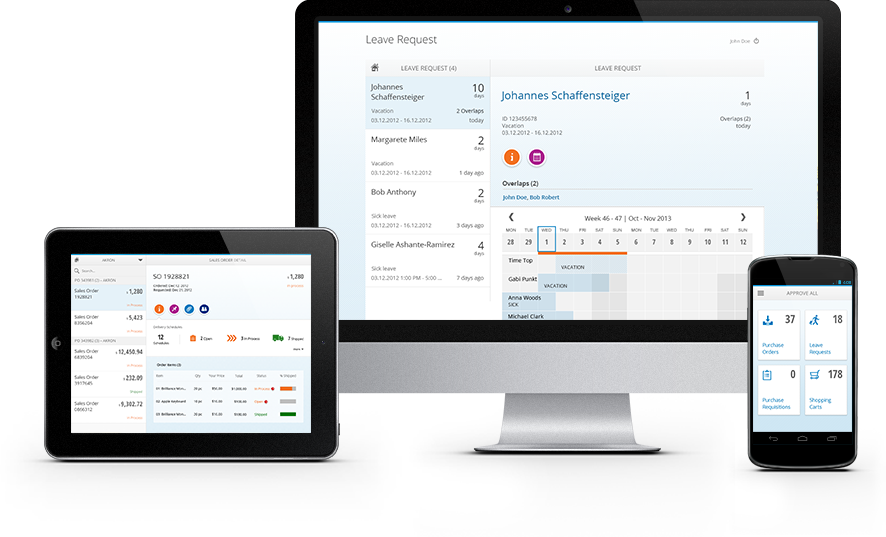
HD3C07 – Buttons and Toolbars

|  |  |
| --- | --- |
| **Product**  HANA Platform  **Level**  Undergraduate/Graduate  Beginner  **Focus**  Application Development  **Author** Ross Hightower  Leigh Jin | MOTIVATION  This case describes creating buttons and toolbars using SAPUI5.  PREREQUISITES  HD3C06 – The Base Application |



# Create the Buttons view

Create two new files in the **view** folder named **Buttons.view.xml** and **Buttons.controller.js**.

### Buttons.view.xml

Add the following code to the **Buttons.view.xml** file:

|  |
| --- |
| <mvc:View  height=*"100%"*  controllerName=*"ui5.controller.Buttons"*  xmlns=*"sap.m"*  xmlns:mvc=*"sap.ui.core.mvc"* >  <Page  enableScrolling=*"false"*  showNavButton = *"true"*  navButtonPress = *"handleNavButtonPress"*  title=*"Start Coding Buttons!"* >    </Page>  </mvc:View> |

Listing

This code creates a view that contains a <Page> element that includes a header with a title and a back button. The event handler, handleNavButtonPress, defined in the controller, is assigned to the press event.

### Buttons.controller.js

Add the following code to the **Buttons.controller.js** file:

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function(Controller) {  "use strict";  return Controller.extend("ui5.controller.Buttons", {  onInit: function() {  this.router = sap.ui.core.UIComponent.getRouterFor(this);  },  handleNavButtonPress: function() {  this.router.navTo("Master", {  from: "Buttons"  });  },  onPress: function(evt) {  jQuery.sap.require("sap.m.MessageToast");  sap.m.MessageToast.show("Pressed: " + evt.getSource().getId());  }  });  }); |

Listing

The onPress method uses a toast message to show a button’s ID when it is clicked and the handleNavButtonPress uses the router to navigate back to the Master view.

### views.json

Add the following to the **views.json** ViewsCollection array.

|  |
| --- |
| {  "icon" : "play",  "number" : "2",  "title" : "Buttons and Toolbars",  "info" : "Buttons"  } |

Listing

Make sure you include a comma between this object and the previous object.



### Component.js

Add the route to the router in the **Component.js** file.

