.

1) From the toolbar, click on the **Open Perspective** button


Resource - IBM WebSphere Application Server Toolkit, V6.1
File Edit Navigate Search Project Run Window Help
Resource
Navigator
Open Perspective

2) Select **J2EE**

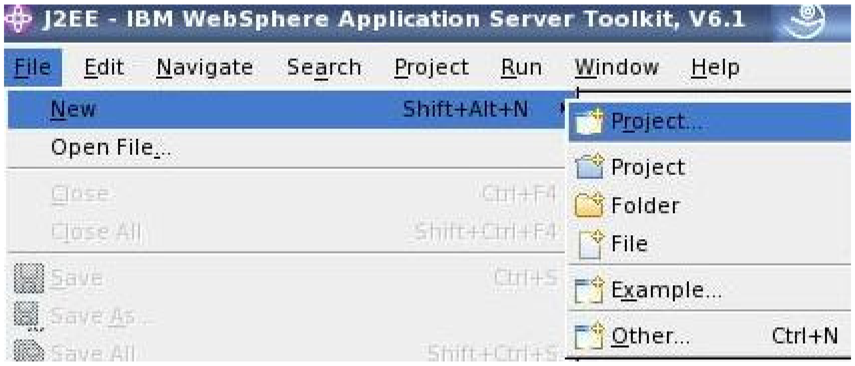


Part 2: Create a portlet project and application

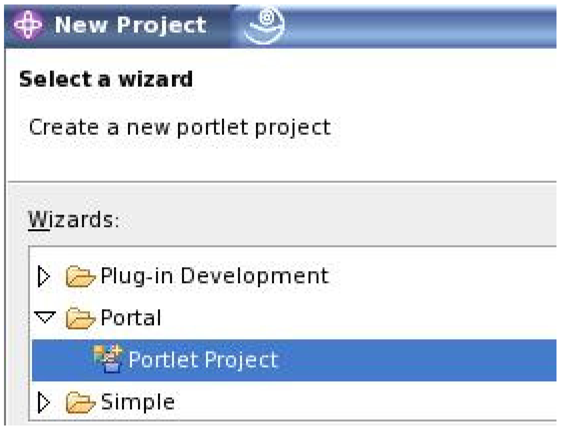
Create portlet projects as a foundation for developing portlet applications in the product workbench.

____ 1. Create a new portlet application using the New Portlet Project wizard

__ a. From the AST menubar, select **File → New → Project**.

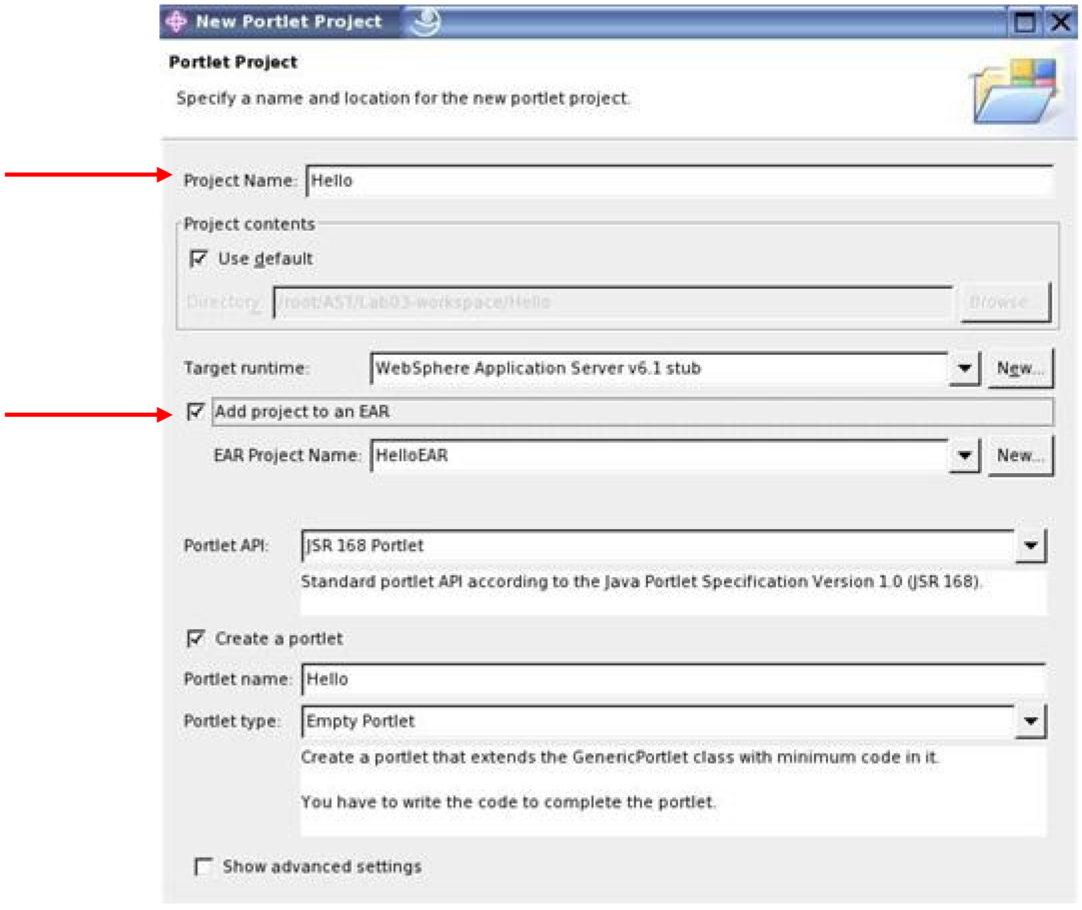


__ b. In the Select a Wizard panel, locate and expand Portal. Select **Portlet Project.** Click **Next**



__ c. In the **Portlet Project** window --

1) Enter a Project Name of **Hello**
2) Select **Add project to an EAR**
3) Click **Next**



New Portlet Project

Portlet Project

Specify a name and location for the new portlet project.

Project Name: Hello

Project contents

[✓] Use default

Directory: /root/AST/Lab03-workspace/Hello

Target runtime: WebSphere Application Server v6.1 stub

[✓] Add project to an EAR

EAR Project Name: HelloEAR

Portlet API: JSR 168 Portlet

Standard portlet API according to the Java Portlet Specification Version 1.0 (JSR 168).

[✓] Create a portlet

Portlet name: Hello

Portlet type: Empty Portlet

Create a portlet that extends the GenericPortlet class with minimum code in it.

You have to write the code to complete the portlet.

[ ] Show advanced settings

By default, portlets that you develop in this tool will follow the JSR 168 (Standard) portlet API according to the Java™ Portlet Specification Version 1.0 (JSR 168).

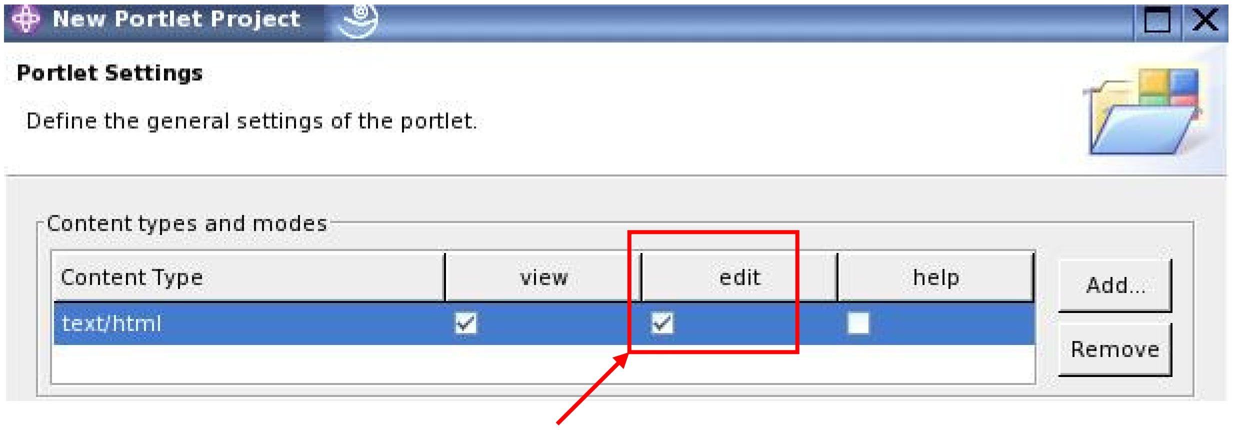
You will create an initial portlet named **Hello**. An Empty Portlet creation type will perform the following tasks:

1) Generate a skeleton class extending GenericPortlet
2) Add the necessary elements to the portlet.xml for the new portlet definition
3) Generate default resource bundles

__ d. In the Portlet Settings window --

1) In the **Content types and modes** section, select **edit**

2) Click **Finish** and wait for the workspace to build.



Portlet Settings

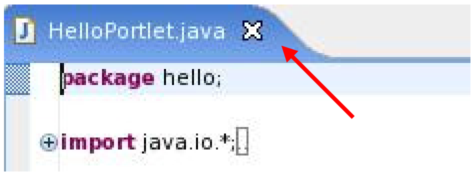
Define the general settings of the portlet.

Content types and modes

Content Type	view	edit	help
text/html	✓	✓	

____ 2. Examine the Hello Dynamic Web Project

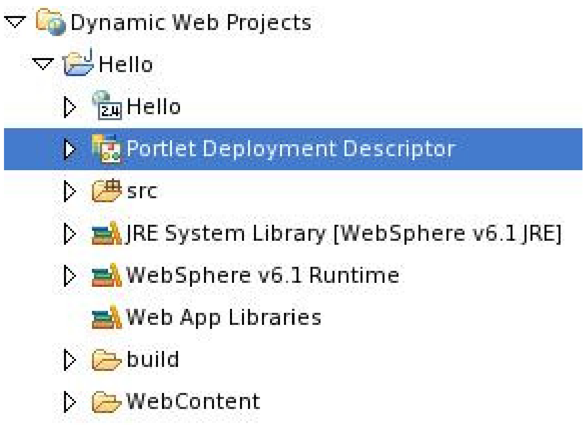
__ a. Close the **HelloPortlet.java** editor. You will return to this file shortly.



