

# NetBeans 6.5.1, GlassFish v 2.1, Web Space Server 10 Creating a Healthcare Facility JSR286-compliant Portlet

[Michael.Czapski@sun.com](mailto:Michael.Czapski@sun.com)

June 2009

## Abstract

SOA is sometimes shown as a series of 4 layers with Presentation Layer (SOA 1) at the top. Each layer consumes services exposed by layers under it. Service interfaces are described using WSDL. Web Services are the means to decouple functional layers. Functionality in one layer can be swapped in and out without disturbing other layers. Presentation layer is often implemented as JSR-168-compliant or JSR-286-compliant Portlets, exposed through a standards-based Portal.

Here I will walk through development of a JSR-286-compliant Visual Web JSF Portlet, which will use a Web Service as a data provider. I use the NetBeans 6.5.1 IDE, part of the GlassFish ESB v2.1 installation, the Portal Pack 3.0.1 NetBeans Plugin and the JSF Portal Bridge provided by the Web Space Server 10. The Portlet will use JSF components provided by Project Woodstock. The technology will be introduced in a practical manner.

This is not a tutorial on JavaServer Faces, Visual Web JSF, Project Woodstock or Portlet development.

## Introduction

In some views SOA is represented as a series of 4 layers: Presentation Layer (SOA 1), Business Process Layer (SOA 2), Business Service Layer (SOA 3) and Technical Layer (SOA 4). Typically each layer higher up in the hierarchy consumes services exposed by the layer under it. So the Presentation Layer would consume services provided by the Business Process or Business Service Layers. Service interfaces are described using Web Services Description Language (WSDL), sheltering service consumers from details of service implementation. Web Services are seen as the technical means to implement the decoupled functional layers in a SOA development. Decoupling allows implementations of business functionality at different layers to be swapped in and out without disturbing other layers in the stack. The SOA 1, Presentation Layer, is often implemented as JSR-168-compliant or JSR-286-complaint Portlets, exposed through a standards-based Portal.

The business idea is that patients are looked after in various healthcare facilities. Applications need to allow selection of a facility and to access facility details for display to human operators. A relational holds details of facilities which are a part of the healthcare enterprise. Facility list and details are available through a web service. This web service will be used to construct the JSR-286-compliant Portlet that provides a user view into the facilities and facility details. This Portlet will be deployed to the Sun FOSS Web Space Server 10 Portal.

Previous documents in this series, "GlassFish ESB v 2.1 - Creating a Healthcare Facility Web Service Provider", at [http://blogs.sun.com/javacapsfieldtech/entry/glassfish\\_esb\\_v\\_2\\_1](http://blogs.sun.com/javacapsfieldtech/entry/glassfish_esb_v_2_1) and "NetBeans 6.5.1 and GlassFish v 2.1 - Creating a Healthcare Facility Visual Web Application", [http://blogs.sun.com/javacapsfieldtech/entry/netbeans\\_6\\_5\\_1\\_and](http://blogs.sun.com/javacapsfieldtech/entry/netbeans_6_5_1_and), walked the reader through the process of implementing a GlassFish ESB v2.1-

based web service which returns facility list and facility details, and a Visual Web JSF Web Application which used that Web Service to display facility list and details.

In this document I will walk through the process of developing a JSR-286-compliant Visual Web JSF Portlet, deployed to the Sun Web Space Server 10 Portal, which will use the Web Service as a data provider. We will use the NetBeans 6.5.1 IDE, which comes as part of the GlassFish ESB v2.1 installation, the Portal Pack 3.0.1 NetBeans Plugin and the JSF Portal Bridge infrastructure provided by the Web Space Server 10. The Portlet will be implemented as a Visual Web JavaServer Faces Portlet using JSF components provided by Project Woodstock. The Portlet will introduce the technology in a practical manner and show how a web service can be used as a data provider, decoupling the web application from the data stores and specifics of data provision.

Note that this document is not a tutorial on JavaServer Faces, Visual Web JSF, Project Woodstock components or Portlet development. Note also that all the components and technologies used are either distributed as part of the NetBeans 6.5, as part of the GlassFish ESB v2.1, as part of the Web Space Server 10 or are readily pluggable into the NetBeans IDE. All are free and open source.

## Prerequisites

To work through this material certain pre-requisites have to be met.

The GlassFish ESB v2.1 installation must be available. Install GlassFishESB v2.1, <https://open-esb.dev.java.net/Downloads.html>, following standard installation steps detailed at the OpenESB site.

Sun Web Space Server 10 functionality must be added to the GlassFish ESB v 2.1 installation. This is discussed in "Adding Sun WebSpace Server 10 Portal Server functionality to the GlassFish ESB v2.1 Installation", [http://blogs.sun.com/javacapsfieldtech/entry/adding\\_sun\\_webpace\\_server\\_10](http://blogs.sun.com/javacapsfieldtech/entry/adding_sun_webpace_server_10). This implies availability of the MySQL RDBMS installation, also discussed in that document.

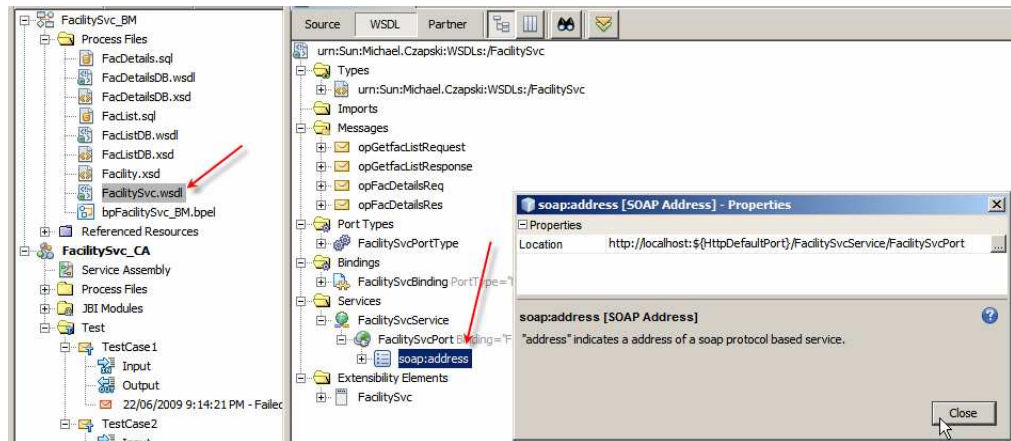
To allow web services and portlets to coexist in the same GlassFish Application Server instance the infrastructure must be configured with an alternate portal servlet context root. This is discussed in "Making Web Space Server And Web Services Play Nicely In A Single Instance Of The Glassfish Application Server", [http://blogs.sun.com/javacapsfieldtech/entry/making\\_web\\_space\\_server\\_and](http://blogs.sun.com/javacapsfieldtech/entry/making_web_space_server_and).

The multi-operation web service, which provides faculty list and facility details, must be developed and deployed. This is discussed in "GlassFish ESB v 2.1 - Creating a Healthcare Facility Web Service Provider", at [http://blogs.sun.com/javacapsfieldtech/entry/glassfish\\_esb\\_v\\_2\\_1](http://blogs.sun.com/javacapsfieldtech/entry/glassfish_esb_v_2_1).

## Determining Service Endpoint and WSDL Location

This document assumes that the portlet will use the web service developed elsewhere as a data provider. To make it possible we need to know the endpoint location of the service and the location of the WSDL. This information is available if one knows where to look.

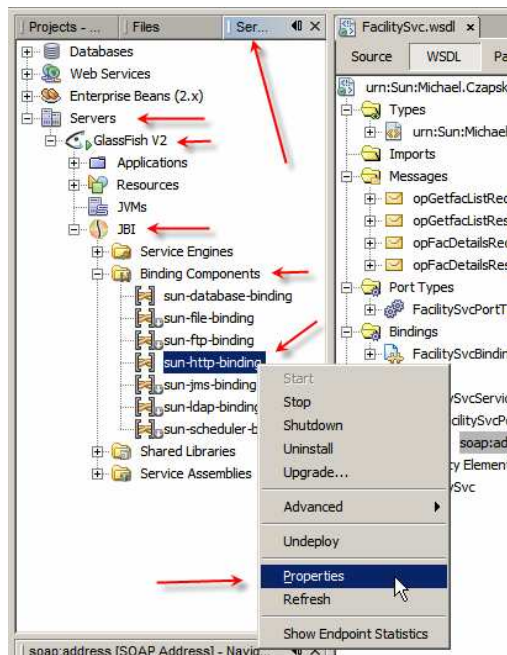
Let's open the FacilitySvc.wsdl document in project FacilitySvc\_BM and inspect the properties of the soap:address node under the FacilitySvcService node.



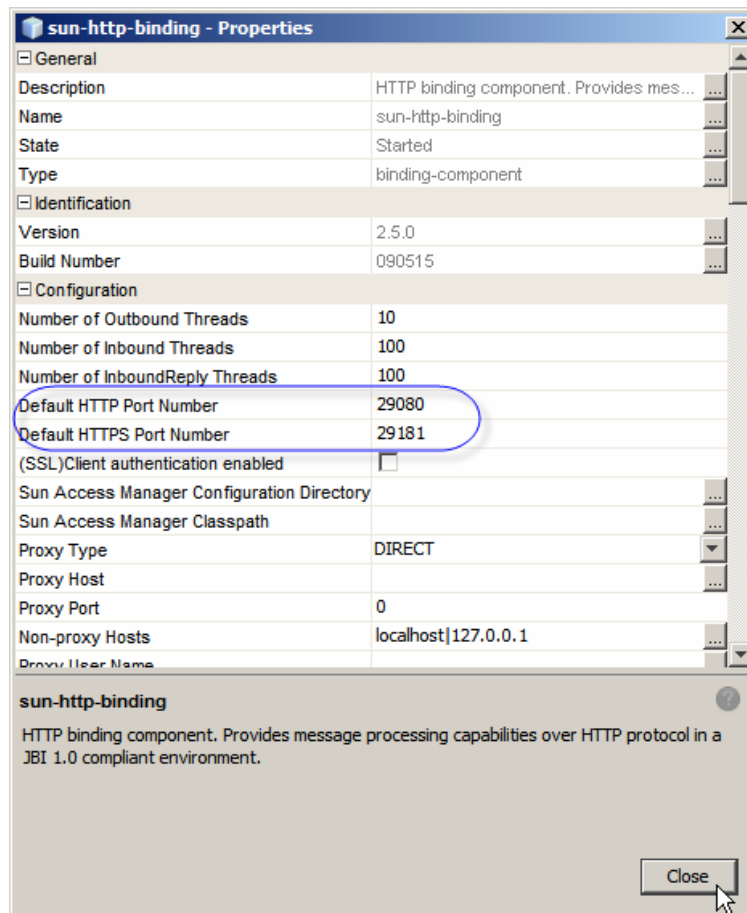
Note the Location property value:

`http://localhost:${HttpDefaultPort}/FacilitySvcService/FacilitySvcPort`

The HttpDefaultPort is the port which SOAP/HTTP BCs use. At CA deployment time this variable gets replaced with the actual port. To find out what this port is let's switch to the Services tab in Netbeans, expand Servers, expand JBI, expand Binding Components, right-click sun-http-binding and choose Properties.



Observe the Default HTTP Port Number property value. For my installation this will be 29080. For a default installation it will be 9080. It can be changed.



So, the final service endpoint URL, from the soap:address Location property earlier, will be:

```
http://localhost:29080/FacilitySvcService/FacilitySvcPort
```

This URL is the service location.

The WSDL for this service can be accessed, using the regular convention, at:

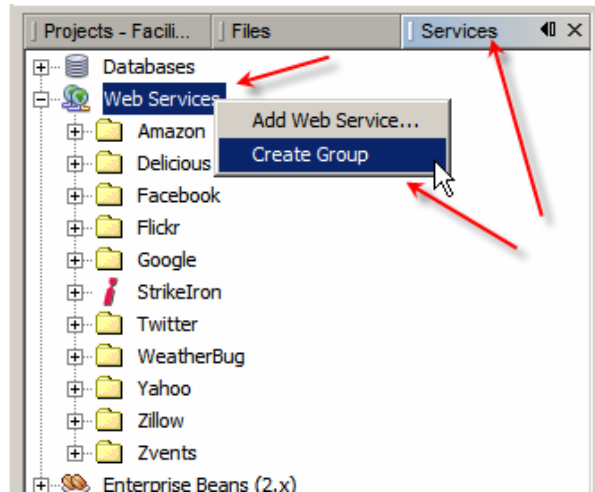
```
http://localhost:29080/FacilitySvcService/FacilitySvcPort?WSDL
```

With this knowledge we can use the service in related projects.

## Making Service Available for use in a Portlet

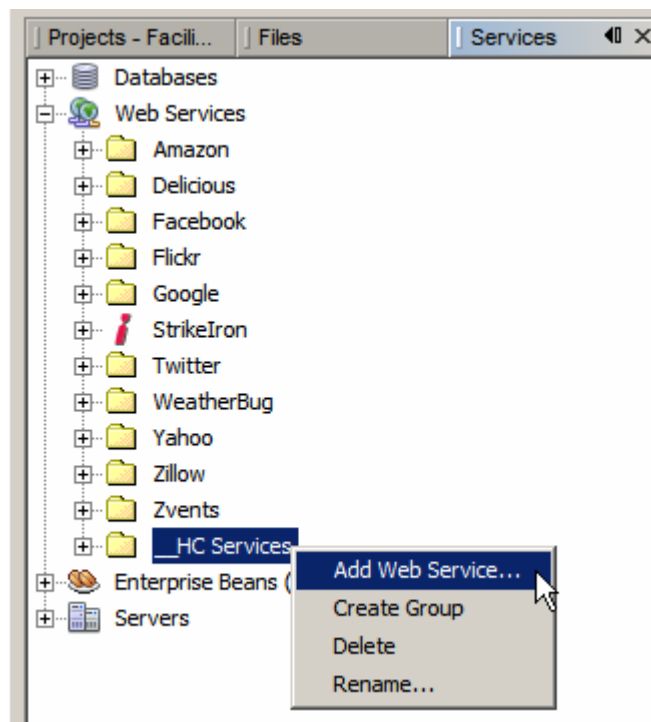
If you created a Web Application, discussed in “NetBeans 6.5.1 and GlassFish v 2.1 – Creating a Healthcare Facility Visual Web Application”, [http://blogs.sun.com/javacapsfi/eldtech/entry/netbeans\\_6\\_5\\_1\\_and](http://blogs.sun.com/javacapsfi/eldtech/entry/netbeans_6_5_1_and), the web service reference will already be available for use. You can skip this section. If not, follow this section to get the service in the right place in NetBeans.

To make a web service available for use as a data provider in a web application, or a Portlet, we must “introduce” it to the NetBeans IDE. Switch to the Services Tab, right-click on the Web Services node and choose Create Group.

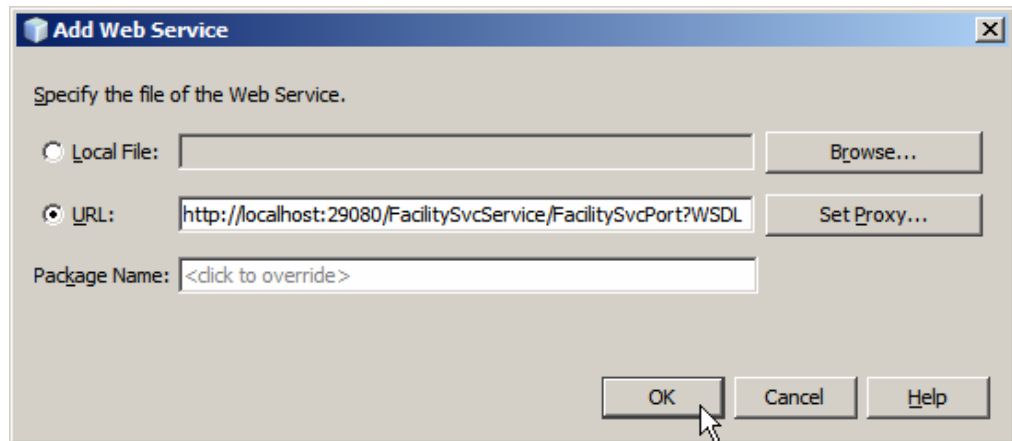


Name this group “\_\_HC Services”.

