

NetBeans 6.5.1, GlassFish v 2.1, Web Space Server 10 Patient Lookup Portlet with a Google Map, Route and Directions

Michael.Czapski@sun.com

July 2009

Table of Contents

Abstract.....	1
Introduction.....	1
Prerequisites.....	4
Copy Patient Lookup Project.....	4
Add Google Route Map and Directions components.....	8
Summary	24
References	24

Abstract

In this walkthrough I will add a Google Map showing a Route between patient's home address location and the location of the facility of record, as well as directions to follow along this route, to the Visual Web JSF Portlet developed in "Patient Lookup Visual Web JSF Portlet with a nicer looking Google Map", at http://blogs.sun.com/javacapsfieldtech/entry/patient_lookup_visual_web_jsf.

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. The portlet will also have a map showing the route along which to travel between patient's home location and the location of the facility of record, and directions to follow to get there. This Portlet will be deployed to the Sun FOSS Web Space Server 10 Portal.

The previous document [12], walked through development and deployment of the Patient Lookup Portlet with a better looking Google Map centered at the location identified by patient address, if any. In this document I will add to the Patient Lookup Portlet developed in [12] another map showing the route between patient's location and that of the facility of record as well as the directions to follow along that route.

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.

Patient Lookup Route ⋮ - + ✕

Choose Facility ▼

Enter Local ID *

Patient Lookup Route ⋮ - + ✕

Facility A RED MEDICAL CENTRE (ARMC)

Local ID 0439334

Patient Name ANNE-MARIE POHL

Gender FEMALE (F)

Race ()

Ethnic Origin ()

Religion ()

Language ()

Marital Status ()

Address 164 Edwin Street North
Croydon, NSW, 2132, AU

Medicare Number 2437547403

Date of Birth 19211013

Patient Lookup Route ⋮ - + ✕

ARMC / 0439334
ANNE-MARIE POHL
164 Edwin Street North, Croydon, NSW,
2132, AU

Google search the map

Patient Lookup Route [Close] [Zoom In] [Zoom Out] [Refresh]

Directions | Map | Details | Search

Map | Satellite | Hybrid | Terrain

Google search the map

Patient Lookup Route [Close] [Zoom In] [Zoom Out] [Refresh]

Route | Map | Details | Search

5. Turn left at Frederick St	0.9 km
6. Continue on Wattle St	0.7 km
7. Continue on Dobroyd Pde	1.4 km
8. Continue on City West Link Rd	2.9 km
9. Continue on Victoria Rd	0.5 km
10. Continue on Western Dstr	3.8 km
11. Continue on Bradfield Hwy	2.0 km
12. Slight left at Pacific Hwy	3.3 km
13. Turn left at Anglo Rd	0.3 km
14. Turn right at Greenwich Rd	0.3 km
15. Turn right to stay on Greenwich Rd	12 m
16. Turn right at Pacific Hwy	0.6 km
Destination will be on the left	

53 Pacific Hwy, St Leonards NSW 2065, Australia

Map data ©2009 MapData Sciences Pty Ltd, PSMA

Note that this walkthrough builds on the Patient Lookup Portlet with a nicer looking Google Map, built previously, and 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]

- Patient Lookup Visual Web JSF Portlet with a nicer looking Google Map has been deployed and tested [12]

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

Copy Patient Lookup Project

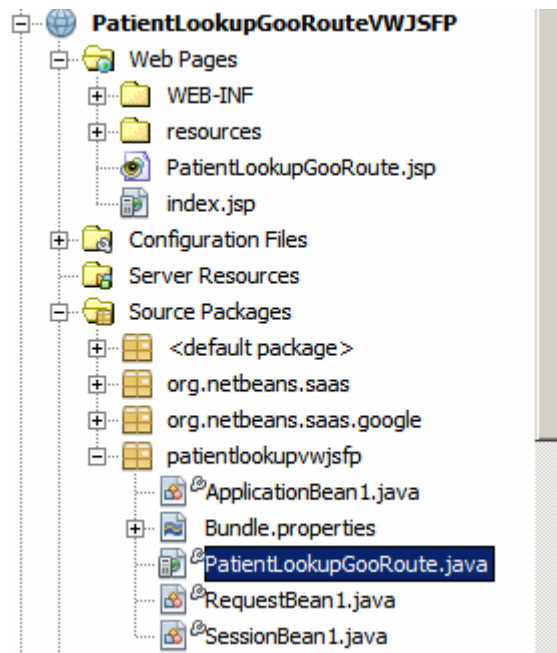
This document assumes that the Patient Lookup with a nicer looking Google Map Portlet, developed in [12], is available for cloning.

To save the effort we will copy the project PatientLookupGooMapBetterVWJSFP [12] and use it as the basis for elaboration.

Right-click the name of the project and choose Copy. Name the new project PatientLookupGooRouteVWJSFP 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 PatientLookupGooMapBetter.jsp page and choose Refactor -> Rename. Change the name to PatientLookupGooRoute, check the "Apply Rename on Comments" and click the "Refactor" button.

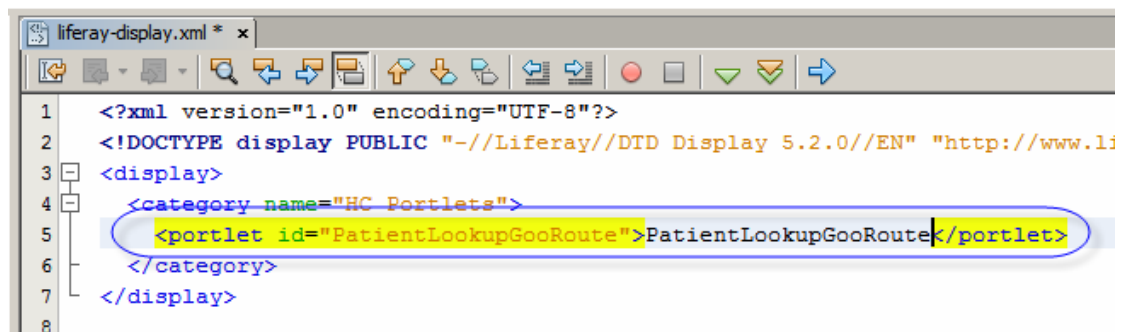
Note that the portlet backing class was also renamed.