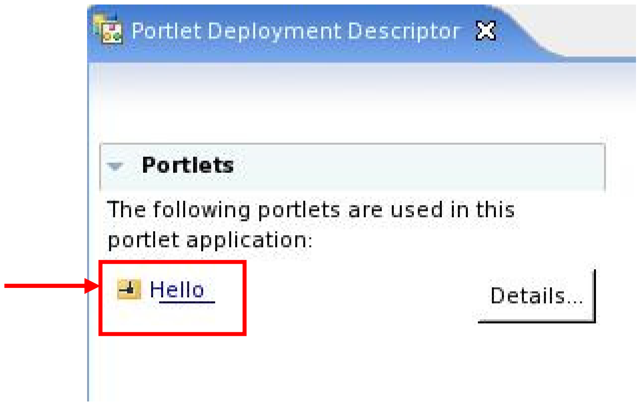
```
package hello;

import java.io.*;
```

__ b. From the **Project Explorer** view, expand the **Hello** web project

__ c. Select the **Portlet Deployment Descriptor** and **double-click**

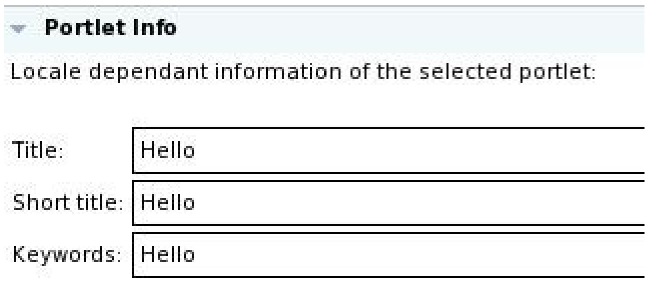


__ d. From the **Overview** section of the Portlet Deployment Descriptor, you can see the new portlet wizard created a Hello portlet and automatically added it to the Portlet Deployment Descriptor. Click on the **Hello** portlet link to display additional information about the Hello portlet.



The following portlets are used in this portlet application:

__ e. From the details section, you can configure additional properties of the portlet. For example, you could change the Title from the Portlet Info section.



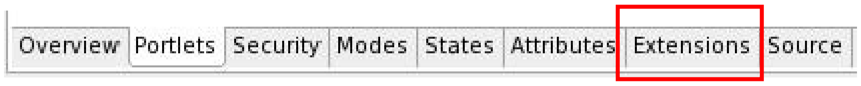
Locale dependant information of the selected portlet:

Title: Hello

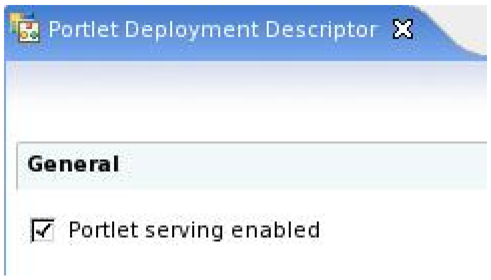
Short title: Hello

Keywords: Hello

__ f. Locate the tabs at the bottom of the Portlet Deployment Descriptor and click on **Extensions**

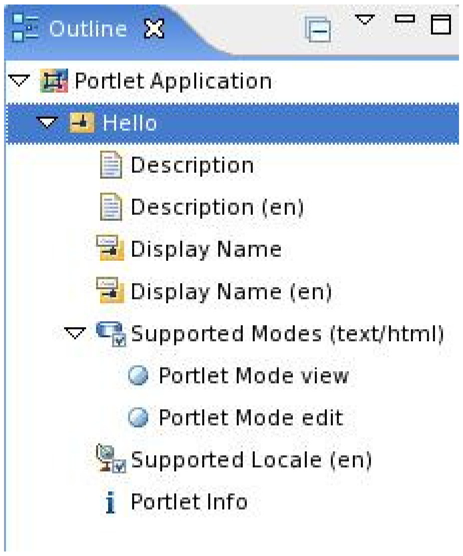


__ g. With **Portlet serving enabled** you can request a portlet directly through a Uniform Resource Locator (URL) to display its content without portal aggregation. The PortletServingServlet servlet registers each Web application that contains portlets. It is similar to the FileServingServlet servlet of the Web container that serves resources. The PortletServingServlet servlet allows you to directly render a portlet into a full browser page by a URL request.



☑ Portlet serving enabled

__ h. Locate the **Outline** view to the right of the graphical Portlet Deployment Descriptor. The Outline view allows you to quickly see the entire contents of the Portlet Deployment Descriptor. **Expand Portlet Application**. Clicking on entries in the Outline view will adjust to Portlet Deployment Descriptor view accordingly. For example, click on **Supported Locale (en)** and the Portlet Deployment Desciptor view will adjust to display this section.

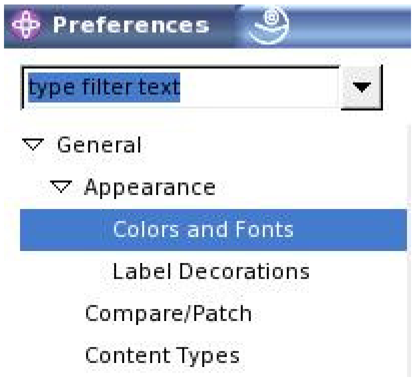


__ i. Close the **Portlet Deployment Descriptor** view.

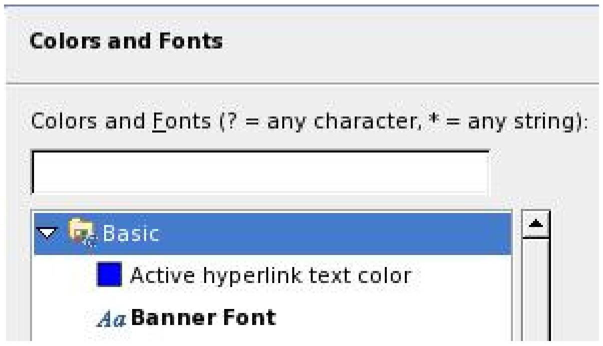
____ 3. Before working with the **Hello portlet** source code, change the default text font size

__ a. From the AST menubar, select **Window > Preferences**

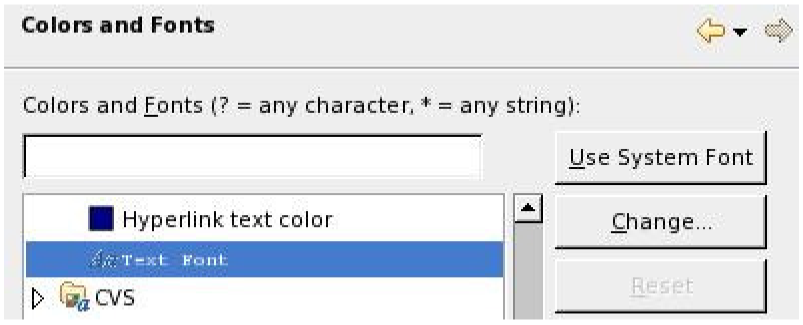
__ b. Expand **General → Appearance**. Select **Colors and Fonts**



__ c. In the Colors and Fonts workspace, expand **Basic**


Colors and Fonts
Colors and Fonts (? = any character, * = any string):
Basic
Active hyperlink text color
Banner Font

__ d. Locate and select **Text Font**



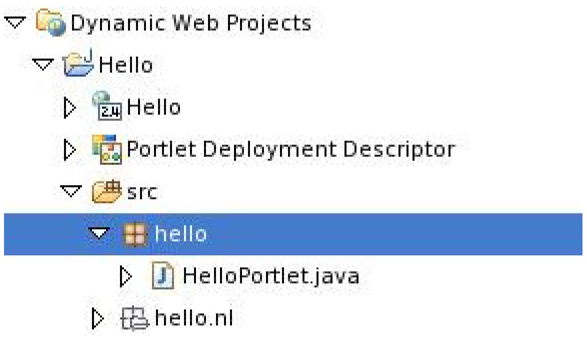
__ e. Click **Use System Font** or **Change…**

__ f. Click **Apply**

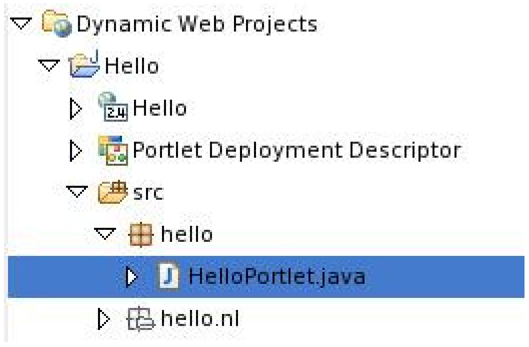
__ g. Click **OK** to close the Preferences Window

____ 4. Modify the Hello portlet.

__ a. From the **Hello** web project, expand the **src** folder. Expand **hello**.



__ b. Select **HelloPortlet.java** and **double-click** to open it in the Java editor.



Portlet Modes

Portlets perform different tasks and create content according to their current function. A portlet mode indicates the function a portlet is performing, at a point in time. A portlet mode specifies the kind of task the portlet should perform and what content it should generate. When invoking a portlet, the portlet container provides the mode for the current request to the portlet. Portlets can programmatically change their portlet mode while processing an action request.

JSR 168 defines three categories of portlet modes.

View

Display the portlet output.

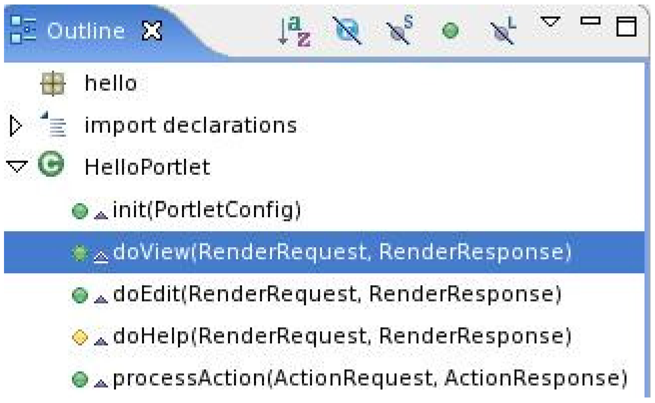
Edit

Display one or more views that let the user personalize portlet settings.

