NetBeans 6.5.1, GlassFish v 2.1, Web Space Server 10
Patient Lookup Visual Web JSF Portlet with a Nicer Google Map

Michael.Czapski@sun.com
July 2009

Table of Contents

Abstract

In this walkthrough I will add a nicer looking Google Map, with Google Search functionality, to the Visual Web JSF Portlet, developed in "GlassFish v 2.1, Web Space Server 10 - Patient Lookup Visual Web JSF Portlet with a basic Google Map", at http://blogs.sun.com/javacapsfieldtech/entry/glassfish_v_2_1_web.

Introduction

The business idea behind the functionality developed in this walkthrough is that patients are looked after in various healthcare facilities. Healthcare workers need to lookup patient details such as their identifier, gender, birth date or address. A relational database holds patient details as well as other information of relevance such as descriptions of various coded values. Patient details are available through a web service. Facility list and details, used to narrow down the search for patients to a specific facility, are available through a web service. These web services will be used to construct the Portlet that will allow patient search and a display of patient details with display a Google Map, centered at patient's address, if one is available and is valid for the purpose of mapping. This Portlet will be deployed to the Sun FOSS Web Space Server 10 Portal.

The previous document [11], walked through development and deployment of the Patient Lookup Portlet with a basic Google Map centered at the location identified by patient address, if any. It was a fairly basic Google Map, as Google maps go. In this document I will add a better Google Map to the Patient Lookup Portlet developed in [11].

Other documents in this series, see pre-requisites, walked the reader through the process of implementing GlassFish ESB v2.1-based web services which return facility list and facility details as well as patient details.

To give you some idea of what we will get at the end of the process here are screenshots of the completed portlet running out of the Web Space Server Portal.

Sun microsystems

Welcome | HC Portlet

Patient Lookup Map Better

Choose Facility: SYDNEY TECHNICAL HOSPITAL

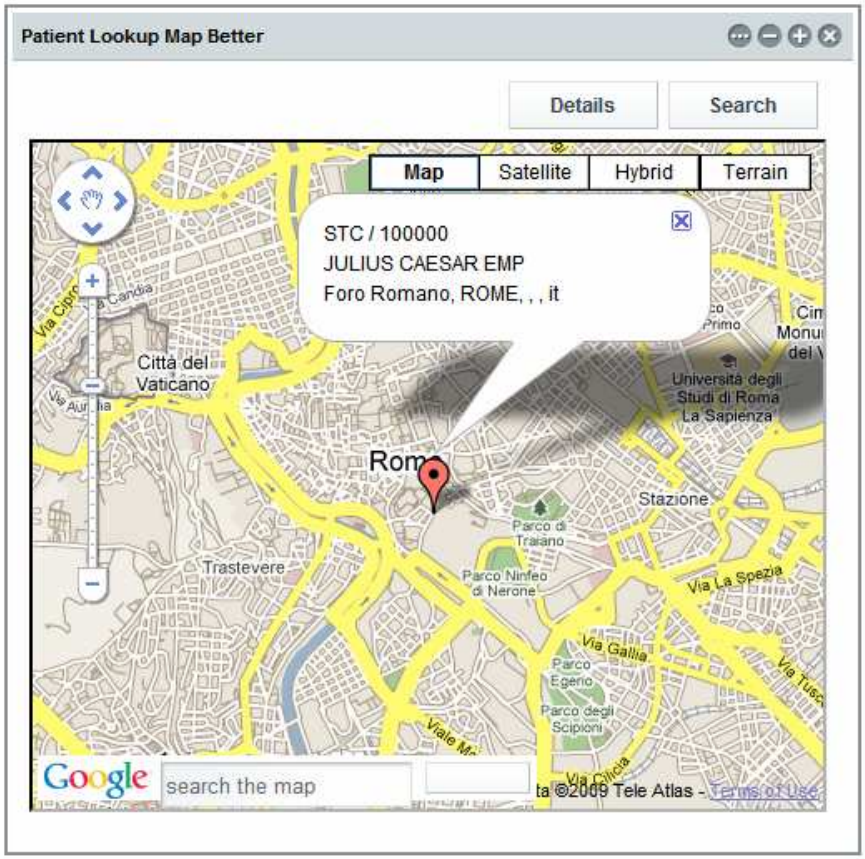
Enter Local ID *: 100000

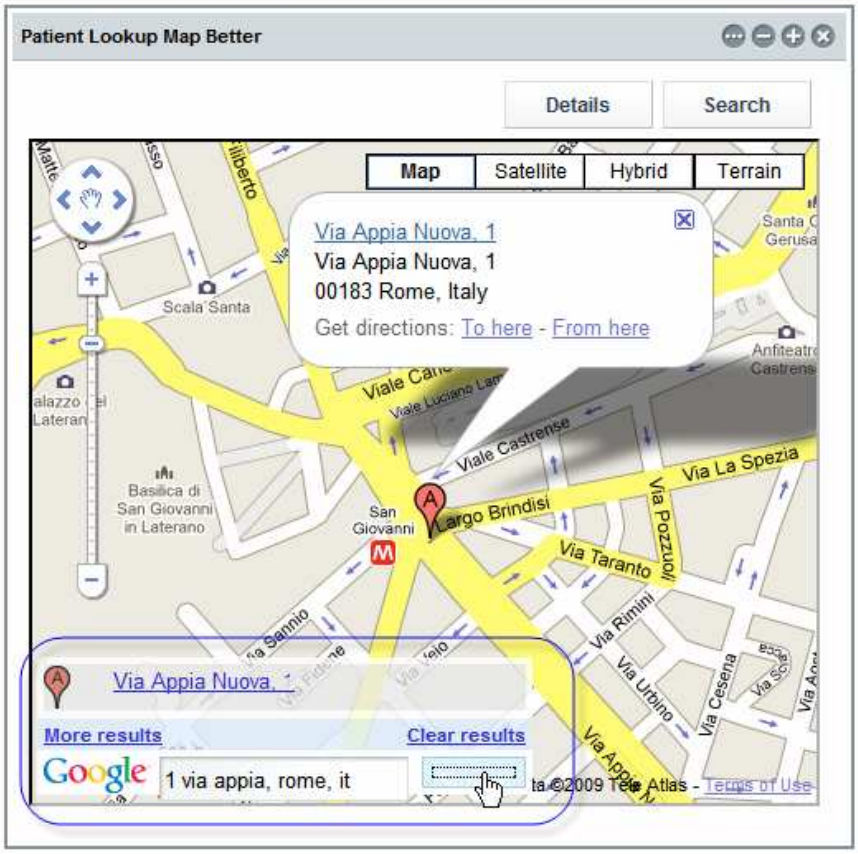
Lookup

Patient Lookup Map Better

Map | Search

Facility	SYDNEY TECHNICAL HOSPITAL (STC)
Local ID	100000
Patient Name	JULIUS CAESAR EMP
Gender	MALE (M)
Race	()
Ethnic Origin	()
Religion	()
Language	()
Marital Status	()
Address	Foro Romano
	ROME, , , it
Medicare Number	
Date of Birth	-1000713


Patient Lookup Map Better
Details
Search
Map
Satellite
Hybrid
Terrain
STC / 100000
JULIUS CAESAR EMP
Foro Romano, ROME, , , it
Citta del Vaticano
Università degli Studi di Roma La Sapienza
Stazione
Parco di Traiano
Parco Ninfeo di Nerone
Trastevere
Via La Spezia
Via Gallia
Parco Egerio
Parco degli Scipioni
Google
search the map
©2009 Tele Atlas - Terms of Use


Patient Lookup Map Better
Details
Search
Map
Satellite
Hybrid
Terrain
Via Appia Nuova, 1
Via Appia Nuova, 1
00183 Rome, Italy
Get directions: To here - From here
Scala Santa
Basilica di San Giovanni in Laterano
San Giovanni
Viale Castrense
Via La Spezia
Largo Brindisi
Via Taranto
Via Pozzuoli
Via Rimini
Via Urbino
Via Cesena
Via Appia Nuova, 1
More results
Clear results
Google
1 via appia, rome, it
©2009 Tele Atlas - Terms of Use

Note that this walkthrough builds on the Patient Lookup Portlet with basic Google Map, built previously, but deals exclusively with Visual Web JSF portlet-related technologies, Java Script and Google Maps API.