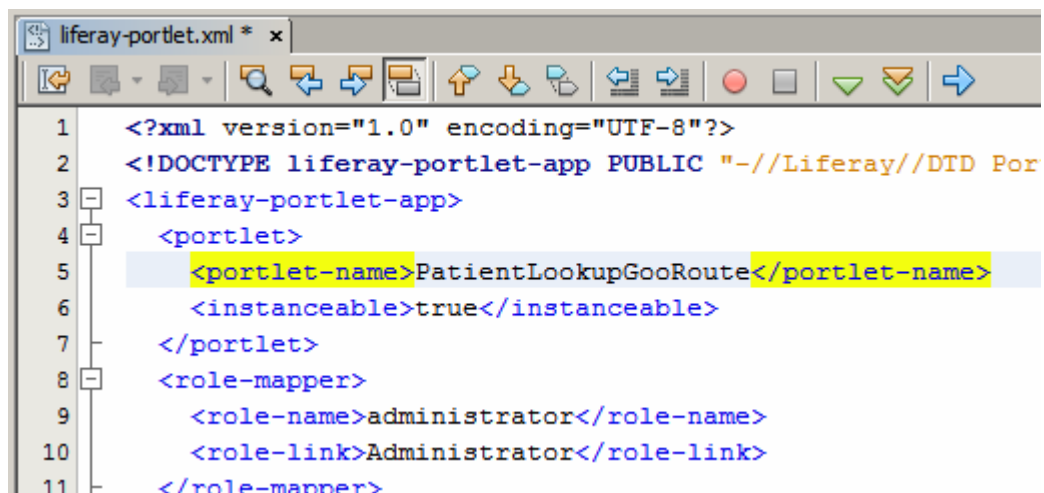
Alas, there are a few configuration files which must be manually modified.

Expand the "Configuration Files" folder.

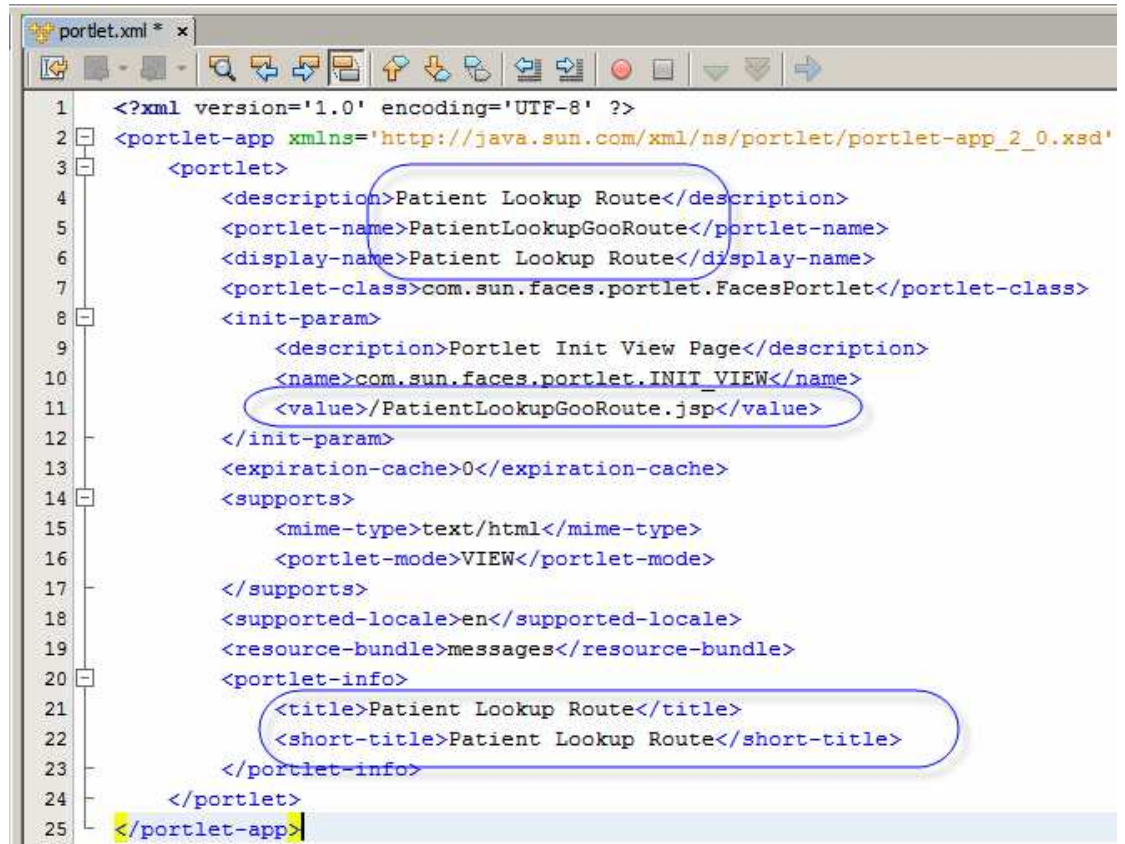
Open the liferay-display.xml and update portlet name and id.



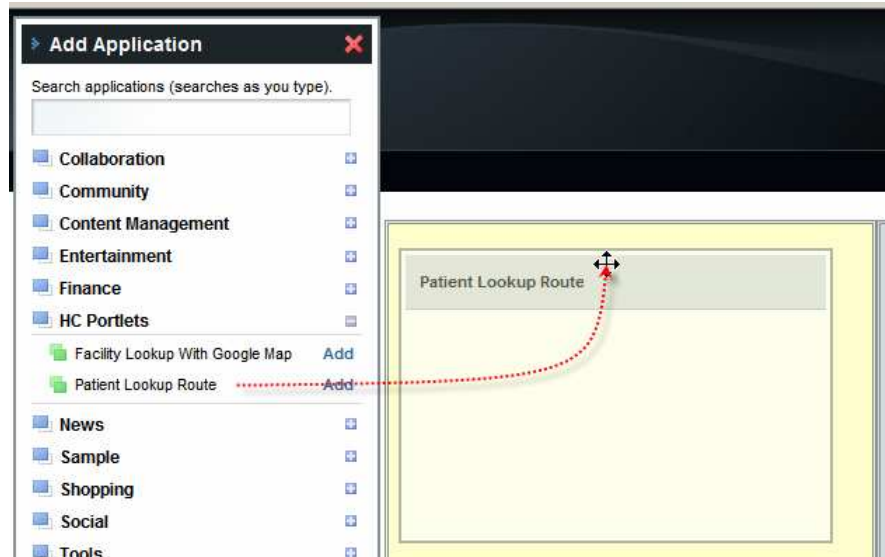
Open liferay-portlet.xml and update portlet-name.



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.



