SAP NetWeaver
How-To Guide

How To... Build a Navigation Menu with the Navigation API for Ajax Framework Page

Applicable Releases:

SAP NetWeaver 7.0 EhP2

IT Practice

User Productivity Enablement

IT Scenario

Running an Enterprise Portal

Version 1.0

May 2010

HOW-TO GUIDE

THE BEST-RUN BUSINESSES RUN SAP™

Document History

Document Version	Description
1.00	First official release of this guide

Typographic Conventions

Type Style	Description
Example Text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, graphic titles, and table titles
`Example text`	File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

Icons

Icon	Description
	Caution
	Note or Important
	Example
	Recommendation or Tip

Table of Contents

1. Business Scenario 1

2. Background Information 1

3. Prerequisites 1

4. Step-by-Step Procedure 2

4.1 Set up the application project 2

4.1.1 Create the application project 2

4.1.2 Create a portal component 2

4.1.3 Create a JSP file: lsapi_demo.jsp 4

4.1.4 Create a JavaScript file: lsapi_demo.js 4

4.1.5 Create a Cascading Style Sheet (CSS) file 4

4.1.6 Add images 5

4.2 Write the Code 6

4.2.1 Attach files to the portal component 6

4.2.2 Implement JSP file: lsapi_demo.jsp 7

4.2.3 Implement JavaScript file lsapi_demo.js 8

4.3 Deploy your application 10

4.4 Test your application 11

5. Appendix 14

1. Business Scenario

This tutorial walks you through the steps necessary for the implementation of basic dynamic tree navigation (DTN) with the help of the Navigation API.

You call the Navigation API methods to retrieve information about the current state of navigation and use this information to render the navigation tree by building the DOM (Document Object Model). You also use the Navigation API to handle navigation events, such as expanding a node.

2. Background Information

The Ajax Framework Page (AFP) is based on the Ajax technology that provides improved performance, dynamic navigation and enhanced user experience. The Ajax technology determines that most of the page's functionality takes place on the client side, for example, navigation hierarchy is cached on the client side.

AFP provides the JavaScript client-side Navigation API that exposes the navigation hierarchy and provides functionality to control navigation in the portal. You can use the Navigation API to trigger and manipulate the navigation iViews and/or regular portal navigation

The Navigation API is comprised of the following objects:

- **NavigationNode:** represents a node in the navigation tree
- **AFPModel**: provides read-only access to the navigation tree of the current user. For more information please visit section: 4.2.3 Implement JavaScript file lsapi_demo.js
- **AFPController**: controls access to navigation events. For more information please visit section: 4.2.2 Implement JSP file lsapi_demo.jsp
- **AFPService:** enables actions, such as navigation to a specific node

For more information please visit http://wiki.sdn.sap.com/wiki/display/AFP

3. Prerequisites

The following are the prerequisites that must be in place before using this How-To-Guide.

- SAP NetWeaver 7.0 EhP2 Java Server and NetWeaver Developer Studio
- Experience in JavaScript programming

  Familiarity with DOM. More information: http://www.w3.org/DOM
- Familiarity with portal application development.

  More information: Running an Enterprise Portal: http://help.sap.com/saphelp_nw70ehp1/helpdata/EN/19/4554426dd13555e10000000a1550b0/frameset.htm in the SAP NetWeaver Developer's GuideSoftware
- Refer to SAP Note 1166135 (central note for Ajax Framework Page Navigation). This note includes known issues and limitations

4. Step-by-Step Procedure

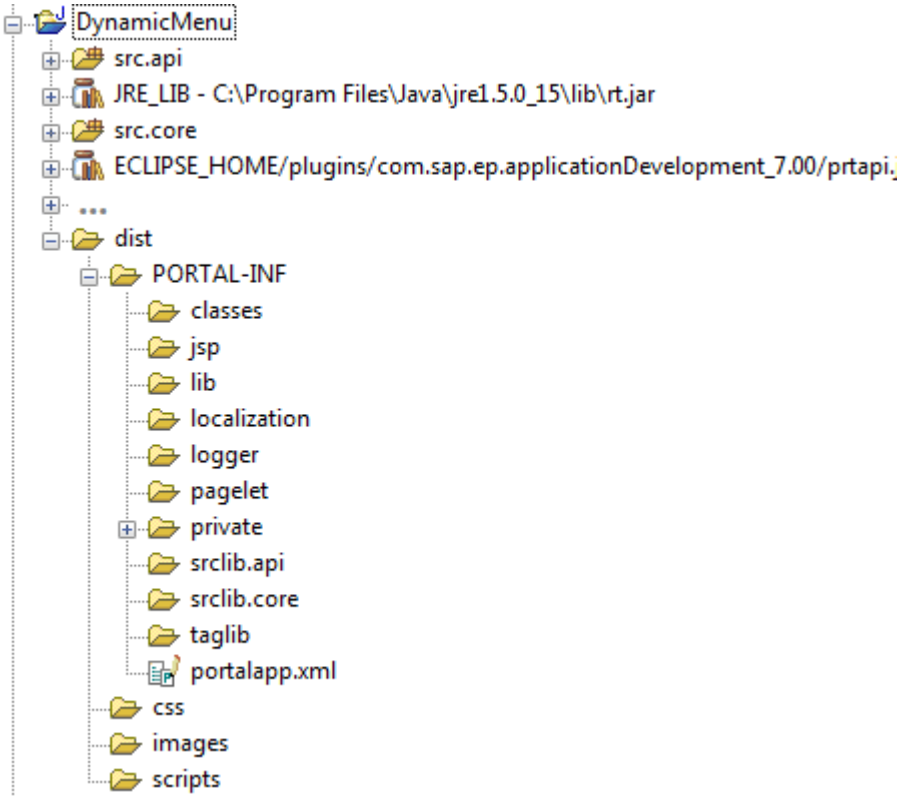
You can use the Navigation API in JavaScript files, which are included in the portal applications that you develop in the SAP NetWeaver Developer Studio. The workflow is the same as the one used for regular portal applications: after completing development, you package your portal application into a PAR file and deploy it to the portal