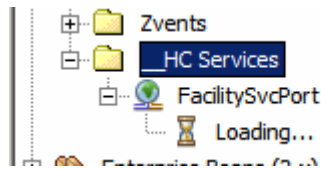
Right-click on the \_\_HC Services node and choose Add Web Service...



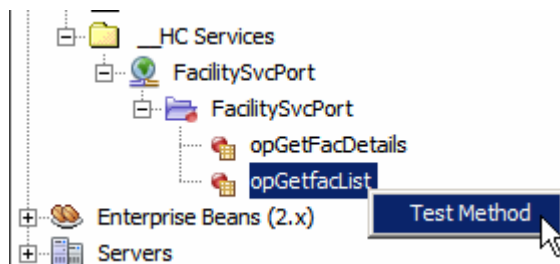
Enter service WSDL URL and click OK. The service WSDL URL for me will be <http://localhost:29080/FacilitySvcService/FacilitySvcPort?WSDL>



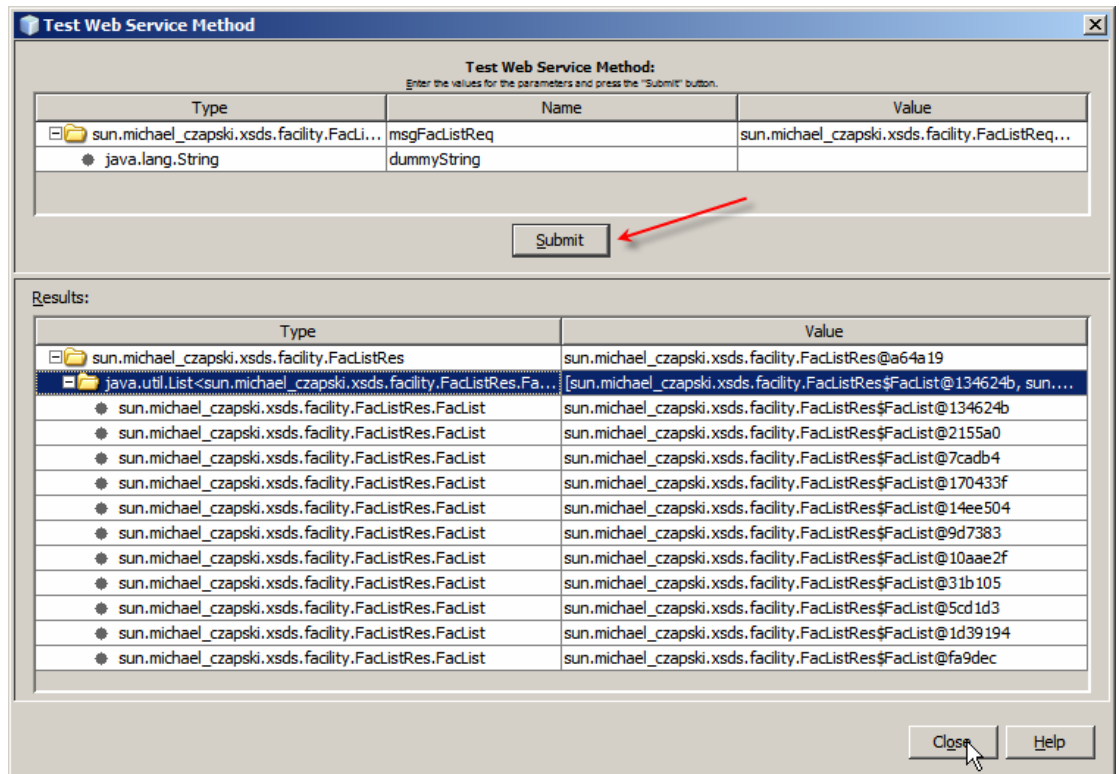
Expand \_\_HC Services -> FacilitySvcPort, wait for the WSDL to be loaded and appropriate classes to be generated, compiled and packaged.



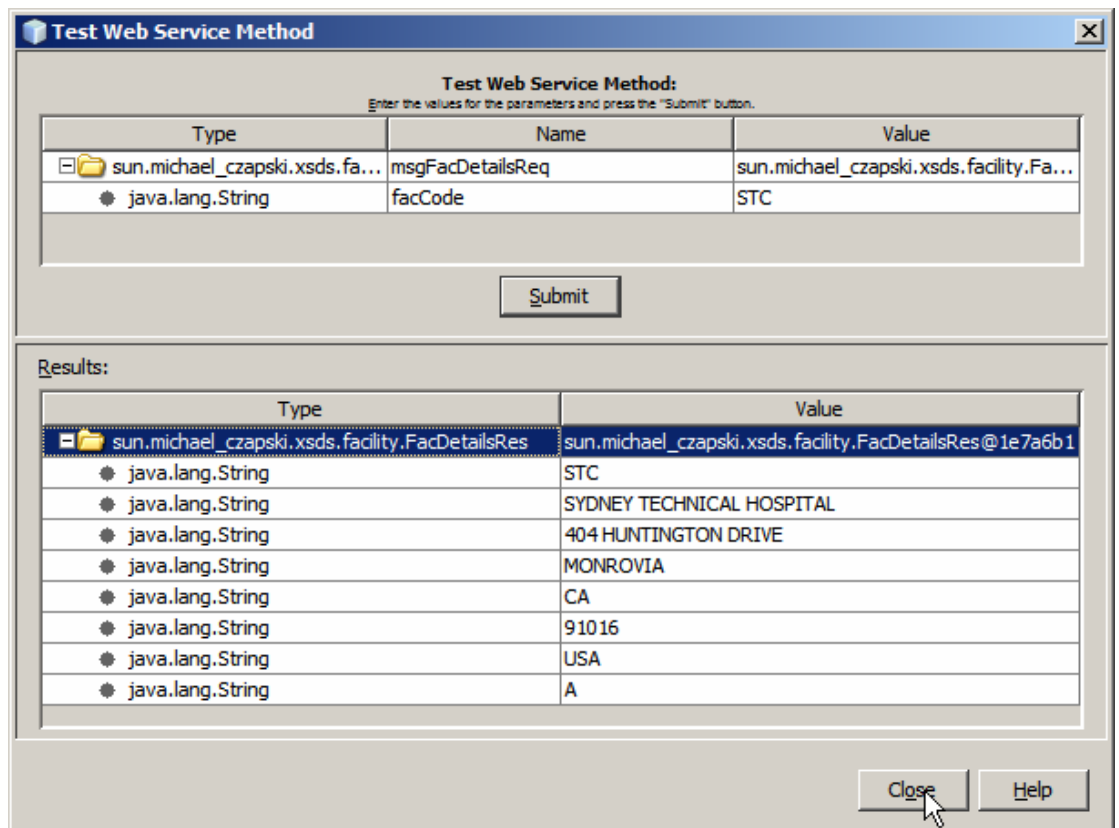
Expand the node tree all the way to the operations, right-click on opFacList and choose Test Method.



Click Submit button and, when execution is completed, inspect the results.



Now execute the test for the opGetFacDetails, providing facility code of STC.

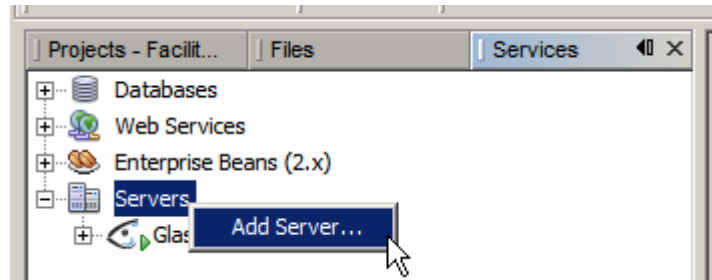


This is yet another method of testing web services in NetBeans. We have valid references to service operations, ready to be used in web applications and portlets.

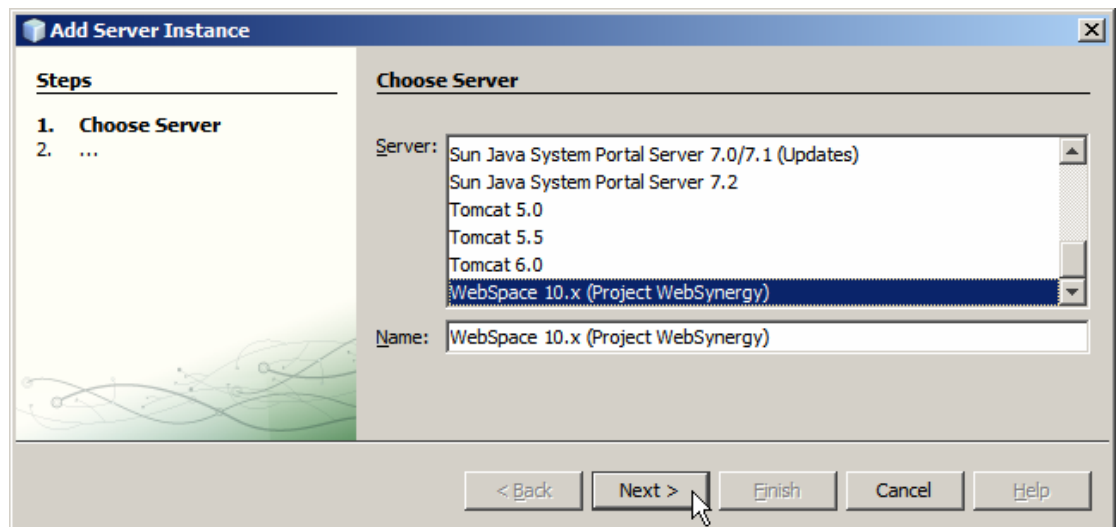
## Add Web Space Server 10 to the list of Servers

Before we get to create a Portlet we must add the Web Space Server to the list of Servers in the Services Tab.

Switch to the Services Tab in NetBeans IDE, right-click Servers and choose Add Server.

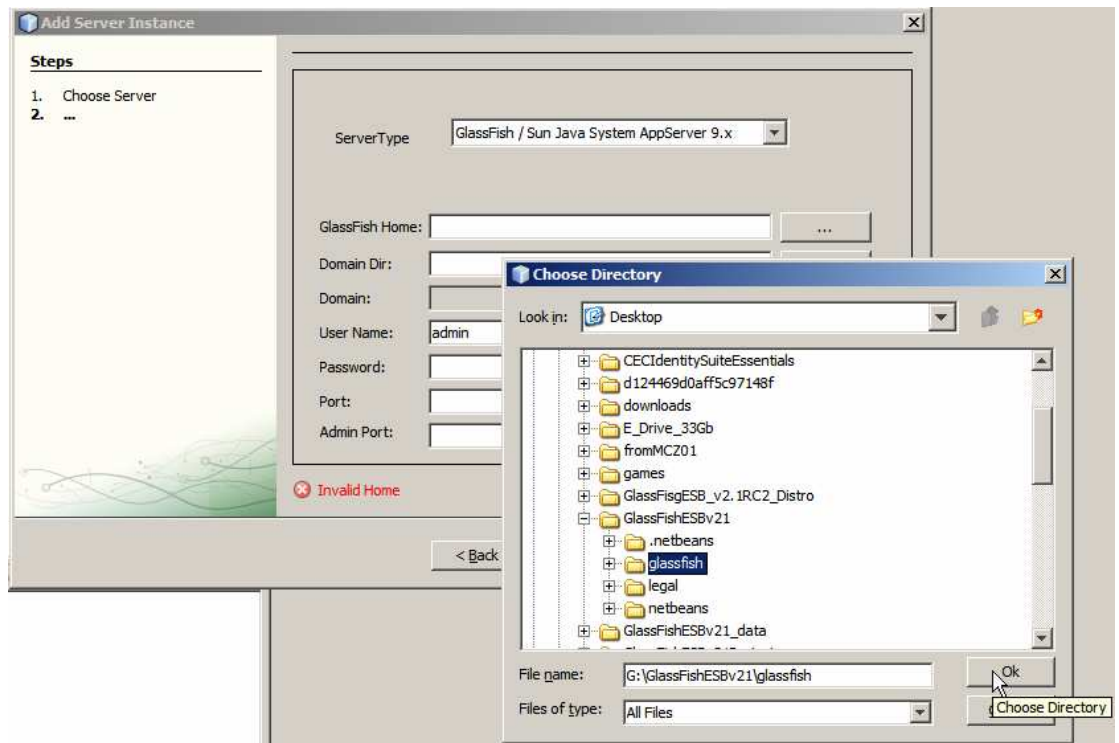


Choose WebSpace 10.x (Project WebSynergy) and click Next.

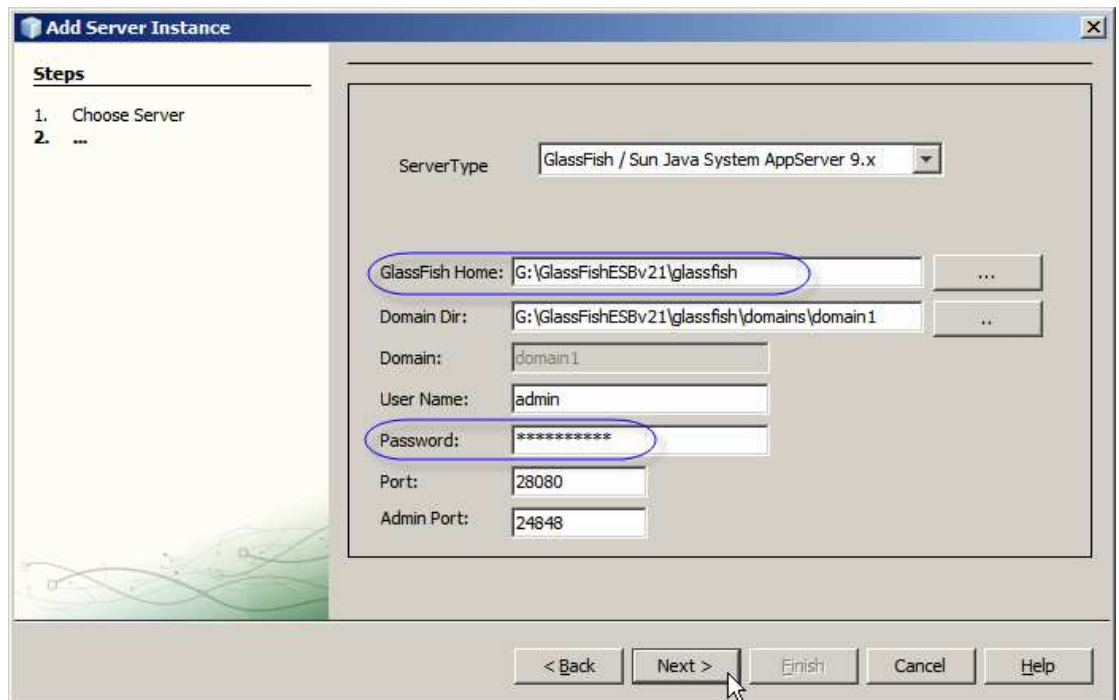


Locate the GlassFish Home directory (this will be the GlassFish instance to which you added WebSpace Server functionality).