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



Patient Lookup Route [Close] [Refresh] [Zoom In] [Zoom Out]

Choose Facility: SYDNEY TECHNICAL HOSPITAL [v]

Enter Local ID*: 100000

Lookup

Patient Lookup Route [Close] [Refresh] [Zoom In] [Zoom Out]

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 Route [Close] [Refresh] [Zoom In] [Zoom Out]

Details Search

Map Satellite Hybrid Terrain

STC / 100000
JULIUS CAESAR EMP
Foro Romano, ROME, , it

Google search the map

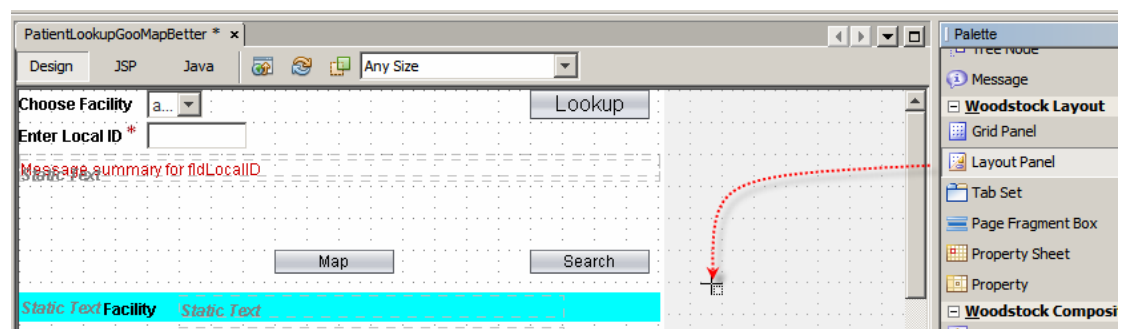
©2009 Tele Atlas - Terms of Use

It works for me.

Add Google Route Map and Directions 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. In [12] we used the fact that we are authenticated and that we can execute some of the Google JavaScript scripts to get and manipulate Google Maps objects for a better looking Google map. In this document we will add another Google Map with a route between two points marked on the map, and a separate panel with directions to follow to get from the starting point to the finishing point of the route.

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



Change the id property to lpRoute, panellayout property to “Grid Layout” and style property attributes to: font-size:12px, position: relative, height: 448px, width: 480px, left: 0px, top: 0px. Once done, the new panel will appear below all existing panels in 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 lpRoute panel. Set new buttons properties id: btnSearch03, 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 from the lpMapBetter panel to the lpRoute panel. Set new buttons properties id: btnView03, style: font-size: 12px; left: 287px; top: 0px; position: absolute; width: 90px.

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 lpRoute panel and paste the following text as the first child of the structure, just above the first button, btnSearch03.

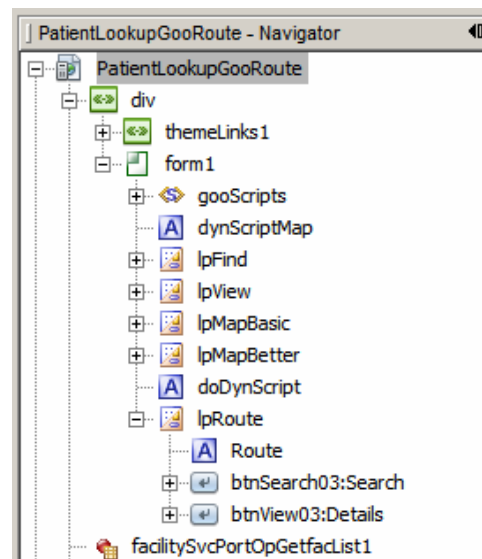
```
<h:outputText escape="false" id="Route" style="border-width: 2px; border-style: inset; height: 400px; left: 0px; top: 35px; position: absolute; width: 475px;"/>
```

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

```
109 </webuijsf:panellayout>
110 <h:outputText binding="#{PatientLookupGooRoute.doDynScript}" escape="false" id="doDynScript"
111 <webuijsf:panellayout binding="#{PatientLookupGooRoute.lpRoute}" id="lpRoute" style="font-siz
112 <h:outputText escape="false" id="Route" style="border-width: 2px; border-style: inset; he
113 <webuijsf:button actionExpression="#{PatientLookupGooRoute.btnSearch01_action}" id="btnSe
114 style="style: font-size: 12px; left: 383px; top: 0px; position: absolute; width: 90px
115 <webuijsf:button actionExpression="#{PatientLookupGooRoute.btnView02_action}" id="btnView
116 style="style: font-size: 12px; left: 287px; top: 0px; position: absolute; width: 90px
117 </webuijsf:panellayout>
118 </webuijsf:form>
```

This creates a SPAN element with id of Route, placed somewhat lower than the top of the panel.

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



We added a new panel to contain the map with the route. Now we need to add a panel to contain the directions.

Drag a Woodstock Layout Panel component to the canvas anywhere outside the existing layout panels.

Change the id property to lpDirections, panellayout property to "Grid Layout" and style property attributes to: font-size:12px, height: 448px; left: 0px; position: relative; width: 480px;. Once done, the new panel will appear below all existing panels in 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 lpDirections panel. Set new button's properties id: btnSearch04, 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 from the lpMapBetter panel to the lpDirections panel. Set new buttons properties id: btnView04, style: font-size: 12px; left: 287px; top: 0px; position: absolute; width: 90px.

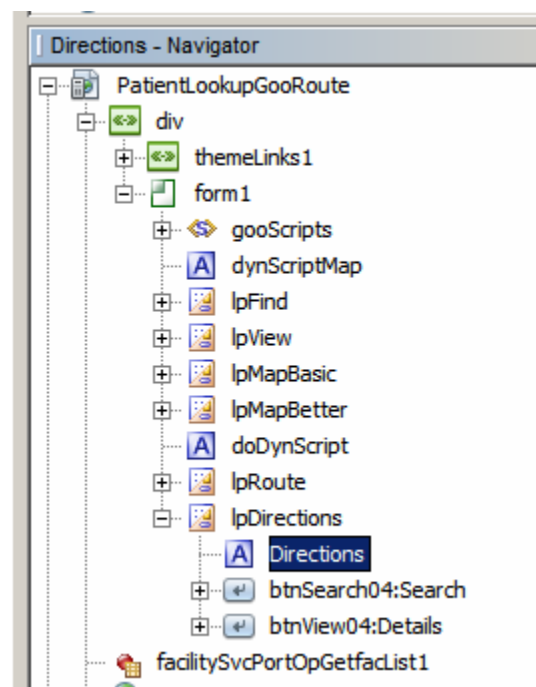
We will obtain Directions using a Google JavaScript functions. The actual directions will need to be displayed in a container on the page. Let's create this container. Switch to the JSP View tab, scroll down to the definition of the lpDirections panel and paste the following text as the first child of the structure, just above the first button, btnSearch04.