Prerequisites

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

It is assumed that:

- MySQL RDBMS is installed and available, as discussed in [1]
- GlassFish ESB v2.1 is installed, as discussed in [2]
- Sun Web Space Server Portal is installed, as discussed in [3]
- Web Space Server is configured as discussed in [4]
- Facility Service Web Service is implemented and deployed, as discussed in [5]
- Patient Service Web Service is implemented and deployed, as discussed in [6]
- Patient Lookup Portlet with basic Google Map has been developed and tested [11]

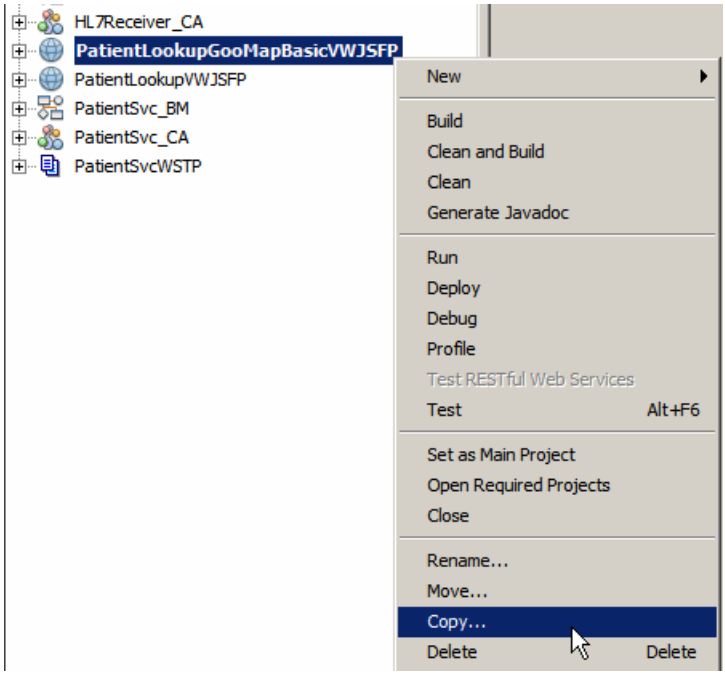
Unless these pre-requisites are met, you will not be able to complete this walkthrough.

Copy Patient Lookup Project

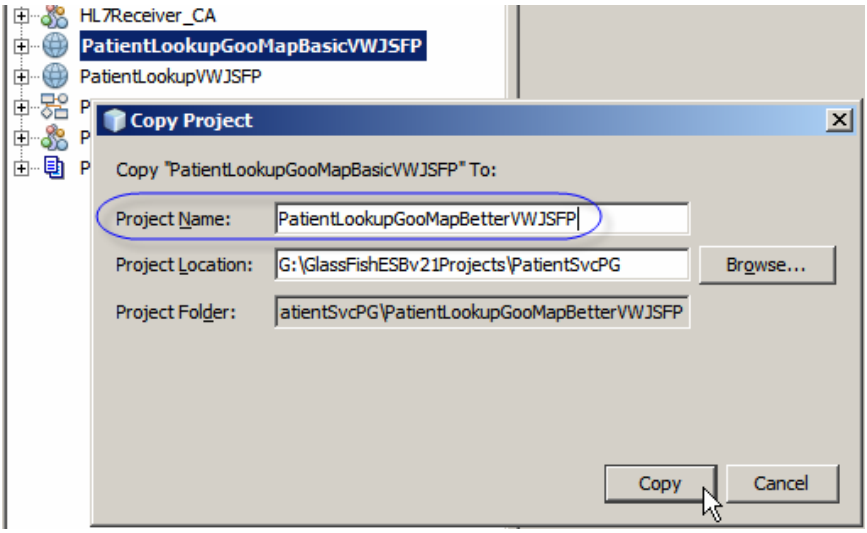
This document assumes that the Patient Lookup with basic Google Map Portlet, developed in [11], is available for cloning.

To save the effort we will copy the project PatientLookupGooMapBasicVWJSFP [11] and use it as the basis for elaboration.

Right-click the name of the project and choose Copy.

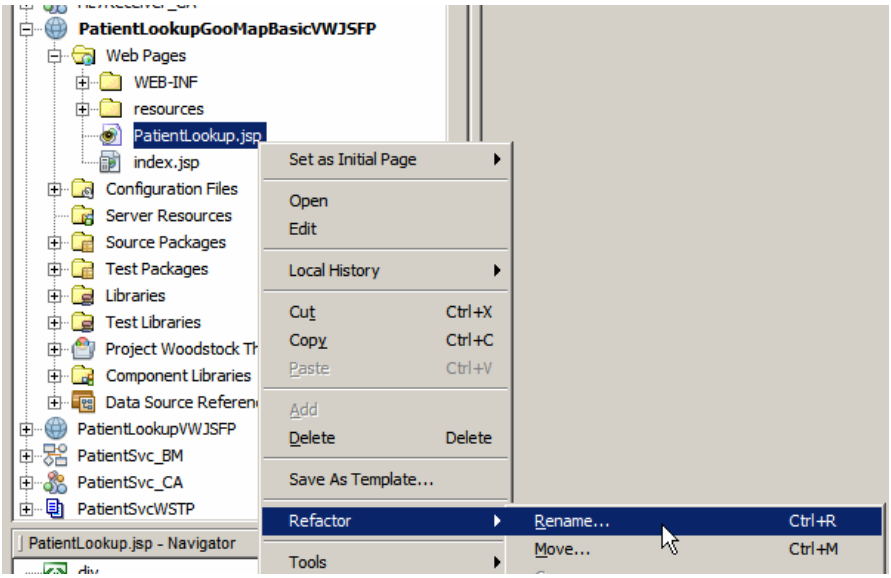


Name the new project PatientLookupGooMapBetterVWJSFP and click the Copy button.

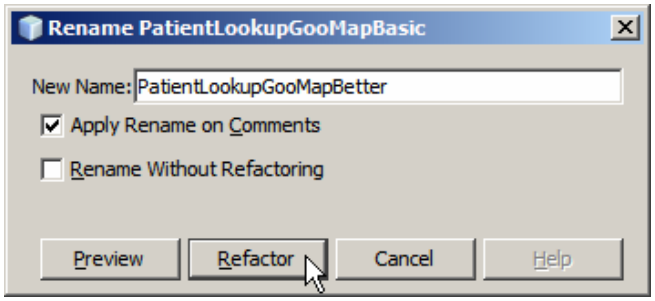


Right-click the name of the new project and choose “Set as Main Project”.

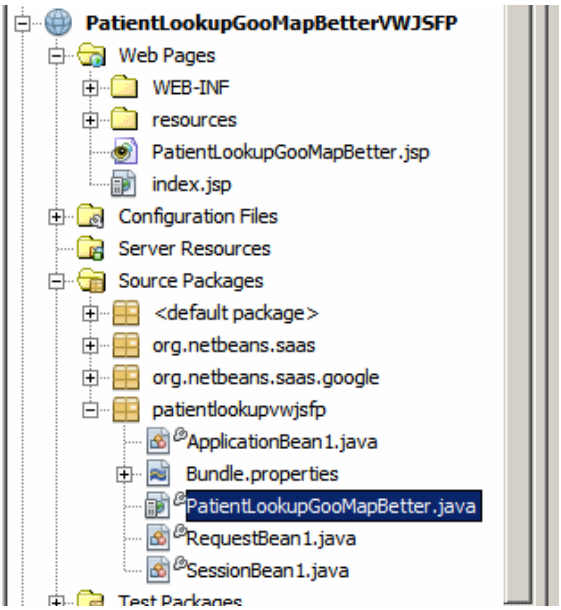
Expand the project’s Web Pages folder, right-click on the PatientLookupGooMapBasic.jsp page and choose Refactor -> Rename.



Change the name to PatientLookupGooMapBetter, check the “Apply Rename on Comments” and click the “Refactor” button.



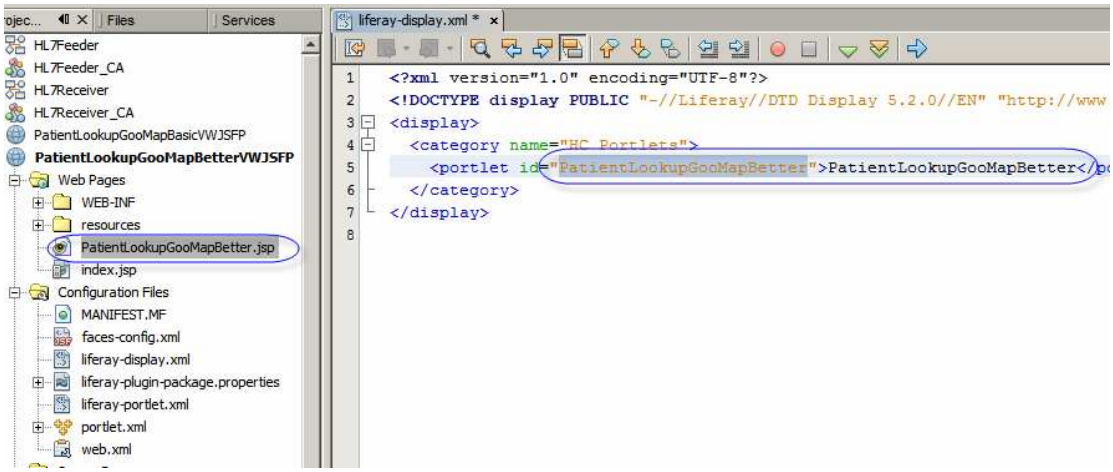
Note that the portlet backing class was also renamed.