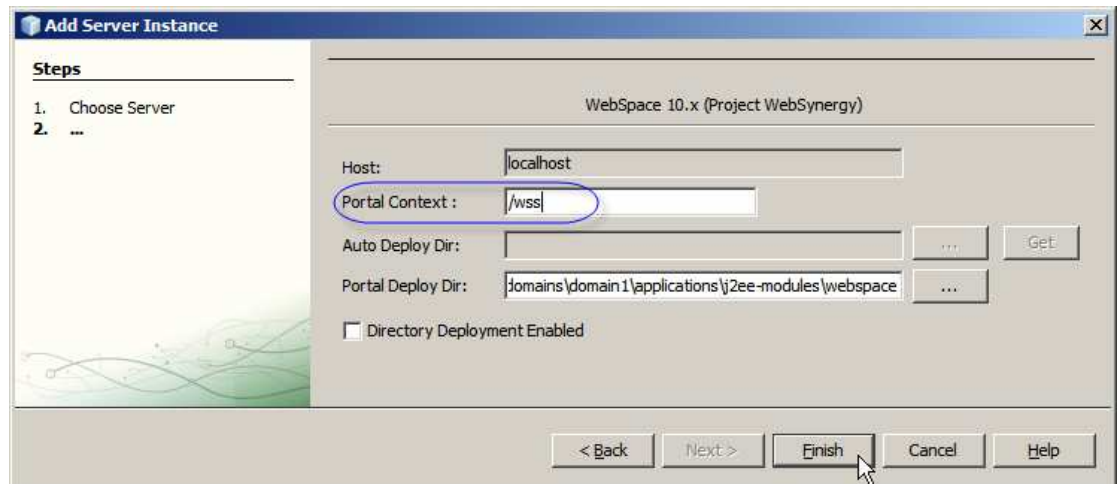




Once you have chosen the right directory all other details will be filled in, except admin password. Provide the admin password and click Next.



Recall that in one of the pre-requisites we changed the servlet context root for the portal. We must reflect this in the service we are adding. Change the Portal Context from "/" to "/wss", or whatever you used for the portal context. Finish the wizard.



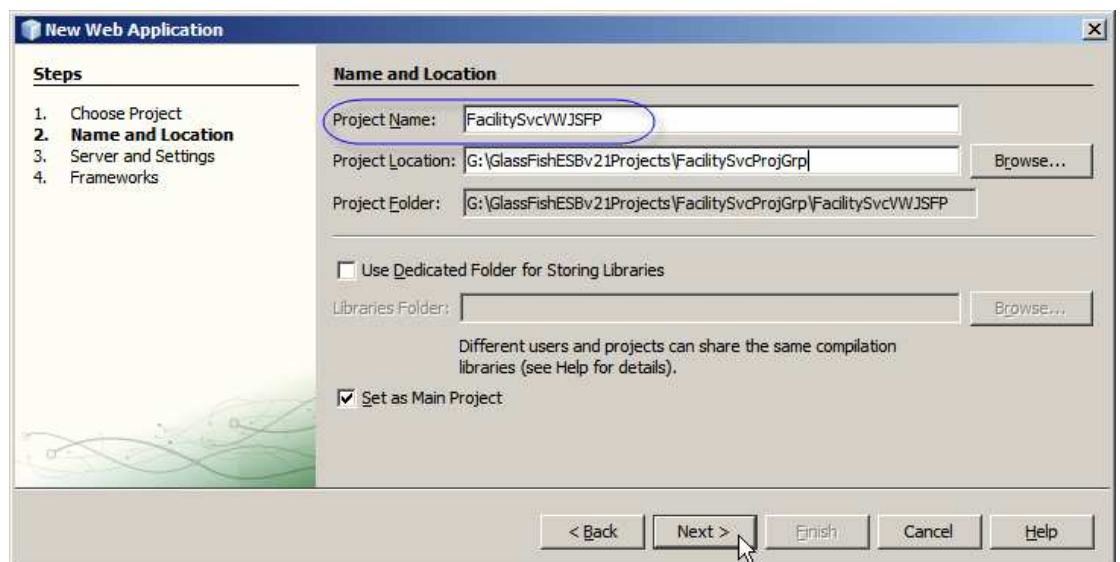
This adds the WebSpace 10.x server to the list. This server is where we will be deploying the portlet we will create shortly.

## Create Visual Web JSF Portlet

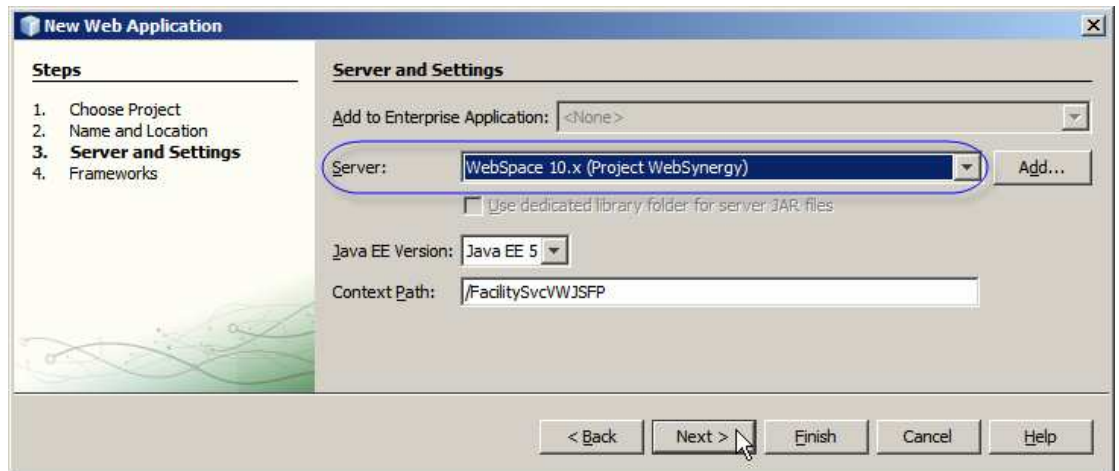
Of the variety of methods available in NetBeans to develop portlets I have chosen the Visual Web JSF Portlet method. The reason is that it is visual, easy and quick, if one knows what one is doing. I know enough to be dangerous but not enough to help you out if you get into trouble with JSF, Visual Web JSF or Portlet design. There is a plethora of material on the Internet on the different aspects of these technologies. Do what I do – research and experiment. This is a practical cookbook for the specific portlet I built. Feel free to learn the technology and ad-lib.

Switch back to the Projects tab.

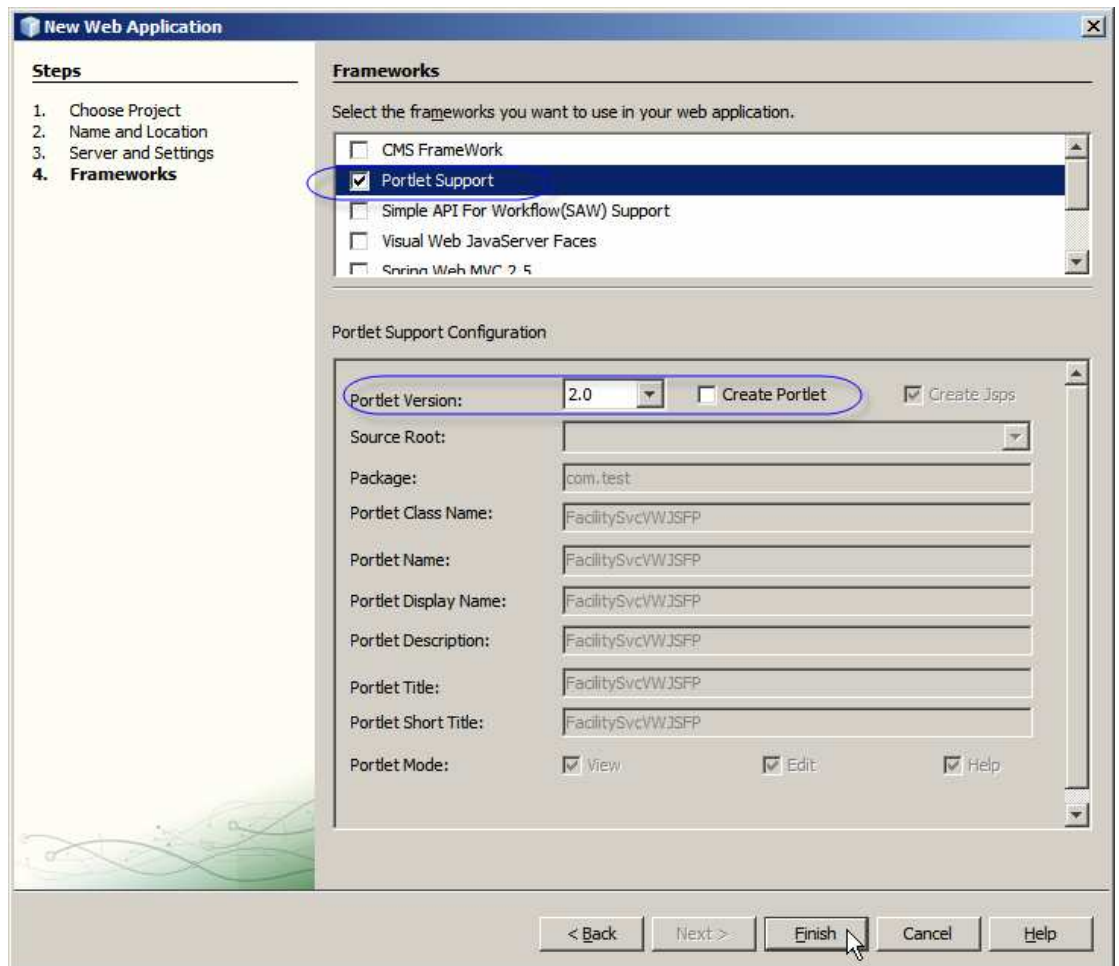
Create a New Project -> Java Web -> Web Application, FacilitySvcVWJSFP.



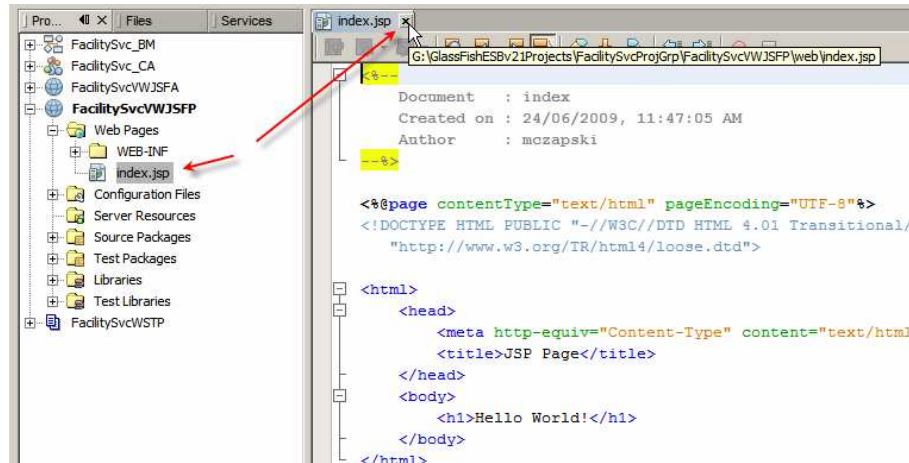
Choose WebSpace 10.x as Server and accept defaults for other settings in the panel.



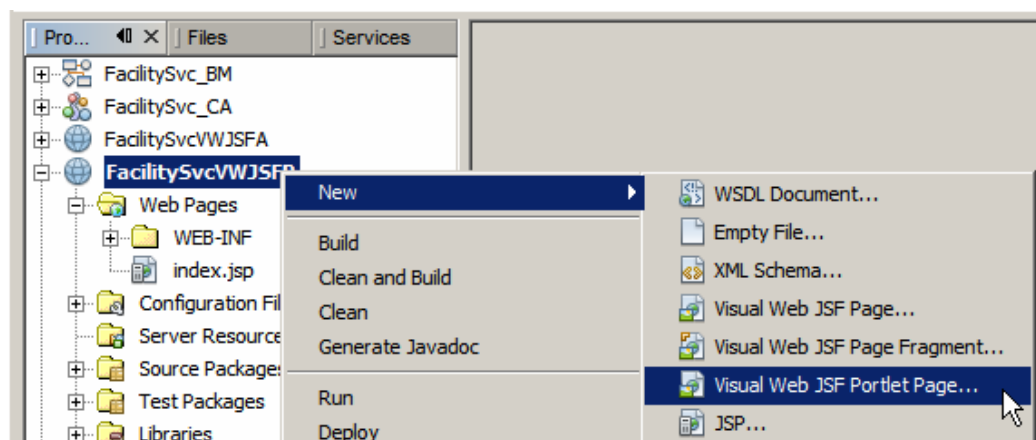
Choose portlet support but do not create a portlet.



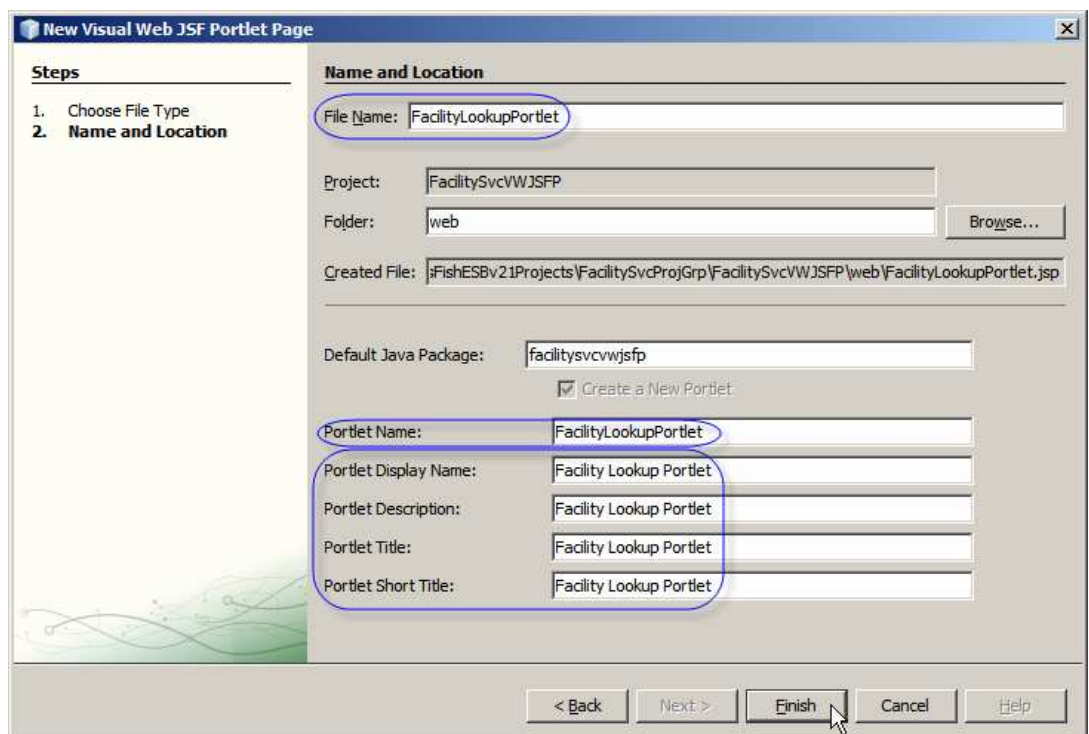
Close index.jsp – we will not be using it.