4.1 Set up the application project

4.1.1 Create the application project

In the SAP NetWeaver Developer Studio

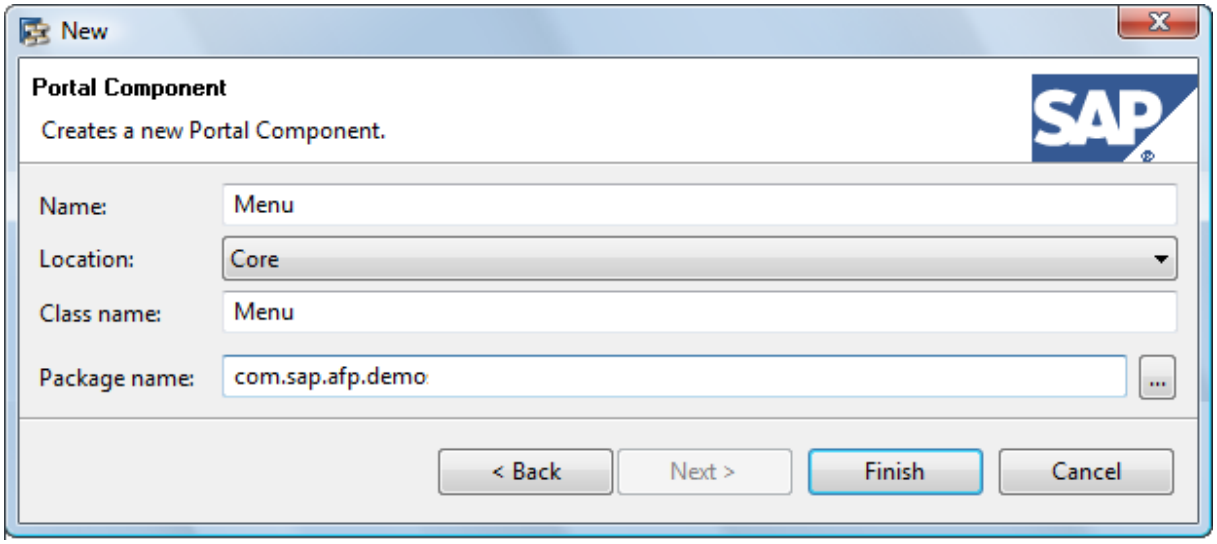
1. From the File menu, select *New* → *Other*. The New window is displayed.
2. Select Portal Application, and then *Create* → *Portal Application Project*. Click Next.
3. In the **`Project name`** textbox, enter a name for the project, such as *DynamicMenu*
4. Click Finish.
5. The wizard creates the following:



4.1.2 Create a portal component

1. From the File menu, select *New* → *Other*. The New window is displayed.
2. Select Portal Application, and then *Create a New Portal Application Object*. Click Next.
3. Select the project to which you want to add the portal component (*DynamicMenu*). Click Next.

4. Select *Portal Component* → *AbstractPortalComponent*. Click Next.
5. Enter the following fields:
    a. **Name:** Name of component, which is defined by the name attribute of the <component> element for this component in the portalapp.xml.
    b. **Location:** The location of the new Java class file for this component, which determines whether other applications can reference the classes.
        i. Core (default): Places the file in the src.core project directory and makes it private.
        ii. API: Places the file in the src.api and makes it public.
    c. Class name: The Java class file for this component.
    d. Package name: The package name for the Java class file for this component.



6. Click Finish.
7. The wizard does the following:
    a. Creates an entry for the component in the application's portalapp.xml file, as shown below

```
<components>
  <component name="Menu">
    <component-config>
      <property name="ClassName" value="com.sap.afp.demo.Menu"/>
    </component-config>
    <component-profile/>
  </component>
</components>
```

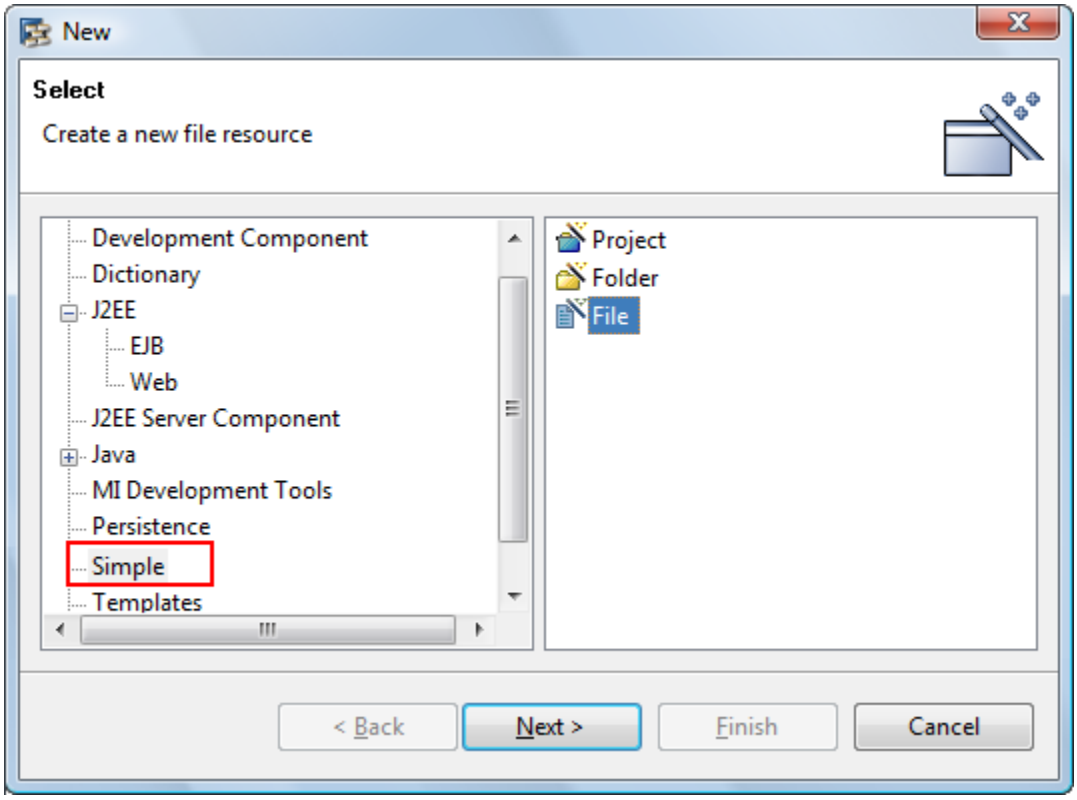
    b. Creates a class in the src.core folder called myPortalComp that extends AbstractPortalComponent. A blank doContent method is also created, which is the only method that must be implemented.

```
public class Menu extends AbstractPortalComponent{
    public void doContent(IPortalComponentRequest request,
IPortalComponentResponse response){

    }
}
```