Alas, there are a few configuration files which must be manually modified.

Expand the “Configuration Files” folder.

Open the liferay-display.xml and update portlet name and id.

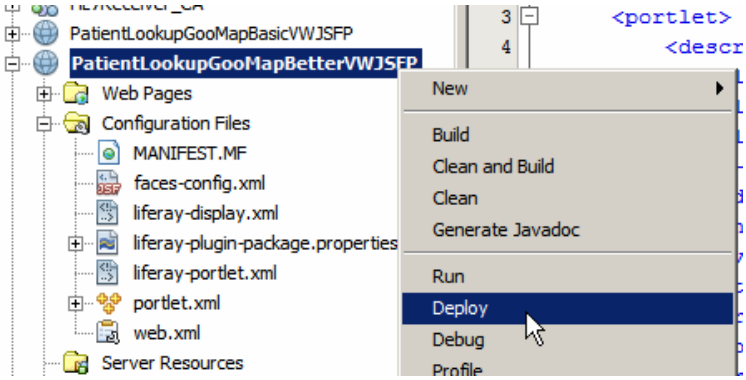


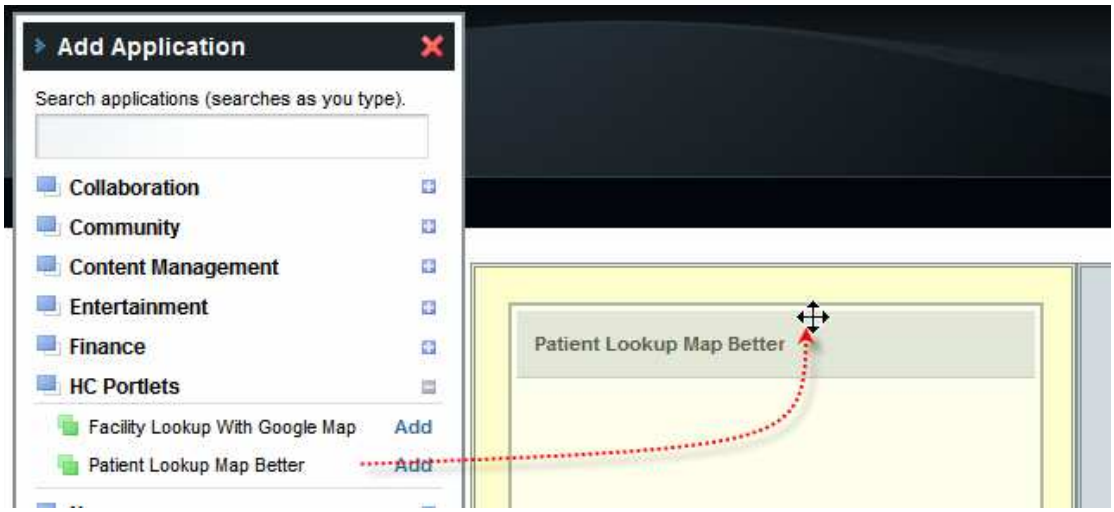
Open liferay-portlet.xml and update portlet-name.

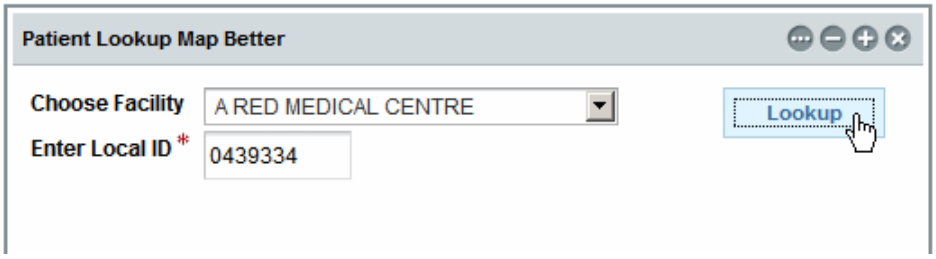
Open portlet.xml and update description, portlet-name, display-name, title and short-title. Of these only the portlet-name and init-param -> value are critical.

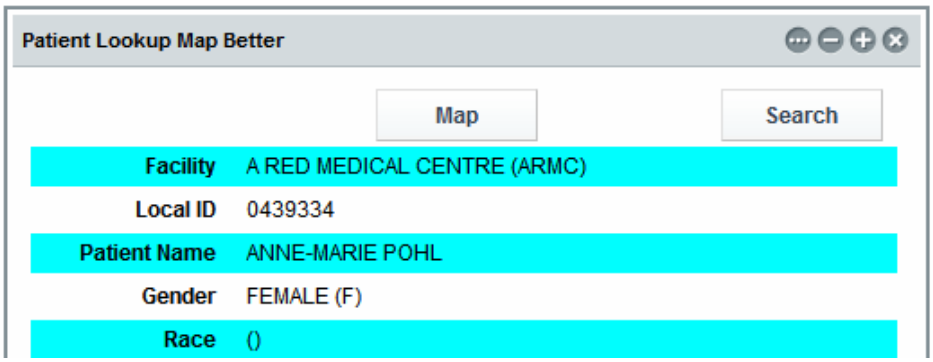
```
1  <?xml version='1.0' encoding='UTF-8' ?>
2  <portlet-app xmlns='http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd'
3      <portlet>
4          <description>Patient Lookup Map Better</description>
5          <portlet-name>PatientLookupGooMapBetter</portlet-name>
6          <display-name>Patient Lookup Map Better</display-name>
7          <portlet-class>com.sun.faces.portlet.FacesPortlet</portlet-class>
8          <init-param>
9              <description>Portlet Init View Page</description>
10             <name>com.sun.faces.portlet.INIT_VIEW</name>
11             <value>/PatientLookupGooMapBetter.jsp</value>
12         </init-param>
13         <expiration-cache>0</expiration-cache>
14         <supports>
15             <mime-type>text/html</mime-type>
16             <portlet-mode>VIEW</portlet-mode>
17         </supports>
18         <supported-locale>en</supported-locale>
19         <resource-bundle>messages</resource-bundle>
20         <portlet-info>
21             <title>Patient Lookup Map Better</title>
22             <short-title>Patient Lookup Map Better</short-title>
23         </portlet-info>
24     </portlet>
25 </portlet-app>
```

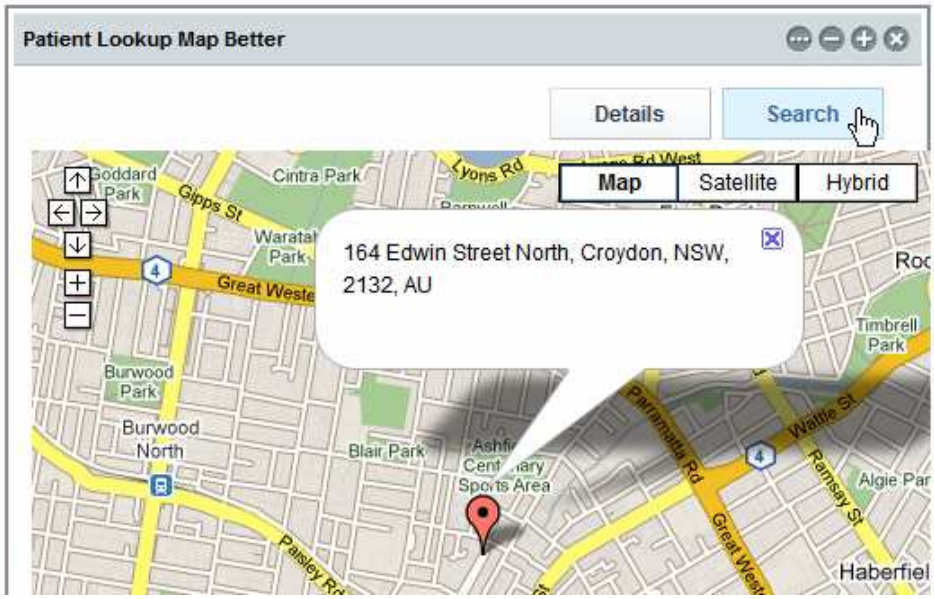
Once done, deploy the portlet and exercise it in the browser to make sure it still functions.