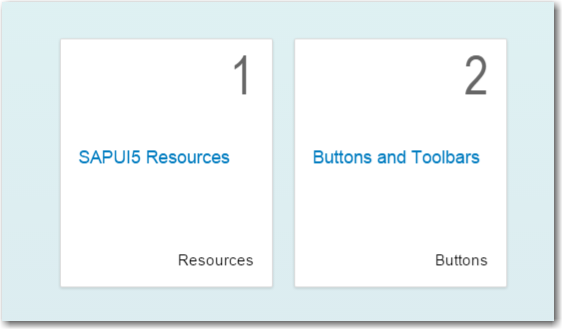
|  |
| --- |
| {  pattern : "Buttons",  name : "Buttons",  view : "Buttons",  targetAggregation : "pages",  } |

Listing

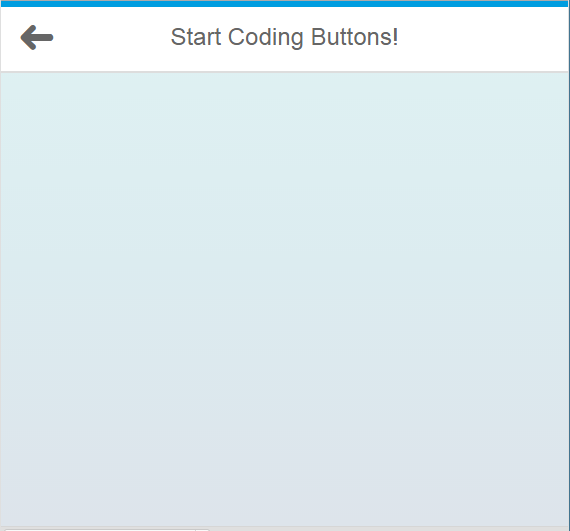
Make sure you add a comma between the routes.



Run the application by right-clicking the **index.html** file and clicking **.**



If you click on it, the Buttons view will load. It doesn’t have much on except the header and a back button.



## Add Some Buttons

### Buttons.view.xml

To add some buttons add the following code to the **Buttons.view.xml** file. The code is inserted immediately before the </Page> tag:

|  |
| --- |
| <Panel width="60%" class="panelShadow">  <headerToolbar>  <Toolbar height="3rem">  <Text text="Buttons" class="sapMH4FontSize"/>  <ToolbarSpacer />  <Button icon="sap-icon://settings" press="onPress"/>  <Button icon="sap-icon://drop-down-list" press="onPress"/>  </Toolbar>  </headerToolbar>  <content>  <Button text="Default" press="onPress" />  <Button type="Accept" text="Accept" press="onPress" />  <Button type="Reject" text="Reject" press="onPress" />  <Button type="Transparent" text="Submit" press="onPress" />  <Button type="Up" text="Up" press="onPress" />  <Button type="Unstyled" text="Unstyled" press="onPress" />  <Button text="Emphasized" press="onPress" type="Emphasized" />  </content>  </Panel> |

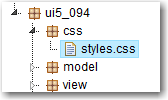
Listing



This shows a variety of buttons inside a Panel control. The Panel has a header with a Toolbar. The properties in the button elements can be found under the Properties tab in the page referenced above.

### styles.css

Notice the class attribute of the Panel. We will use this to style the Panel. Create a package in your project called **css** and then create a file inside the css package called **styles.css.**



Insert the following CSS code into the new file:

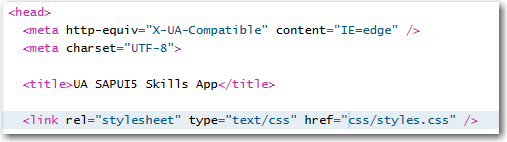
|  |
| --- |
| *.panelShadow* {  box-shadow: *10px* *10px* *5px* *#888888*;  margin-left: *auto*;  margin-right: *auto*;  margin-top: *20px*;  border: *solid* *1px*;  } |