4.1.3 Create a JSP file: lsapi_demo.jsp

In the *dist* → *PORTAL-INF* → *jsp* subfolder

1. Right click the *jsp* subfolder and select *New* → *Other*
2. Go to *Simple* → *file*. Click *Next*



3. Enter **`lsapi_demo.jsp`** in the *File name* field.
4. Click Finish

4.1.4 Create a JavaScript file: lsapi_demo.js

In the *dist* → *scripts* subfolder

1. Right click the *scripts* subfolder and select *New* → *Other*
2. Go to *Simple* → *file*. Click Next
3. Enter **`lsapi_demo.js`** in the *File name* field.
4. Click Finish

4.1.5 Create a Cascading Style Sheet (CSS) file

In the *dist* → *css* subfolder

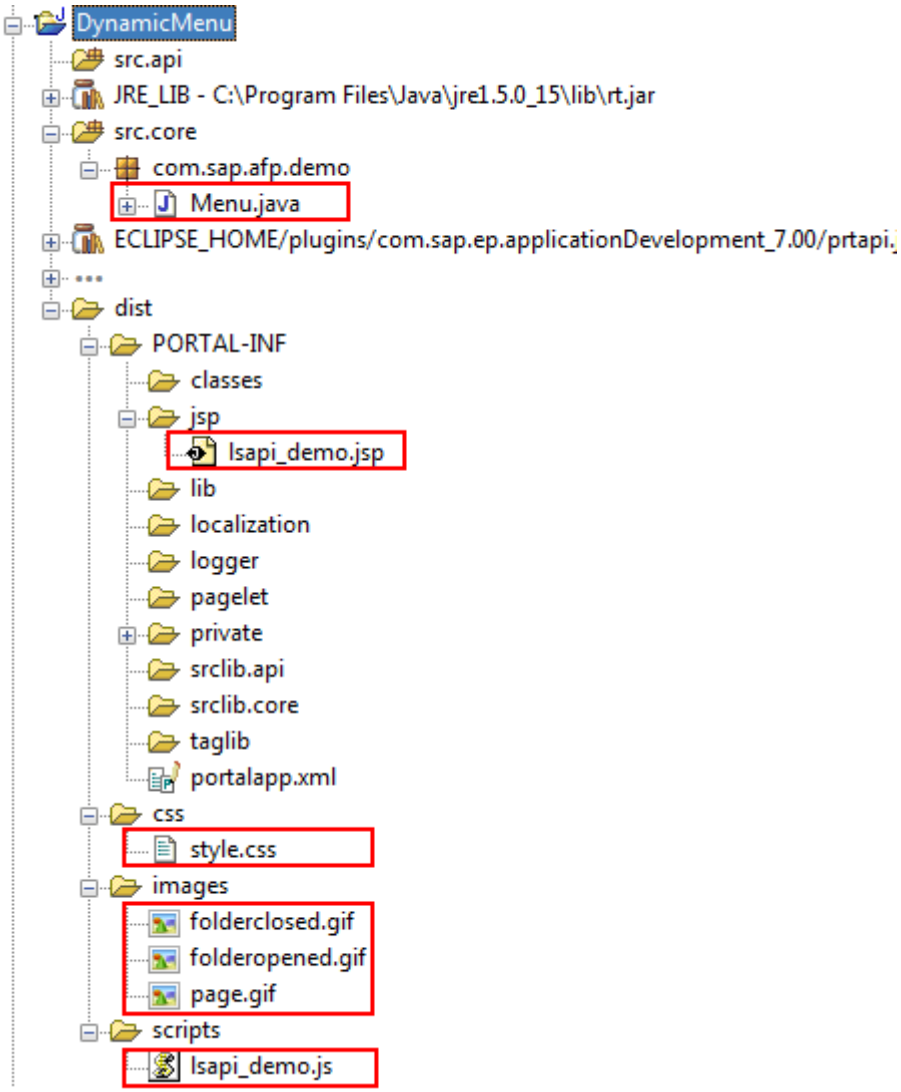
1. Right click the *css* subfolder and select *New* → *Other*
2. Go to Simple and select file. Click Next
3. Enter **`style.css`** in the *File Name* field. Click Finish
4. Paste the following style classes within your style.css file

```
A:link {
   text-decoration: none;
   font-family: "Helvetica Neue", "Lucida Grande", Helvetica, Arial,
Verdana, sans-serif;
    font-size: 12px;
    color: #666;
}
A:visited {
   text-decoration: none;
   font-family: "Helvetica Neue", "Lucida Grande", Helvetica, Arial,
Verdana, sans-serif;
    font-size: 14px;
    color: #ddd;
}
A:active {text-decoration: none}

A:hover {
   text-decoration: none;
    font-family:Georgia,serif;
    font-weight: bold;
   letter-spacing:2px;
}

.sap-TreeNodes {padding: 0 0 0 20px;}

.sap-TreeNodeIcon-FolderClosed, .sap-TreeNodeIcon-FolderOpened, .sap-
TreeNodeIcon-Page {
   padding: 0 4px 0 4px;
   width: 19px;
   height: 19px;
}

.sap-TreeNodeIcon-FolderClosed {
   background:url(../images/folderclosed.gif);
}

.sap-TreeNodeIcon-FolderOpened {
   background:url(../images/folderopened.gif);
}

.sap-TreeNodeIcon-Page {
   background:url(../images/page.gif);
}
```