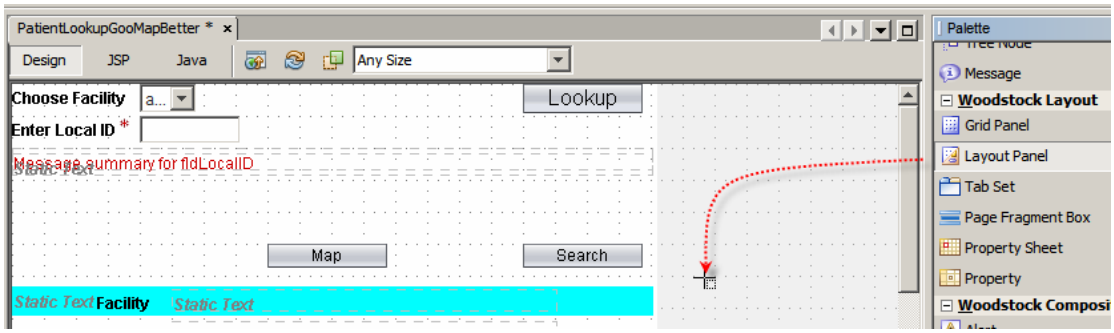


It works for me.

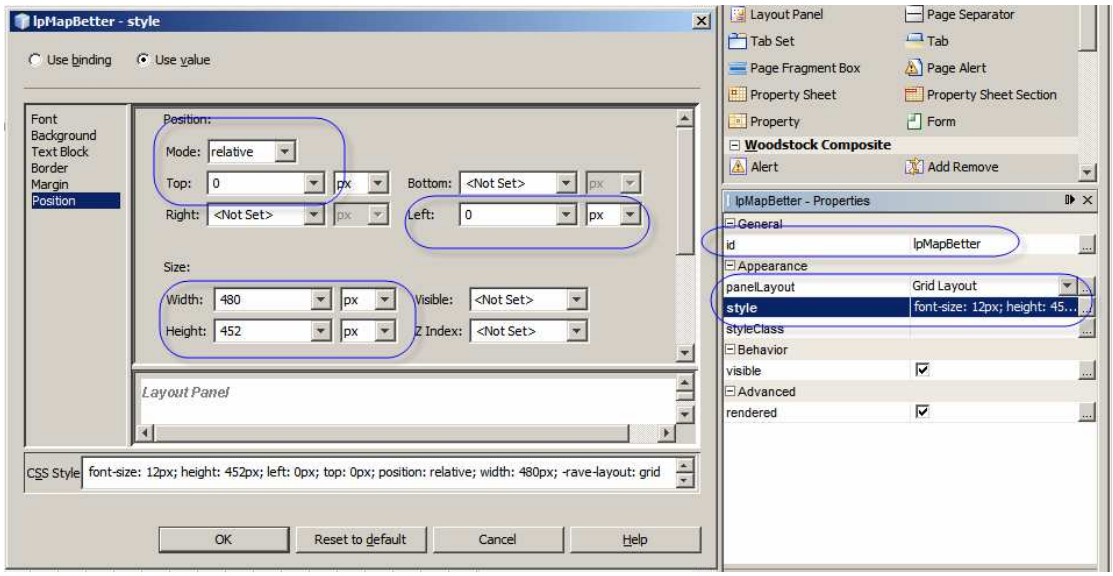
Add Google Map components

In [11] we developed a portlet that invokes the Google Map REST service, gratuitously provided by the NetBeans and Google teams. This service returns a HTML fragment, which includes Java Script scripts and other elements. We “injected” this HTML fragment into an outputText container of our portlet. As a consequence the HTML content, returned by the Google Map service, was rendered in the browser. During that process the Google Maps service-provided Java Script scripts executed and caused the content of the DIV element to be dynamically replaced with the Google Map. At the same time the page got authenticated with the Google Map service and a series of Google JavaScript scripts became available to be dynamically executed. We will use the fact that we are authenticated and that we can execute some of the Google JavaScript scripts to get and manipulate Google Maps objects.

Open the PatientLookupGooMapBetter.jsp in Design mode. Drag the Woodstock Layout Panel to the canvas anywhere outside the existing layout panels.



Change the id property to lpMapBetter, panelLayout property to “Grid Layout” and style property attributes to: font-size:12px, position: relative, height: 452px, width: 480px, left: 0px, top: 0px.



The new panel now appears below all existing panels in the Design View.

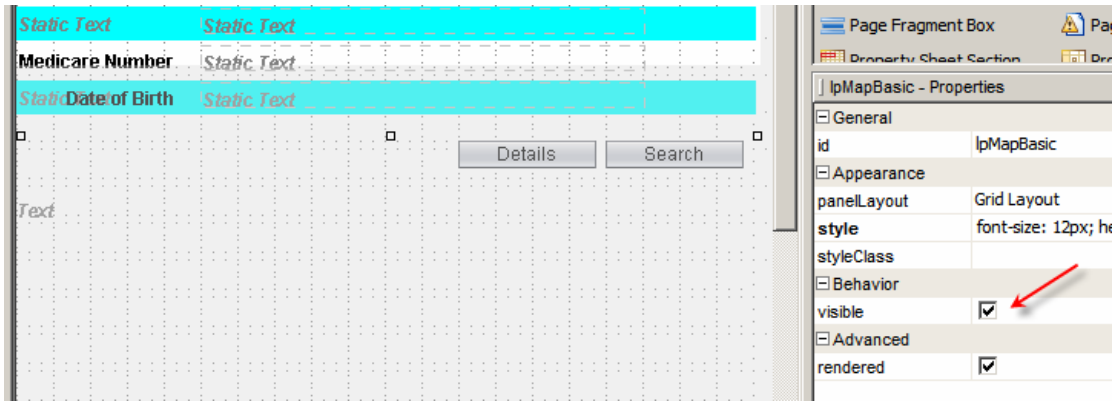
Right-click the panel and choose “Add Binding Attribute” so that we can manipulate visible property of the panel in the Java class.

Copy the search button from the lpView panel and paste it into the lpMapBetter panel. Set new buttons properties id: btnSearch05, style: font-size: 12px; left: 383px; top: 0px; position: absolute; width: 90px.

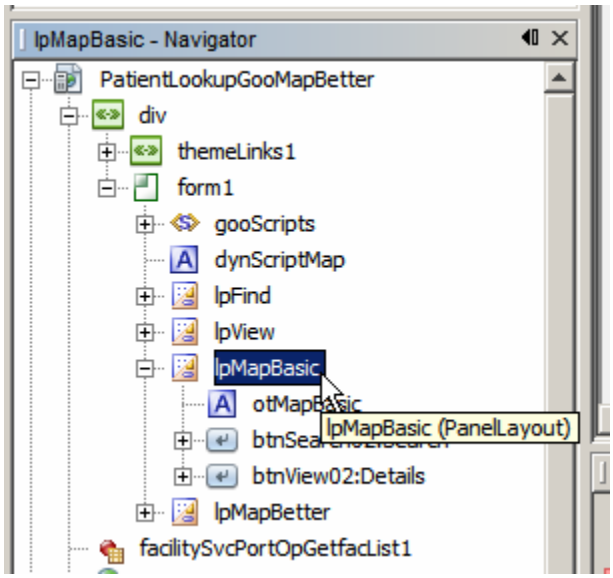
Right-click on the button and notice that the Edit Action option has the “btnSearch01_action() Event Handler” specified. We will leave it as is since all of the search buttons should do the same thing – return the user to the Lookup panel.

Copy the Details button form the lpMapBasic panel to the lpMapBetter panel. Set new buttons properties id: btnView05, style: font-size: 12px; left: 287px; top: 0px; position: absolute; width: 90px.

Once the buttons are copied, select the lpMap panel and set its visible property to false. We will have a better map to look at so we will hide the original map.



Note that once you uncheck the "visible" checkbox the panel and its content will disappear from the Design View window. You can still access it through the Navigator panel.



We will create a new Google Map, using a Google JavaScript functions which give us explicit control over certain aspects of the Map's appearance and functionality. The actual map will need to be displayed in a container on the page. Let's create this container. Switch to the JSP View tab, scroll down to the definition of the lpMapBetter panel and paste the following text as the first chilled of the structure, just above the first button, btnSearch03.