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

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

```
118 <webuijsf:panelLayout binding="#{PatientLookupGooRoute.lpDirections}" id="lpDirections" style="height: 448px; ]
119 <h:outputText escape="false" id="Directions" style="border-width: 2px; border-style: inset; height: 400px; ]
120 <webuijsf:button actionExpression="#{PatientLookupGooRoute.btnSearch01_action}" id="btnSearch04"
121 style="font-size: 12px; left: 383px; top: 0px; position: absolute; width: 90px" text="Search"/>
122 <webuijsf:button actionExpression="#{PatientLookupGooRoute.btnView02_action}" id="btnView04"
123 style="font-size: 12px; left: 287px; top: 0px; position: absolute; width: 90px" text="Details"/>
124 </webuijsf:panelLayout>
```

This creates a SPAN element with id of Directions, placed somewhat lower than the top of the panel. Switch to the Design mode. Inspect the hierarchy in the Navigator panel to make sure all components are ordered and nested correctly.



Now we are ready to add some code to get the Google Map with the Route and the Directions.

The production of the route map and directions will be accomplished by a JavaScript script.

Switch to the JSP View mode and scroll to the bottom of the existing JavaScript script. Insert the following text before the `</webuijsf:script>` tag.

```
function doDirections(vFromAddress, vToAddress) {
    var vRouteID = "";
    var vDirectionsID = "";
    var vObj = document.getElementsByTagName('SPAN');
    for (var i = 0; i < vObj.length; i++) {
        if (vObj[i].id.lastIndexOf(":Route") > 0) {
            vRouteID = vObj[i].id;
        }
        if (vObj[i].id.lastIndexOf(":Directions") > 0) {
            vDirectionsID = vObj[i].id;
        }
    }
}

if (GBrowserIsCompatible()) {
    var map = new GMap2(document.getElementById(vRouteID));
    map.setUIToDefault();
    map.enableGoogleBar();
    var gdir
        = new GDirections(map, document.getElementById(vDirectionsID));
    gdir.load("from: " + vFromAddress + " to: "
        + vToAddress, { "locale": "en_GB", "getSteps":true});
}
}
```

The relevant fragment of the JSP should look like this:



```
39 );
40 map.setUIToDefault();
41 map.enableGoogleBar();
42 }
43 function doDirections(vFromAddress, vToAddress) {
44     var vRouteID = "";
45     var vDirectionsID = "";
46     var vObj = document.getElementsByTagName('SPAN');
47     for (var i = 0; i < vObj.length; i++) {
48         if (vObj[i].id.lastIndexOf(":Route") > 0) {
49             vRouteID = vObj[i].id;
50         }
51         if (vObj[i].id.lastIndexOf(":Directions") > 0) {
52             vDirectionsID = vObj[i].id;
53         }
54     }
55 }
56 if (GBrowserIsCompatible()) {
57     var map = new GMap2(document.getElementById(vRouteID));
58     map.setUIToDefault();
59     map.enableGoogleBar();
60     var gdir = new GDirections(map, document.getElementById(vDirectionsID));
61     gdir.load("from: " + vFromAddress + " to: " + vToAddress, { "locale": "en_GB", "getSteps":true});
62 }
63 }
64
65 </webuijsf:script>
66 <h:outputText binding="#{PatientLookupGooRoute.dynScriptMap}" escape="false" id=
67 <webuijsf:panellayout binding="#{PatientLookupGooRoute.lpFind}" id="lpFind" styl
```

Notice the following, in the script:

1. This JavaScript script defines, but does not execute, the function `doDirections(vFromAddress, vToAddress)`, which accepts two parameters, the patient address formatted in a way acceptable to the Google Map API and the facility address formatted in a way acceptable to the Google Map API.
2. The first for loop inspects all SPAN elements in the Document Object Model, looking for one whose name ends with `:Route`, and one whose name ends in `:Directions`, saving the IDs of these container elements as a variable values 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. Change the appearance of the map controls by setting UI to defaults
5. Add Google Search control to the map
6. Get directions between patient's address and facility address and overlay the map with the route between the two points

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

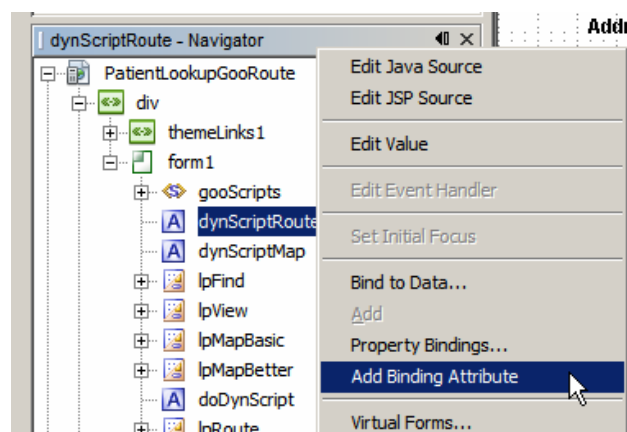
We will need to inject more JavaScript code at runtime to invoke this function and pass it appropriate parameters. To do this we need to add another `outputText` element to the page, immediately below the script block we just added.

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

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

```
64 |
65 | </webuijsf:script>
66 | <h:outputText escape="false" id="dynScriptRoute" style="visibility: hidden" />
67 | <h:outputText binding="#{PatientLookupGooRoute.dynScriptMap}" escape="false" id="dynS
68 | <webuijsf:panelLayout binding="#{PatientLookupGooRoute.lpFind}" id="lpFind" style="he
69 | <webuijsf:label id="label1" style="font-size: 12px; left: 0px; top: 0px; positior
```

Switch to Design View, expand the document elements hierarchy in the Navigator pane, right-click the `dynScriptRoute` `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 facility address. 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.

The Details/View panel has a button, Map, which allows the user to switch to the Map panel. We now have two more panels, the Route panel and the Directions panel. We need to add two more buttons, Route and Directions to all relevant panels – `lpView`, `lpMap`, `lpRoute` and `lpDirections`, so that the user can switch between panels. Each panel will be “missing” the button that would switch to it if clicked in another panel, much as we have done for the View and Map panels. The Map panel does not have the Map button and the View panel does not have the Details button.