Help

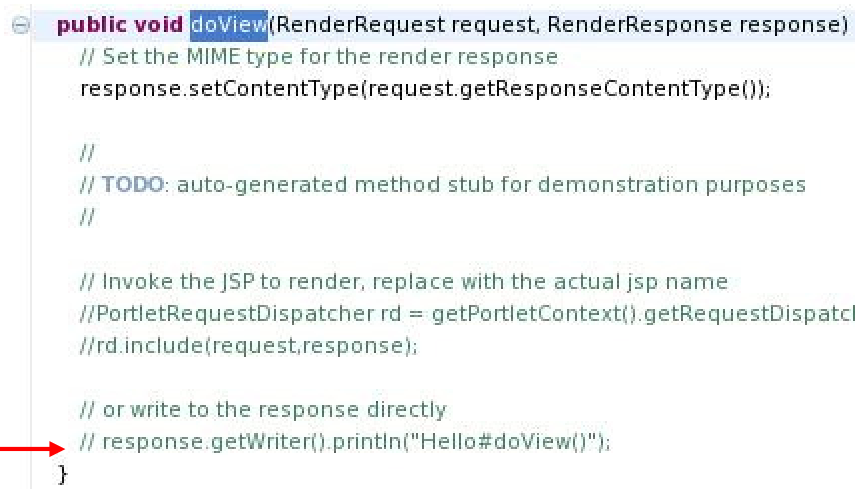
Display help views.

__ c. From the Outline view, click on **doView()**


Outline
hello
import declarations
HelloPortlet
init(PortletConfig)
doView(RenderRequest, RenderResponse)
doEdit(RenderRequest, RenderResponse)
doHelp(RenderRequest, RenderResponse)
processAction(ActionRequest, ActionResponse)

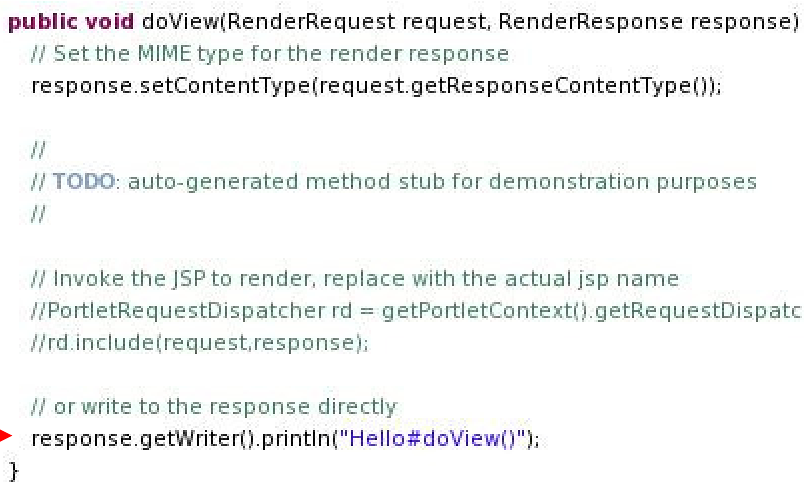
__ d. In the **doView()** method, locate the line

```
// response.getWriter().println("Hello#doView()");
```



```
public void doView(RenderRequest request, RenderResponse response)
  // Set the MIME type for the render response
  response.setContentType(request.getResponseContentType());

  //
  // TODO: auto-generated method stub for demonstration purposes
  //

  // Invoke the JSP to render, replace with the actual jsp name
  //PortletRequestDispatcher rd = getPortletContext().getRequestDispatcl
  //rd.include(request,response);

  // or write to the response directly
  // response.getWriter().println("Hello#doView()");
}
```

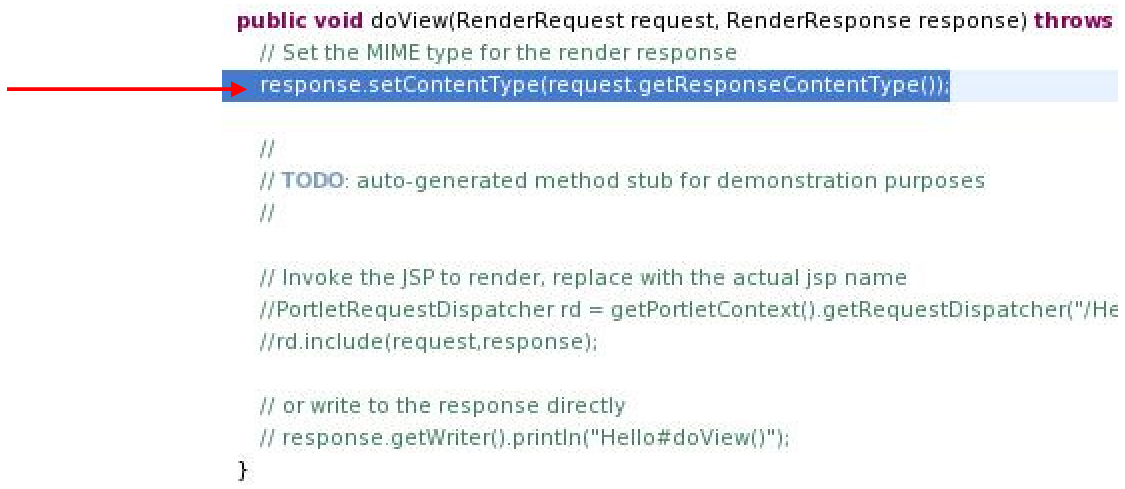
__ e. Remove the two forward slashes from beginning of this line – i.e. uncomment this line.



```
public void doView(RenderRequest request, RenderResponse response)
  // Set the MIME type for the render response
  response.setContentType(request.getResponseContentType());

  //
  // TODO: auto-generated method stub for demonstration purposes
  //

  // Invoke the JSP to render, replace with the actual jsp name
  //PortletRequestDispatcher rd = getPortletContext().getRequestDispatc
  //rd.include(request,response);

  // or write to the response directly
  response.getWriter().println("Hello#doView()");
}
```

__ f. This simple change is all that is needed to have a functional portlet. The portlet wizard automatically generates this standard code as a quick starting point. You will now make similar changes the **doEdit()** method of the Hello portlet.

__ g. In the **doView()** method, locate and highlight the line
`response.setContentType(request.getResponseContentType());`



```
public void doView(RenderRequest request, RenderResponse response) throws
    // Set the MIME type for the render response
    response.setContentType(request.getResponseContentType());

    //
    // TODO: auto-generated method stub for demonstration purposes
    //

    // Invoke the JSP to render, replace with the actual jsp name
    //PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher("/He
    //rd.include(request,response);

    // or write to the response directly
    // response.getWriter().println("Hello#doView()");
}
```

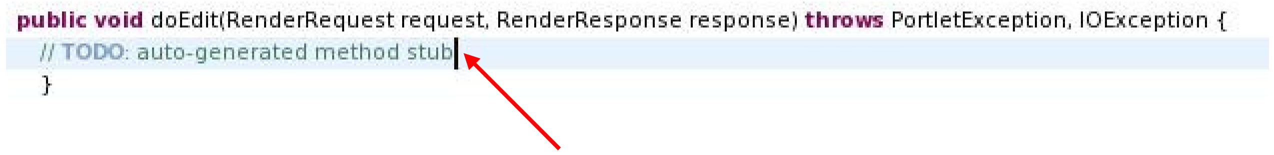
__ h. Use **CTRL+C** to copy this line.

__ i. From the Outline view, click on **doEdit()**

__ j. In the **doEdit** method, locate the line.

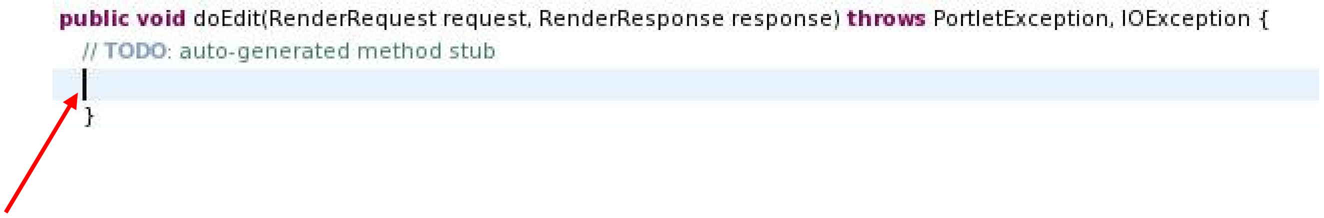
`// TODO: auto-generated method stub`

__ k. **Position** your **cursor** at the end of this line.



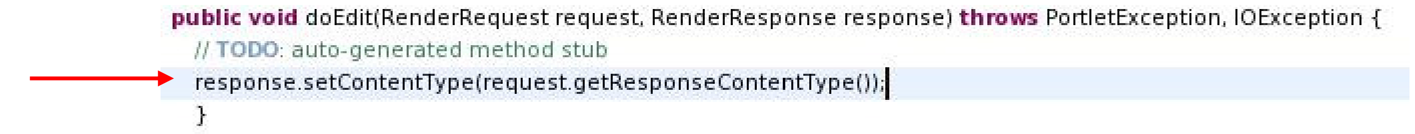
```
public void doEdit(RenderRequest request, RenderResponse response) throws PortletException, IOException {
    // TODO: auto-generated method stub
}
```

__ l. Press **Enter** to create a new line.



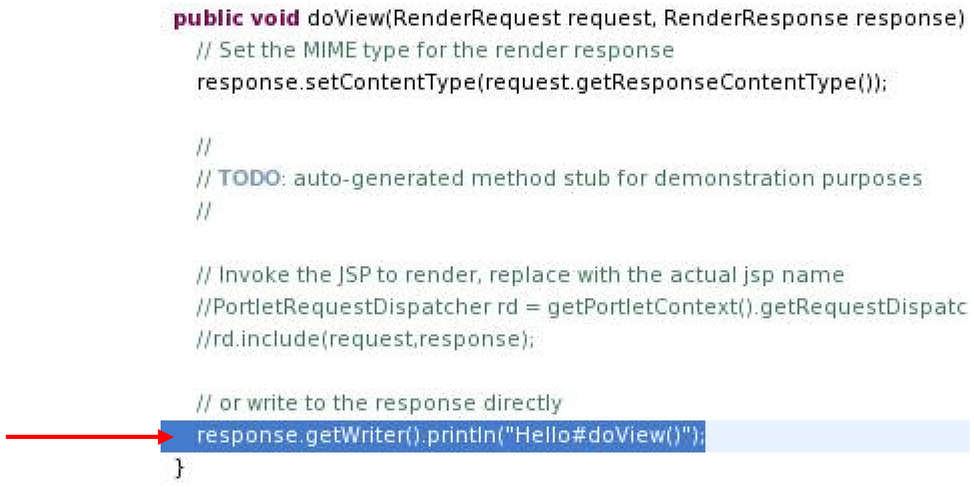
```
public void doEdit(RenderRequest request, RenderResponse response) throws PortletException, IOException {
    // TODO: auto-generated method stub

}
```