Listing

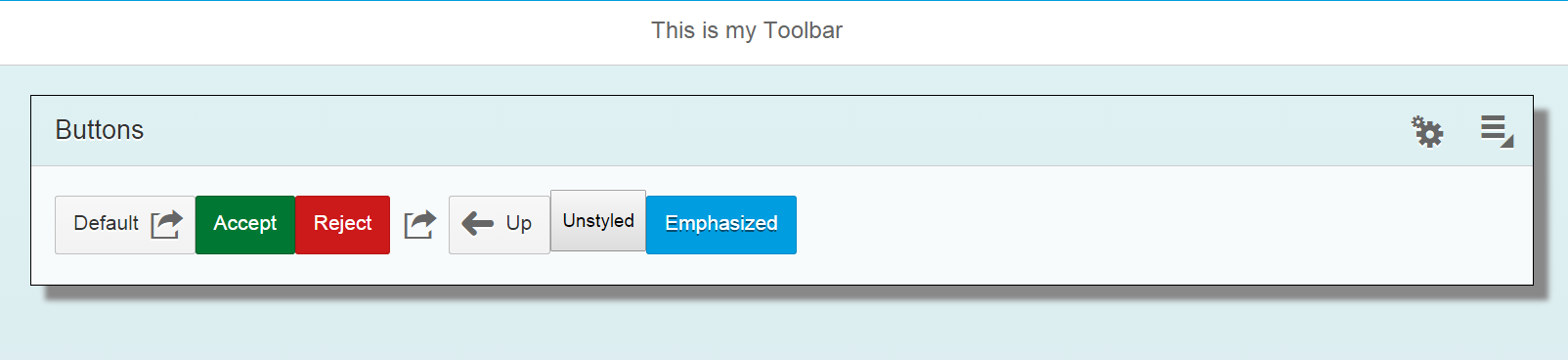
Now, edit the **index.html** file to add the reference to the styles.css file:

|  |
| --- |
| <link rel="stylesheet" type="text/css" href="css/styles.css" /> |

Listing



Run the application or refresh it if the page is already showing. When you navigate to the Buttons view there should now be a Panel showing a variety of button types.

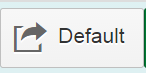


If you click a button, a toast message will appear showing the button’s id which is assigned, in this case, in the order the buttons appear in the file.

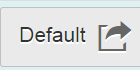
You can add icons to buttons by using the icon property. Add **icon=*"sap-icon://action"*** to the first button.



|  |
| --- |
| The SAPUI5 icons are in the form of a font which makes them more compatible with a mobile environment than images would be. You can find all the icons using the Icon Explorer at <https://openui5.hana.ondemand.com/iconExplorer.html> |



Change the order of the icon and the text using **iconFirst=*'false'****.*



To create an icon only button, use the transparent type and replace the text with an icon.



## Accounting for Screen Size

You can control for the visibility of buttons based on screen size by adding one of the following:

sapUiHideOnPhone - will be hidden if the screen has 600px or more

sapUiHideOnTablet - will be hidden if the screen has less than 600px or more than 1023px

sapUiHideOnDesktop - will be hidden if the screen is smaller than 1024px

sapUiVisibleOnlyOnPhone - will be visible if the screen has less than 600px

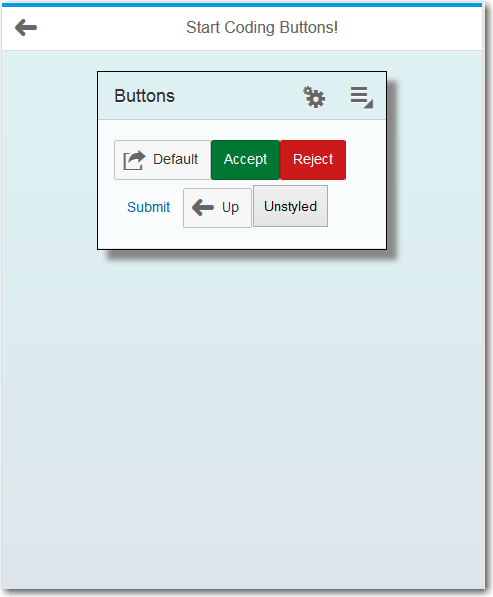
sapUiVisibleOnlyOnTablet - will be visible if the screen has 600px or more but less than 1024px

sapUiVisibleOnlyOnDesktop - will be visible if the screen has 1024px or more

Add **class=*"sapUiHideOnPhone"***to theemphasized button in Buttons.view.xml.