Let’s drag a new Button Woodstock component to the `lpView` panel and set button properties to: `is: btnDirections01`, `style: left: 0px; top: 0px; position: absolute; width: 90px`, `text: Directions`. Right-click the button and choose “Add Binding Attribute”. Right-click the button and choose “Edit action Event Handler”. This will switch NetBeans to Java View with a skeleton of the `btnDirections01_action()` method displayed. We will fill in the skeleton later. For now switch back to the Design View.

Copy `btnDirections01` button and paste it into `lpMapBetter` panel. Set the properties to: `id: btnDirections05`, `style: left: 0px; top: 0px; position: absolute; width: 90px`. Right-click the button and choose “Add Binding Attribute”.

Copy the `lpView` -> `btnDirections01` button and paste it into the `lpRoute` panel. Set properties: `id: btnDirections03`, `style: left: 0px; top: 0px; position: absolute; width: 90px`. Right-click the button and choose “Add Binding Attribute”.

We have a Directions button in all panels except the search panel and the directions panel itself.

Let’s now add a Route button to panels `lpView`, `lpMapBetter` and `lpDirections`, leaving out the search panel and the route panel.

Let’s drag a new Button Woodstock component to the `lpView` panel and set button properties to: `id: btnRoute01`, `style: left: 95px; top: 0px; position: absolute; width: 90px`, `text: Route`. Right-click the button and choose “Add Binding Attribute”. Right-click the button and choose “Edit action Event Handler”. This will switch NetBeans to Java View with a skeleton of the `btnRoute01_action()` method displayed. We will fill in the skeleton later. For now switch back to the Design View.

Copy `btnRoute01` button and paste it into `lpMapBetter` panel. Set the properties to: `id: btnRoute05`, `style: left: 95px; top: 0px; position: absolute; width: 90px`. Right-click the button and choose “Add Binding Attribute”.

Copy the `lpView` -> `btnRoute01` button and paste it into the `lpDirections` panel. Set properties: `id: btnDirections04`, `style: left: 95px; top: 0px; position: absolute; width: 90px`. Right-click the button and choose “Add Binding Attribute”.

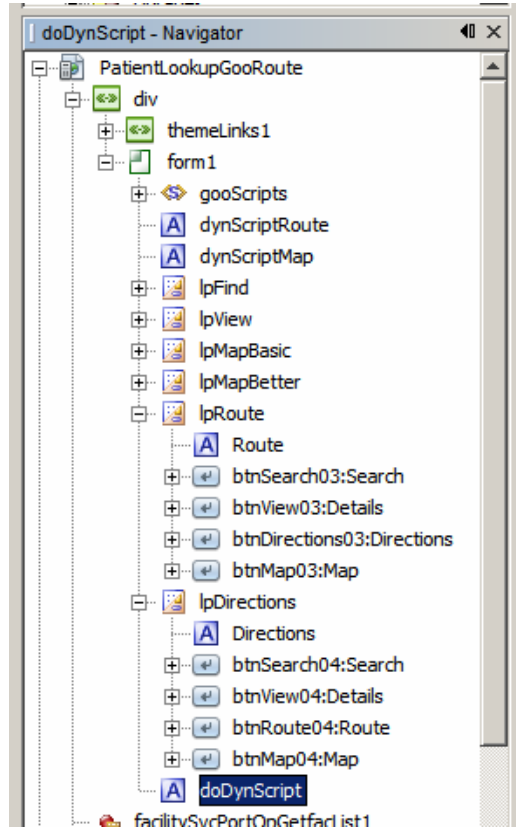
We have a Route button in all panels except the search panel and the route panel itself.

To complete adding visual components we need to add the Map button to the two new panels.

Copy the `lpView` -> `btnMap01` button and paste it into the `lpRoute` panel. Set properties: `id: lpMap03`, `style: left: 191px; top: 0px; position: absolute; width: 90px`. Right-click the button and choose “Add Binding Attribute”.

Copy the IpView -> btnMap01 button and paste it into the IpDirections panel. Set properties: id: IpMap04, style: left: 191px; top: 0px; position: absolute; width: 90px. Right-click the button and choose "Add Binding Attribute".

To ensure the components are correctly nested inspect the Navigator panel The outputText doSynScript must be the last child of the form form1.



In the original portlet we are testing to see if patient address is available. If it is not we hide the Map button in the View / Details panel. If we can not display a map with patient's location then we can not display the route or directions either – we must hide them as well. This is why we added binding attributes to the Route and Directions buttons. Furthermore, even if patient's address is available it does not necessarily mean that facility address is available. Both are needed to obtain a route and directions. So if the facility address is not available we must hide the Route and Directions buttons in the IpView and IpMapBetter panels, so as to prevent a user from switching to these panels.

Let's now add 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. Add statements setting visibility property of the panel IpRoute and Ip Directions to false.

```

482  @Override
483  public void prerender() {
484      log("====>>> prerender");
485      init();
486
487      FacListReq flReq = new FacListReq();
488      flReq.setDummyString("DummyString");
489      facilitySvcPortOpGetfacList1.setMsgFacListReq(flReq);
490
491      lpFind.setVisible(true);
492      lpView.setVisible(false);
493      lpMapBetter.setVisible(false);
494      lpRoute.setVisible(false);
495      lpDirections.setVisible(false);
496      stMsgLocalID.setVisible(false);
497  }

```

Scroll to the location in the btnLookup_action() method where panel visibility is set and add statements that set the visibility of panels lpRoute and lpDirectins to false. When the lookup button is clicked we will either get redirected to the View panel or will remain on the Lookup panel.

```

616
617  // have data, set to display details
618  //
619      lpFind.setVisible(false);
620      lpView.setVisible(true);
621      lpMapBetter.setVisible(false);
622      lpRoute.setVisible(!blHaveRecord);
623      lpDirections.setVisible(!blHaveRecord);
624
625  // additional map
626  //

```

Scroll down to the btnSearch01_action() method and add statements that set the visibility of panels lpRoute and lpDirectins to false.

```

664  public String btnSearch01_action() {
665      log("====>>> btnSearch01_action");
666      lpFind.setVisible(true);
667      lpView.setVisible(false);
668      lpMapBetter.setVisible(false);
669      lpRoute.setVisible(false);
670      lpDirections.setVisible(false);
671      fldLocalID.setValue("");
672      return null;
673  }

```

Repeat the process for method btnView02_action().

```

683 public String btnView02_action() {
684     log("====>>> btnView02_action");
685     lpFind.setVisible(false);
686     lpView.setVisible(true);
687     lpMapBetter.setVisible(false);
688     lpRoute.setVisible(false);
689     lpDirections.setVisible(false);
690     return null;
691 }