Right-click on the project name and create New -> Visual Web JSF Portlet Page ...

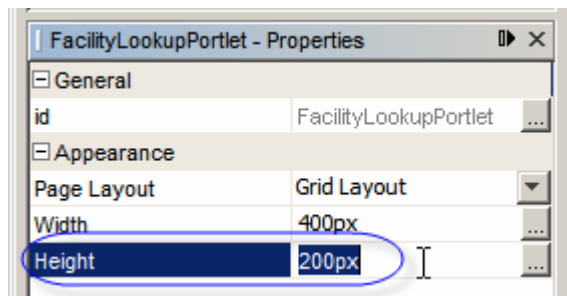


Name the page FacilityLookupPortlet, provide Portlet Name (which must be an Identifier) and modify description and title text if you feel so inclined.

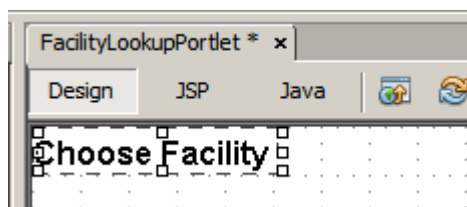
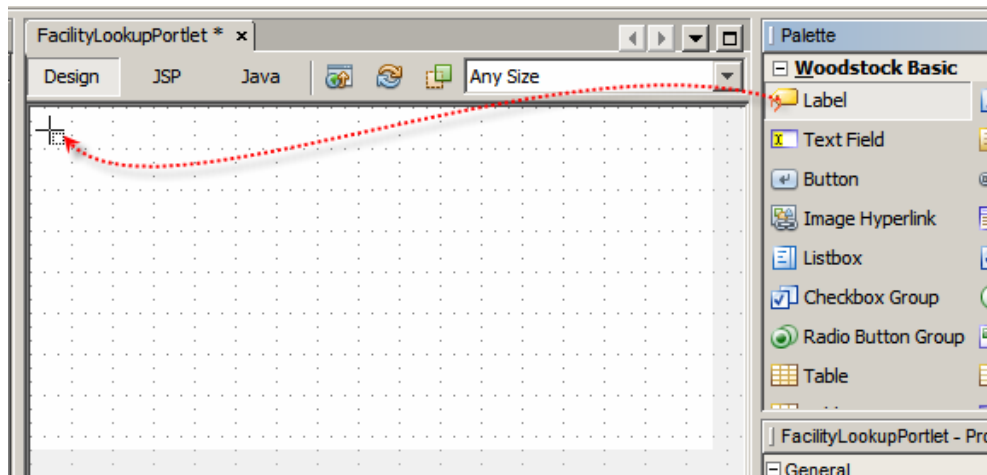


Finish.

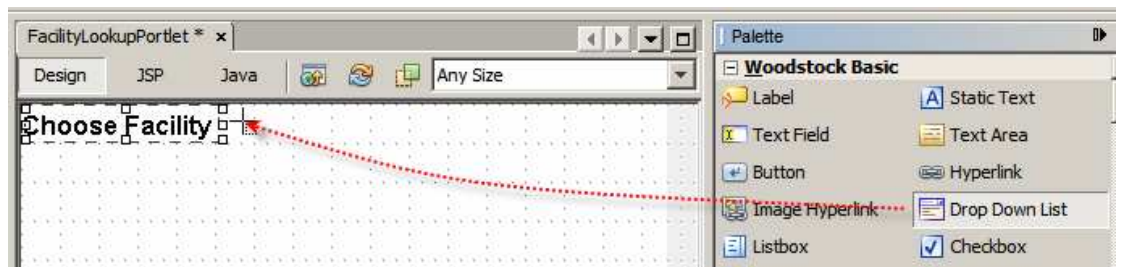
Change the Height property of the portlet to 200px. We can get away with less still. Experiment.

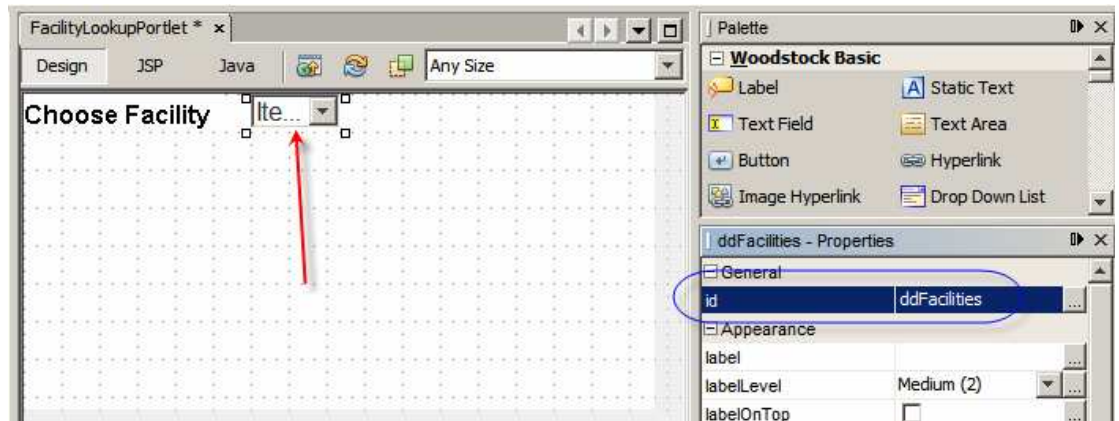


Drag the Woodstock Basic -> Label component onto the canvas and change its text property to read "Choose Facility".

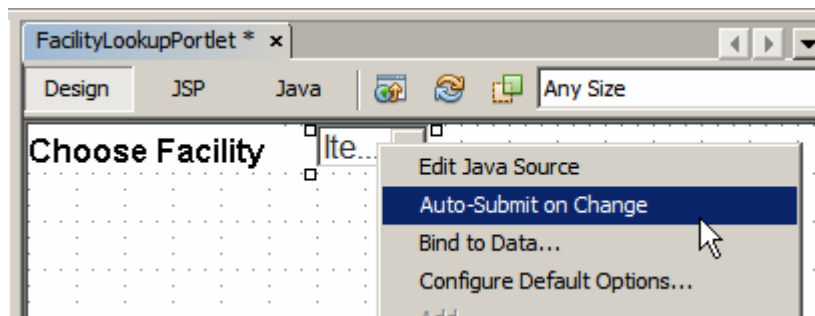


Drag Woodstock Basic -> Drop Down List component to the canvas to the right of the label. Change the General -> Id property of the component to ddFacilities.



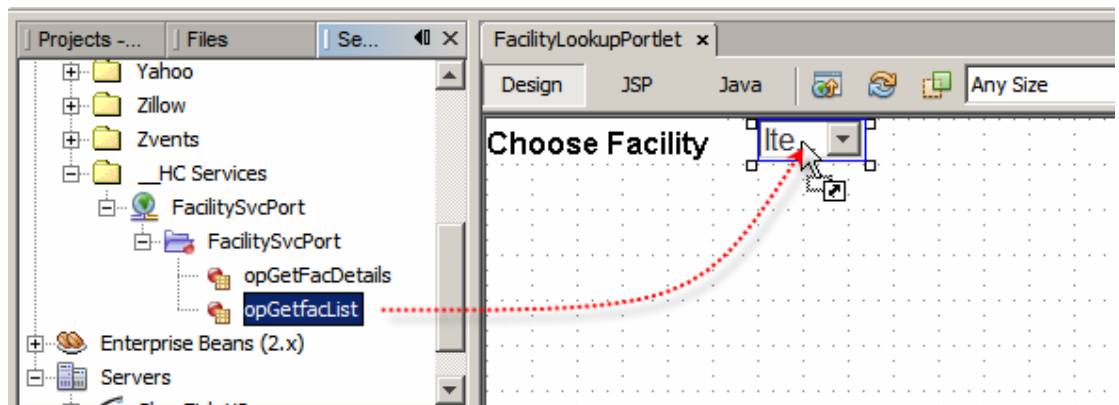


Right-click on the drop down component and choose Auto-Submit On Change.

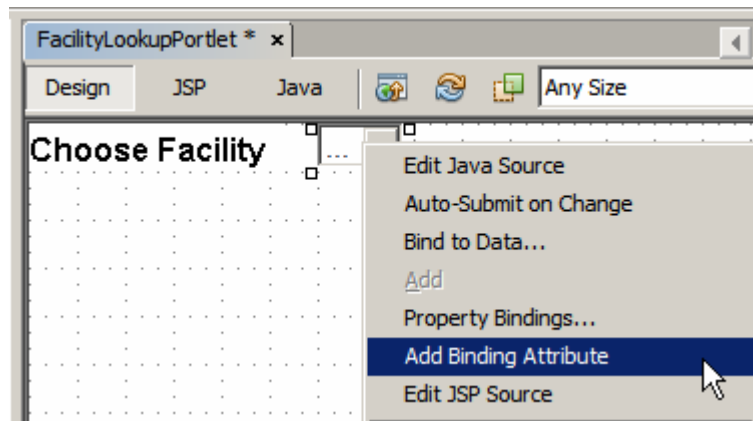


Save the project.

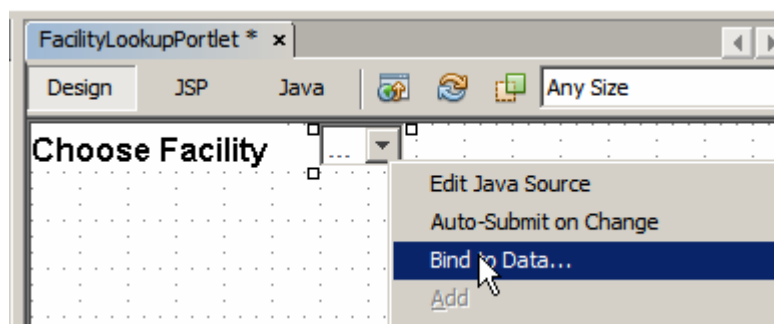
Switch from the Project view to Services view, locate the opGetFacList operation on the web service we added to NetBeans view of web services, drag it onto the FacilityLookupPortlet JSF page canvas and drop it right over the top of the drop down component.



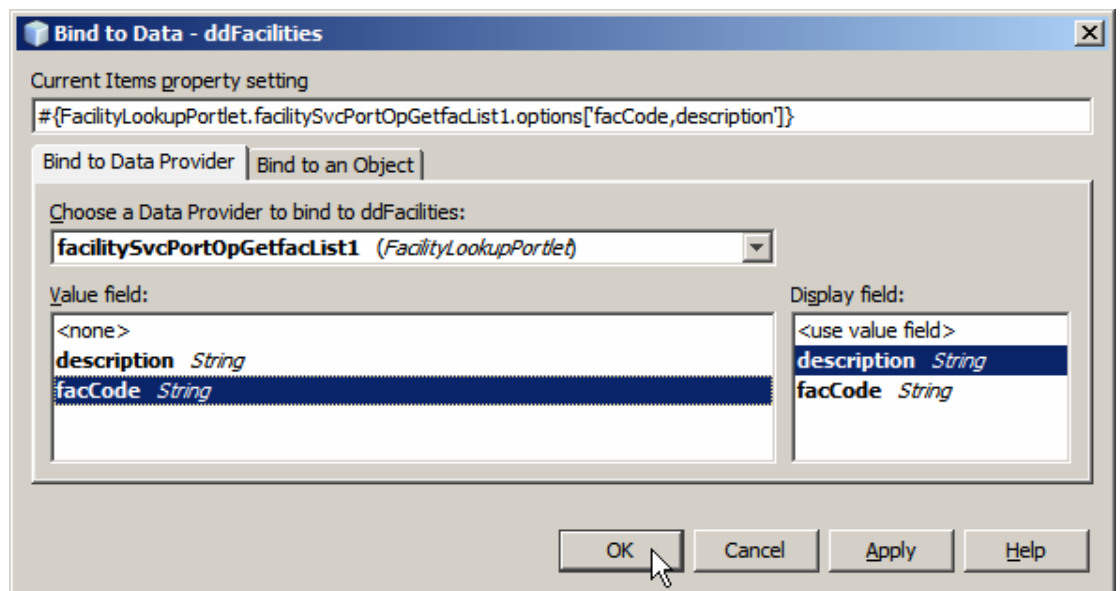
Right-click the drop down component and choose Add Binding Attribute.



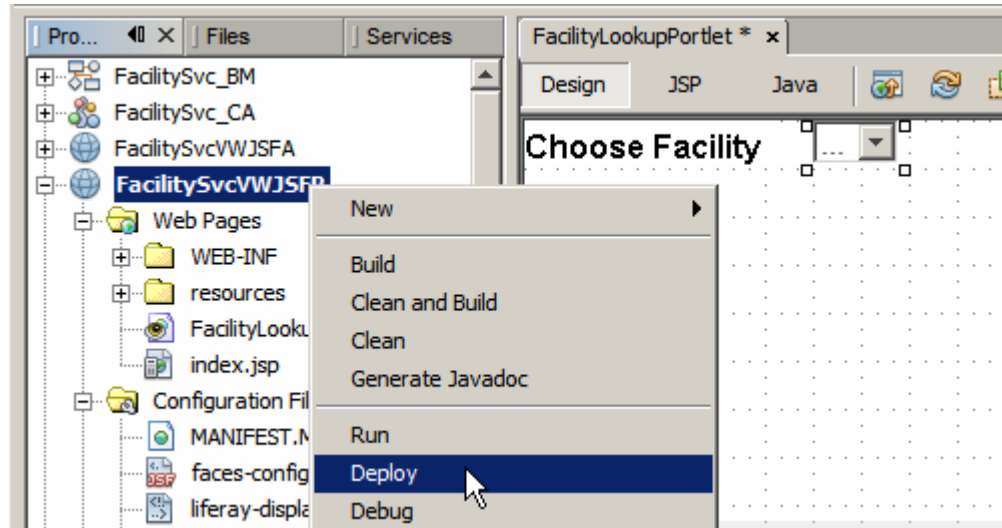
Right-click the drop down component and choose Bind to Data.



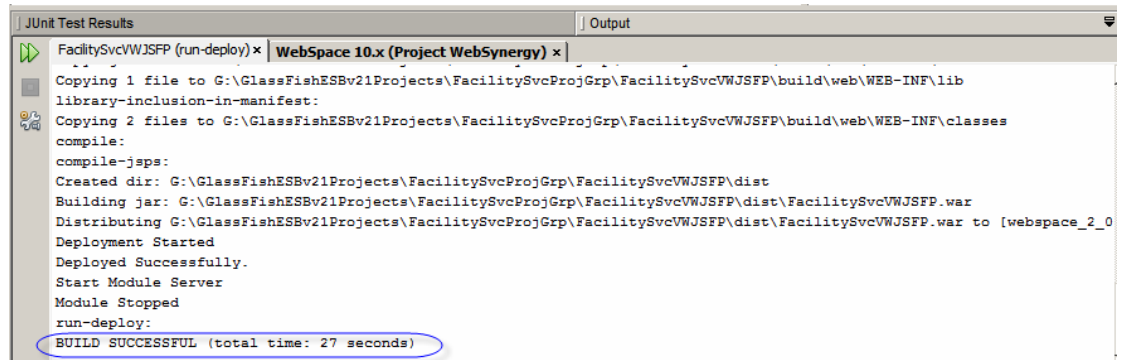
Make sure facCode is selected in the Value field list and description is selected in Display field list, the click OK.



Switch to the project view, right-click the name of the project and choose Deploy.