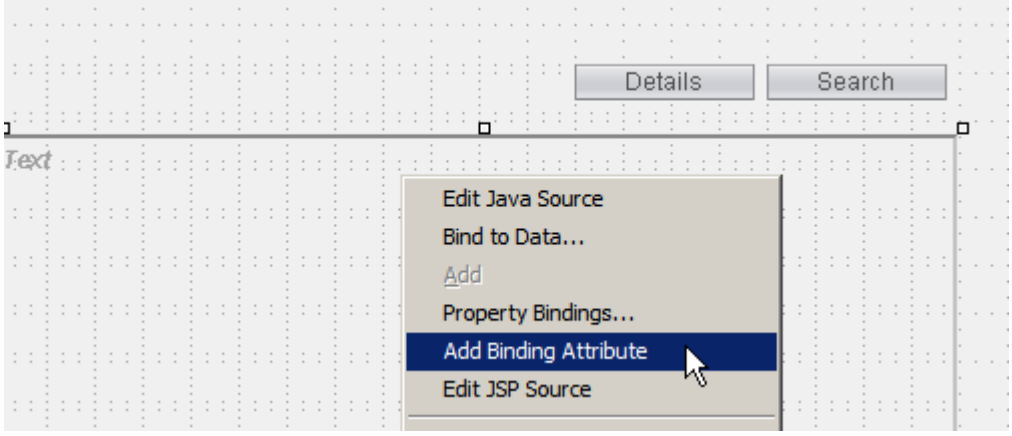
```
<h:outputText escape="false" id="betterMap" style="border-width: 2px;
border-style: inset; height: 400px; left: 0px; top: 35px; position:
absolute; width: 475px"/>
```

The JSP text with the outputText pasted should look like that shown below.

```
</webuijsf:panelLayout>
<webuijsf:panelLayout id="lpMapBetter" style="font-size: 12px; height: 452px; left: 0px; top: 0px; position: relati
    <h:outputText escape="false" id="BetterMap" style="border-width: 2px; border-style: inset; height: 400px; left:
    <webuijsf:button actionExpression="#{PatientLookupGooMapBetter.btnSearch01_action}" id="btnSearch1"
        style="font-size: 12px; left: 383px; top: 0px; position: absolute; width: 90px" text="Search"/>
    <webuijsf:button actionExpression="#{PatientLookupGooMapBetter.btnView02_action}" id="btnView03"
        style="font-size: 12px; left: 287px; top: 0px; position: absolute; width: 90px" text="Details"/>
</webuijsf:panelLayout>
```

This creates a SPAN element with id of betterMap, placed somewhat lower then the top of the panel. Switch to the Design mode, right-click on the new outputText element

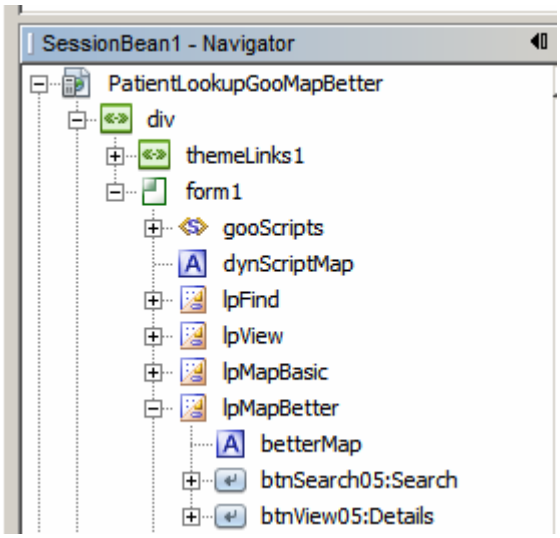
and choose “Add Binding Attribute”. We will need to access this component in the Java code later.



The result, in JSP View, should look something like this (where I re-formatted the text to somewhat improve readability).

```
103  <webuijsf:panelLayout binding="#{PatientLookupGooMapBetter.lpMapBetter}" id="lpMapBetter" style="fo
104      <h:outputText binding="#{PatientLookupGooMapBetter.betterMap}" escape="false" id="betterMap" st
105      <webuijsf:button actionExpression="#{PatientLookupGooMapBetter.btnSearch01_action}" id="btnSear
106          style="font-size: 12px; left: 383px; top: 0px; position: absolute; width: 90px" text="Searc
107      <webuijsf:button actionExpression="#{PatientLookupGooMapBetter.btnView02_action}" id="btnView05
108          style="font-size: 12px; left: 287px; top: 0px; position: absolute; width: 90px" text="Detai
109  </webuijsf:panelLayout>
```

Switch to the Design mode and inspect the hierarchy in the Navigator panel to make sure all components are ordered and nested correctly.



Now we are ready to add the Java code to get the better looking Google Map and manipulate the components we added.

The production of the better looking map will be accomplished by a JavaScript script. Switch to the JSP View mode and scroll to the source around where line 14 is in the following picture.

```
11          <p:portletPage>
12              <div style="height: 450px; position: relative; w
13                  <webuijsf:themeLinks binding="#{PatientLooku
14                  <webuijsf:form binding="#{PatientLookupGooMa
15                      <webuijsf:panelLayout binding="#{Patient
16                          <webuijsf:label id="label1" style="f
17                          <webuijsf:dropDown binding="#{Patien
18                              items="#{PatientLookupGooMapBett
19                          <webuijsf:label for="fldLocalID" id=
```

Paste the following text as the first child of the form1 node, between lines 14 and 15.

```
                    <webuijsf:script id="gooScripts">
function doMapFixed(sAddress, vLabelHTML) {
    var vRouteID = "";
   var vObj = document.getElementsByTagName('SPAN');
   for (var i = 0; i &lt; vObj.length; i++) {
         if (vObj[i].id.lastIndexOf(":betterMap") &gt; 0) {
               vRouteID = vObj[i].id;
         }
   }

    var map = new GMap2(document.getElementById(vRouteID));
    var geocoder = new GClientGeocoder();
    geocoder.getLatLng(
        sAddress,
        function(point) {
          if (!point) {
            alert(address + " not found");
          } else {
            map.setCenter(point, 13);
            var marker = new GMarker(point);
            map.addOverlay(marker);
            marker.openInfoWindowHtml(vLabelHTML);
          }
        }
      );
   map.setUIToDefault();
   map.enableGoogleBar();
}
                    </webuijsf:script>
```

The relevant fragment of the JSP should look like this:

```
11          <p:portletPage>
12              <div style="height: 450px; position: relative;
13                  <webuijsf:themeLinks binding="#{PatientLoo
14                  <webuijsf:form binding="#{PatientLookupGoo
15                      <webuijsf:script id="gooScripts">
16  function doMapFixed(sAddress, vLabelHTML) {
17      var vRouteID = "";
18      var vObj = document.getElementsByTagName('SPAN');
19      for (var i = 0; i &lt; vObj.length; i++) {
20          if (vObj[i].id.lastIndexOf(":betterMap") &gt; 0) {
21              vRouteID = vObj[i].id;
22          }
23      }
24
25      var map = new GMap2(document.getElementById(vRouteID))
26      var geocoder = new GClientGeocoder();
27      geocoder.getLatLng(
28          sAddress,
29          function(point) {
30            if (!point) {
31              alert(address + " not found");
32            } else {
33              map.setCenter(point, 13);
34              var marker = new GMarker(point);
35              map.addOverlay(marker);
36              marker.openInfoWindowHtml(vLabelHTML);
37            }
38          }
39        );
40      map.setUIToDefault();
41     map.enableGoogleBar();
42  }
43                      </webuijsf:script>
```

Notice the following, in the script:

1. This JavaScript script defines, but does not execute, the function **`doMapFixed(sAddress, vLabelHTML)`**, which accepts two parameters, sAddress – the patient address formatted in a way acceptable to Google Maps API and vLabelHTML – the HTML markup that will be used by the GoogleMap API to present a balloon/label describing the location on the map.

2. The first for loop inspects all SPAN elements in the Document Object Model, looking for one whose name ends with **`:betterMap`**, then setting saving the ID of that container element as a variable value for use later

3. Create a new GMap2 object, passing the ID of the container element into which to inject the map markup - http://code.google.com/apis/maps/documentation/reference.html

4. Modify the map object such that the map is centered at the Loattitude/Longtitude of the patient's address and the label marking that location uses the HTML markup provided

5. Change the appearance of the map controls by setting UI to defaults

6. Add Google Search control to the map

Items 3-6 in the list above are strictly Google Map API-related.

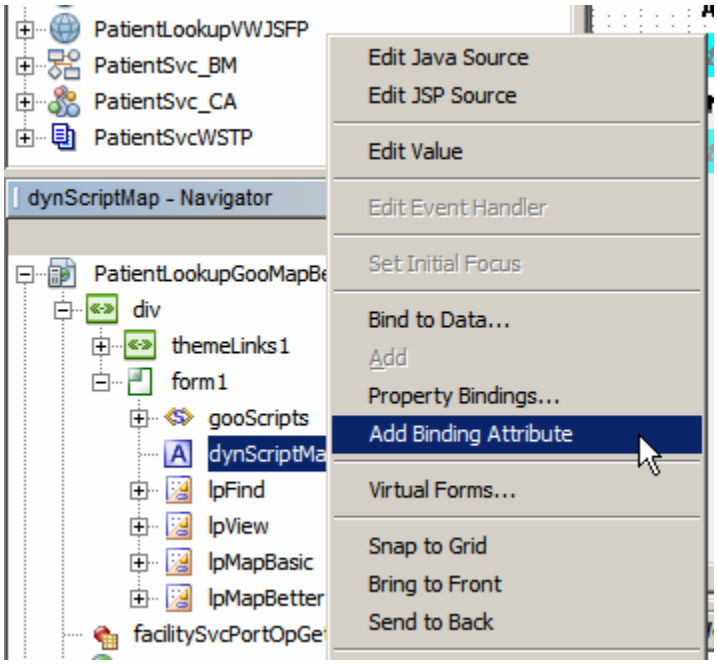
We will need to inject more JavaScript code at runtime to invoke this function and pass it appropriate parameters. To do this we need to add another outputText element to the page, immediately below the script block we just added.