__ m. Use **CTRL+V** to paste



```
public void doEdit(RenderRequest request, RenderResponse response) throws PortletException, IOException {
  // TODO: auto-generated method stub
  response.setContentType(request.getResponseContentType());
  }
```

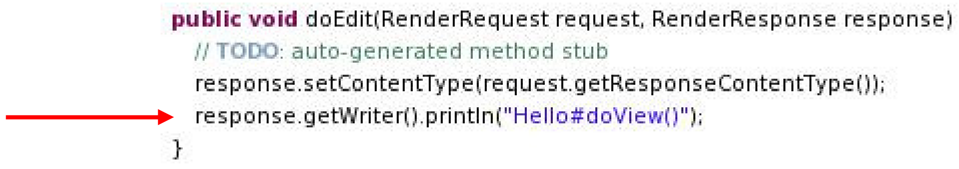
__ n. Return to the **doView()** method. Use the mouse to select the line `response.getWriter().println("Hello#doView()");`



```
public void doView(RenderRequest request, RenderResponse response)
  // Set the MIME type for the render response
  response.setContentType(request.getResponseContentType());

  //
  // TODO: auto-generated method stub for demonstration purposes
  //

  // Invoke the JSP to render, replace with the actual jsp name
  //PortletRequestDispatcher rd = getPortletContext().getRequestDispatc
  //rd.include(request,response);

  // or write to the response directly
  response.getWriter().println("Hello#doView()");
}
```

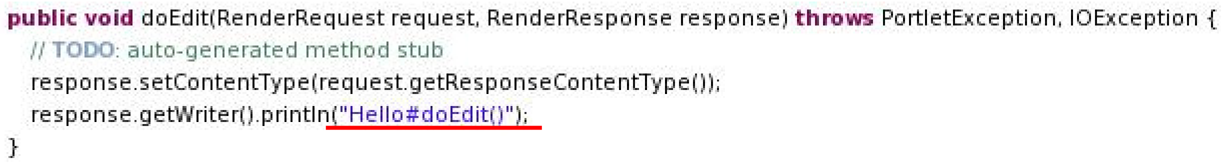
__ o. Use **CTRL+C** to copy this line.

__ p. Returning to the **doEdit()** method, use **CTRL+V** to paste below `response.setContentType(request.getResponseContentType());`



```
public void doEdit(RenderRequest request, RenderResponse response)
  // TODO: auto-generated method stub
  response.setContentType(request.getResponseContentType());
  response.getWriter().println("Hello#doView()");
}
```

__ q. Change the new line to read: **`response.getWriter().println("Hello#doEdit()");`**



```
public void doEdit(RenderRequest request, RenderResponse response) throws PortletException, IOException {
  // TODO: auto-generated method stub
  response.setContentType(request.getResponseContentType());
  response.getWriter().println("Hello#doEdit()");
}
```

__ r. Use **CTRL+S** to save this file.

__ s. Close the **HelloPortlet.java** editor

Part 3: Configuring the WebSphere test environment

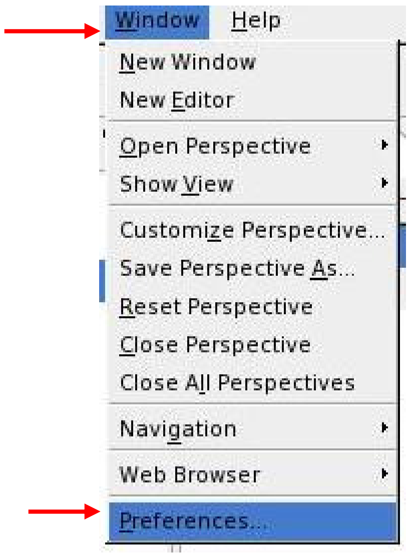
The *WebSphere® test environment* is a runtime environment that is integrated into the workbench for testing applications that are targeted for WebSphere Application Server.

The test environment for WebSphere Application Server v6.1 requires a ***full*** installation of the WebSphere Application Server and is enabled through a **Run server with resources within the workspace** publishing setting.

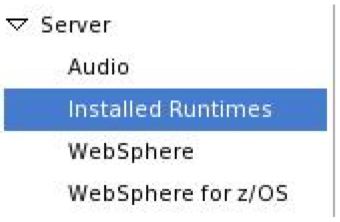
For this lab, WebSphere Application Server v6.1 is already installed on your machine and there is no need to install a test environment.

____ 1. **Configure and set the WebSphere test environment**

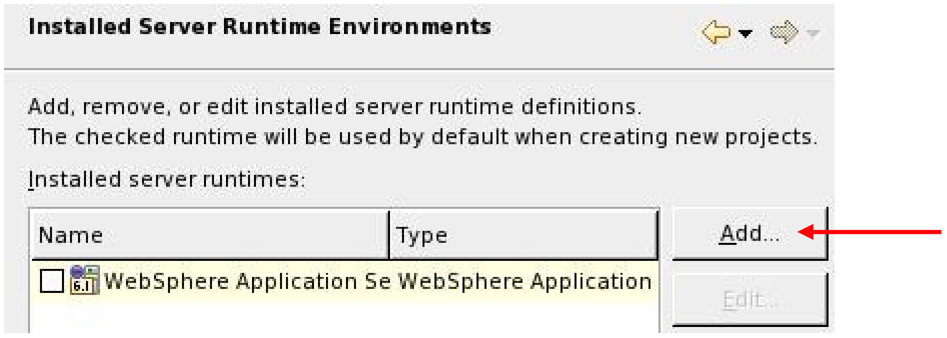
__ a. From the Application Server Toolkit menubar, select **Window → Preferences**…



__ b. On the left-hand menu, expand **Server**, select **Installed Runtimes**



__ c. In the **Installed Server Runtime Environments** panel, click **Add…**



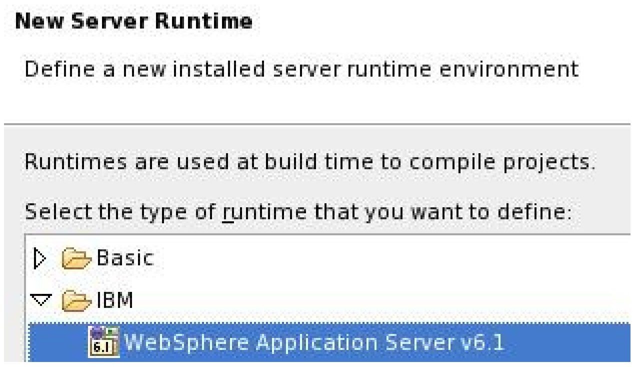
Installed Server Runtime Environments

Add, remove, or edit installed server runtime definitions.
The checked runtime will be used by default when creating new projects.

Installed server runtimes:

Name	Type
WebSphere Application Se	WebSphere Application

__ d. In the **New Server Runtime** dialog, accept the default server runtime of WebSphere Application Server v6.1 and click **Next.**



New Server Runtime

Define a new installed server runtime environment

Runtimes are used at build time to compile projects.

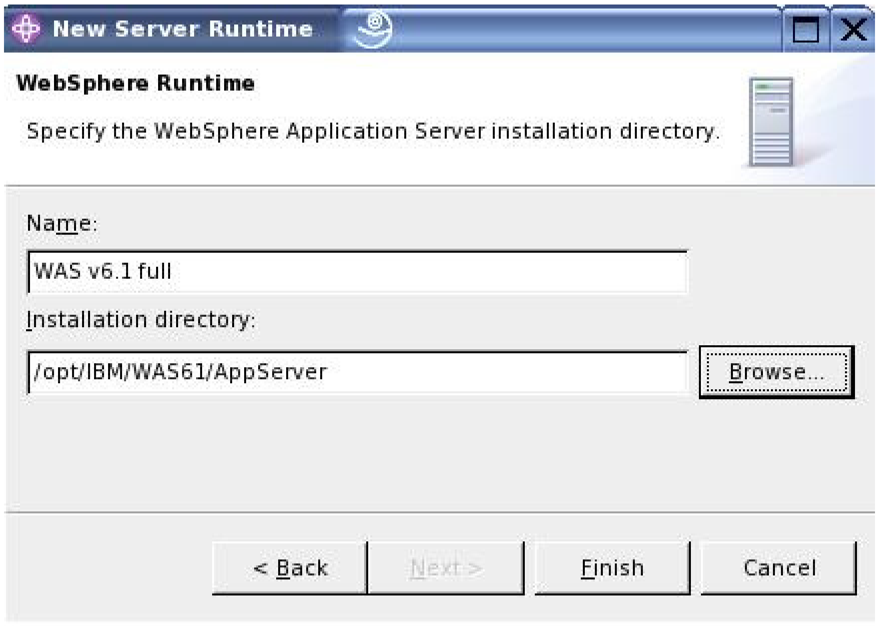
Select the type of runtime that you want to define:

__ e. In the WebSphere Runtime panel -

1) Enter a **Name** of **`WAS v6.1 full`**

2) Set the installation directory to **`/opt/IBM/WAS61/AppServer –`** use the **Browse…** button

3) Click **Finish**



New Server Runtime

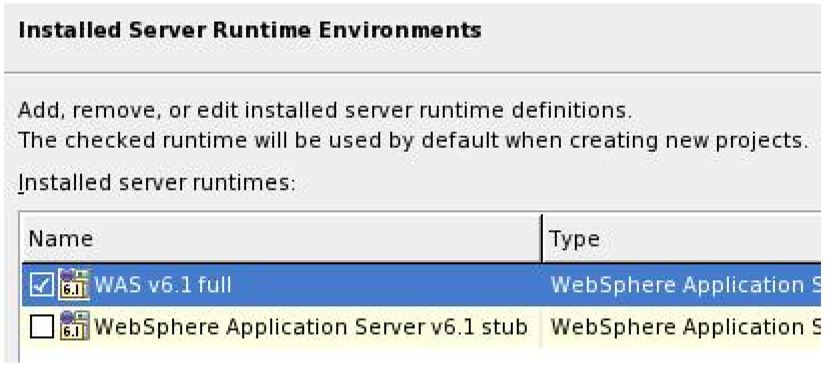
WebSphere Runtime

Specify the WebSphere Application Server installation directory.