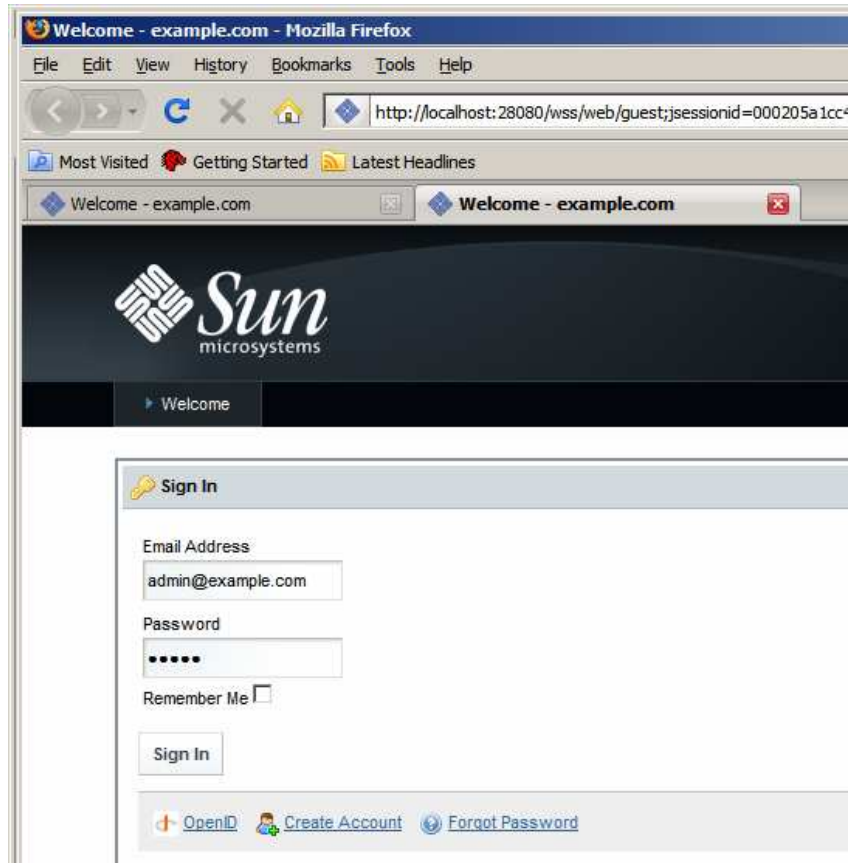


Once the deployment completes successfully you will see messages similar to the following.



Start your favorite web browser, for example Mozilla Firefox, connect to <http://localhost:28080/wss> (change port if you use a different one, change portal context if you use different one) and sign in as a portal user. If you have no other user created there is always [admin@example.com](mailto:admin@example.com) with password admin.

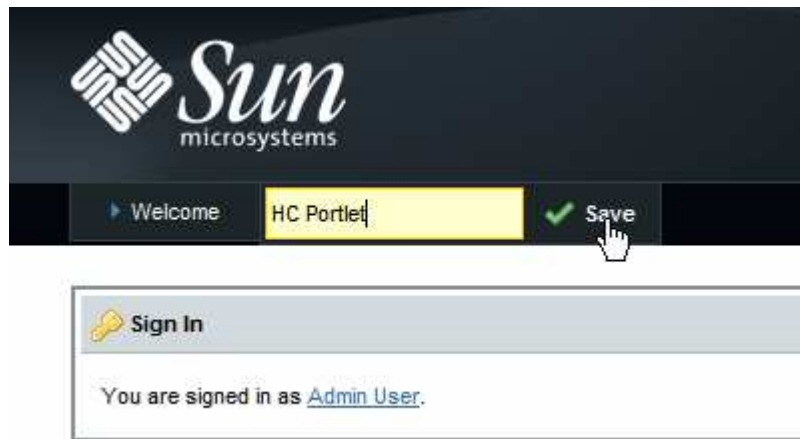




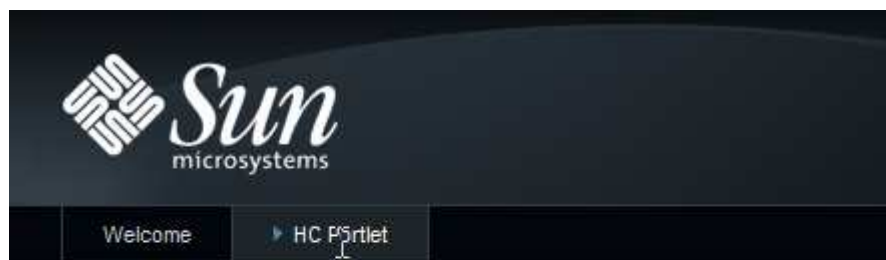
Click on Add Page.



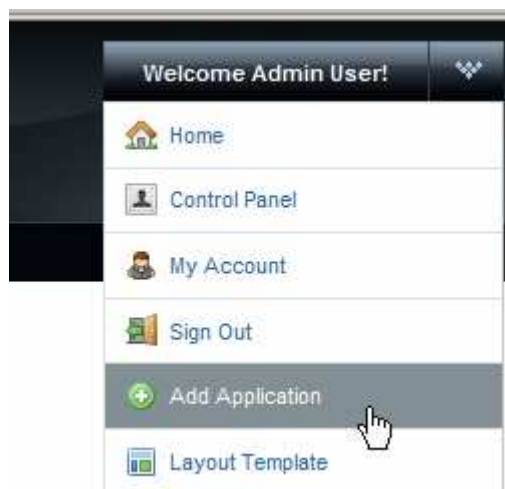
Name the page "HC Portlet" and click Save.



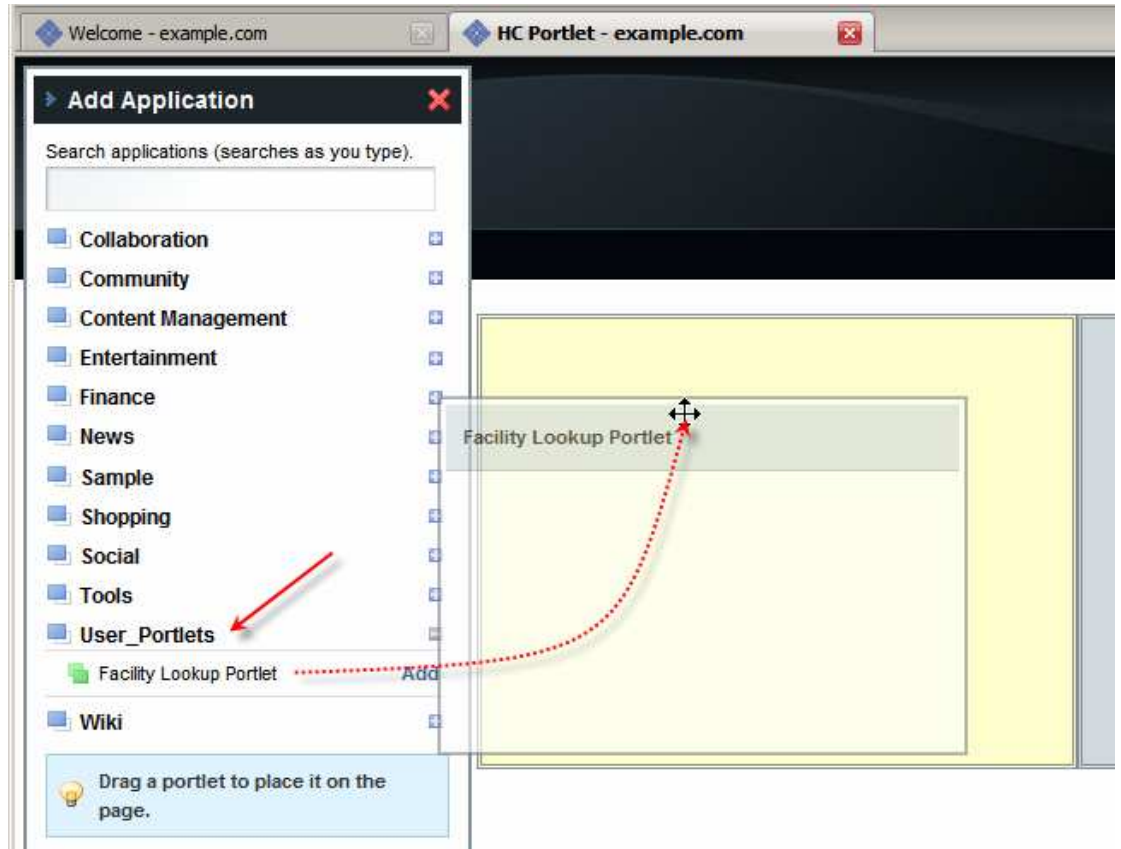
Click on the HC Portlet tab to switch to that page.



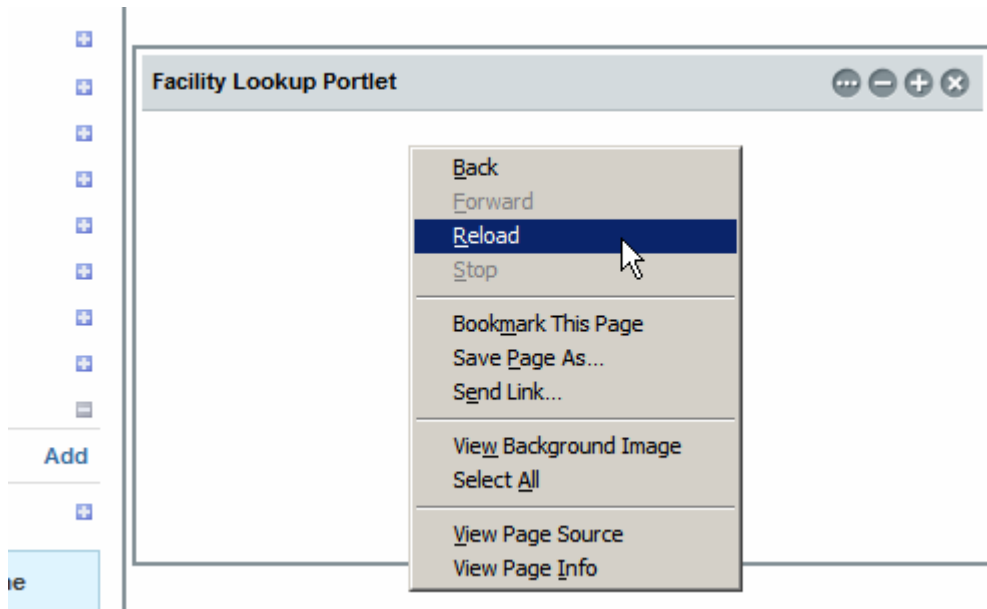
Pull down the menu and choose Add Application



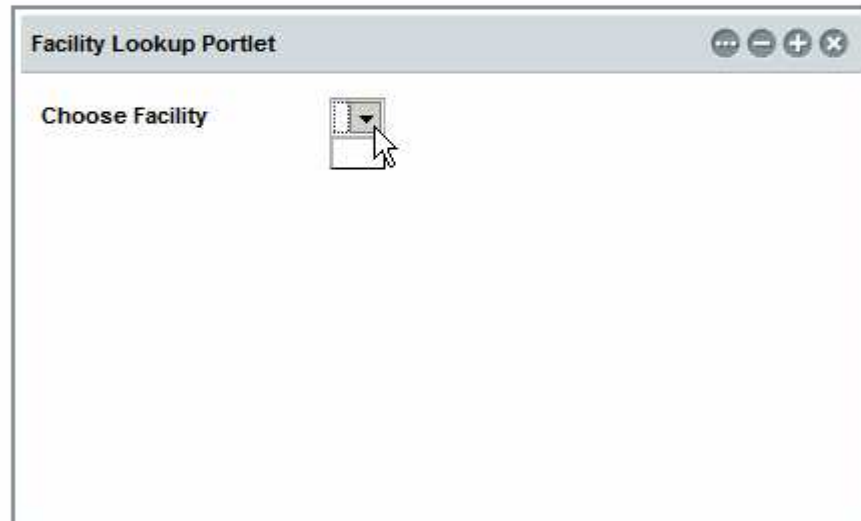
Expand User Portlets menu item and drag the Facility Lookup Portlet to the canvas.



You may need to right-click on the portlet and choose Reload to get the portlet to display properly.

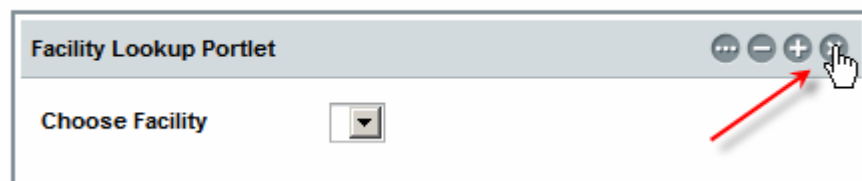


Our portlet is now visible with no facilities in the drop down.

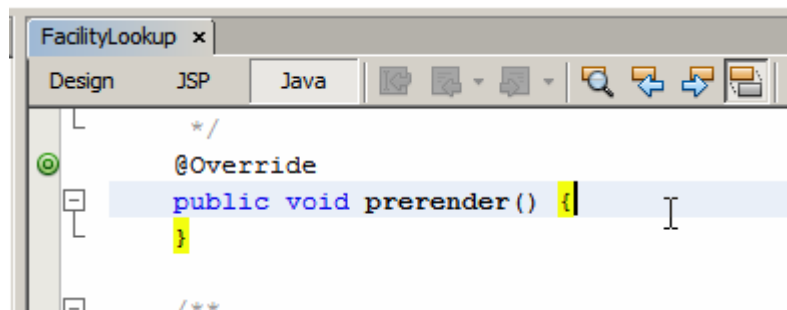


Note that the drop-down is empty. We need to initialize the call to the web service by providing dummy string value.

Remove the portlet so we can add it again later, once we modified it.



Switch to NetBeans, switch to Java Tab and scroll down to the prerender() method.

