HD3C11 – Google Maps

|  |  |
| --- | --- |
| **Product and Focus**  HANA Platform/SAPUI5 | **MOTIVATION**  Maps are a ubiquitous feature of mobile applications. This cases uses the Google Map API to display a map.  **PREREQUISITES**  HD3C06 – The Base Application |
| **Target Audience**  Undergrduate/Graduate Beginner to Intermediate |
| **Author**  Ross Hightower |
| https://bgoerke.files.wordpress.com/2013/05/section1.png | |

# The Google Map API

The [Google Map API](https://developers.google.com/maps/) is easy to use, flexible and powerful way to display maps using JavaScript and HTML. Although, it is usually free there are some restrictions as described [here](https://developers.google.com/maps/licensing). We will barely scratch the surface in this case but if you want to learn more, [here](https://developers.google.com/maps/tutorials/) is an excellent tutorial and [here](https://developers.google.com/maps/documentation/javascript/) is the JavaScript API documentation.

# Create a GoogleMap View

### Add the View to the App Navigation

Add a new object for the Lists view to the views.json file. You can find and icon [here](https://openui5.hana.ondemand.com/iconExplorer.html). Also, add a route to the Component.js file.

Create a new xml view and controller in the view folder called **GoogleMap**.

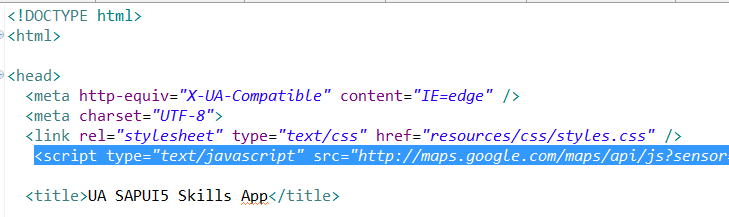
### Index.html

Add the code shown below to the index.html file.

|  |
| --- |
| <script type=*"text/javascript"* src=*"http://maps.google.com/maps/api/js?sensor=false&libraries=places"*></script> |

Listing 1

This line initializes the Google Map API.



### GoogleMap.view.xml

Add the code shown below to the GoogleMap.view.xml file:

|  |
| --- |
| <mvc:View controllerName=*"ui5.controller.GoogleMap"*  xmlns:core=*"sap.ui.core"*  xmlns:mvc=*"sap.ui.core.mvc"*  xmlns:l=*"sap.ui.layout"*  xmlns:html=*"http://www.w3.org/1999/xhtml"*  xmlns:commons=*"sap.ui.commons"*  xmlns=*"sap.m"*>  <Page id=*"mainPage"* enableScrolling=*"false"*  showNavButton=*"true"* navButtonPress=*"handleNavButtonPress"*  title=*"Start Coding Maps!"*>  <VBox fitContainer=*"true"* justifyContent=*"Center"* alignItems=*"Center"*>  <HBox fitContainer=*"true"* justifyContent=*"Center"* alignItems=*"Center"*>  <Input id=*"inpSearch"* editable=*"true"* value=*""* maxLength=*"80"*/>  <Button id=*"bntSearch"* text=*"Search"* tap=*"mapSearch"* />  </HBox>  <HBox id=*"map\_canvas"* fitContainer=*"true"* justifyContent=*"Center"* alignItems=*"Center"* class=*"*myMap*"* />  </VBox>  </Page>  </mvc:View> |

Listing 2

This code creates a VBox control that contains two HBox controls. The first HBox contains an Input control and a Button control. These will be used to enter and submit an address. The second HBox control will contain the map.