4.1.6 Add images

This exercise uses 3 images: folderclosed.gif, floderopened.gif and page.gig (Dimensions: 24x22)

1. In the Windows explorer, search for the NetWeaver Developer Studio workspace and save 3 images in <local drive>:\<MWDS workspace folder>\DynamicMenu\dist\images
2. Go to the NetWeaver Developer Studio and refresh the *Package Explorer* view
3. Your Portal project should look like the image below



4.2 Write the Code

4.2.1 Attach files to the portal component

1. Open Menu.java class located in *src.core* → *com.sap.afp.demo* → *Menu.java*
2. Paste the following code into the portal component's doContent() method. This code attaches the JSP file, JavaScript and the CSS file to the portal component

```
public void doContent(
   IPortalComponentRequest request,
   IPortalComponentResponse response) {

   IResource jsFunctions = request.getResource(IResource.SCRIPT,
      "scripts/lsapi_demo.js");
   response.include(request, jsFunctions);

   IResource cssStyle = request.getResource(IResource.CSS,
      "css/style.css");
   response.include(request, cssStyle);
```

```
    IResource jspPage =  request.getResource(IResource.JSP,
       "jsp/lsapi_demo.jsp");
    response.include(request, jspPage);
}
```

3. Organize import CTRL + SHIFT + O
4. Save changes

4.2.2 Implement JSP file: lsapi_demo.jsp

1. Open lsap_demo.jsp with the JSP editor
2. Enter the following heading

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Menu</title>
</head>
<body>
</body>
</html>
```

3. Between the **<body>** tags, add a ** ** element named *menuContainer*. This element is referenced from the JavaScript code for rendering the tree by injecting DOM elements.

```
<div id="menuContainer"
style="height:100%;width:100%;position:relative;"> </div>
```

4. After the ** ** tag, add the following JavaScript code snippet to declare and initialize global variables needed to build the navigation tree, and to call the **registerOnNavigate(createNavigationMenu)** method.

```
<script language="javascript">
//show the tree from the specified level
   var NM_START_FROM_LEVEL = 0;
   var flag_clickFromDTN = false;
   var menuContainer = document.getElementById("menuContainer");
   var selected_dtn_node;

   //LSAPI initial registration for navigation events
   EPCM.getSAPTop().LSAPI.AFPPlugin.controller.registerOnNavigate(create
NavigationMenu);
</script>
```

Important

registerOnNavigate(func, onlyNode)

Enables calling a notification function each time a navigation event occurs. You call this function at the beginning of your JavaScript code to retrieve the initial state of the navigation tree during the first page load.

Parameters

- **Func:** a notification function to call when navigation is performed. The function receives the current navigation node as a parameter: **func(currentNode)**
- **onlyNode**: If true, the callback function receives only the node itself rather than a pointer to the tree, which can be used to improve performance. Otherwise, the node is received with additional data, which allows exploring the tree

Return: None

5. Save the changes

4.2.3 Implement JavaScript file lsapi_demo.js

Add the following code to the lsapi_demo.js file

1. Open lsapi_demo.js
2. Add the **createNavigationMenu** notification function. This function uses an LSAPI call to retrieve the navigation nodes relative to the current node

```
function createNavigationMenu(currentNode){

   //check if click event received from the tree (DTN)
   if (flag_clickFromDTN)
   {
      flag_clickFromDTN = false;
      return;
   }

   //nothing is selected
   selected_dtn_node = null;

   //the container is empty
   menuContainer.innerHTML = "";

   var sapTreeEl = document.getElementById("menuContainer");

   EPCM.getSAPTop().LSAPI.AFPPlugin.model.getNavigationSubTree(null,draw
   Tree,sapTreeEl);
}
```

Important

getNavigationSubTree(name, callback, argument)

Returns an array of child nodes of the specified navigation node along with their subtrees.

Parameters

- **name:** The ID of the navigation node whose child nodes are to be returned. If set to null, the initial (top-level) nodes are returned
- **callback**: A callback function to return the result. The callback function receives the following parameters:
  Nodes: An array of NavigationNode objects representing the children of the specified navigation node. If the node ID is set to null, the initial (top-level) nodes are returned

Additional Parameter: The additional parameter passed to the getNavigationSubTree() method

- **`argument (optional):`** Any JavaScript object. The object is passed to the callback function

`Return`: None. The result is returned asynchronously through the callback function

3. Add the **`drawTree`** callback function.

For each node in the array, it calls the **`renderNode`** function

```
function drawTree(nodes, container) {
  for (var i=0; i<nodes.length; i++) {
     var renderedNode = renderNode(nodes[i]);
     container.appendChild(renderedNode);
  }
}
```

4. Add the **`renderNode`** function

This function renders the node according to the position of the node in the tree, if it doesn't have children, it displays the folder closed icon; otherwise it displays the page icon. It also add a div element to hide or display the children of a particular node.