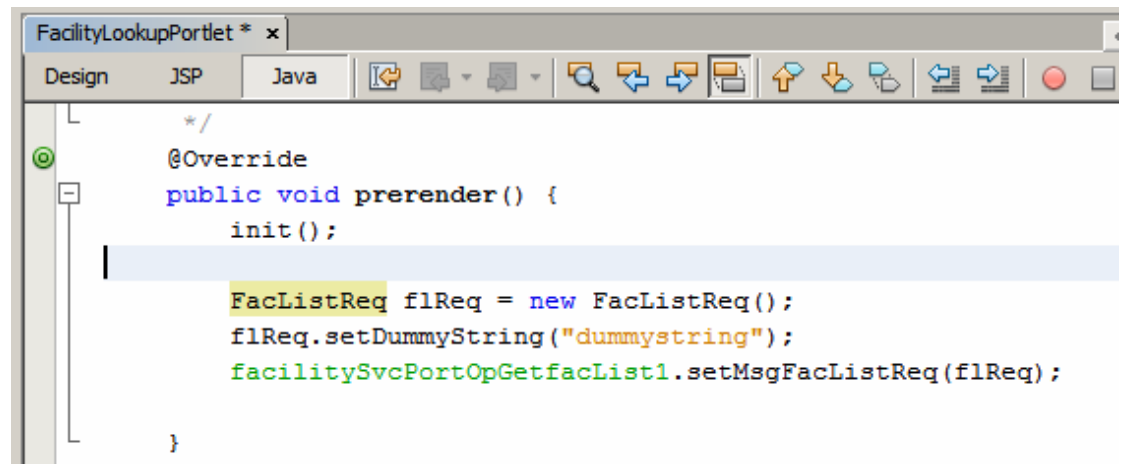


Into the prerender method insert the following code.

```
init();  
  
FacListReq flReq = new FacListReq();
```

```
flReq.setDummyString("dummystring");
facilitySvcPortOpGetfacList1.setMsgFacListReq(flReq);
```

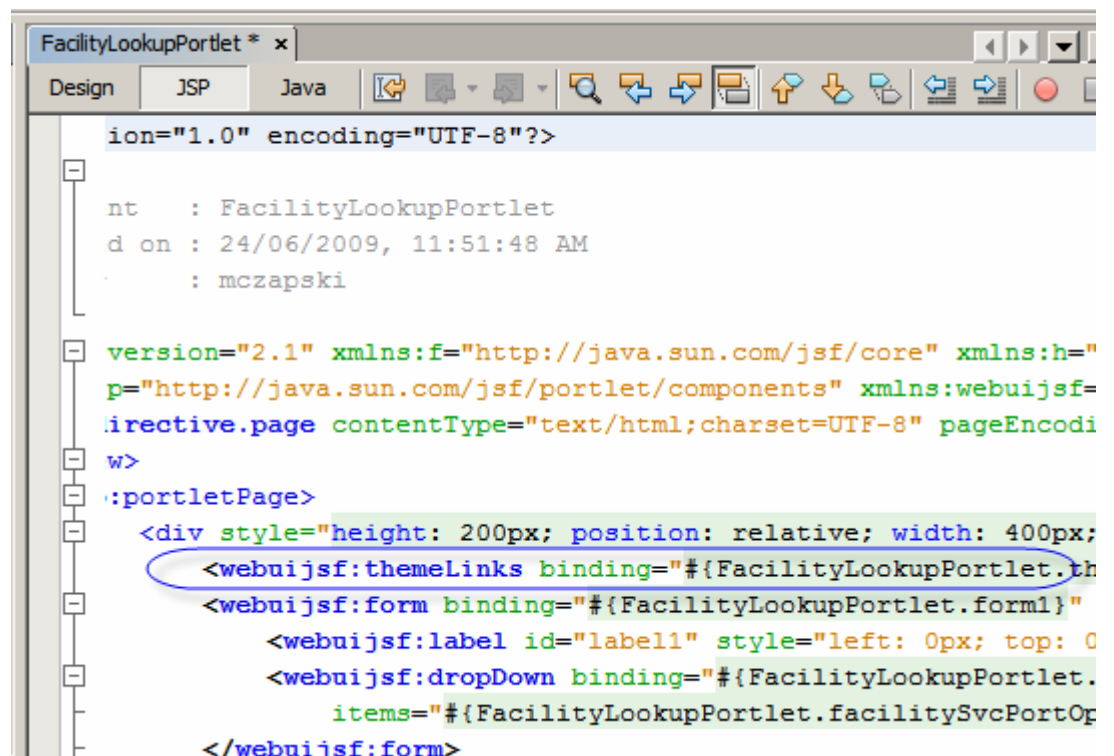
Right-click anywhere in the source window and choose Fix Imports, then choose Format. This will add appropriate imports and reformat the source. We added a call to `init()` method because, unlike in a Visual Web JSF Application, in the Visual Web JSP Portlet it does not get executed. The only method of interest, created by default in the JSF, that is executed is the `prerender()` method. `init()` method contains invocation of various methods generated by the addition of the web service data provider so we must execute it to initialize all there is to be initialized.



```
FacilityLookupPortlet * x
Design JSP Java
@Override
public void prerender() {
    init();

    FacListReq flReq = new FacListReq();
    flReq.setDummyString("dummystring");
    facilitySvcPortOpGetfacList1.setMsgFacListReq(flReq);
}
```

Before we deploy the portlet let's switch to the JSP view, locate the line that reads `themeLinks`, scroll to the far right and remove the string `'webuiAll="true"'`. This is necessary if the web browser is a Firefox, as it is in my case. If we don't do this an obscure and annoying bug is triggered when displaying this portlet in Firefox.

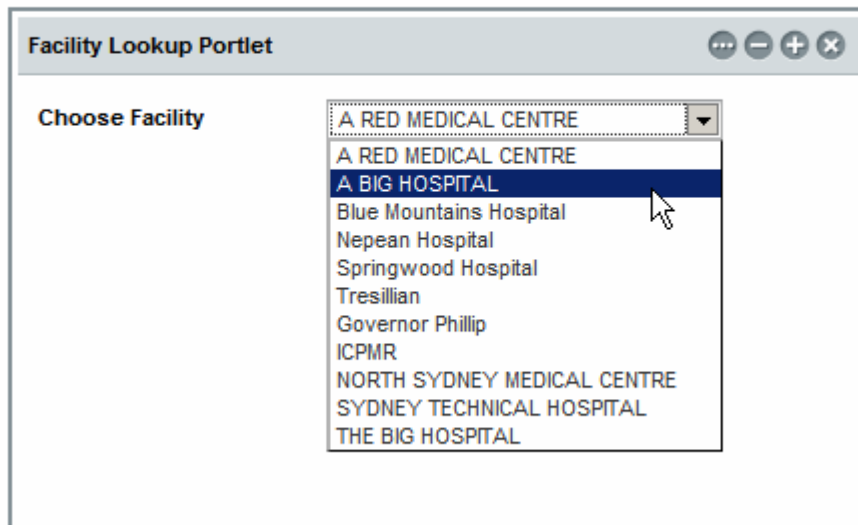


```
FacilityLookupPortlet * x
Design JSP Java
ion="1.0" encoding="UTF-8"?>
nt : FacilityLookupPortlet
d on : 24/06/2009, 11:51:48 AM
: mczapski
version="2.1" xmlns:f="http://java.sun.com/jsf/core" xmlns:h="
p="http://java.sun.com/jsf/portlet/components" xmlns:webuijsf=
irective.page contentType="text/html; charset=UTF-8" pageEncodi
w>
:portletPage>
<div style="height: 200px; position: relative; width: 400px;
<webuijsf:themeLinks binding="#{FacilityLookupPortlet.th
<webuijsf:form binding="#{FacilityLookupPortlet.form1}"
<webuijsf:label id="label1" style="left: 0px; top: 0
<webuijsf:dropDown binding="#{FacilityLookupPortlet.
items="#{FacilityLookupPortlet.facilitySvcPortOp
</webuijsf:form>
```

```
t: grid">  
id="themeLinks1" parseOnLoad="false" webuiAll="true"/>  
: absolute" text="Choose Facility"/>
```

Let's now deploy the portlet again, add it to the page and see what we get.

As before, you may need to reload the page after adding the portlet. Once the portlet displays the facility list will be populated.

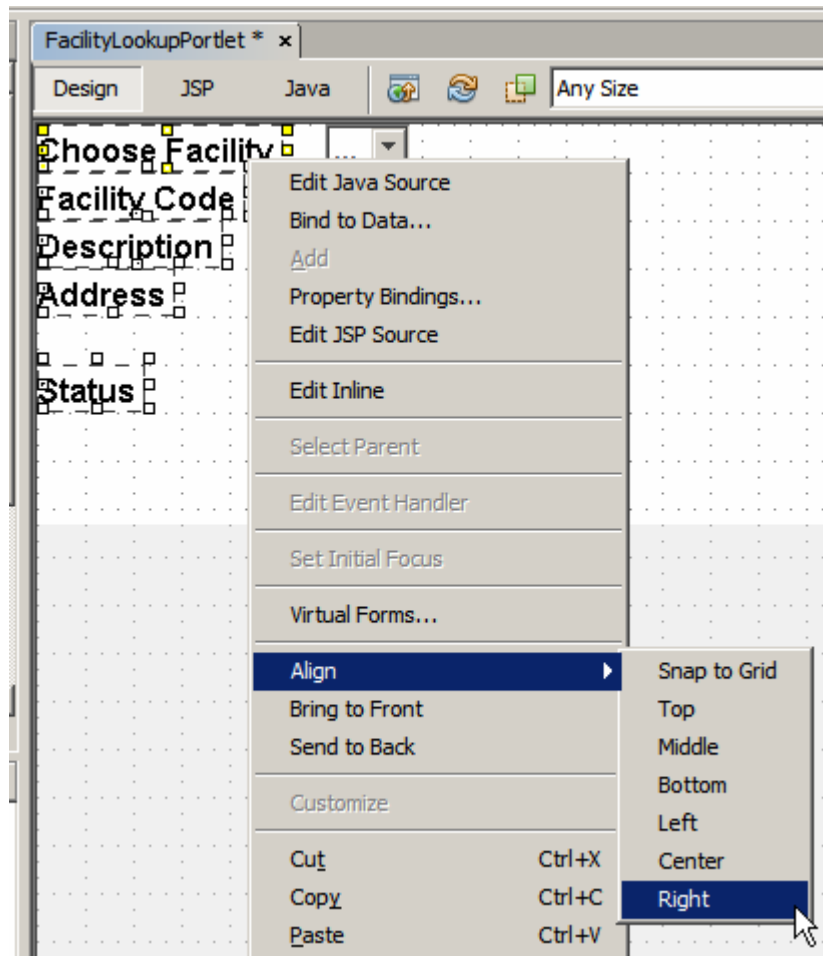


Let's remove the portlet for the next round of changes.

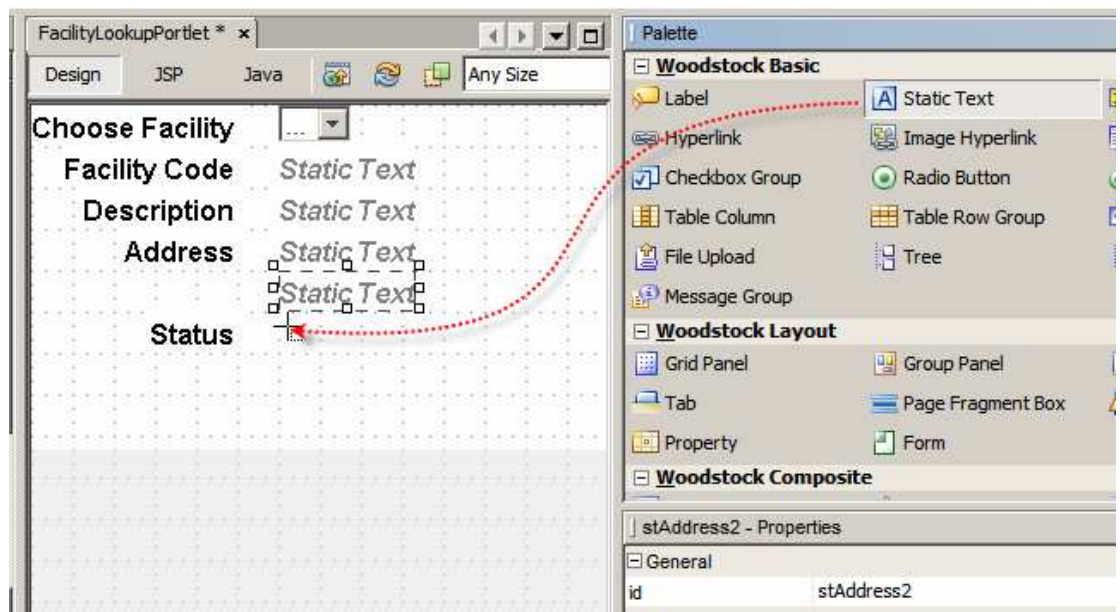
Now that we have the drop down being populated we would like to display details of the selected facility each time facility changes in the dropdown.

Let's switch to Design view, then add 4 labels and 5 static fields to contain descriptions and values of the various facility properties.

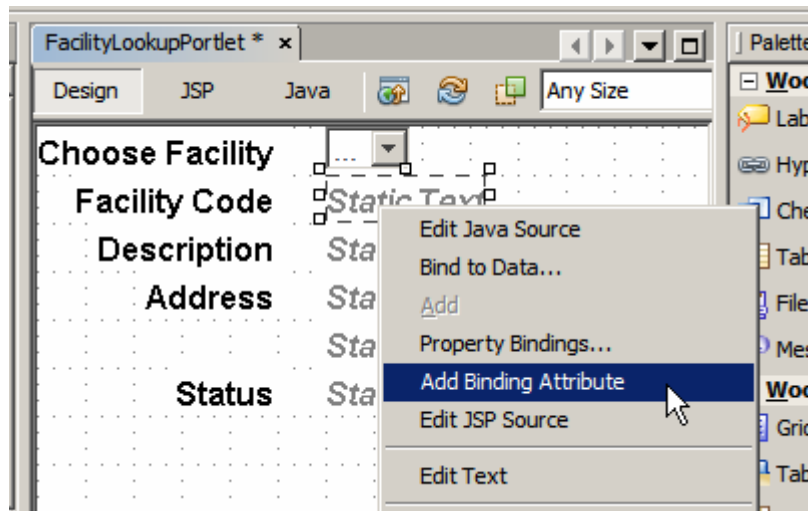
Let's add labels for "Facility Code", "Description", "Address" and "Status", select all labels, right-click and choose Align and choose Right.



Let's now drag 5 static fields and give them ID of stFacCode, stDescription, stAddress1, stAddress2 and stStatus respectively.

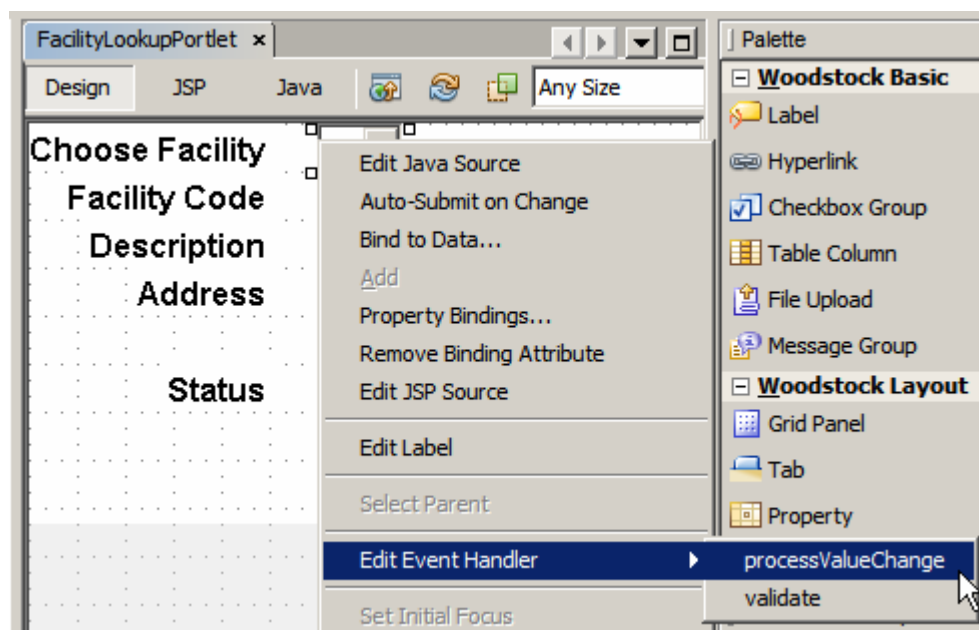


Now right-click on each static field in turn and choose Add Binding Attribute.



This will allow us to manipulate contents of the fields at runtime.

So far nothing happens if we change the value of the ddFacilities drop-down. Let's add a handler to be executed when the value changes. Right-click the drop down component, choose Edit Event Handler and choose processValueChange.