Run the application by selecting Run in application preview from the Run icon then change the size to a phone size.You will see the button disappear when the screen size is small enough.



## Add a Custom Header

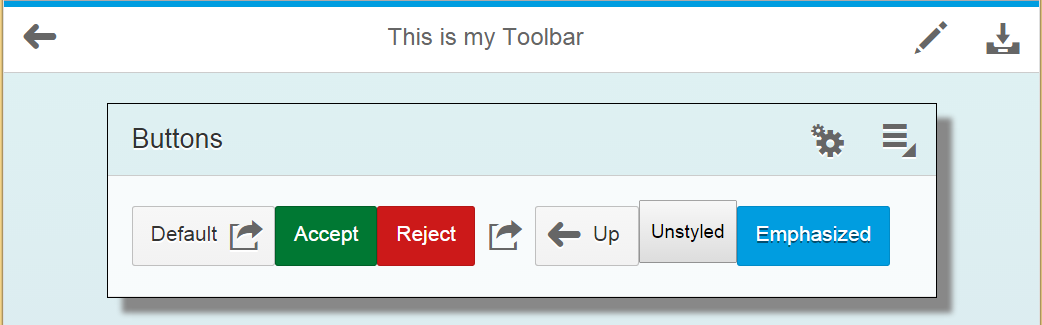
### Buttons.view.xml

To add a header with a toolbar add this code to **Buttons.view.xml** above the Panel:

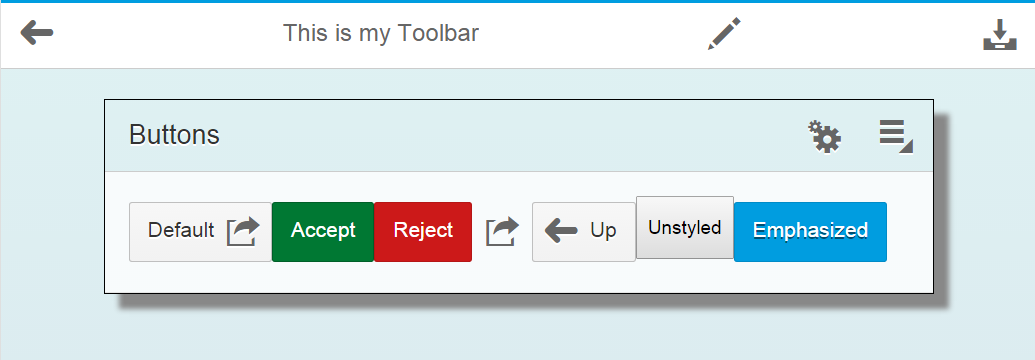
|  |
| --- |
| <customHeader>  <Toolbar>  <Button type="Back" press="*handleNavButtonPress*" />  <ToolbarSpacer/>  <Label text="This is my Toolbar" />  <ToolbarSpacer/>  <Button icon="sap-icon://edit" press="onPress" />  <Button icon="sap-icon://inbox" press="onPress" />  </Toolbar>  </customHeader> |

Listing





The custom header replaces the existing header. The <ToolbarSpacer/> tag is used to divide the toolbar into sections. Add another between the two buttons. Not very attractive but it illustrates how space items on a toolbar.



## Add a Toolbar in the Footer

You can also add a toolbar in a footer.