```

Repeat the process for the method btnMap01_action().

```

675 public String btnMap01_action() {
676     log("====>>> btnMap01_action");
677     lpFind.setVisible(false);
678     lpView.setVisible(false);
679     lpMapBetter.setVisible(true);
680     lpRoute.setVisible(false);
681     lpDirections.setVisible(false);
682     return null;
683 }

```

Fill in skeleton btnRoute01_action method body with the following code:

```

704 public String btnRoute01 action() {
705     lpFind.setVisible(false);
706     lpView.setVisible(false);
707     lpMapBetter.setVisible(false);
708     lpRoute.setVisible(true);
709     lpDirections.setVisible(false);
710     return null;
711 }

```

Fill in skeleton btnDirections01_action method body with the following code:

```

695 public String btnDirections01_action() {
696     lpFind.setVisible(false);
697     lpView.setVisible(false);
698     lpMapBetter.setVisible(false);
699     lpRoute.setVisible(false);
700     lpDirections.setVisible(true);
701     return null;
702 }

```

We will now add new code to the end of the btnLookup_action() method to get the address of the facility of record for the patient, if any, work out whether we can use it, set the dynamic parameters to the JavaScript script that prepares the route map and directions, and set the page to execute the scripts at page render time.

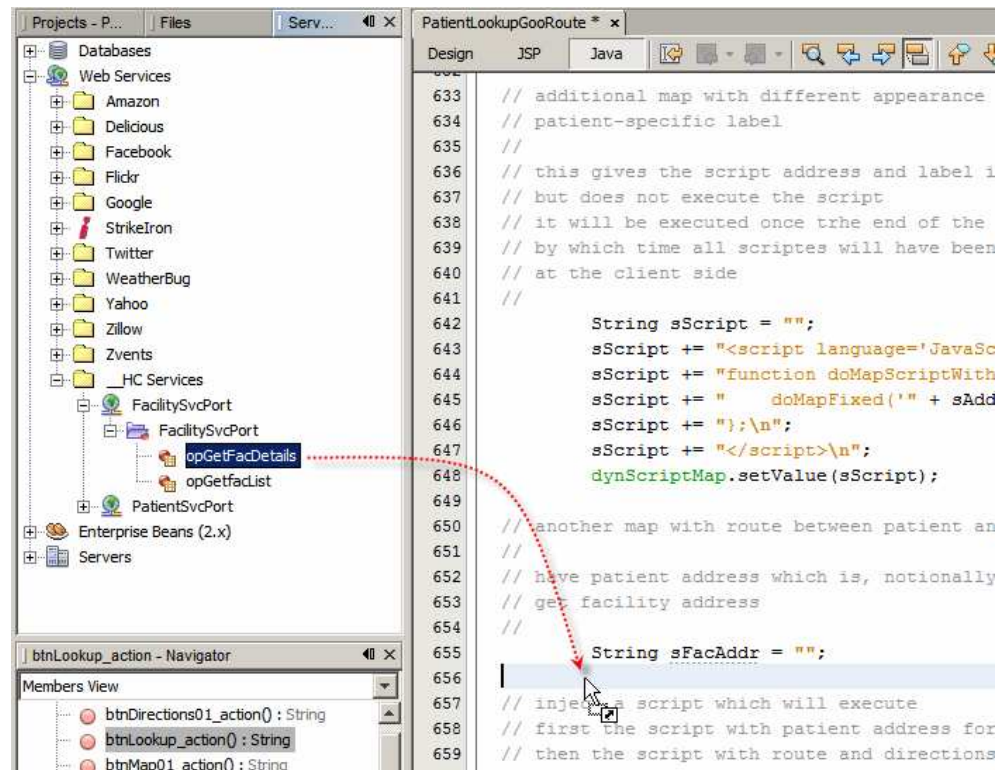
After the statement "dynScriptMap.setValue(sScript);", just before the comment "// inject a script which will execute" begin inserting additional code, starting with the following:


```

// another map with route between patient and facility
//
// have patient address which is, notionally, mappable
// get facility address
//
String sFacAddr = "";

```

Now switch from the Project Explorer tab to the Services Explorer tab, expand Web Services node through FacilitySvcPort and drag the web service operation opGetFacilityDetails to the Java source window, dropping it on the line below the String = sFacAdd = "";



Accept the default in the dialogue box that appears.

Replace the slab of boilerplate code that was inserted, starting with try and ending with the brace closing the catch, with the following code:

```

try {
    FacDetailsReq msgFacDetailsReq = new FacDetailsReq();
    msgFacDetailsReq.setFacCode(patRes.getFACILITY());
    FacilitySvcService service = new FacilitySvcService();
    FacilitySvcPortType port = service.getFacilitySvcPort();
    FacDetailsRes result1 = port.opGetFacDetails(msgFacDetailsReq);

    if (result1.getAddressLine1() != null &&
        result1.getAddressLine1().trim().length() > 0) {
        sFacAddr += result1.getAddressLine1().trim();
        if (result1.getSuburbTown() != null &&
            result1.getSuburbTown().trim().length() > 0) {
            sFacAddr += ", " + result1.getSuburbTown().trim();
            if (result1.getState() != null &&
                result1.getState().trim().length() > 0) {
                sFacAddr += ", " + result1.getState().trim();
            }
        }
    }
}

```

```

    }
    if (result1.getPostCode() != null &&
        result1.getPostCode().trim().length() > 0) {
        sFacAddr += ", " + result1.getPostCode().trim();
    }
    if (result1.getCountry() != null &&
        result1.getCountry().trim().length() > 0) {
        sFacAddr += ", " + result1.getCountry().trim();
    }
}
}

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

```

To resolve issues right-click in the source window and choose Fix Imports.

What we are doing here is invoking the Facility Details web service operation, looking at the result and setting the value of sFacAddr to the address of the facility, formatted for use with the Google Map service, or setting it to an empty string, if one of the conditions of address validity is not met.

For the facility address to be valid, the street address and city/suburb/town must be present. Just how valid such an abbreviated address is will depend on the location of the user. Google Maps service seems to be making assumptions about missing data based on requester's location.

If the address is not empty, and the patient's address seems reasonable, we can inject the JavaScript script which will invoke the Google map function that obtains the route and the directions, given patient's and facility's addresses. If not, we will hide the Route and Directions buttons.