Insert the following text just below the `</webuijsf:script>` tag.

```
<h:outputText escape="false" id="dynScriptMap"
style="visibility: hidden"/>
```

```
40  fault();
41  leBar();
42
43      </webuijsf:script>
44      <h:outputText escape="false" id="dynScriptMap" style="visibility: hidden"/>
45      <webuijsf:panelLayout binding="#{PatientLookupGooMapBetter.lpFind}" id="lpFind" st
46          <webuijsf:label id="label1" style="font-size: 12px; left: 0px; top: 0px; posit
47          <webuijsf:dropDown binding="#{PatientLookupGooMapBetter.ddFacilities}" id="ddl
```

Switch to Design View, expand the document elements hierarchy in the Navigateor pane, right-click the dynScriptMap outputText and choose “Add Binding Attribute”.



The outputText we just added will be populated with a JavaScript function invocation, to which we will pass patient address and the label to display on the map. It is necessary to do this as the values of the two parameters will change from patient to patient therefore can not be hardcoded in the JSP.

Finally, at the end of the page, after all elements are laid out and all the scripts are parsed, we need to trigger the script that will run the dynamic script with address and label text.

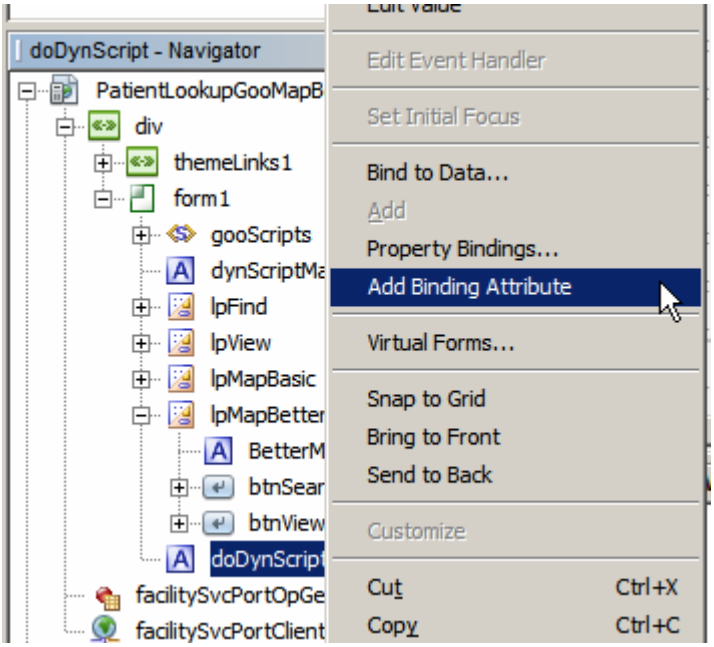
Switch back to the JSP mode, scroll down to the bottom and insert the following code between “</webuijsf:panelLayout>” and “</webuijsf:form>”, that is as the last child of the for1 object.

```
<h:outputText escape="false" id="doDynScript" style="visibility: hidden"/>
```

The JSP code will look like that in the picture below.

```
08          style="font-size: 12px; left: 287px; top: 0px; position: absolute; wi
09      </webuijsf:panelLayout>
10      <h:outputText escape="false" id="doDynScript" style="visibility: hidden"/>
11  </webuijsf:form>
12 'div>
```

Switch to Design mode, expand the page hierarchy in the Navigator pane, right-click on the doDynScript outputText element and choose “Add Binding Attribute”.



Now we will add some Java code to manipulate dynamic objects at runtime, including executing JavaScript scripts to dynamically modify page components.

Switch to Java mode and scroll to the prerender() method. Replace the statement setting visibility property of the panel lpMapBasic with a statement setting visibility of the panel lpMapBetter to false.

```
377         FacListReq flReq = new FacListReq();
378         flReq.setDummyString("DummyString");
379         facilitySvcPortOpGetfacList1.setMsgFacListReq(flReq);
380
381         lpFind.setVisible(true);
382         lpView.setVisible(false);
383         lpMapBetter.setVisible(false);
384         stMsgLocalID.setVisible(false);
385     }
```

Scroll to the end of the btnLookup_action() method and replace the statement that sets the visibility of the panel lpMapBasic to false with one that set the visibility of the panel lpMapBetter to false. When the lookup button is clicked we will either get redirected to the View panel or will remain on the Lookup panel.

```
507     // have data, set to display details
508     //
509             lpFind.setVisible(false);
510             lpView.setVisible(true);
511             lpMapBetter.setVisible(false);
512
513             return null;
514         }
```

Scroll down to the btnSearch01_action() method and replace the statement that sets the visibility of the panel lpMapBasic to false with one that set the visibility of panel the lpMapBetter to false as well.

```
516         public String btnSearch01_action() {
517             log("===>>> btnSearch01_action");
518             lpFind.setVisible(true);
519             lpView.setVisible(false);
520             lpMapBetter.setVisible(false);
521             fldLocalID.setValue("");
522             return null;
523         }
```

Repeat the process for method btnView02_action().

```
533         public String btnView02_action() {
534             log("===>>> btnView02_action");
535             lpFind.setVisible(false);
536             lpView.setVisible(true);
537             lpMapBetter.setVisible(false);
538             return null;
539         }
540     }
```

Modify the code in the method btnMap01_action() replacing lpMapBasic with lpMapBetter.

```
525         public String btnMap01_action() {
526             log("===>>> btnMap01_action");
527             lpFind.setVisible(false);
528             lpView.setVisible(false);
529             lpMapBetter.setVisible(true);
530             return null;
531         }
```

We will now add new code to the end of the btnLookup_action() method to prepare the address and the label, and trigger the JavaScript that will get us the improved Google Map.

At the end of the btnLookup_action() method, just before the statement “return null;” begin inserting additional code, stating with the following:

```
// additional map
//
// assemble label for the additional map
```

```
//
String sMapLabel = "";
sMapLabel += patRes.getFACILITY() + " / " + patRes.getLOCALID() + "<br>";
sMapLabel += stPatNames.getValue() + "<br>";
sMapLabel += sAddress;
```

Now add the following statements to define a JavaScript function that will pass the patient address and label text to the better map creation function. You can use what values are available to construct the label. This code makes a 3-line label with patient ID on one line, patient names on the second line and patient address on the last line.

```
// additional map with different appearance and
// patient-specific label
//
// this gives the script address and label input
// but does not execute the script
// it will be executed once trhe end of the page is rendered
// by which time all scriptes will have been "rendered"
// at the client side
//
String sScript = "";
sScript += "<script language='JavaScript'>\n";
sScript += "function doMapScriptWithParams() {\n";
sScript += "    doMapFixed('" + sAddress + "', '" + sMapLabel + "');\n";
sScript += "};\n";
sScript += "</script>\n";
dynScriptMap.setValue(sScript);
```

Finally, add the following statements to set the JavaScript function that will create the better looking Google Map, so it is executed when the browser gets to the script as it renders the page.

```
// inject a script which will execute
// first the script with patient address for fixed map
//
sScript = "";
sScript += "<script language='JavaScript'>\n";
sScript += "doMapScriptWithParams();\n";
sScript += "</script>\n";
doDynScript.setValue(sScript);
```

Note that this JavaScript fragment does not define a function, as did the previous script, but rather invokes a previously defined function as soon as the browser gets to render this part of the page (the end).