### Buttons.view.xml

Add code like this below the buttons in **Buttons.view.xml**:

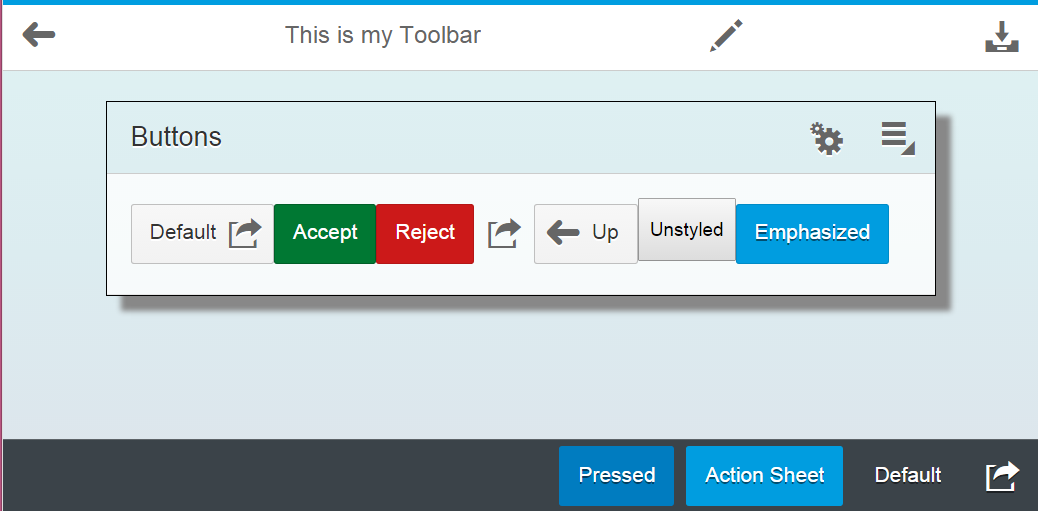
|  |
| --- |
| <footer>  <Toolbar>  <ToolbarSpacer/>  <ToggleButton text=*"Pressed"* enabled=*"true"* pressed=*"true"* press=*"onPress"* />  <Button type=*"Emphasized"* text=*"Action Sheet"* press=*"onPress"* />  <Button text=*"Default"* press=*"onPress"* />  <Button icon=*"sap-icon://action"* press=*"onPress"* />  </Toolbar>  </footer> |

Listing



The first button is a ToggleButton which will toggle between two states.

Note that the single <ToolbarSpacer/> at the top of the toolbar code right justifies the contents.

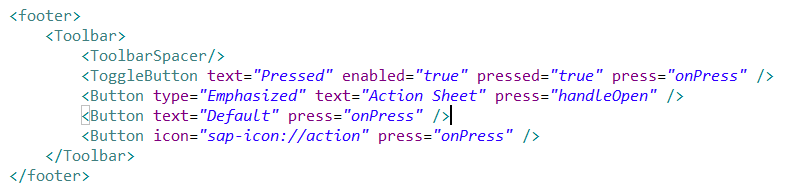


## Add an Action Sheet

The next step is to add an Action Sheet to the Action Sheet button in the footer toolbar. Start by modifying the code for the Action Sheet button in **Buttons.view.xml** as shown below.

### Buttons.view.xml

Start by modifying the code for the Action Sheet button in **Buttons.view.xml** as shown below. Change the press handler to **handleOpen**.



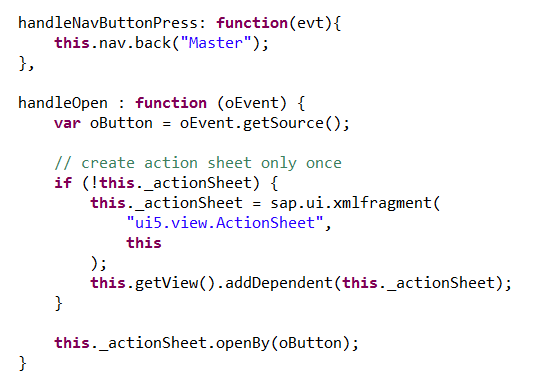
### Buttons.controller.js

Next add the following code to the **Buttons.controller.js** file.