```
function renderNode(node) {
  var expanded = false;

  var icon = document.createElement("span");
  icon.className = (node.hasChildren()?"sap-TreeNodeIcon-
FolderClosed":"sap-TreeNodeIcon-Page");

  var title = document.createElement("a");
  title.innerHTML = node.getTitle();
  title.href="javascript:void(0)";
  title.onclick = function() {
     EPCM.doNavigate(node.getName());
  };

  var childrenEl = null;

  var nodeEl = document.createElement("div");
  nodeEl.appendChild(icon);
  nodeEl.appendChild(title);

  icon.onclick = function() {
     expanded = !expanded;

     icon.className = (expanded?"sap-TreeNodeIcon-FolderOpened":"sap-
TreeNodeIcon-FolderClosed");
     if (expanded) {
        if (childrenEl==null)    {
           childrenEl = document.createElement("div");
           childrenEl.className = "sap-TreeNodes";
           nodeEl.appendChild(childrenEl);

           node.getChildren(drawTree, childrenEl);
        }
        childrenEl.style.display = '';
     } else {
```

```
            childrenEl.style.display = 'none';
        }
    }

    return nodeEl;
}
```

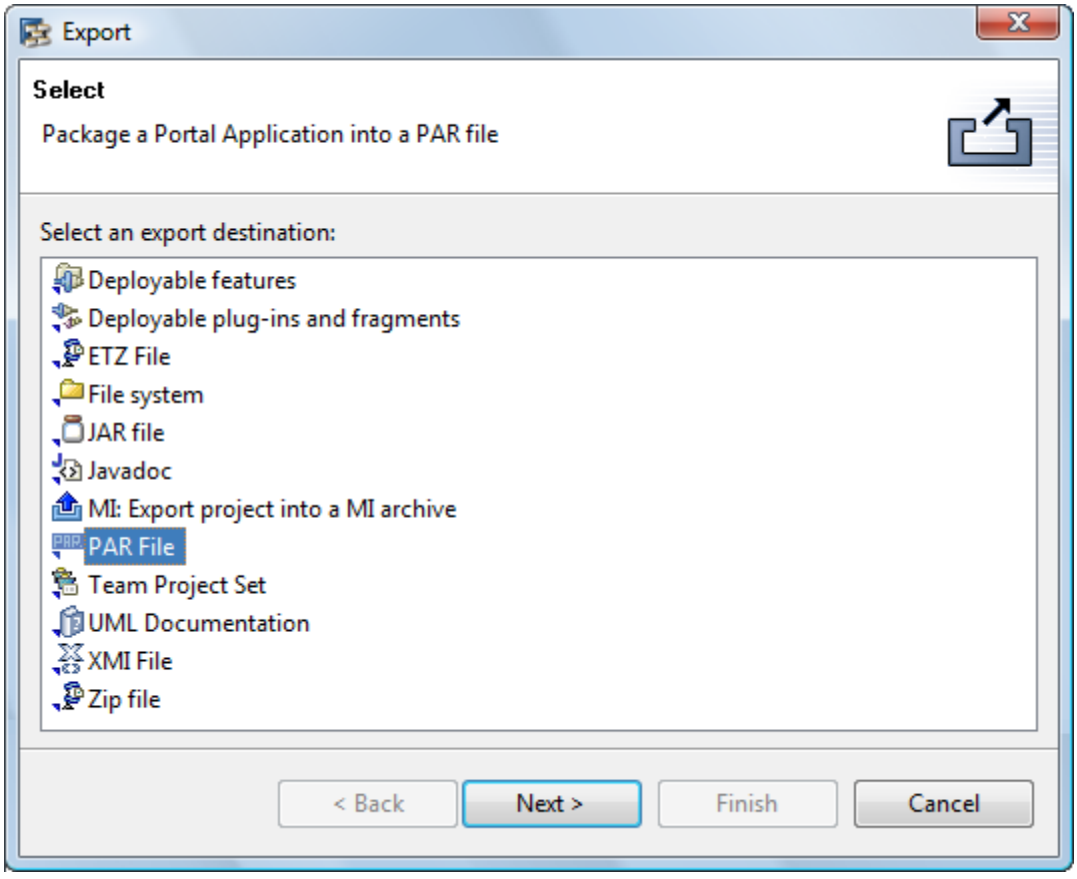
5. Save the changes

4.3 Deploy your application

Once you have created a portal application, you need to deploy it to the portal

A single PAR can be uploaded from the NetWeaver Developer Studio directly into a portal via the PAR Export feature of the portal plug-in for Eclipse

1. Choose *File* → *Export* → *PAR File*. Choose Next.

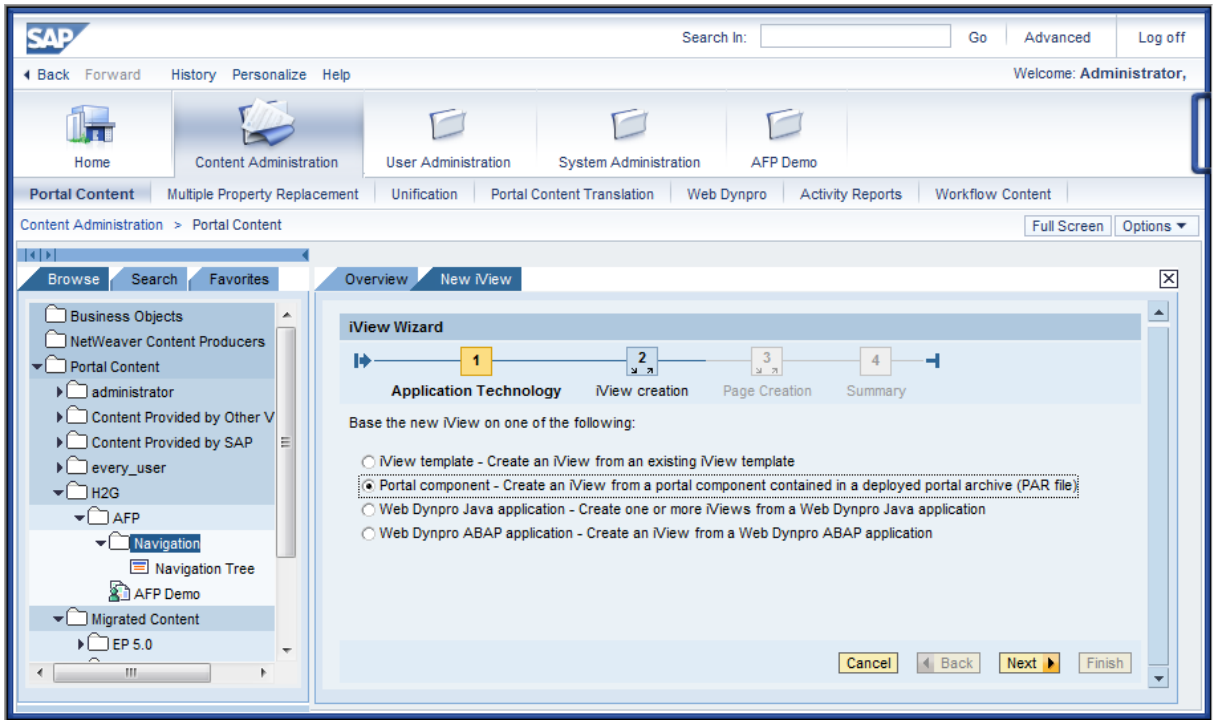


2. Select the project where the PAR file is located. Choose Next.
3. Enter data as required. If you want to deploy the PAR, choose Deploy PAR.
4. Choose Finish.

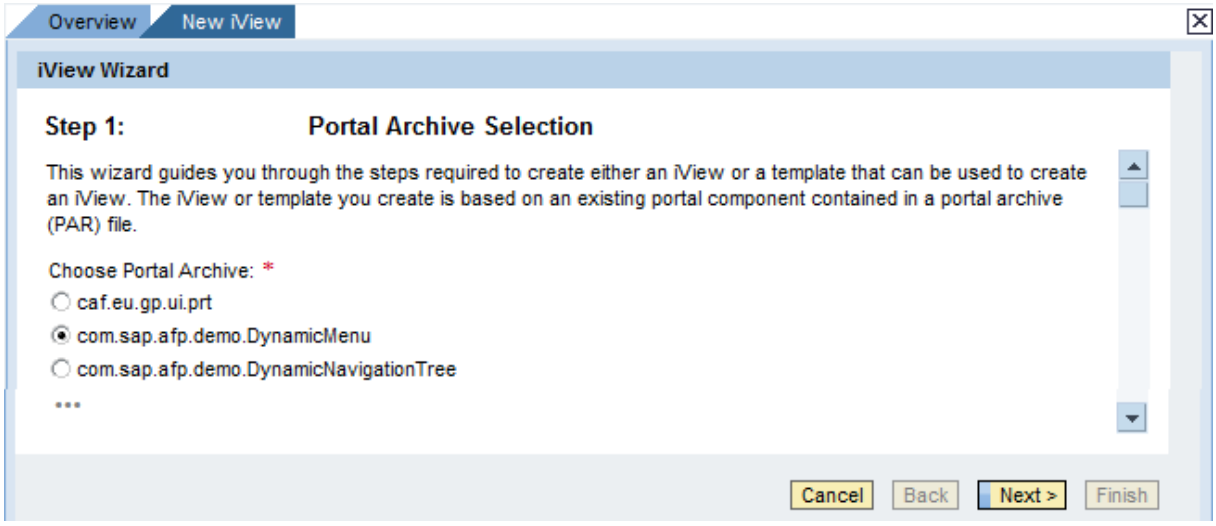
4.4 Test your application