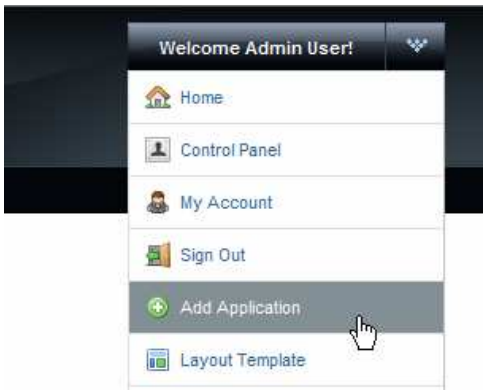
The Java code looks like this:

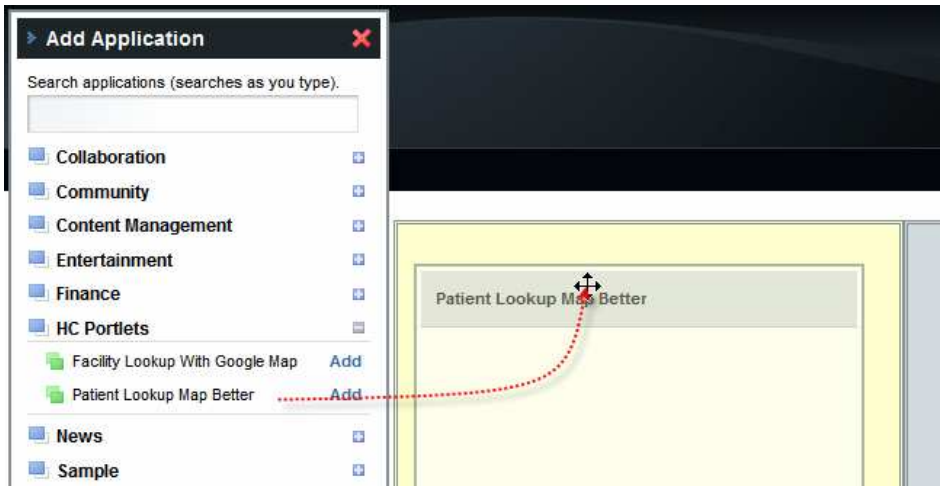
```
521  // additional map
522          //
523          // assemble label for the additional map
524          //
525          String sMapLabel = "";
526          sMapLabel += patRes.getFACILITY() + " / " + patRes.getLOCALID() + "<br>";
527          sMapLabel += stPatNames.getValue() + "<br>";
528          sMapLabel += sAddress;
529
530  // additional map with different appearance and
531  // patient-specific label
532  //
533  // this gives the script address and label input
534  // but does not execute the script
535  // it will be executed once trhe end of the page is rendered
536  // by which time all scriptes will have been "rendered"
537  // at the client side
538  //
539          String sScript = "";
540          sScript += "<script language='JavaScript'>\n";
541          sScript += "function doMapScriptWithParams() {\n";
542          sScript += "    doMapFixed('" + sAddress + "', '" + sMapLabel + "');\n";
543          sScript += "};\n";
544          sScript += "</script>\n";
545          dynScriptMap.setValue(sScript);
546
547  // inject a script which will execute
548  // first the script with patient address for fixed map
549  // then the script with route and directions, if any
550  //
551          sScript = "";
552          sScript += "<script language='JavaScript'>\n";
553          sScript += "doMapScriptWithParams();\n";
554          sScript += "</script>\n";
555          doDynScript.setValue(sScript);
556
557          return null;
```

All done. Let's deploy and exercise this portlet.

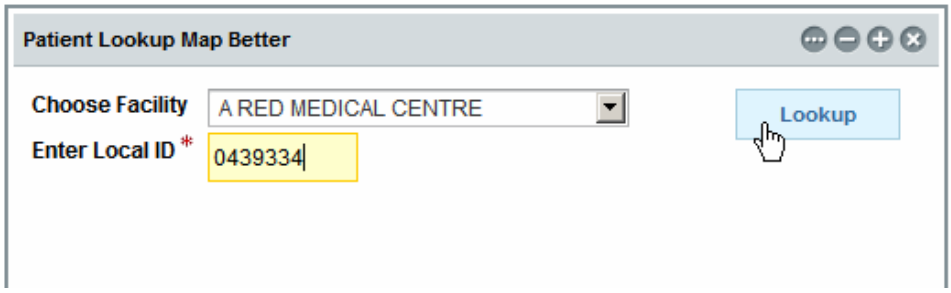
Right-click on the project name and choose Deploy.

Now that the portlet is deployed we need to add it to the portal page. If the early version of the portlet is on the portal page it needs to be removed.

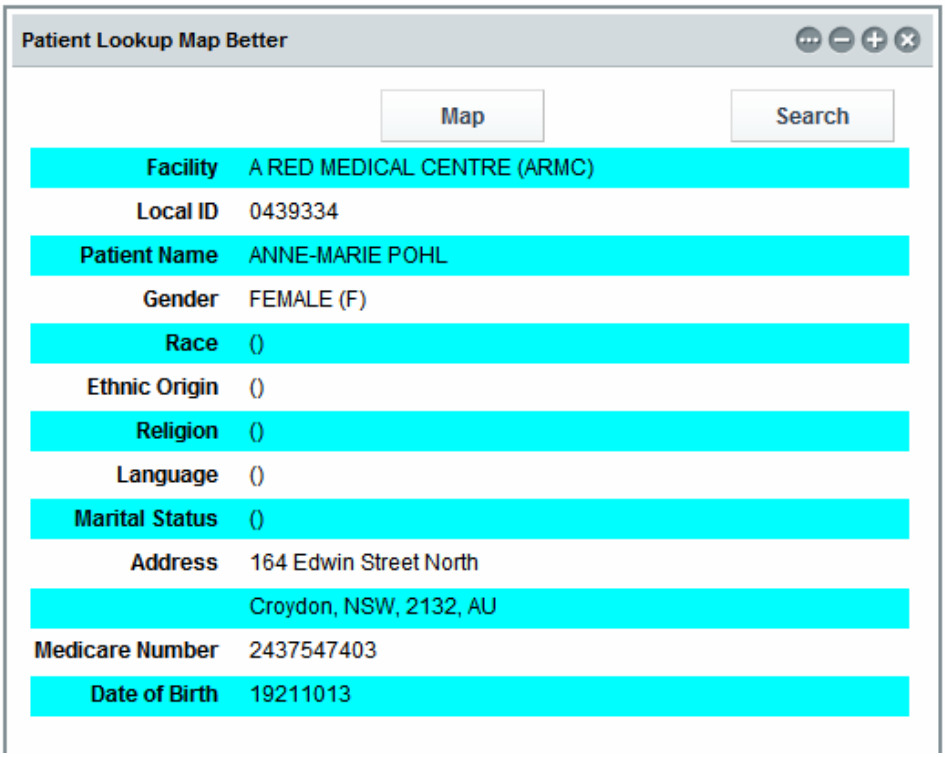




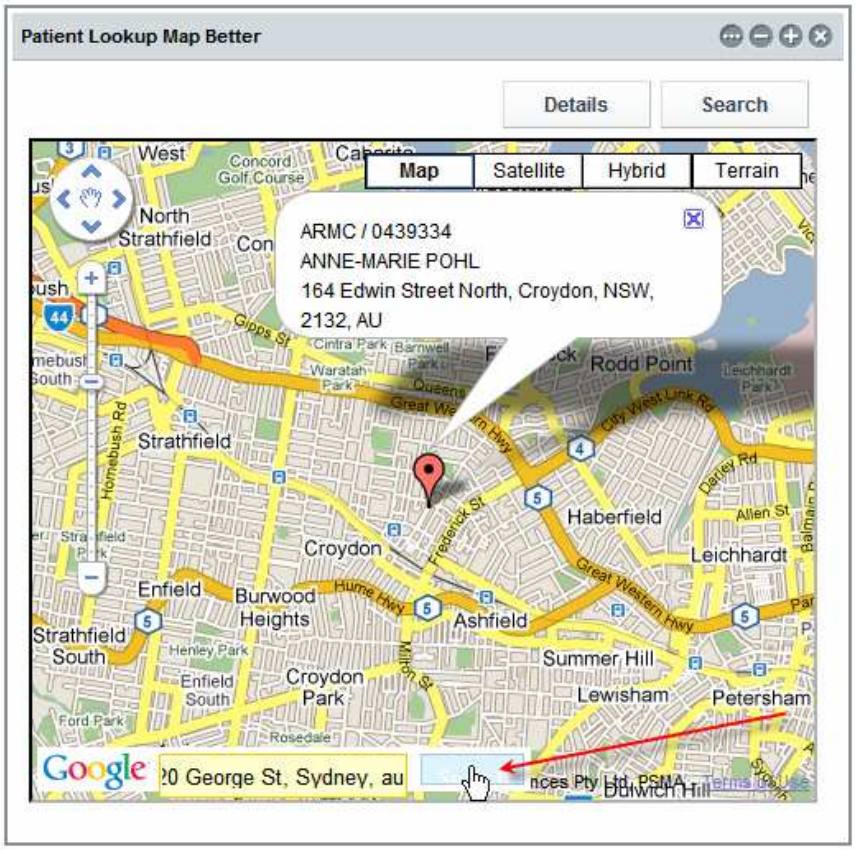
Choose A RED MEDICAL CENTRE from the list of facilities and enter 0439334 as Local ID. Click the Lookup button.