|  |
| --- |
| handleOpen : **function** (oEvent) {  **var** oButton = oEvent.getSource();  // create action sheet only once  **if** (!**this**.\_actionSheet) {  **this**.\_actionSheet = sap.ui.xmlfragment(  "ui5.view.ActionSheet",  **this**  );  **this**.getView().addDependent(**this**.\_actionSheet);  }  **this**.\_actionSheet.openBy(oButton);  } |

Listing

Don’t forget to insert a comma between the functions. This function is called when the Action Sheet button is pressed. The first time the method is called, it will instantiate the action sheet fragment (created below), and add it to the view. The last statement opens the ActionSheet next to the button that was clicked.

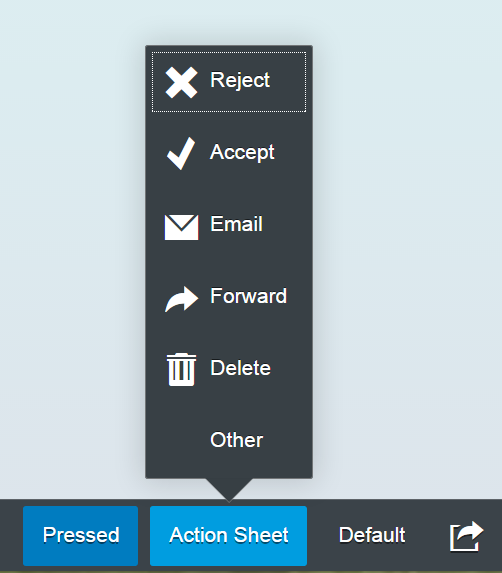


Now, create a new file in the **view** folder called **ActionSheet.fragment.xml** and insert the code shown below:

|  |
| --- |
| <core:FragmentDefinition  xmlns=*"sap.m"*  xmlns:core=*"sap.ui.core"*>  <ActionSheet  title=*"Choose Your Action"*  showCancelButton=*"true"*  placement=*"Top"*>  <buttons>  <Button text=*"Reject"* icon=*"sap-icon://decline"* press=*"onPress"* />  <Button text=*"Accept"* icon=*"sap-icon://accept"* press=*"onPress"* />  <Button text=*"Email"* icon=*"sap-icon://email"* press=*"onPress"* />  <Button text=*"Forward"* icon=*"sap-icon://forward"* press=*"onPress"* />  <Button text=*"Delete"* icon=*"sap-icon://delete"* press=*"onPress"* />  <Button text=*"Other"* />  </buttons>  </ActionSheet>  </core:FragmentDefinition> |

Listing

This code creates a code fragment that is instantiated when the Action Sheet button is tapped or clicked. Notice the code is similar to a view but doesn’t include the enclosing <mvc:View></mvc:View> tags.



## Add a Menu

Finally, let’s add a [menu](https://sapui5.hana.ondemand.com/sdk/explored.html#/entity/sap.ui.unified.Menu/samples).

### Menu.fragment.xml

The menu content is included in a separate file called **Menu.fragment.xml**. Create the file in the **view** package. Enter the code show below

|  |
| --- |
| <core:FragmentDefinition  xmlns="sap.m"  xmlns:core="sap.ui.core"  xmlns:u="sap.ui.unified">  <u:Menu>  <u:MenuItem  text="My 1st Item"  select="handleMenuItemPress"  icon="sap-icon://save"/>  <u:MenuItem  text="My 2nd Item"  select="onPress" />  <u:MenuItem  text="My 3rd Item">  <u:Menu>  <u:MenuItem  text="1st Sub Item"  select="onPress" />  <u:MenuItem  text="2nd Sub Item"  select="onPress" />  <u:MenuItem  text="3rd Sub Item but inactive"  enabled="false" />  </u:Menu>  </u:MenuItem>    <u:MenuItem  text="My 4th Item"  startsSection="true"  select="onPress" />  <u:MenuItem  text="My 5th Item"  select="onPress" />  <u:MenuTextFieldItem  label="Find"  enabled="true"  startsSection="true"  icon="sap-icon://filter"  select="handleTextFieldItemPress" />  </u:Menu>  </core:FragmentDefinition> |