Name: WAS v6.1 full

Installation directory: /opt/IBM/WAS61/AppServer

__ f. In the Installed Server Runtime Environments panel, select **WAS v6.1 full** as the default runtime



Installed Server Runtime Environments

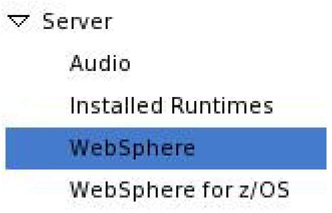
Add, remove, or edit installed server runtime definitions.
The checked runtime will be used by default when creating new projects.

Installed server runtimes:

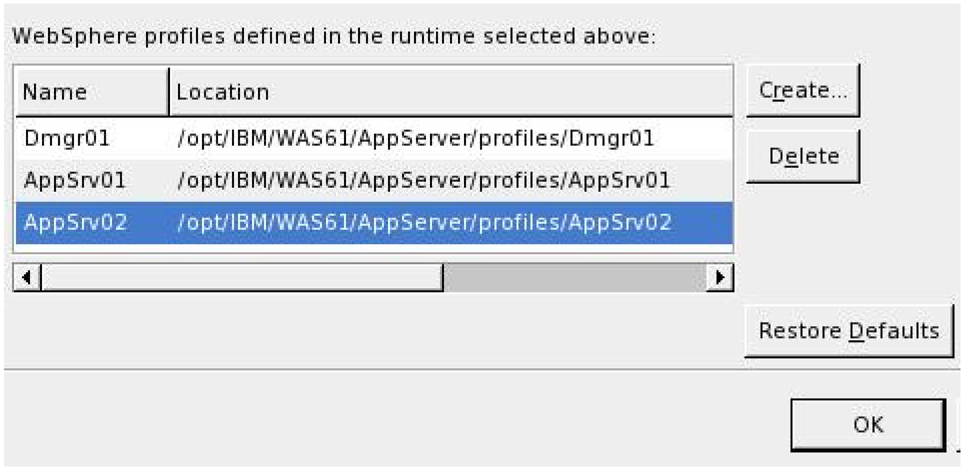
Name	Type
✓ WAS v6.1 full	WebSphere Application S
☐ WebSphere Application Server v6.1 stub	WebSphere Application S

__ g. If you have already created WebSphere profiles in the previous labs, skip to the next page and begin with **2) Add a new WebSphere Application Server v6.1 test server**

__ h. If you do not have any existing WebSphere profiles, return to the left-hand side preferences and select **Server → WebSphere**



__ i. On the right-hand side of the view, the configured WebSphere runtimes and profiles are shown. Click on the **Create…** button. Create a new Application Server profile, with the Typical Profile options. When finished, click **OK** to return to the J2EE perspective.



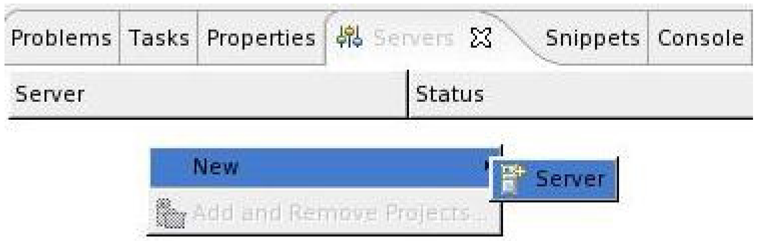
WebSphere profiles defined in the runtime selected above:

Name	Location
Dmgr01	/opt/IBM/WAS61/AppServer/profiles/Dmgr01
AppSrv01	/opt/IBM/WAS61/AppServer/profiles/AppSrv01
AppSrv02	/opt/IBM/WAS61/AppServer/profiles/AppSrv02

____ 2. **Add a new WebSphere Application Server v6.1 test server**

__ a. Locate and select the **Servers** view. The Servers view allows you to manage the servers. This view displays a list of all your servers and configurations that are associated with that server. You can use this view to start, start in debug mode, start in profile mode, restart, or stop the servers.

__ b. Right-click in the Servers view and select **New → Server**



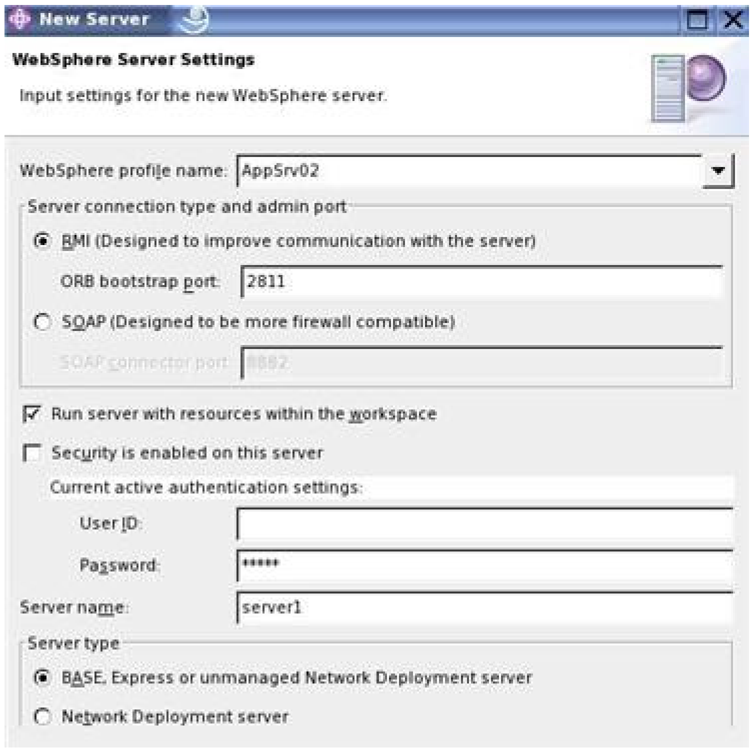
__ c. In the **Define a New Server** panel, ensure the server type is **WebSphere v6.1 Server** and the Server Runtime is **WAS v6.1 full**. Click **Next**.

__ d. In the WebSphere Server Settings:

1) Select a WebSphere profile name of **AppSrv02** (or another profile you have created). Notice the tool automatically determines the correct WebSphere Administration port of 2811.

2) **Uncheck** 'Security is enabled on this server'. Also notice that the tool is designed to work with Base and Express servers, as well as a Network Deployment topology.

3) Click **Next**



New Server

WebSphere Server Settings

Input settings for the new WebSphere server.

WebSphere profile name: AppSrv02

Server connection type and admin port

(●) RMI (Designed to improve communication with the server)

ORB bootstrap port: 2811

( ) SOAP (Designed to be more firewall compatible)

SOAP connector port: 8882

[✓] Run server with resources within the workspace

[ ] Security is enabled on this server

Current active authentication settings:

User ID:

Password: *****

Server name: server1

Server type

(●) BASE, Express or unmanaged Network Deployment server

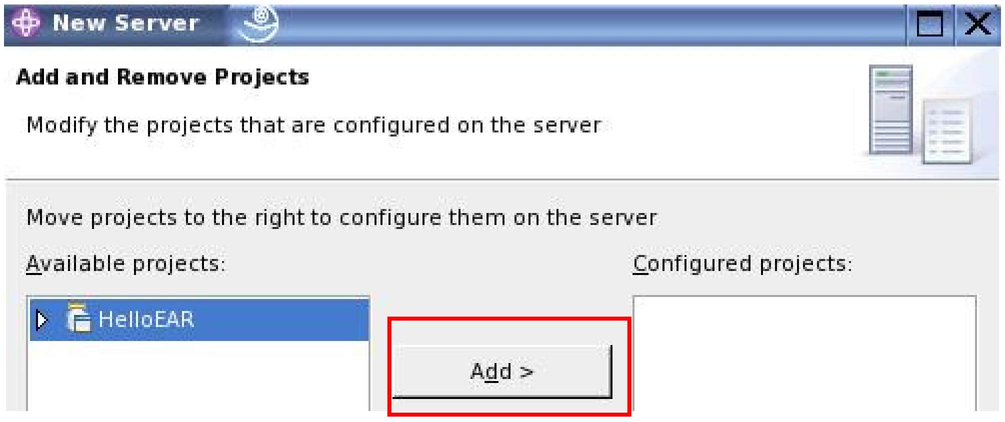
( ) Network Deployment server

__ e. In the **Add and Remove Projects** panel

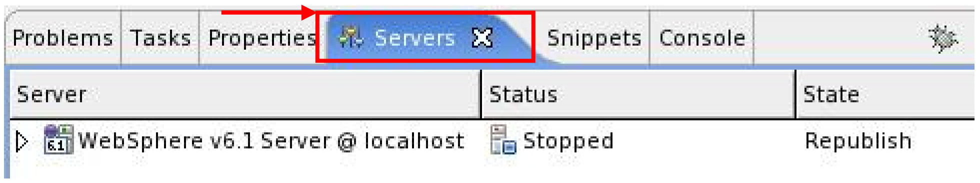
1) Select **HelloEAR**

2) Click **Add >** to add the **HelloEAR** project to the server.

3) Click **Finish** when you are done.

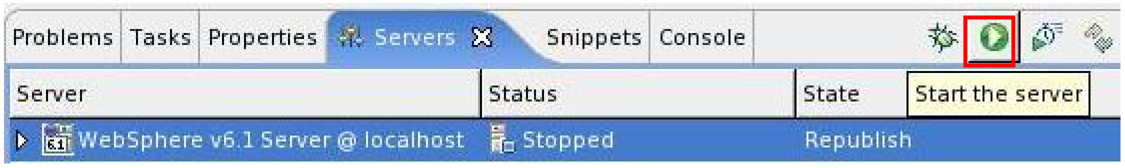

New Server
Add and Remove Projects
Modify the projects that are configured on the server
Move projects to the right to configure them on the server
Available projects:
Configured projects:
HelloEAR
Add >

__ f. In the **Servers** view, you should now have a WebSphere v6.1 Server in the Stopped state.



Server	Status	State
WebSphere v6.1 Server @ localhost	Stopped	Republish

__ g. Highlight **WebSphere v6.1 Server @ localhost**. Click the **Start** icon.



Server	Status	State
WebSphere v6.1 Server @ localhost	Stopped	Republish

Part 4: Testing the Hello portlet

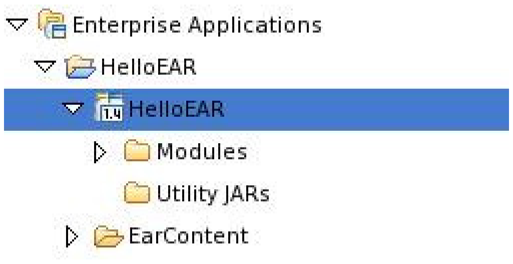
You can invoke each portlet by its context root and name with the URL mapping /<portlet-name> that is created for each portlet. The context root and name has the following format:

```
http://<host>:<port>/<context-root>/<portlet-name>
```

For example, http://localhost:9080/portlets/TestPortlet1

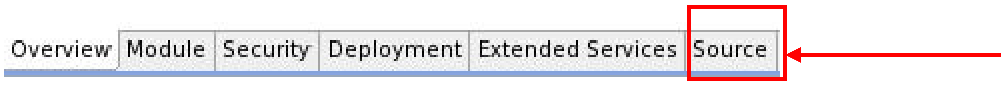
____ 1. Determine the context root for the **Hello** Web Application

__ a. From the Project Explorer view, expand **Enterprise Applications → HelloEAR** and select the **HelloEAR** Application Deployment Descriptor.