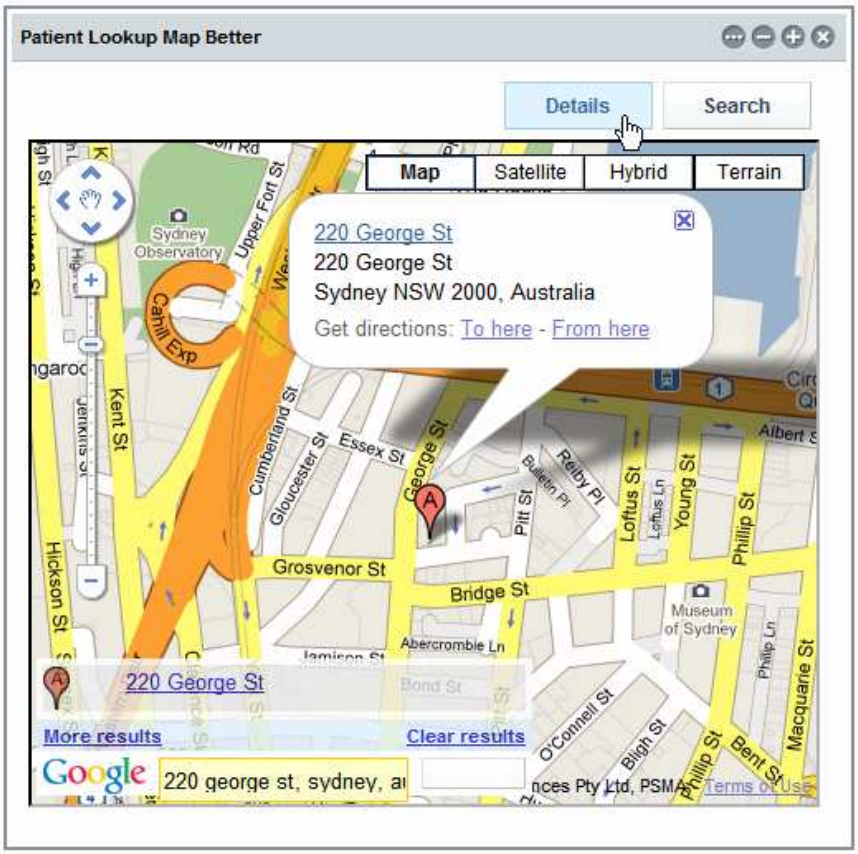
Click the Map button.



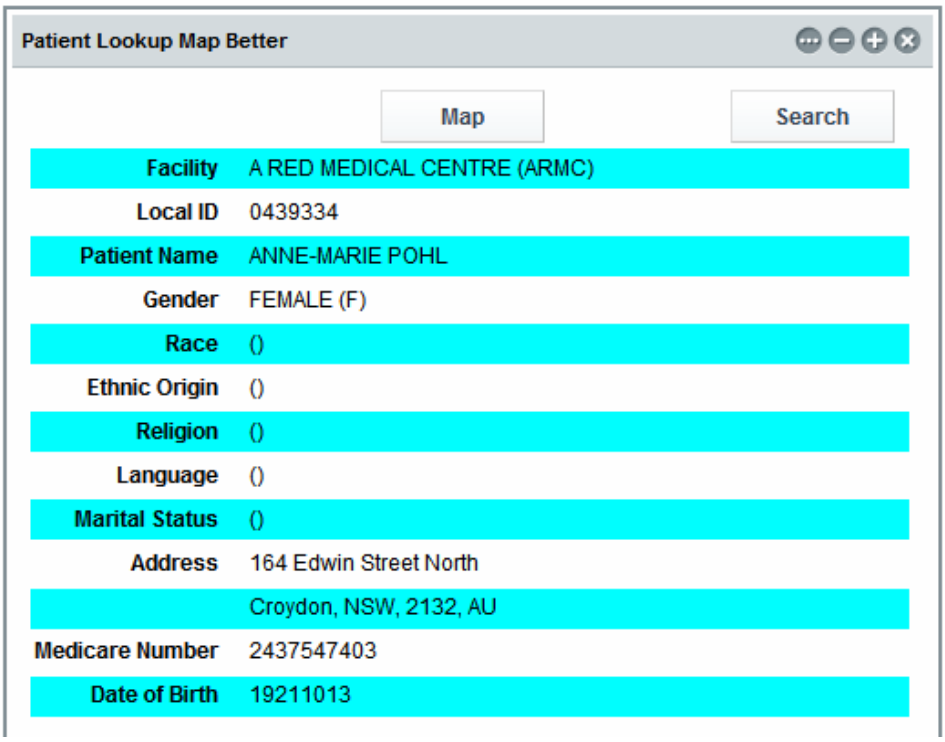
Enter an address into the Google Search entry box and click the Search button.



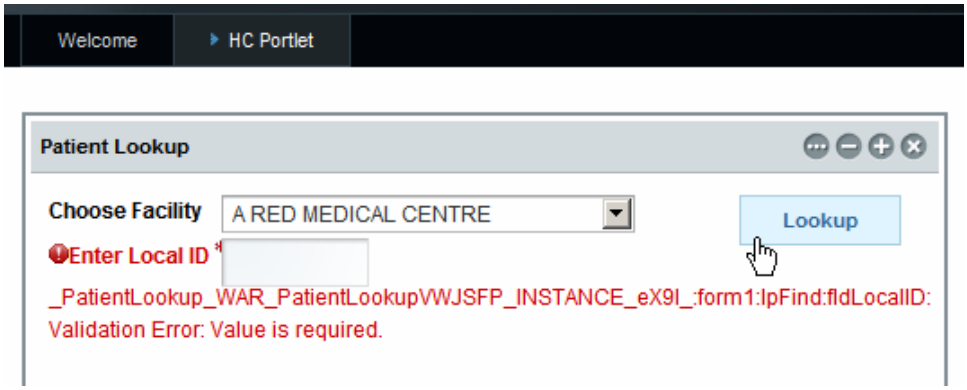
Click the Details button.



Click the Search button.



Click the Search button, noting that the Local ID field is cleared. Without entering anything into the Local ID field click the Lookup button. Notice the error message.



Enter a local id which is not in the database, for example 11. Click the Lookup button and notice the error message.

This error message was explicitly set in the btnLookup_action() method when the web service invocation returned with no record.

We are done. This is what it took to add a better looking Google Map to the portlet created before.

Summary

In this document we elaborated on a design of a JSR-286-compliant Visual Web JSF Portlet, deployed to the Sun Web Space Server 10 Portal, which used the Facility Service and the Patient Service Web Service as data providers. We added a panel with a better looking Google Map, obtained by directly manipulating Google Maps API JavaScript functions.

References

[1] MySQL Community Server and GUI Tools - Getting, Installing and Configuring, at http://blogs.sun.com/javacapsfieldtech/entry/mysql_community_server_and_gui.

[2] GlassFish ESB v2.1 download and installation, https://open-esb.dev.java.net/Downloads.html

[3] Adding Sun WebSpace Server 10 Portal Server functionality to the GlassFish ESB v2.1 Installation, http://blogs.sun.com/javacapsfieldtech/entry/adding_sun_webspace_server_10

[4] 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.

[5] GlassFish ESB v 2.1 - Creating a Healthcare Facility Web Service Provider, http://blogs.sun.com/javacapsfieldtech/entry/glassfish_esb_v_2_1

[6] GlassFish ESB v2.1, MySQL v5.1 - Creating a Patient Service Web Service Provider, http://blogs.sun.com/javacapsfieldtech/entry/glassfish_esb_v2_1_mysql1

[7] GlassFish ESB v2.1, MySQL v5.1 - Make HL7 v2.3.1 Delimited Messages from Custom Delimited Records with HL7 Encoder and HL7 BC, http://blogs.sun.com/javacapsfieldtech/entry/glassfish_esb_v2_1_mysql

[8] Healthcare Facility Mashup Portlet with Google Map - GlassFish v 2.1, Web Space 10, Web Service and REST Service, http://blogs.sun.com/javacapsfieldtech/entry/healthcare_facility_mashup_portlet_with

[9] GlassFish ESB v2.1, Web Space Server 10 - Creating a Patient Lookup Visual Web JSF Portlet, http://blogs.sun.com/javacapsfieldtech/entry/creating_a_patient_lookup_visual

[10] Healthcare Facility Mashup Portlet with Google Map - GlassFish v 2.1, Web Space 10, Web Service and REST Service, http://blogs.sun.com/javacapsfieldtech/entry/healthcare_facility_mashup_portlet_with

[11] GlassFish v 2.1, Web Space Server 10 - Patient Lookup Visual Web JSF Portlet with a basic Google Map, http://blogs.sun.com/javacapsfieldtech/entry/glassfish_v_2_1_web