Add the following statements below the closing brace of the try-catch statement:

```

btnRoute01.setVisible(false);
btnDirections01.setVisible(false);

```

This is what the new code should look like:

```

651         sScript += "</script>\n";
652         dynScriptMap.setValue(sScript);
653
654         // another map with route between patient and facility
655         //
656         // have patient address which is, notionally, mappable
657         // get facility address
658         //
659         String sFacAddr = "";
660
661         try {
662             FacDetailsReq msgFacDetailsReq = new FacDetailsReq();
663             msgFacDetailsReq.setFacCode(patRes.getFACILITY());
664             FacilitySvcService service = new FacilitySvcService();
665             FacilitySvcPortType port = service.getFacilitySvcPort();
666             FacDetailsRes result1 = port.opGetFacDetails(msgFacDetailsReq);
667
668             if (result1.getAddressLine1() != null && result1.getAddressLine1().trim().length() > 0) {
669                 sFacAddr += result1.getAddressLine1().trim();
670                 if (result1.getSuburbTown() != null && result1.getSuburbTown().trim().length() > 0) {
671                     sFacAddr += ", " + result1.getSuburbTown().trim();
672                     if (result1.getState() != null && result1.getState().trim().length() > 0) {
673                         sFacAddr += ", " + result1.getState().trim();
674                     }
675                     if (result1.getPostCode() != null && result1.getPostCode().trim().length() > 0) {
676                         sFacAddr += ", " + result1.getPostCode().trim();
677                     }
678                     if (result1.getCountry() != null && result1.getCountry().trim().length() > 0) {
679                         sFacAddr += ", " + result1.getCountry().trim();
680                     }
681                 }
682             }
683
684             } catch (Exception ex) {
685                 ex.printStackTrace();
686             }
687
688             btnRoute01.setVisible(false);
689             btnDirections01.setVisible(false);
690

```

Immediately following the two statements that hide the two buttons add the following code:

```

if (sFacAddr.trim().length() > 0) {
    // can try to display route from patient to facility
    // and directions
    //
    btnRoute01.setVisible(true);
    btnDirections01.setVisible(true);

    log("===>>> From " + sAddress + " to " + sFacAddr);

    // provide address details to the script
    // the script will not get executed yet - it will be
    // when the page gets rendered further
    //
    sScript = "";
    sScript += "<script language='JavaScript'>\n";
    sScript += "function doRouteScriptWithParams() { doDirections('"
        + sAddress + "', '" + sFacAddr + "');}";
    sScript += "</script>\n";
    dynScriptRoute.setValue(sScript);
}

```

If the facility address is valid we set visibility of the route and directions buttons to true and inject a script that supplies patient and facility address to the script that actually gets the new Google map.

Finally, insert invocation of the new script into the code block that already invokes the doMapScriptWithParams, to set the JavaScript function that will create the better looking Google Map and the Route and Directions objects, 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 += "doRouteScriptWithParams();\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:

```

696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733

        btnRoute01.setVisible(false);
        btnDirections01.setVisible(false);

        if (sFacAddr.trim().length() > 0) {

            // can try to display route from patient to facility
            // and directions
            //
            btnRoute01.setVisible(true);
            btnDirections01.setVisible(true);

            log("====>>> From " + sAddress + " to " + sFacAddr);

            // provide address details to the script
            // the script will not get executed yet - it will be
            // when the page gets rendered further
            //
            sScript = "";
            sScript += "<script language='JavaScript'>\n";
            sScript += "function doRouteScriptWithParams() { doDirections('"
            sScript += "</script>\n";
            dynScriptRoute.setValue(sScript);
        }

// inject a script which will execute
// first the script with patient address for fixed map
// then the script with route and directions, if any
//
sScript = "";
sScript += "<script language='JavaScript'>\n";
sScript += "doMapScriptWithParams();\n";
sScript += "doRouteScriptWithParams();\n";
sScript += "</script>\n";
doDynScript.setValue(sScript);

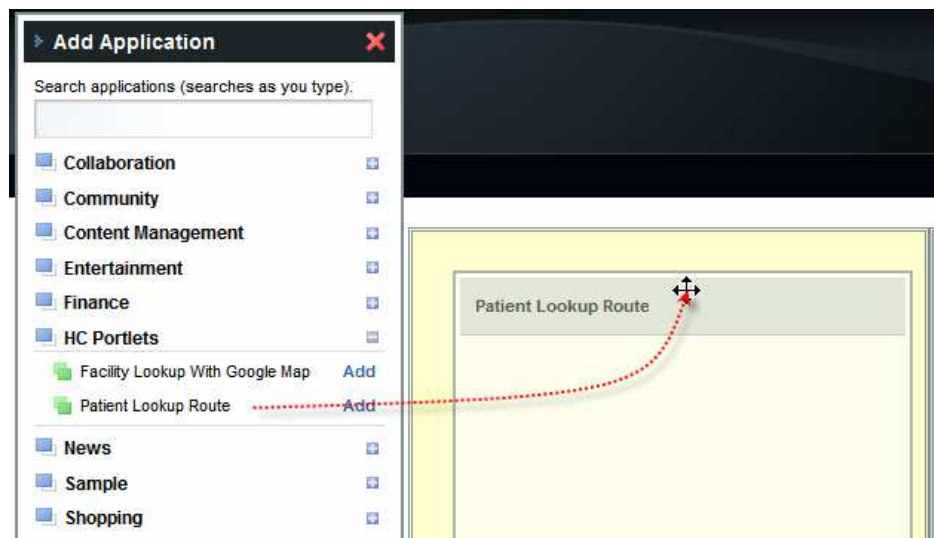
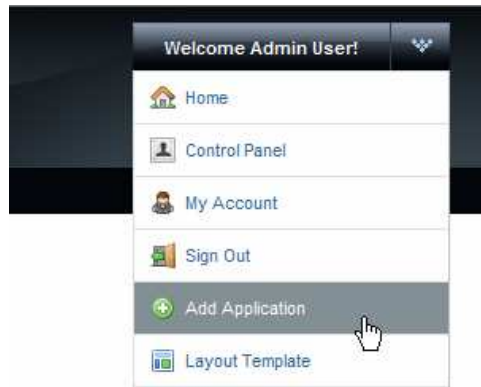
return null;
}

```

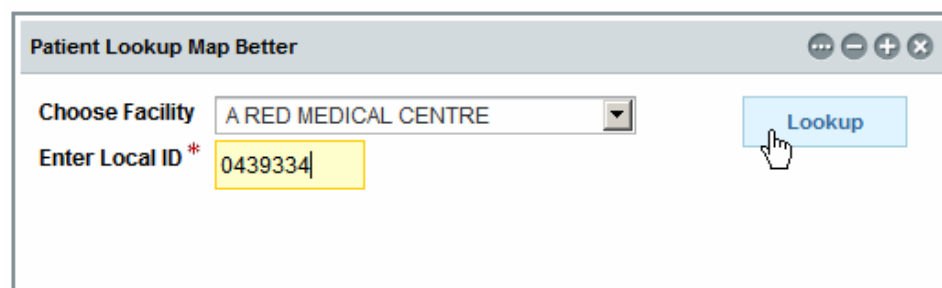
All done. Let's deploy and exercise this portlet.