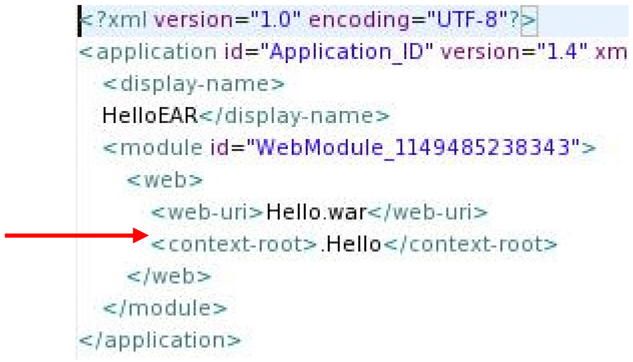


__ b. Double-click on the **HelloEAR** Application Deployment Descriptor to open the graphical editor

__ c. Locate the tabs at the bottom of the view and click on **Source**



__ d. Locate the **`<context-root>`** section of the file. Notice the context root is **.Hello** (with a leading period). This is the standard naming convention for portlet applications.

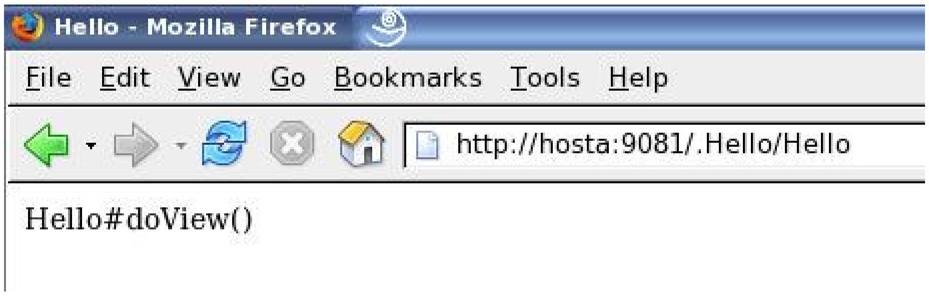


```
<?xml version="1.0" encoding="UTF-8"?>
<application id="Application_ID" version="1.4" xm
  <display-name>
  HelloEAR</display-name>
  <module id="WebModule_1149485238343">
    <web>
      <web-uri>Hello.war</web-uri>
      <context-root>.Hello</context-root>
    </web>
  </module>
</application>
```

__ e. **Close** the Application Deployment Descriptor editor

__ f. **Start** a browser.

__ g. Enter a URL of http://localhost:9081/.Hello/Hello

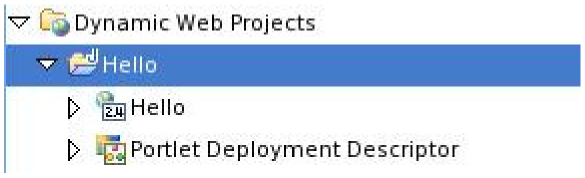

Hello - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://hosta:9081/.Hello/Hello

Hello#doView()

You have just displayed a portlet using the PortletServingServlet. You can only display one portlet at a time using the PortletServingServlet servlet. If you want to aggregate multiple portlets on the page, you need to use the aggregation tag library.

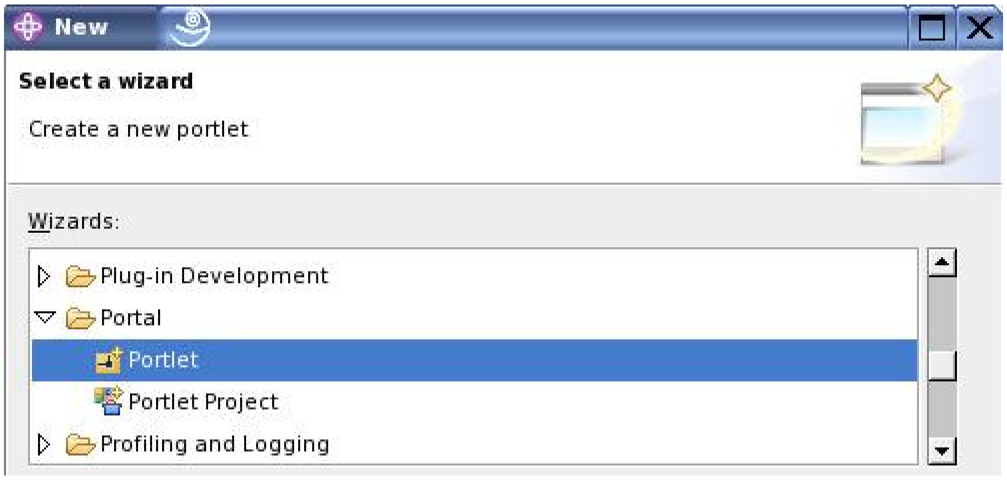
_____ 2. Create a portlet using Java Server Pages

__ a. From the Project Explorer view, select the **Hello** web project

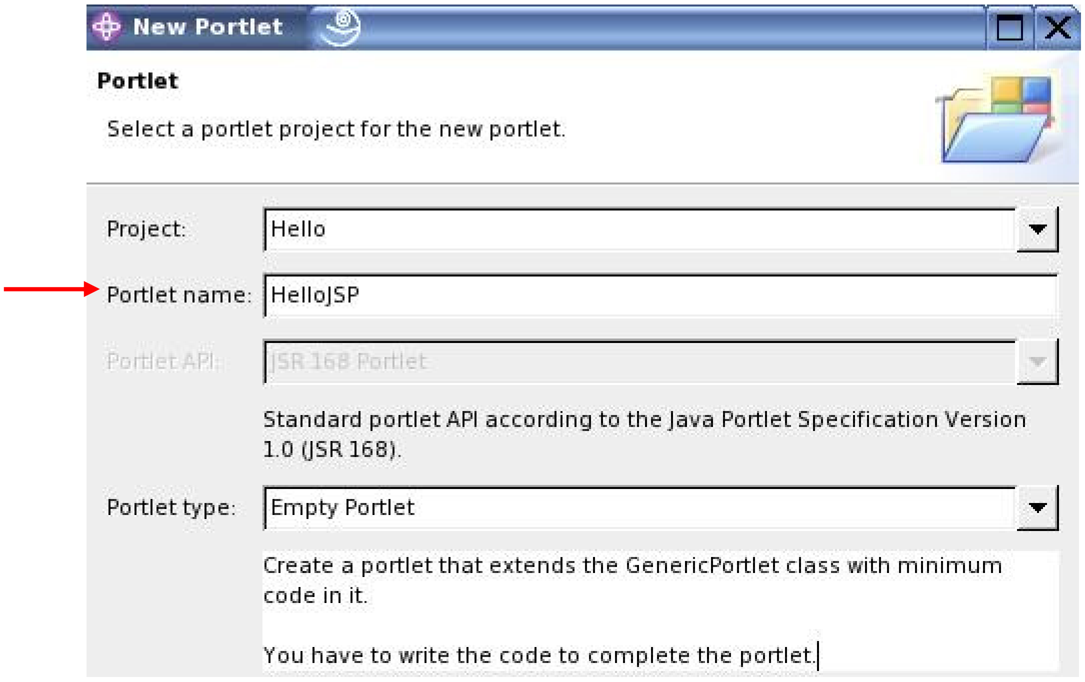


__ b. Use the **CTRL+N** key to open the new creation wizard.

__ c. Locate and expand **Portal**. Select **Portlet.** Click **Next**



__ d. Enter a Portlet name of **HelloJSP.** Click **Next**



Portlet

Select a portlet project for the new portlet.

Project: Hello

Portlet name: HelloJSP

Portlet API: JSR 168 Portlet

Standard portlet API according to the Java Portlet Specification Version 1.0 (JSR 168).

Portlet type: Empty Portlet

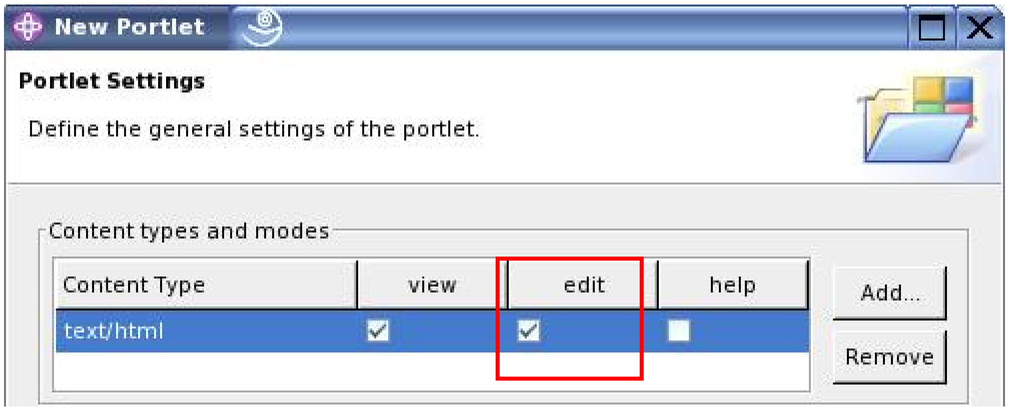
Create a portlet that extends the GenericPortlet class with minimum code in it.

You have to write the code to complete the portlet.