This switches display to the Java source code mode, inserts a skeleton `ddFacilities_processValueChange()` method and allows us to add custom Java code.



```

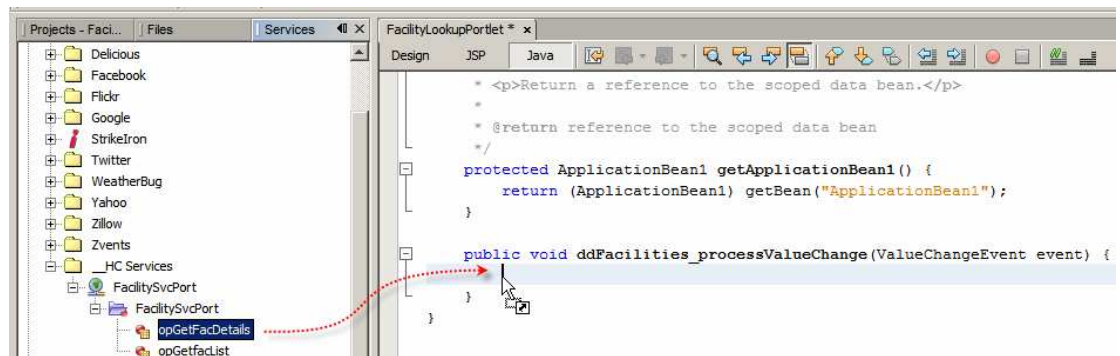
    * <p>Return a reference to the scoped data bean.</p>
    *
    * @return reference to the scoped data bean
    */
    protected ApplicationBean1 getApplicationBean1() {
        return (ApplicationBean1) getBean("ApplicationBean1");
    }

    public void ddFacilities_processValueChange(ValueChangeEvent event) {
    }
}

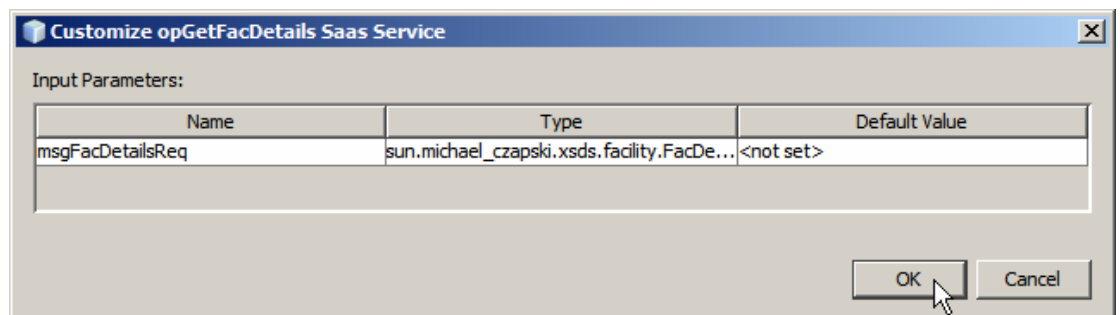
```

What we need to do here is to look at the new value of the ddFacility (facility code), use this code to look up facility details and populate facility details-related static fields with the property values for the designated facility.

Let's switch to the Services Tab, locate the opGetFacDetails web service operation and drag it onto the Java canvas inside the ddFacilities\_processValueChange method.



Accept defaults in the dialog box that results.



A slab of boilerplate code will be added. I re-formatted it for better visibility.

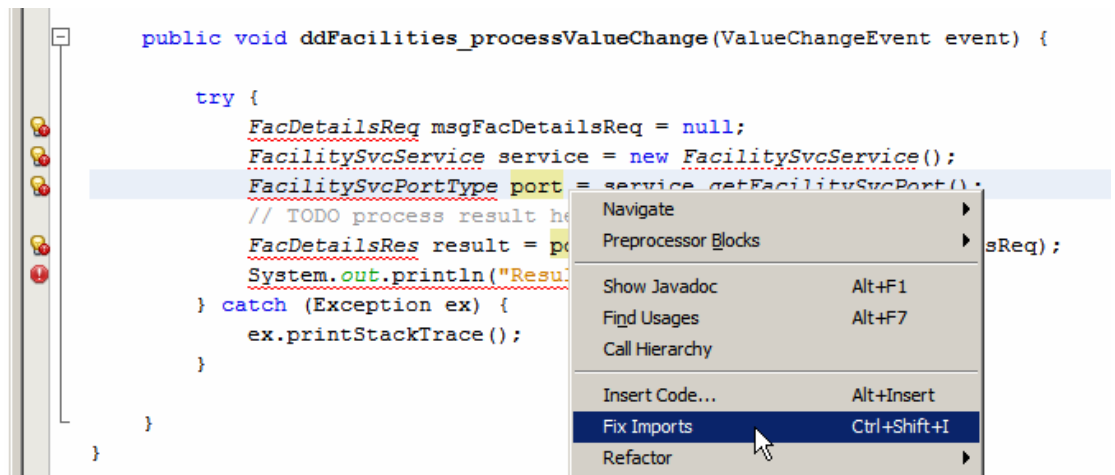
```

public void ddFacilities_processValueChange(ValueChangeEvent event) {

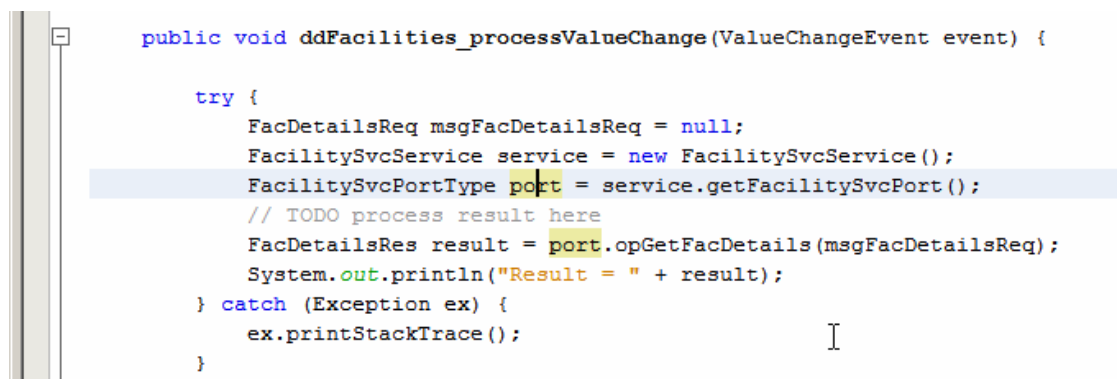
    try {
        sun.michael_czapski.xsds.facility.FacDetailsReq msgFacDetailsReq
            = null;
        sun.michael_czapski.wsdl.facilitysvc.FacilitySvcService service
            = new sun.michael_czapski.wsdl.facilitysvc.FacilitySvcService();
        sun.michael_czapski.wsdl.facilitysvc.FacilitySvcPortType port
            = service.getFacilitySvcPort();
        // TODO process result here
        sun.michael_czapski.xsds.facility.FacDetailsRes result
            = port.opGetFacDetails(msgFacDetailsReq);
        System.out.println("Result = " + result);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

```

Let's remove `sun.michael_czapski.xsds.facility` and `sun.michael_czapski.wsdl.facilitysvc` strings to minimize clutter and use NetBeans IDE facilities to resolve references through the Java import mechanism. Right-click anywhere inside the Java source window and choose Fix Imports.



The brokenness will be resolved and appropriate import statements will be added.



Looking at the statement `FacDetailsReq msgFacDetailsReq = null;` we deduce that we have an opportunity to populate the request message with the facility code and that we will get a response containing facility details 4 statements later. Let's populate the request message with the new facility code chosen in the drop down.

```

public void ddFacilities_processValueChange(ValueChangeEvent event) {

    try {
        FacDetailsReq msgFacDetailsReq = new FacDetailsReq();
        msgFacDetailsReq.setFacCode(((String)event.getNewValue());
        FacilitySvcService service = new FacilitySvcService();
        FacilitySvcPortType port = service.getFacilitySvcPort();
        // TODO process result here
        FacDetailsRes result = port.opGetFacDetails(msgFacDetailsReq);
        System.out.println("Result = " + result);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

```

Now the service execution (port.opGetFacDetails(...)) will give us details for the chosen facility. We can use the response message to populate all the static fields we created earlier.

As you enter this code note how NetBeans helps through code completion and other clever tricks.

```

FacilitySvcService service = new FacilitySvcService();
FacilitySvcPortType port = service.getFacilitySvcPort();
FacDetailsRes result = port.opGetFacDetails(msgFacDetailsReq);

```

stFacCode.setV

|  |      |
|--|------|
| setValue(Object value)                                   | void |
| setValueBinding(String name, ValueBinding binding)       | void |
| setValueExpression(String name, ValueExpression binding) | void |
| setVisible(boolean visible)                              | void |

```

} catch (Exception ex) {
    ex.printStackTrace();
}

```

```

FacilitySvcPortType port = service.getFacilitySvcPort();
FacDetailsRes result = port.opGetFacDetails(msgFacDetailsReq);

```

Object value

```

stFacCode.setValue(result.get

```

|                   |          |
|-------------------|----------|
| getAddressLine1() | String   |
| getClass()        | Class<?> |
| getCountry()      | String   |
| getDescription()  | String   |
| getFacCode()      | String   |
| getPostCode()     | String   |
| getState()        | String   |
| getStatus()       | String   |
| getSuburbTown()   | String   |

```

    System.out.println("Result = " + result);
} catch (Exception ex) {
    ex.printStackTrace();
}
}

```

Output

```

try {
    FacDetailsReq msgFacDetailsReq = new FacDetailsReq();
    msgFacDetailsReq.setFacCode((String)event.getNewValue());
    FacilitySvcService service = new FacilitySvcService();
    FacilitySvcPortType port = service.getFacilitySvcPort();
    FacDetailsRes result = port.opGetFacDetails(msgFacDetailsReq);

    stFacCode.setValue(result.getFacCode());
    stDescription.setValue(result.getDescription());
    stAddress1.setValue(result.getAddressLine1());
    stAddress2.setValue(result.getSuburbTown() + ", " +
        result.getState() + ", " + result.getPostCode());
    stStatus.setValue(result.getState());

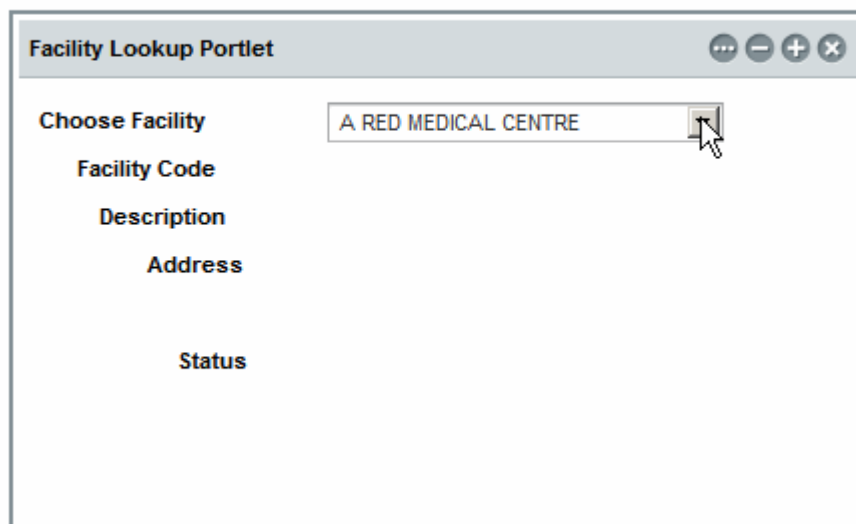
    System.out.println("Result = " + result);
} catch (Exception ex) {
    ex.printStackTrace();
}

```

Our ddFacility\_processValueChange() method body is completed. Each time we change the value in the drop down we will get the details of the facility displayed.

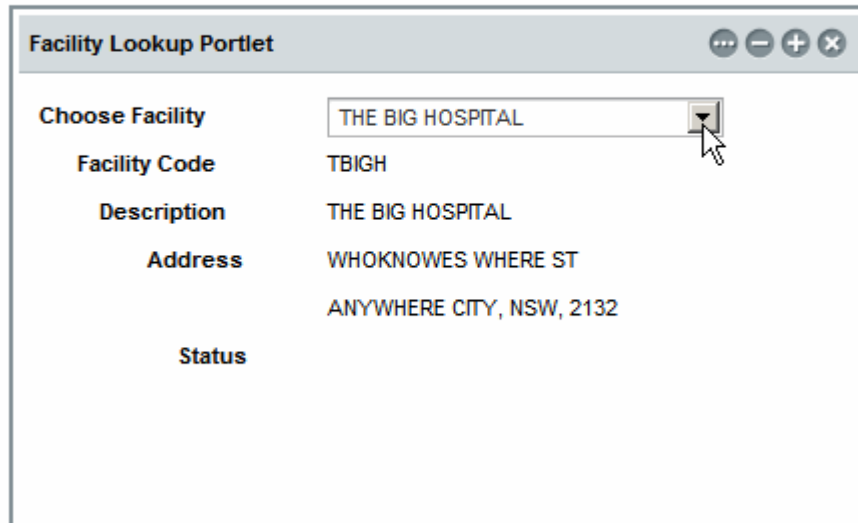
Let's deploy the portlet, as it stands now, add it to the page and see what it looks like and how it behaves.

Notice that we have the populated drop down, as we did before, but no facility details for the selected facility on initial display.



Let's choose a different facility and see what happens.

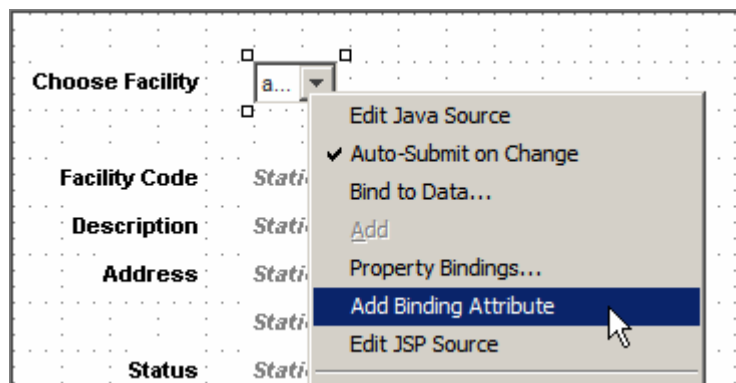
This time details for the new facility got displayed.



The reason we did not see the details when the web page first appeared was because the `ddFacility_provessValueChange` was not executed the first time around. To make it execute we need to add some code to the `prerender()` method to first establish whether this is the first time that the page is displayed, then to work out what the facility code is the first time around and finally to force `ddFacility_processValueChange` method to get executed so the facility details get displayed.

Let's do these things a bit at a time.

Let's switch to the Design mode, right-click on the drop down and choose Add Binding Attribute. If the text reads Remove Binding Attribute we don't need to do this.



Let's switch to Java source mode and scroll down to the `prerender()` method.

At the end of the code already in the pre-render method let's add the following:

```
log("===>>> " + ddFacilities.getValue());
```

Deploy the portlet again, then add it to the page. Look at the server.log to see what this statement displays the first time the application is run, the change facility in the drop down and see what this statement displays the second time around.

On both occasions I see:

```
[#|2009-06-24T15:23:10.046+1000|INFO|sun-appserver2.1|javax.servletContext.log():===>>> null|#]
```

In the portlet I can not tell, by looking at the content of ddFacilities, whether this is the first time the portlet was shown or not. We do know that the ddFacilities\_processValueChange() method was invoked because this is where the static fields get populated and we see values in the static fields when we change the facility selection. Let's confirm the value of the selected facility by adding a log statement to see what the new value is each time the method is executed.