Once deployed, you can create an iView from the applications, and then run the iView and view the content generated by the component. For more information, please visit SAP NetWeaver 7.0 EHP2 Help site

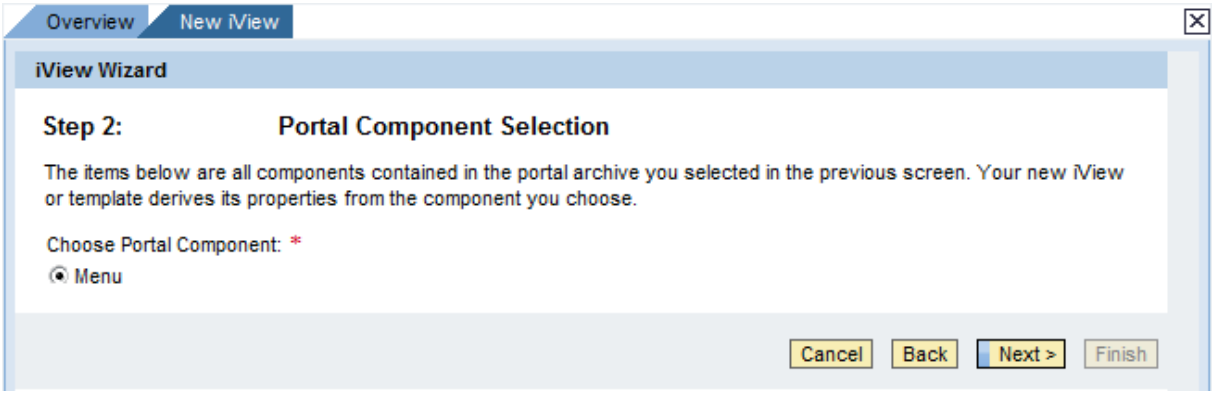
1. Launch the Portal Content Studio by choosing *Content Administration* → *Portal Content*
2. In the Portal Catalog, right-click the folder in which you want to create the iView.
3. In the context menu that appears, choose *New* → *iView*. The iView Wizard is launched.
4. Select Portal Component. Click Next



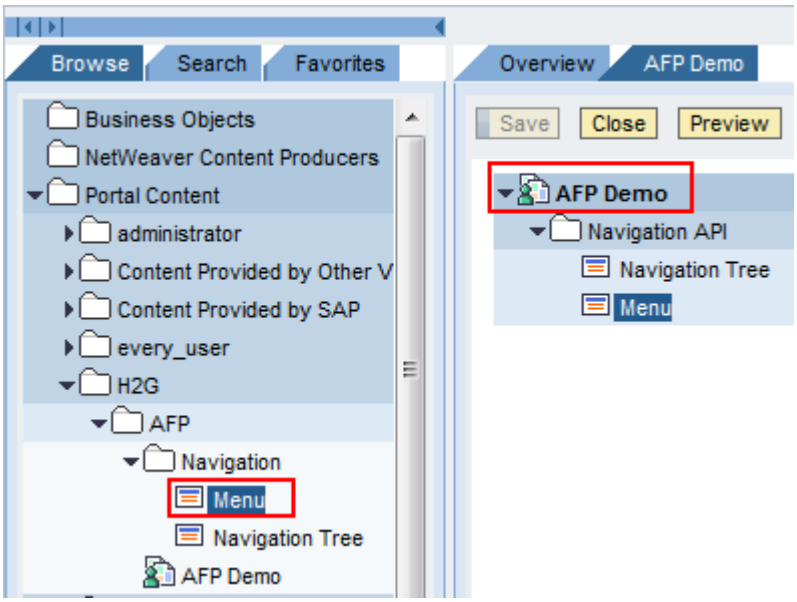
5. Select the DynamicMenu par file. Click Next



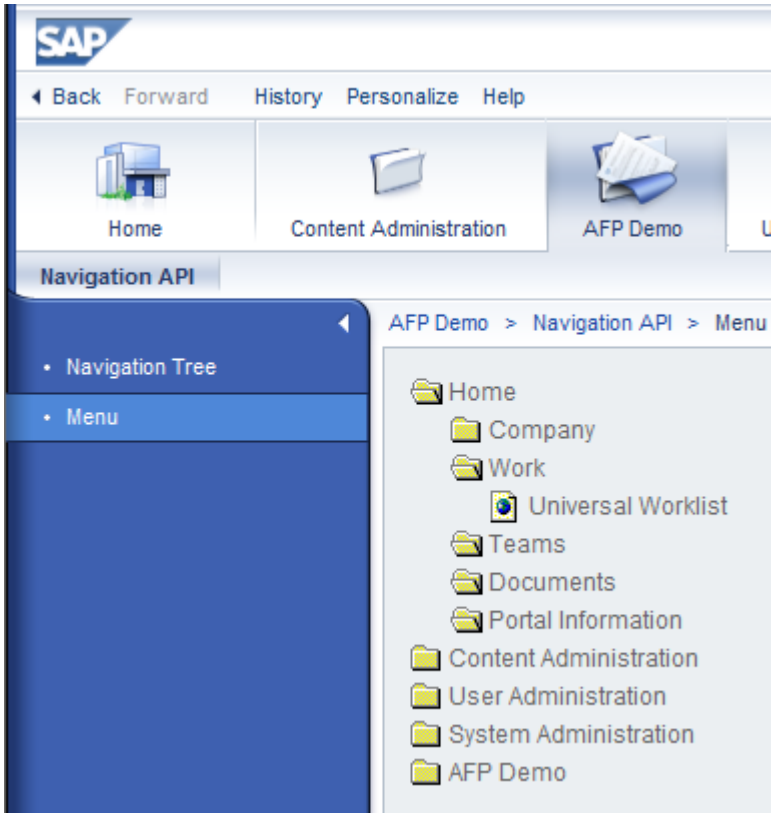
6. Select Menu Portal Component. Click Next



7. Enter General Properties: iView name, ID and Language. Click Next
8. Click Finish
9. Add your new iView to any role



10. Assign the role to your user. Log off and on to the Portal and your new iView should look like the following image



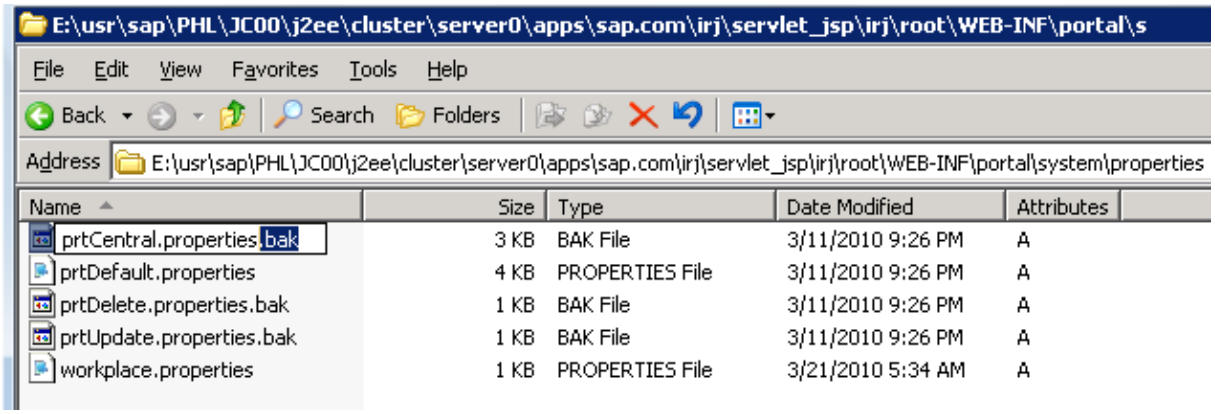