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.

It is expected that at least one facility and at least one patient have reasonable addresses in order to test the route and directions functionality. If you don't see Map, and Route and Direction buttons, verify validity of addresses for the patient and for the patient's facility of record.

Patient Lookup Route

Directions Route **Map** Search

Facility A RED MEDICAL CENTRE (ARMC)
Local ID 0439334

Patient Name ANNE-MARIE POHL
Gender FEMALE (F)

Race ()
Ethnic Origin ()
Religion ()
Language ()

Marital Status ()
Address 164 Edwin Street North
Croydon, NSW, 2132, AU

Medicare Number 2437547403
Date of Birth 19211013

Click the Route button.

Patient Lookup Route

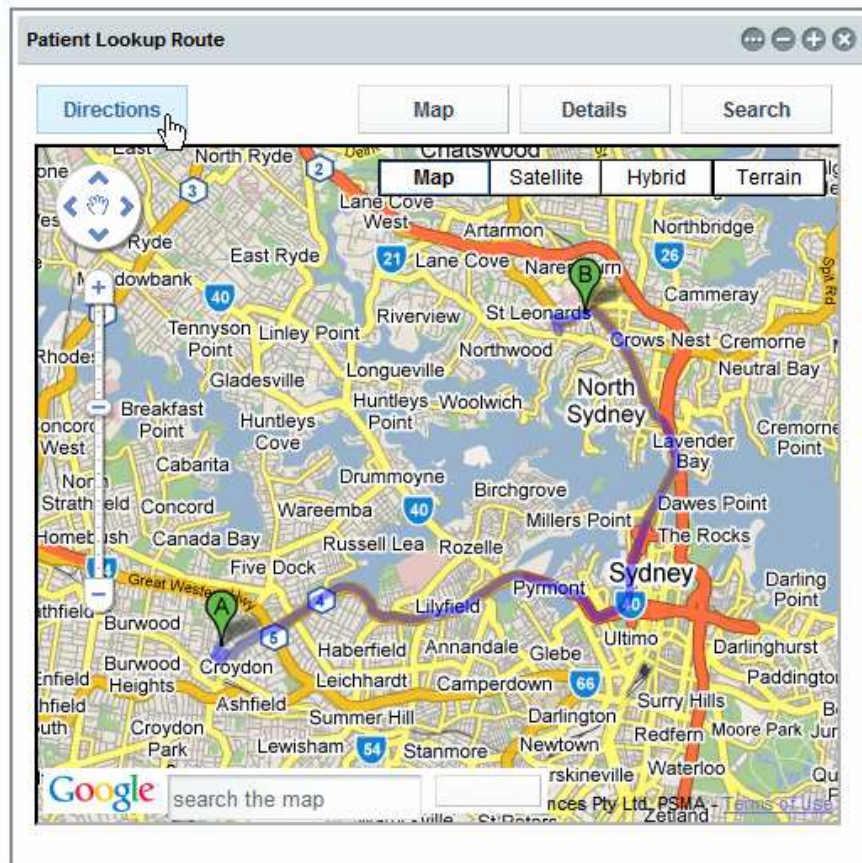
Directions **Route** Details Search

Map Satellite Hybrid Terrain

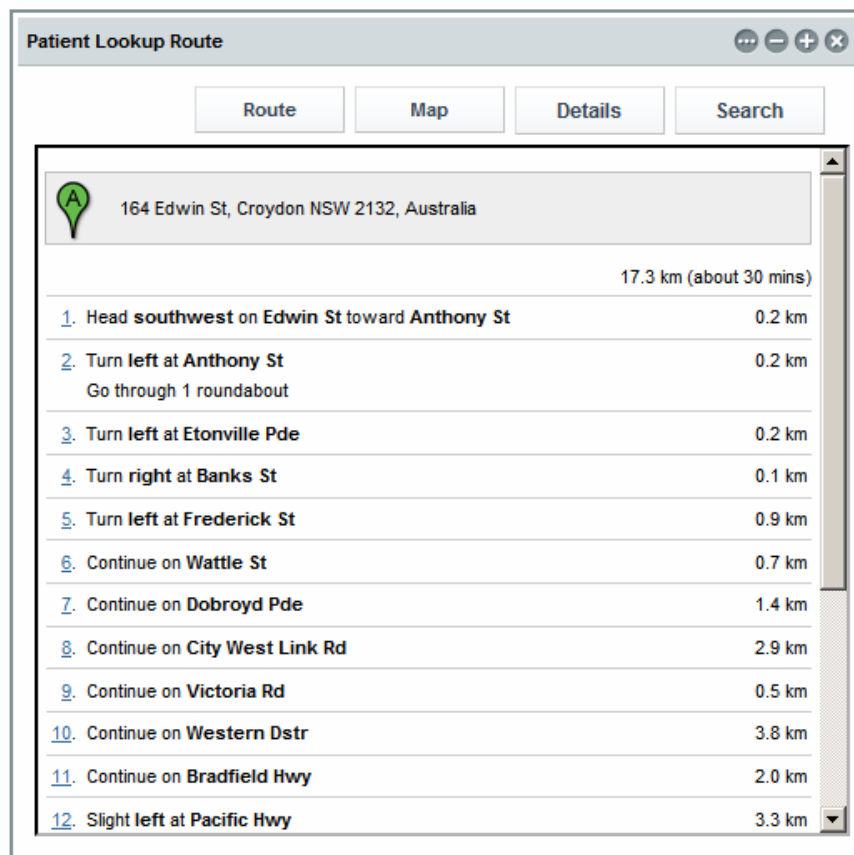
ARMC / 0439334
ANNE-MARIE POHL
164 Edwin Street North, Croydon, NSW,
2132, AU

Google search the map

Click the Directions button.



Click the Search button.



We are done. This is what it took to add a Google Map that shows a Route between two locations, and directions to follow, 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 couple of panels with a Google Map that shows a Route between two locations, and directions to follow, 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_webpace_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
- [12] Patient Lookup Visual Web JSF Portlet with a nicer looking Google Map, http://blogs.sun.com/javacapsfieldtech/entry/patient_lookup_visual_web_jsf