Let's add the following at the end of the ddFacilities\_processValueChange():

```
        result.getState() + ", " + result.getPostCode() + ", " + result.getStateStatus().setValue(result.getState());

        System.out.println("Result = " + result);
    } catch (Exception ex) {
        ex.printStackTrace();
    }

    log("===>>> New Facility " + event.getNewValue());
}
}
```

Let's deploy the portlet, add it to the page, change the facility to Sydney Technical Hospital and see what the server.log says.

```
[#|2009-06-24T15:40:31.390+1000|INFO|sun-appserver2.1|javax.enterprise.system.com.sun.portal.portletContext.log():===>>> New Facility STC|#]

[#|2009-06-24T15:40:31.390+1000|INFO|sun-appserver2.1|debug.com.sun.portal.portletContext.log():===>>> null|#]

[#|2009-06-24T15:40:31.437+1000|INFO|sun-appserver2.1|debug.com.sun.portal.portletContext.log():===>>> null|#]

[#|2009-06-24T15:40:31.453+1000|INFO|sun-appserver2.1|javax.enterprise.system.com.sun.portal.portletContext.log():===>>> null|#]

[#|2009-06-24T15:40:31.453+1000|INFO|sun-appserver2.1|debug.com.sun.portal.portletContext.log():===>>> null|#]
```

The first highlighted log entry comes from the ddFacilities\_processValueChange() method. The second highlighted log entry comes from the prerender() method. Clearly, at one point we know what the chosen facility is but we "forget" it by the time we get to render the page.

Unlike a JSF web application, which gets rendered in a single pass, the portlet gets rendered in two passes (or so I read). This makes it 'forget' local variable values. We need to resort to session variables to pass values of interest between render passes.

Let's add the following code to the `ddFacilities_processValueChange()` method, following the log statement:

```
log("====>>> New Facility " + event.getNewValue());

setValue("#{sessionScope.facilitysvcvwjsfp_FacilityLookupPortlet_facCode}"
, (String) event.getNewValue());
```

The `setValue()` method will add a variable with specific name and value to the appropriate scope. The value of this variable can then be retrieved elsewhere, possibly in a different rendering pass. I modified the magic incantation `"#{sessionScope.varName}"` so that the variable name includes the package and class name. This way I don't have to worry about making sure the variable names are unique across portlets deployed to the same page. I still need to worry about adding the same portlet to the same page more than once but this is an issue that you can address if it worries you.

Let's now go back to the `prerender()` method, get the value of the `sessionScope` variable we are setting in the `ddFacilities_processValueChange()` method and display this value in the log to see what it is at different times.

```
log("====>>> " + ddFacilities.getValue());

String sFacCode
    = (String) getValue("#{sessionScope.facilitysvcvwjsfp_FacilityLookupPortlet_facCode}");
log("====>>> prerender: FacCode " + sFacCode);
```

Let's deploy the portlet and add it again to the page, assuming all the while that we are removing the portlet from the page after each test.

First time around, after deployment, we see log entries from the `prerender` method():

```
[#|2009-06-24T16:27:41.343+1000|INFO|sun-appserver2.1|java
ervletContext.log():====>>> null|#]

[#|2009-06-24T16:27:41.343+1000|INFO|sun-appserver2.1|java
ervletContext.log():====>>> null|#]

[#|2009-06-24T16:27:41.343+1000|INFO|sun-appserver2.1|debu

[#|2009-06-24T16:27:41.343+1000|INFO|sun-appserver2.1|debu

[#|2009-06-24T16:27:41.343+1000|INFO|sun-appserver2.1|java
ervletContext.log():====>>> prerender: FacCode null|#]

[#|2009-06-24T16:27:41.343+1000|INFO|sun-appserver2.1|java
ervletContext.log():====>>> prerender: FacCode null|#]
```

In both tracers facility code (one from the `ddFacility.getValue()` and one from `getValue("#{sessionScope...}")`) are null.

Let's choose Sydney Technical Hospital from the drop down and see what the log says.

```

[#|2009-06-24T16:28:43.906+1000|INFO|sun-appserver2.1|javax.enterprise.system.container.web|webModule[/FacilitySvcVWJSFP] ServletContext.log():===>>> New Facility STC|#]
[#|2009-06-24T16:28:43.906+1000|INFO|sun-appserver2.1|debug.com.sun.portal.portletcontainer|8080-1;|===>>> New Facility STC|#]
[#|2009-06-24T16:28:43.937+1000|INFO|sun-appserver2.1|debug.com.sun.portal.portletcontainer|itySvcVWJSFP;FacilityLookupPortlet;FacilityLookupPortlet_WAR_FacilitySvcVWJSFP_INSTANCE_U3?, PortletName:FacilityLookupPortlet, PortletWindowName:FacilityLookupPortlet_WAR_Facility
[#|2009-06-24T16:28:43.937+1000|INFO|sun-appserver2.1|javax.enterprise.system.container.web|webModule[/FacilitySvcVWJSFP] ServletContext.log():===>>> null|#]
[#|2009-06-24T16:28:43.937+1000|INFO|sun-appserver2.1|debug.com.sun.portal.portletcontainer|8080-1;|===>>> null|#]
[#|2009-06-24T16:28:43.937+1000|INFO|sun-appserver2.1|javax.enterprise.system.container.web|webModule[/FacilitySvcVWJSFP] ServletContext.log():===>>> prerender: FacCode STC|#]
[#|2009-06-24T16:28:43.937+1000|INFO|sun-appserver2.1|debug.com.sun.portal.portletcontainer|8080-1;|===>>> prerender: FacCode STC|#]

```

The first highlighted entry comes from the `ddFacilities_processValueChange()` method. We know at that point what the selected facility code is.

The method sets the session-scoped variable to that value.

In the second highlighted log statement, in `prerender()` method, we get a null from `ddFacilities.getValue()`. We don't know what the facility code just selected is.

In the third highlighted log statement, still in `prerender()` method we get the session-scoped variable to work out what the facility code was in the `ddFacilities_processValueChange()`.

As we can see the first time the `prerender()` method is run the session-scoped variable will be null. On subsequent renders it will contain the value of the most recently selected facility code.

Knowing that this is happening allows us to determine whether we are displaying the first time or the subsequent time.

First time around, as we noticed, we have no idea what the "selected" code is. We may or may not be aware that at this point in the code the `opGetFacList` operation was executed and the list of facilities is available. We do know that the first time around the first facility will be the one "selected" in the drop down. We can exploit both of these to work out what the facility code is.

Let's add the following slab of code to the end of the `prerender()` method.



```

String sFacCode
    = (String) getValue("#{sessionScope.facilitysvcvwjsfp_FacilityLookupPortlet_facCode}");
log("====>>> prerender: FacCode " + sFacCode);

if (sFacCode == null) {

    // get the list of facility objects already assembled
    // get the first item in the list
    // get facility code for first item
    //
    Object[] objFacListAry = facilitySvcPortOpGetfacList1.getResultObjects();
    Object objFirstFac = objFacListAry[0];
    FacListRes.FacList fAcR = (FacListRes.FacList) objFirstFac;

    // force the ddFacilities_processValueChange() method
    // to execute using the facility code as new value
    //
    ValueChangeEvent event =
        new ValueChangeEvent(ddFacilities, null, fAcR.getFacCode());
    ddFacilities_processValueChange(event);
}

```

The `facilitySvcPortOpGetfacList1.getResultObjects()` is a reference to the web service-returned list of facility objects. The first item in that list will give us access to the facility code. We create a new event in which we are setting the new value to that code and explicitly executing `ddFacility_processValueChange` method. This causes the static fields to be populated.

Here is the complete `prerender()` method.

```

public void prerender() {
    init();

    FacListReq flReq = new FacListReq();
    flReq.setDummyString("dummystring");
    facilitySvcPortOpGetfacList1.setMsgFacListReq(flReq);

    log("====>>> " + ddFacilities.getValue());

    String sFacCode
        = (String) getValue("#{sessionScope.facilitysvcvwjsfp_FacilityLookupPortlet_facCode}");
    log("====>>> prerender: FacCode " + sFacCode);

    if (sFacCode == null) {

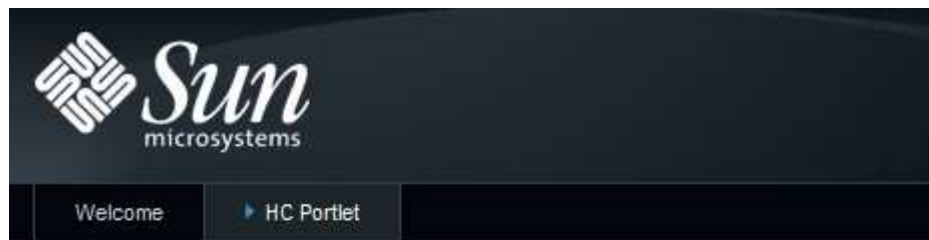
        // get the list of facility objects already assembled
        // get the first item in the list
        // get facility code for first item
        //
        Object[] objFacListAry = facilitySvcPortOpGetfacList1.getResultObjects();
        Object objFirstFac = objFacListAry[0];
        FacListRes.FacList fAcR = (FacListRes.FacList) objFirstFac;

        // force the ddFacilities_processValueChange() method
        // to execute using the facility code as new value
        //
        ValueChangeEvent event =
            new ValueChangeEvent(ddFacilities, null, fAcR.getFacCode());
        ddFacilities_processValueChange(event);
    }
}

```

Let's deploy the portlet, add it to the page again, and see what we get the first time around and what we get when we change the facility selection.

First time around:



Facility Lookup Portlet

|                 |   |
|-----------------|---|
| Choose Facility | A RED MEDICAL CENTRE                      |
| Facility Code   | ARMC                                      |
| Description     | A RED MEDICAL CENTRE                      |
| Address         | SOMEWHERE ST<br>SOMEWHERE CITY, NSW, 2060 |
| Status          | NSW                                       |

Choosing Sydney Technical Hospital:



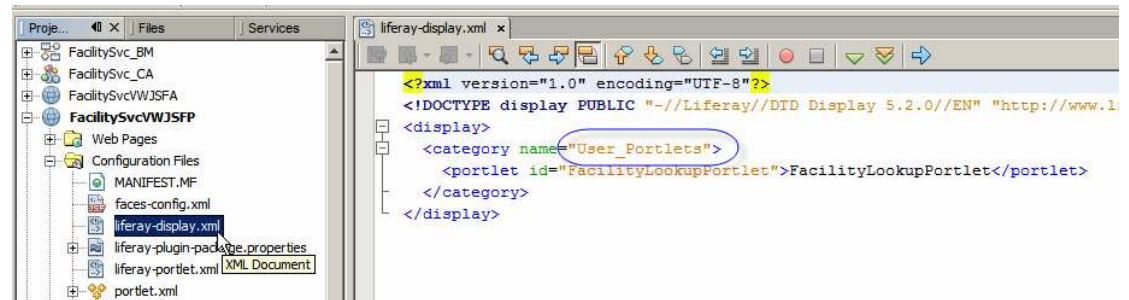
Facility Lookup Portlet

|                 |   |
|-----------------|---|
| Choose Facility | SYDNEY TECHNICAL HOSPITAL                   |
| Facility Code   | STC   |
| Description     | SYDNEY TECHNICAL HOSPITAL                   |
| Address         | 404 HUNTINGTON DRIVE<br>MONROVIA, CA, 91016 |
| Status          | CA  |

The portlet is now complete and behaves the way we expect it to, both the first time it is displayed and each time a different facility is chosen.

Note that the portlet was showing up in the "User Portlets" group of Applications. We can change that to a group with a name of our own choosing.

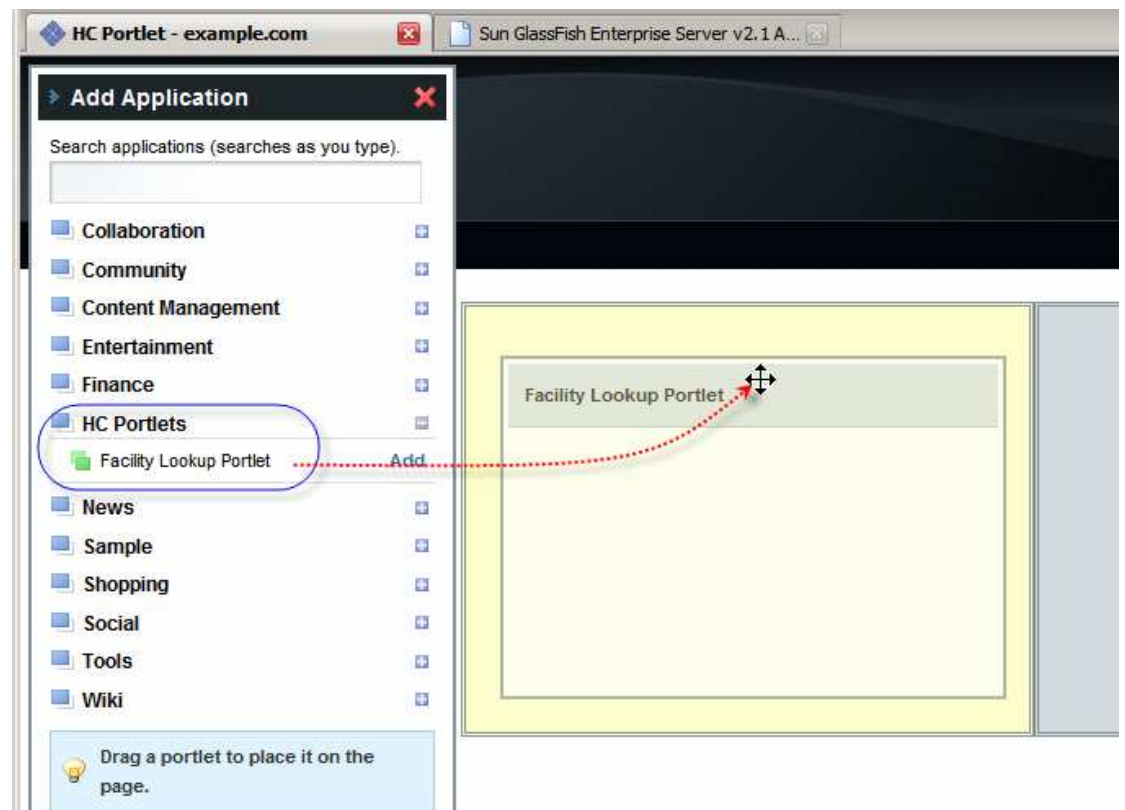
Expand the project FacilitySvcVWJSFP, expand Configuration Files and open liferay-display.xml configuration file in the editor. Change the text in name attribute of the category tag.



For example changing the name to "HC Portlets" will make the portlet appear in a new category, with that name, once it is next deployed. Deploy the portlet to effect the change.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE display PUBLIC "-//Liferay//DI
| <display>
|   <category name="HC Portlets">
|     <portlet id="FacilityLookupPortlet">
|     </category>
| </display>
```

Note the portlet in the new group.



This is it. This is what it took to develop a portlet that used the Facilities web service as a data provider and deploy it to the Web Space Server 10 Portal.

## Summary

In this document we created, deployed and exercised a JSR 286-compliant portlet that provided a list of Healthcare Facilities as a drop down and details of a specific Facility when chosen.

We used the NetBeans 6.5.1 IDE, included with the GlassFish ESB v 2.1 infrastructure, Portal Pack 3.0.1 and Web Space Server 10. We used the Visual Web JavaServer Faces Portlet and JSF Portal Bridge technologies, Project Woodstock JSF components and JBI-based multi-operation web service. The NetBeans IDE tooling assisted in rapidly developing the web application with minimum of custom Java code. This web application, a component in SOA 1, Presentation Layer, consumes SOA 3, Business Service service in a loosely coupled manner.