__ e. In the Portlet Settings window --

1) In the **Content types and modes** section, select **edit**

2) Click **Finish**



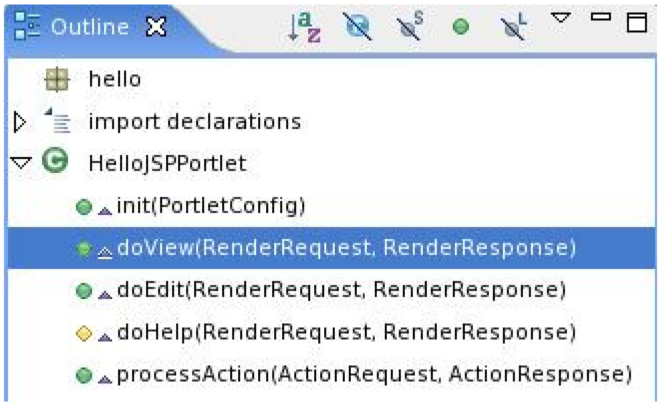
Portlet Settings

Define the general settings of the portlet.

Content types and modes

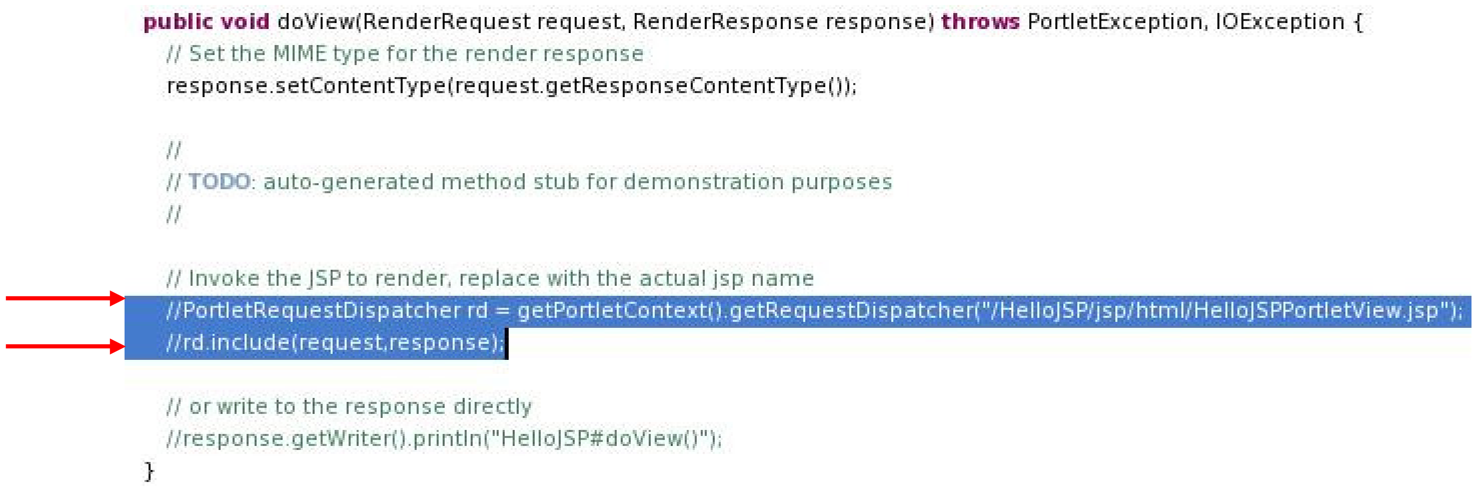
Content Type	view	edit	help
text/html	✓	✓	

__ f. The **HelloJSPPortlet.java** file will now be opened in the editor. In the **Outline** view, select the **doView()** method


Outline
hello
import declarations
HelloJSPPortlet
init(PortletConfig)
doView(RenderRequest, RenderResponse)
doEdit(RenderRequest, RenderResponse)
doHelp(RenderRequest, RenderResponse)
processAction(ActionRequest, ActionResponse)

__ g. In the **doView()** method of the **HelloJSPPortlet.java** file, locate the lines

```
// PortletRequestDispatcher rd =
getPortletContext().getRequestDispatcher("/HelloJSP/...");

// rd.include(request,response);
```

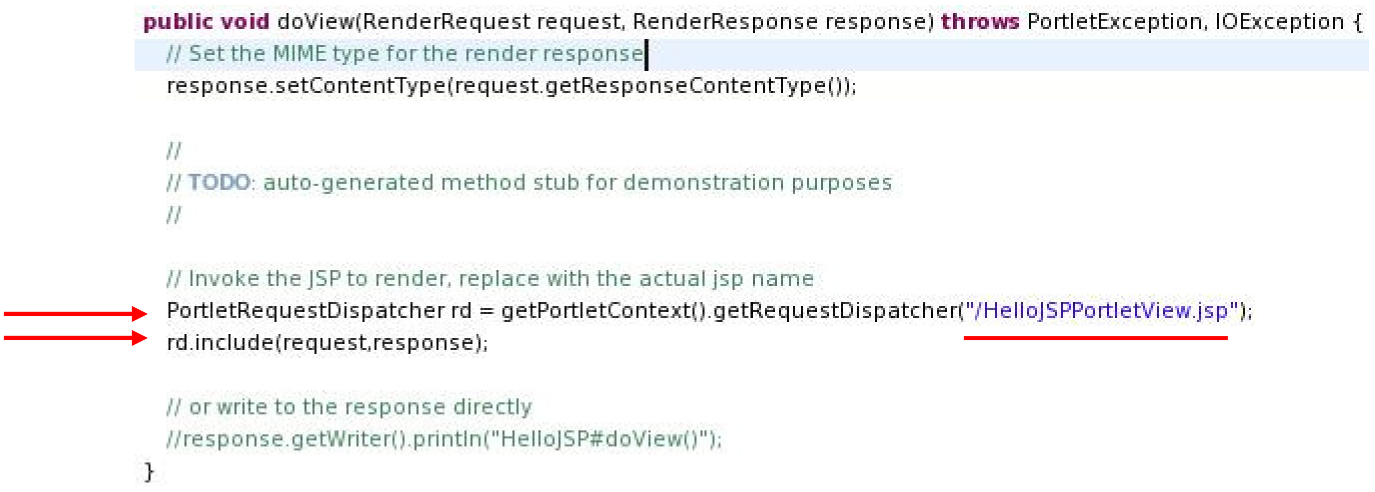


```
public void doView(RenderRequest request, RenderResponse response) throws PortletException, IOException {
  // Set the MIME type for the render response
  response.setContentType(request.getResponseContentType());

  //
  // TODO: auto-generated method stub for demonstration purposes
  //

  // Invoke the JSP to render, replace with the actual jsp name
  //PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher("/HelloJSP/jsp/html/HelloJSPPortletView.jsp");
  //rd.include(request,response);

  // or write to the response directly
  //response.getWriter().println("HelloJSP#doView()");
}
```

__ h. Perform the following actions

1) Remove the two forward slashes from beginning of these lines – i.e. uncomment these lines.

2) Change **`("/HelloJSP/jsp/html/HelloJSPPortletView.jsp")`** to **`("/HelloJSPPortletView.jsp")`**



```
public void doView(RenderRequest request, RenderResponse response) throws PortletException, IOException {
    // Set the MIME type for the render response
    response.setContentType(request.getResponseContentType());

    //
    // TODO: auto-generated method stub for demonstration purposes
    //

    // Invoke the JSP to render, replace with the actual jsp name
    PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher("/HelloJSPPortletView.jsp");
    rd.include(request,response);

    // or write to the response directly
    //response.getWriter().println("HelloJSP#doView()");
}
```

__ i. Using the mouse, select the these two lines.

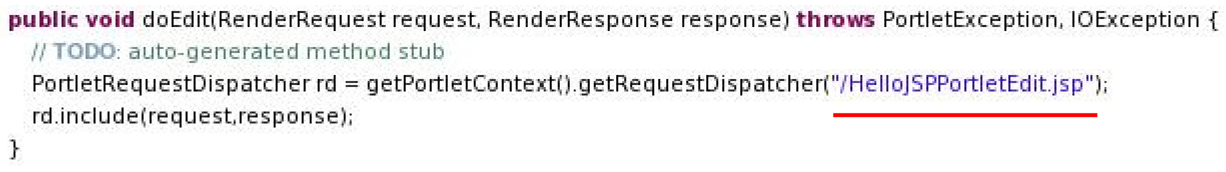
__ j. Use **CTRL+C** to copy these two lines.

__ k. From the Outline view, click on **doEdit()**

__ l. In the **doEdit** method, locate the line **`// TODO: auto-generated method stub`**

__ m. Use **CTRL+V** to paste in the additional lines below **`// TODO: auto-generated method stub`**

__ n. Change **`("/HelloJSPPortletView.jsp") to ("/HelloJSPPortletEdit.jsp")`**

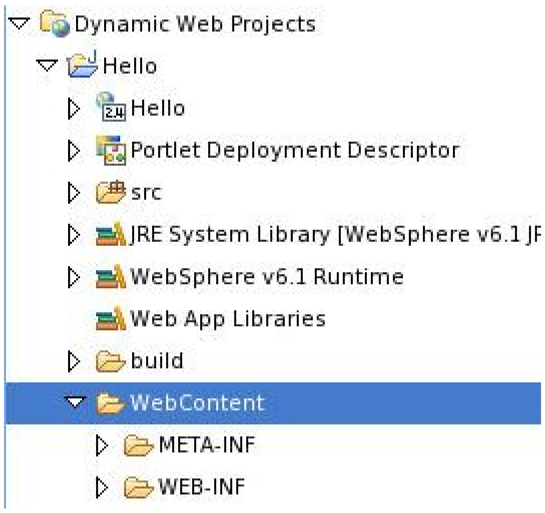


```
public void doEdit(RenderRequest request, RenderResponse response) throws PortletException, IOException {
    // TODO: auto-generated method stub
    PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher("/HelloJSPPortletEdit.jsp");
    rd.include(request,response);
}
```

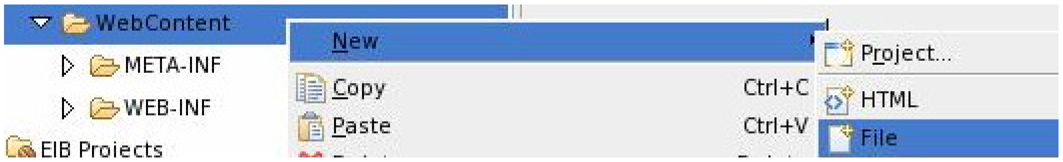
__ o. Use **CTRL+S** to save this file.