Listing

### Buttons.view.xml

Modify the code in the Panel control toolbar in Buttons.view.xml file. Only the bolded section changes. This button will be used to open the menu. We give it an id and change the press event handler.

|  |
| --- |
| <Panel width="80%" class="panelShadow">  <headerToolbar>  <Toolbar height="3rem">  <Text text="Buttons" class="sapMH4FontSize"/>  <ToolbarSpacer />  <Button icon="sap-icon://settings" press="onPress"/>  <Button id='openMenu' icon="sap-icon://drop-down-list" press="**handlePressOpenMenu**"/>  </Toolbar>  </headerToolbar> |

Listing

### Buttons.controller.js

Add the following code to the Buttons.controller.js file.

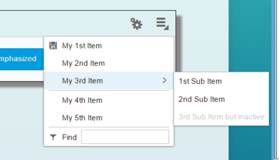
|  |
| --- |
| handlePressOpenMenu: function(oEvent) {  var oButton = oEvent.getSource();  if (!this.\_menu) {  this.\_menu = sap.ui.xmlfragment(  "ui5.view.Menu",  this  );  this.getView().addDependent(this.\_menu);  }  var eDock = sap.ui.core.Popup.Dock;  this.\_menu.open(this.\_bKeyboard, oButton, eDock.BeginTop, eDock.BeginBottom, oButton);  },  handleTextFieldItemPress: function(oEvent) {  var msg = "'" + oEvent.getParameter("item").getValue() + "' entered";  sap.m.MessageToast.show(msg);  } |

Listing

The handlePressOpenMenu function is called when the user presses the button. The function retrieves the button control from the oEvent parameter. It then checks to see if the menu already exists. The first time this function is called, the menu will not exist so the this.\_menu variable is created from the code in the Menu.fragement.xml file. Next, the new menu is attached to the Button’s view as a dependent. The last two lines actually open the menu.

The handleTextFieldItemPress function is invoked when the user enters a value in the input box on the menu and hits enter.

Now when you click the button, the menu opens



# Exercise

1. Add a menu button to the top left corner of master.vew.xml.



1. Add an action sheet to this button. Use the placement attribute of the ActionSheet control to place the ActionSheet below the Button. Note the action sheet opens below the button. The attribute that controls this is in the fragment code.



1. Add a function to Master.controller.js that:
   1. Shows only the case tiles when Cases is selected.
   2. Shows only the exercise tiles when Exercises is selected.
   3. Shows both sets of tiles when Both is selected.

The onPress function used with the action sheet items looks like this:

|  |
| --- |
| onPress : function(evt){  var id = evt.getSource().getId();  switch(id) {  case "idCases":  $("#\_\_xmlview0--caseTiles").fadeIn();  $("#\_\_xmlview0--exerciseTiles").fadeOut();  break;  case "idExercises":  $("#\_\_xmlview0--caseTiles").fadeOut();  $("#\_\_xmlview0--exerciseTiles").fadeIn();  break;  case "idBoth":  $("#\_\_xmlview0--caseTiles").fadeIn();  $("#\_\_xmlview0--exerciseTiles").fadeIn();  break;  }  } |

Listing

This code uses jQuery to hide and show the TileContainers. The $(“#”) construction is how jQuery selects an HTML element based on the element’s ID. SAPUI5 creates the IDs of HTML elements by appending the view name with two hyphens and the id of the element in the XML view file. \_\_xmlview0—caseTiles refers to the case TileContainer.

The idCases, idExercises and idBoth are the ids of the buttons in the ActionSheet.

|  |  |
| --- | --- |
|  |  |