5. Appendix

Appendix A - Cache disabling during development

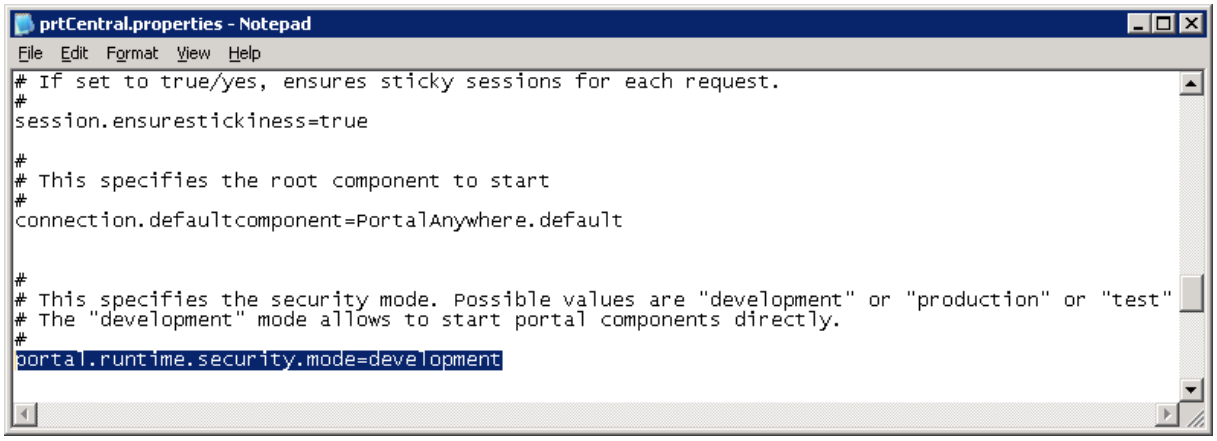
After editing resource files (js, css, jsp, etc.) the changes are not instantly reflected in the http responses because of the HTTP caching.

Flag:

1. Go to the <Portal installation drive>:\usr\sap\<SID>\JC<inst_nr>\j2ee\cluster\server0\apps\sap.com\irj\servlet_jsp\irj\root\WEB-INF\portal\system\properties\prtCentral.properties.bak
2. Remove the .bak extension from the file name.



3. Edit: portal.runtime.security.mode=development

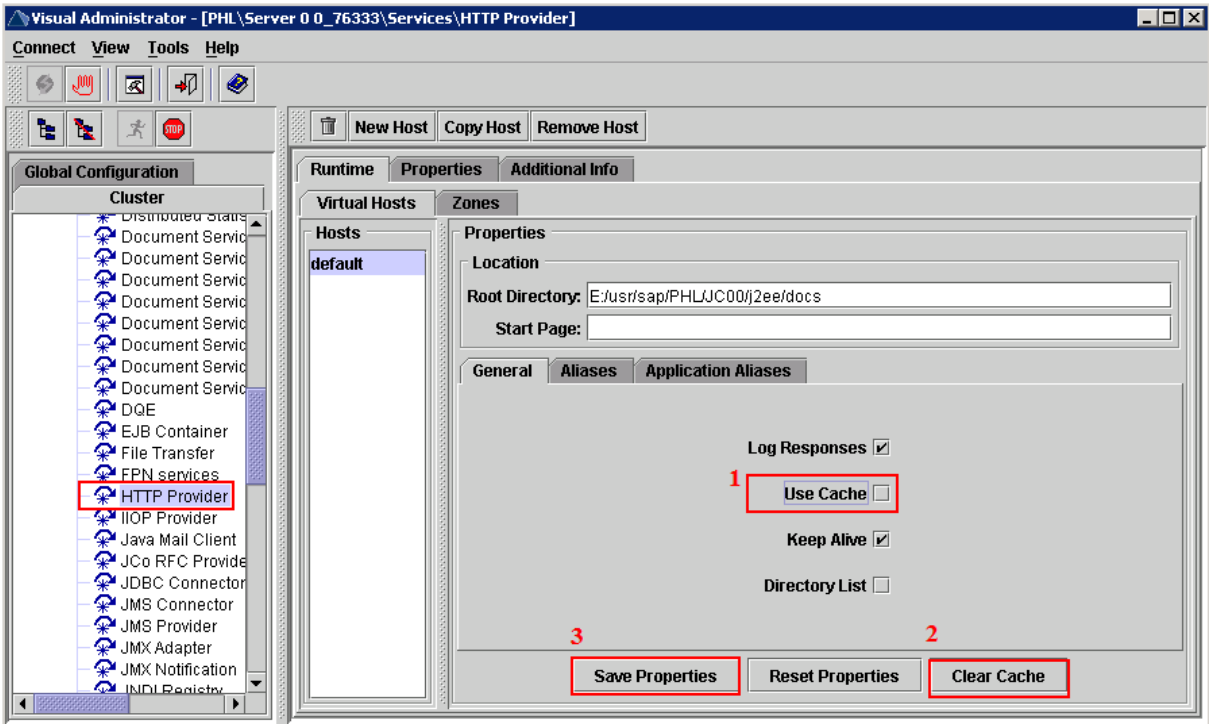


4. Save the changes

Java HTTP Provider

1. Open the Visual Administrator: <Portal installation drive>:\usr\sap\<SID>\JC<inst_nr>\j2ee\admin
2. Go to Cluster → Services → HTTP Provider → Runtime → Virtual Hosts
   a. Uncheck "Use Cache"
   b. "Clear Cache"

c. "Save Properties"



d. Restart the server

www.sdn.sap.com/irj/sdn/howtoguides