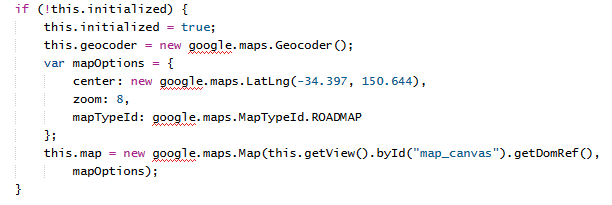
### GoogleMap.controller.js

Add the code shown below to the GoogleMap.controller.js file:

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function(Controller) {  "use strict";  return Controller.extend("ui5.controller.GoogleMap", {  onInit: function() {  this.router = sap.ui.core.UIComponent.getRouterFor(this);  },  handleNavButtonPress: function() {  this.router.navTo("Master", {  from: "SplitAppExercise"  });  },  onAfterRendering: function() {  if (!this.initialized) {  this.initialized = true;  this.geocoder = new google.maps.Geocoder();  var mapOptions = {  center: new google.maps.LatLng(-34.397, 150.644),  zoom: 8,  mapTypeId: google.maps.MapTypeId.ROADMAP  };  this.map = new google.maps.Map(this.getView().byId("map\_canvas").getDomRef(),  mapOptions);  }  },  mapSearch: function() {  var map = this.map;  var address = this.getView().byId("inpSearch").getValue();  this.geocoder.geocode({  'address': address  }, function(results, status) {  if (status == google.maps.GeocoderStatus.OK) {  map.setCenter(results[0].geometry.location);  var marker = new google.maps.Marker({  map: map,  position: results[0].geometry.location  });  } else {  alert('Geocode was not successful for the following reason: ' + status);  }  });  }  });  }); |

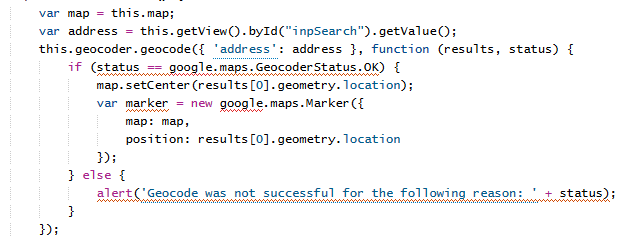
Listing 3

The code in the onInit and handleNavButtonPress should be familiar by now. The onAfterRendering function is called after the HTML elements have been created by SAPUI5. This function is used to manipulate the DOM before the page is shown. In our case it is used to draw the initial map.



The first thing that this function does is check to make sure the map hasn’t been initialized already. It then creates a Google geocoder object and names it this.geocoder. The this refers to the controller and by naming the variable this.geocoder, the variable will be available in every function in the controller. The mapOptions object contains configuration data for the map including the latitude and longitude of Sydney Australia. Feel free to change the initial location. You can find a free geocoding service [here](http://geoservices.tamu.edu/Services/Geocode/Interactive/). You can find the various Google map types [here](https://developers.google.com/maps/documentation/javascript/maptypes). Next, the Google Map API is called to draw the map. The first argument to the Map function is a reference to the DIV element in which we will draw the map is retrieved by the getDomRef() function. Notice the map is placed in a variable called this.map. The this, again, makes this variable available to the other functions in the controller.

The mapSearch function is called when the button is pressed. This code creates a new variable, map, and sets it equal to the map created when the view loaded. It then retrieves the address entered by the user in the Input control. The Google geocode service is used to convert the address to latitude and longitude and a callback is provided to handle the result from the geocoder service..



If the address was successfully geocoded, the map is refreshed with the new coordinates using the setCenter function of the map object. The Marker function sets a marker at the coordinates of the address.

### styles.css

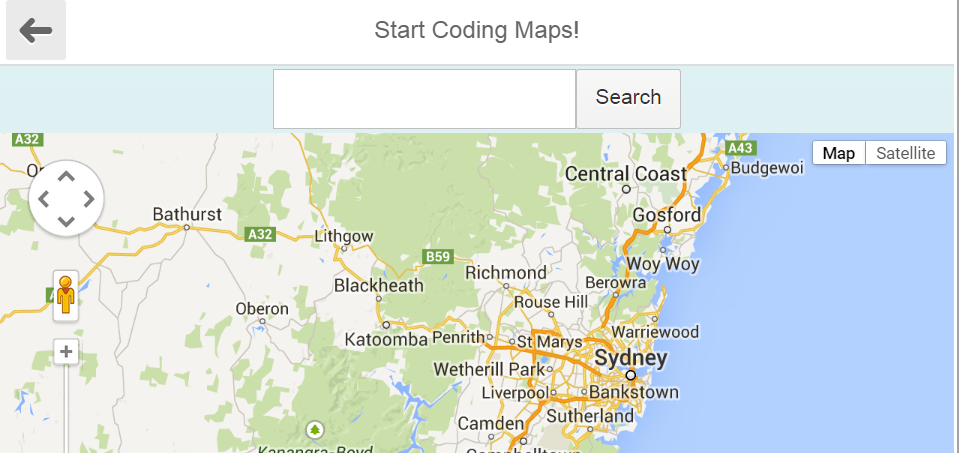
Add the following style to the styles.css file.