__ p. Close the **HelloJSPPortlet.java** editor

__ q. Returning the **Project Explorer** view, select the **WebContent** folder



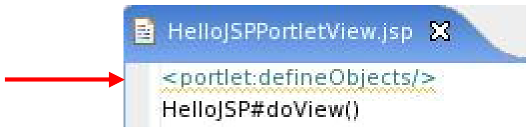
__ r. **Right-click** and select **New → File**



__ s. Enter a file name of **HelloJSPPortletView.jsp** and click **Finish**

__ t. In the **HelloJSPPortletView.jsp** editor, enter the following code

```
<portlet:defineObjects/>

HelloJSP#doView
```


HelloJSPPortletView.jsp

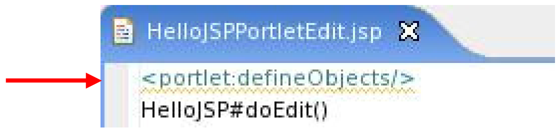
```
<portlet:defineObjects/>
HelloJSP#doView()
```

__ u. Use **CTRL+S** to save the file

__ v. Close the **HelloJSPPortletView.jsp** editor

__ w. Follow the same steps above to create a **HelloJSPPortletEdit.jsp** file**.** The content of the file is:

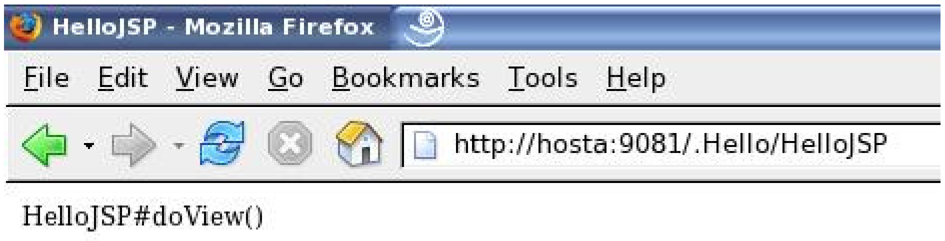
```
<portlet:defineObjects/>

HelloJSP#doEdit
```



```
<portlet:defineObjects/>
HelloJSP#doEdit()
```

__ x. Use **CTRL+S** to save your changes and close the **HelloJSPPortletEdit.jsp** file

__ y. From a browser enter a URL of http://localhost:9081/.Hello/HelloJSP


HelloJSP - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://hosta:9081/.Hello/HelloJSP

```
HelloJSP#doView()
```

You have finished developing a second simple portlet, this time using JSPs for rendering the output.

Part 5: Portlet aggregation using Java Server Pages

The aggregation tag library generates a portlet aggregation framework to address one or more portlets on one page. If you write JavaServer Pages, you can aggregate multiple portlets on one page using the aggregation tag library. This tag library does not provide full featured portal aggregation implementation, but provides a good migration scenario if you already have aggregating servlets and JavaServer Pages and want to switch to portlets.

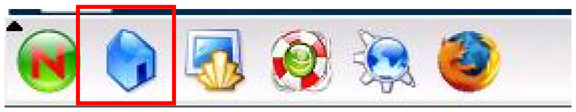
To allow the customer to create a simple portal aggregation, the aggregation tag library also provides the following features.

- Invoke a portlet's action method
- Render multiple portlets on one page
- Provide links to change the portlet's mode or window state
- Display the portlet's title
- Retain the portlet cookie state

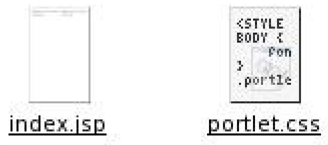
The aggregation tag library and JavaServer Pages that use the aggregation tag library will only work with the WebSphere Application Server portlet container implementation because the protocol between the tags and the container is not standardized.

____ 1. Developing a portlet aggregation JSP is beyond the scope of this lab, so you will import a completed version.

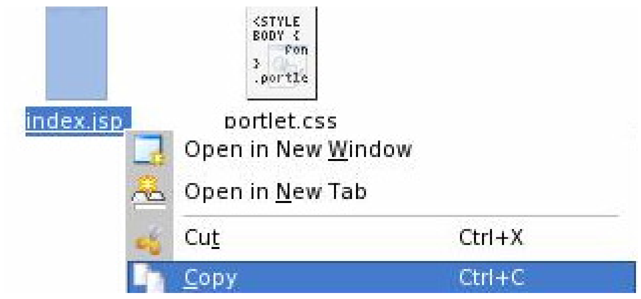
1) From the SLES Desktop, locate the KDE Panel at the bottom of the workspace. Click on the 'Personal Files' icon.



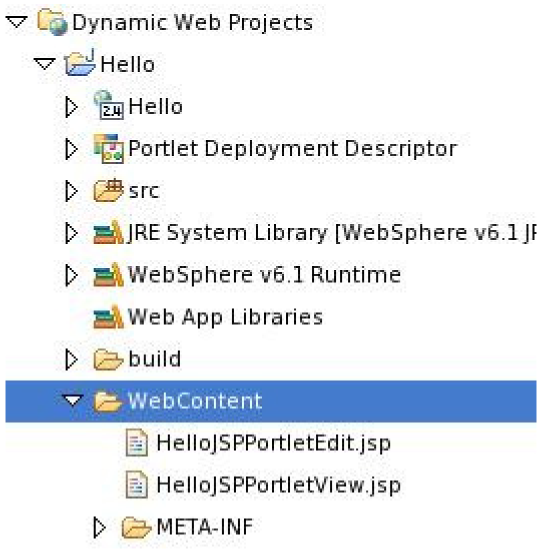
2) Navigate to **`/root/WAS61STEW/hands-on/Lab03-Portlet.`** You should see two files.



3) Right-click on **index.jsp** and select **Copy**.



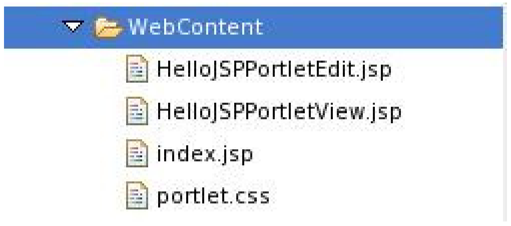
4) Return to the **AST**. In the **Hello** project, select the **WebContent** folder.



5) Enter **CTRL+V** (paste) to copy the file

6) Follow a similar procedure to copy and paste the **portlet.css** file into the **WebContent** folder

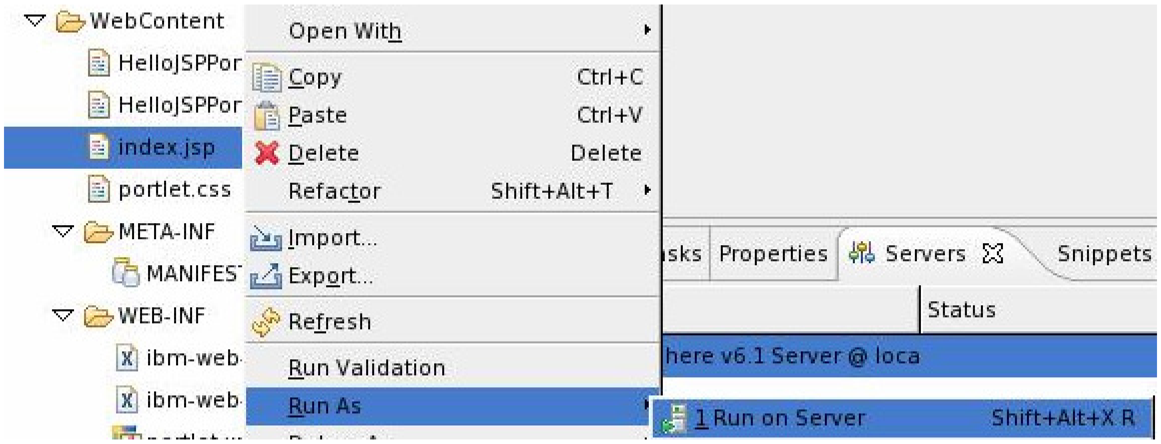
7) When finished, you will have two new files in the **WebContent** folder



____ 2. **Testing portlet aggregation using Java Server Pages**

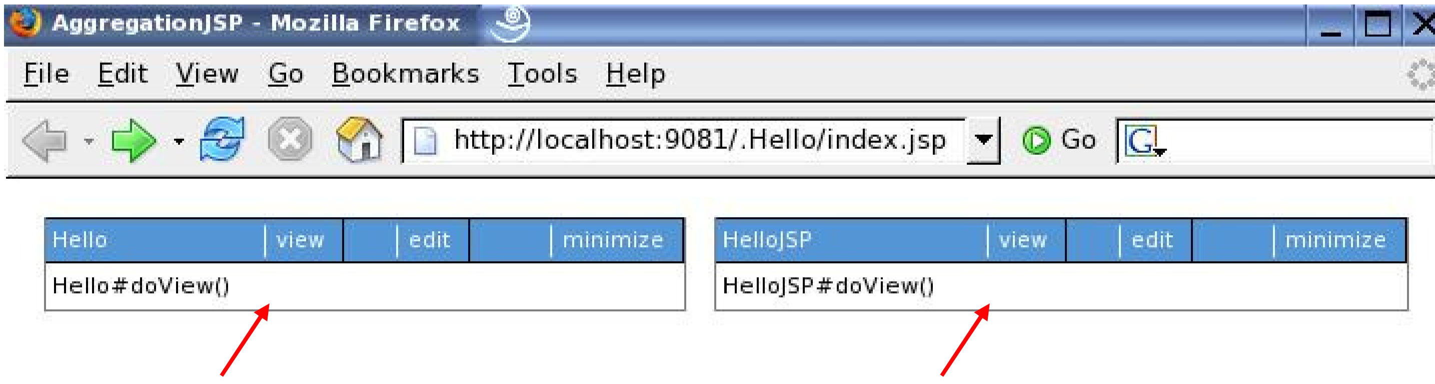
__ a. From the Web Content folder, select **index.jsp**

__ b. **Right-click** and select **Run As → Run on Server**



__ c. Accept the server defaults and click **Finish**.

__ d. From the browser, you should now see both portlets aggregated on the page. Note: If you are not using port 9081, edit **index.jsp** and modify the PortletURLPrefix.


AggregationJSP - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:9081/.Hello/index.jsp
Go
Hello | view | edit | minimize
Hello#doView()
HelloJSP | view | edit | minimize
HelloJSP#doView()

Hello Portlet **HelloJSP Portlet**

__ e. Click on the **view** and **edit** locations in each portlet. This calls the **doView()** and **doEdit()** methods you developed previously. In this simple example, use the browser's back button to return to the **index.jsp** page.

__ f. For additional information on developing with the Portlet aggregation tag library, refer to the WebSphere Application Server v6.1 InfoCenter.

This page is intentionally left blank.