|  |
| --- |
| *.myMap* {  height: *100%*;  width: *100%*  } |

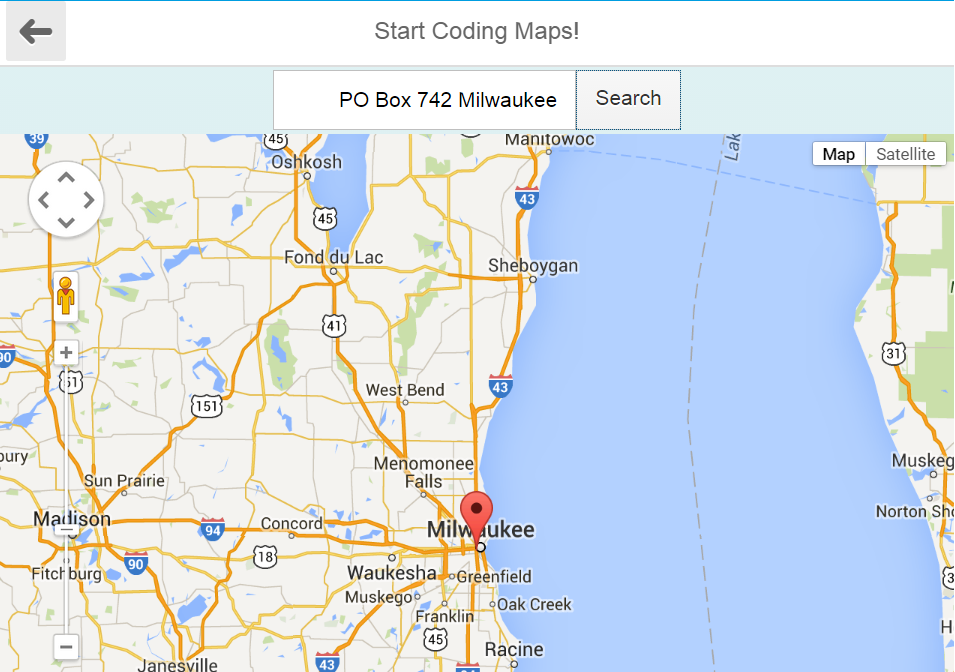
Listing 4

The . in front of myMap is a class selector so the style is assigned to our HBox control used to display the map because it was assigned the myMap class.

Run the application.



Enter an address or a city name and click Search to see a new map:

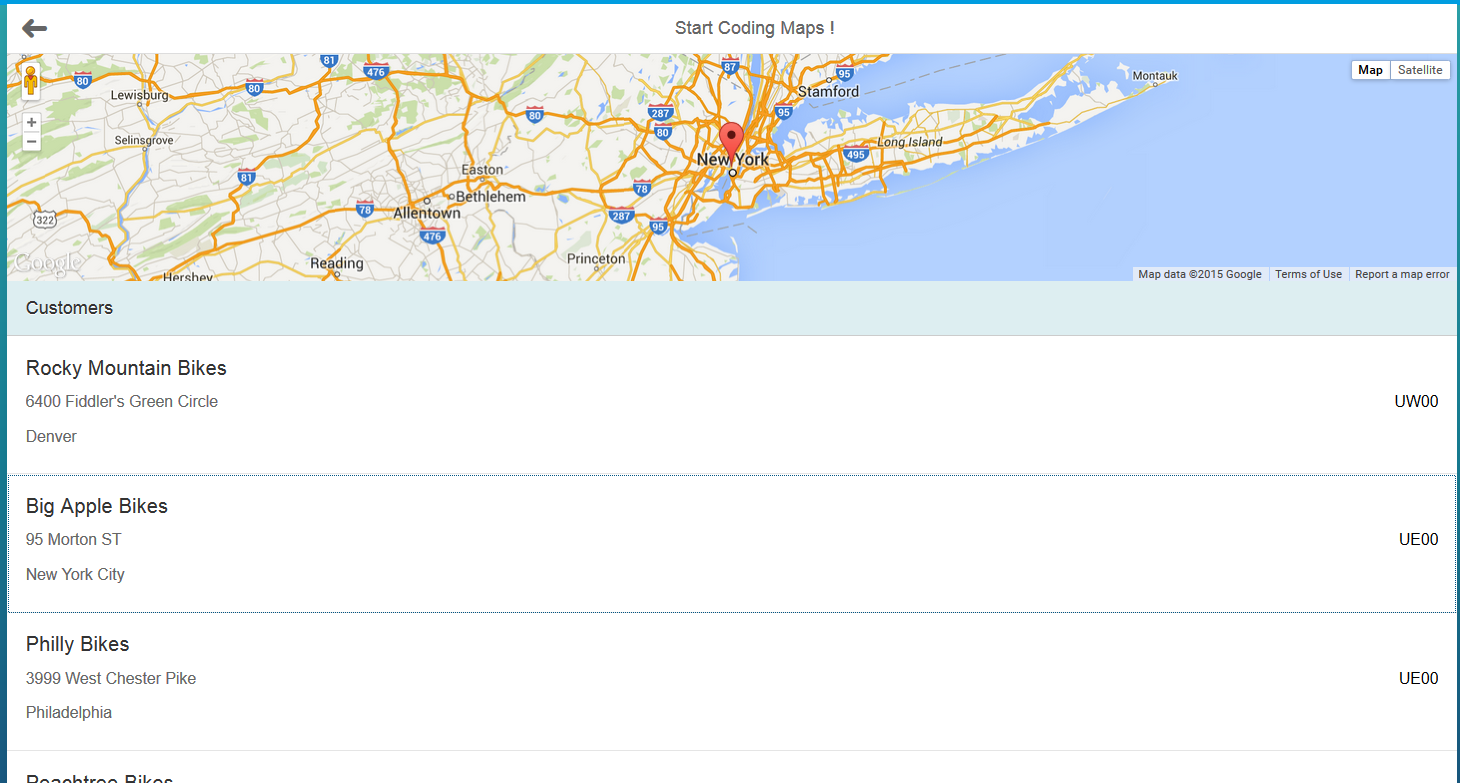


# Exercise

This exercise builds on the exercise at the end of the Lists case. Create a new view that implements the Customer list you created in the Lists case exercise. Add a map to the top of the detail above the list as shown in the figure below.

To make this work you will have to:

* Assign a different id to the HBox in which the map is drawn in this case. Remember to use this id in the getDomRef() function.
* Eliminate the fitContainer="true" justifyContent="Center" alignItems="Center" attributes in the new HBox.
* Create a new style that has a fixed height (in px) and assign it to the new HBox.



The code to draw the map is very similar to the code in the GoogleMap controller. One difference is you assign the mapSearch event handler to the press event of the list items and replace the code that creates the address variable with code shown below. This retrieves the address from the list item that was clicked.

|  |
| --- |
| var context = evt.getSource().getBindingContext('gbi');  var street = evt.getSource().getBindingContext('gbi').getProperty('ADDRESS');  var city = evt.getSource().getBindingContext('gbi').getProperty('CITY');  var postal\_code = evt.getSource().getBindingContext('gbi').getProperty('POSTAL\_CODE');  var address = street + ", " + city + ", " + postal\_code; |

Listing 5

Make sure you edit the code to insert your field names. You can leave the onAfterRendering code as is.