HD3C09 – Lists

|  |  |
| --- | --- |
| **Product and Focus**  HANA Platform/SAPUI5 | **MOTIVATION**  Lists are a staple of most applications. This case describes the SAPUI5 List control.  **PREREQUISITES**  HD1sC03 – The Base Application |
| **Target Audience**  Undergrduate/Graduate Beginner to Intermediate |
| **Author**  Ross Hightower |
| https://bgoerke.files.wordpress.com/2013/05/section1.png | |

# The List Control

Lists of items are a fundamental building block for business applications so time learning this control is well spent.

## Create the List View

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. Next, create two new files in the **view** package called **Lists.view.xml** and **Lists.controller.js.**

### Lists.view.xml

Copy the code below into the **Lists.view.xml** file.

|  |
| --- |
| <mvc:View  controllerName="ui5.controller.Lists"  xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m">  <Page  showNavButton = "true"  navButtonPress = "handleNavButtonPress"  title="Start Coding Lists!" >    <List  id="ShortProductList"  headerText="Products"  items="{gbi>/Products}" >  <StandardListItem  type="Active"  press="handleListItemPress"  title="{gbi>ID}"  description="{gbi>ProductName}" />  </List>  </Page>  </mvc:View> |

Listing 1

The list is bound to the Products in the gbi model. The StandardListItem is defined as the template for the items on the list. SAPUI5 will create a [StandardListItem](https://sapui5.hana.ondemand.com/sdk/explored.html#/entity/sap.m.StandardListItem/samples) for each item in the collection. The {} brackets represent placeholders into which data from the model will be inserted. The names in the brackets are the names of the properties in the objects defined in the model. The gbi> refers to the name of the model.

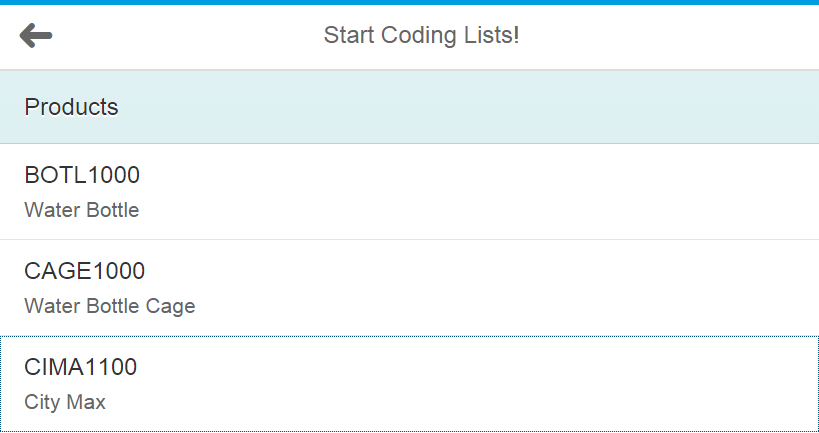
### Lists.Controller.js

Insert the code shown below into the **Lists.controller.js** file.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function(Controller) {  "use strict";  return Controller.extend("ui5.controller.Lists", {  onInit: function() {  this.router = sap.ui.core.UIComponent.getRouterFor(this);  },  handleNavButtonPress: function() {  this.router.navTo("Master", {  from: "Lists"  });  }  });  }); |

Listing 2

Now run the application.



## Add a ListDetail View

Now we’ll add a detail view that appears when a list item is clicked.

### Lists.controller.js

Update the **handleListItemPress** method in **Lists.controller.js** like that shown below:

|  |
| --- |
| handleListItemPress: function(evt){  //Get the path to the selected object and extract the index  var entity = evt.getSource().getBindingContext("gbi").getPath().split("/");  //Navigate to ListDetai and pass the index of the selected object  this.router.navTo("ListDetail", {  from: "Lists",  entity: entity[1]  });    } |

Listing 3

For an explanation of this code see case HD1C01w.

### ListDetail.view.xml

Copy the following code into the **ListDetail.view.xml** file:

|  |
| --- |
| <mvc:View  controllerName="ui5.controller.ListDetail"  xmlns:mvc="sap.ui.core.mvc"  xmlns:commons="sap.suite.ui.commons"  xmlns="sap.m">  <Page  showNavButton = "true"  navButtonPress = "handleNavButtonPress"  title="Product List Detail!" >  <VBox>  <Text text = "{gbi>ID}" />  <Text text = "{gbi>ProductName}" />  <Text text = "{gbi>ProductCategory}" />  <Text text = "{gbi>Divsion}" />  <Text text = "{gbi>Price}" />  <Text text = "{gbi>InternaPrice}" />  <Text text = "{gbi>Color}" />  </VBox>    </Page>  </mvc:View> |

Listing 4

### ListDetail.controller.js

Copy the following code into the **ListDetail.controller.js** file:

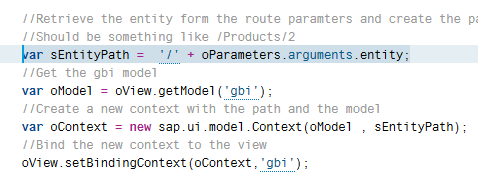
|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function(Controller) {  "use strict";  return Controller.extend("ui5.controller.ListDetail", {  onInit: function() {  this.router = sap.ui.core.UIComponent.getRouterFor(this);  this.router.attachRoutePatternMatched(this.onRouteMatched, this);  },  onRouteMatched: function(oEvent) {  //Get the route paramters  var oParameters = oEvent.getParameters();  //Get a reference to this view  var oView = this.getView();  // Make sure we're in the right place  if (oParameters.name !== "ListDetail") {  return;  }  this.getView().bindElement("gbi>/" + oParameters.arguments.entity);  },  handleNavButtonPress: function() {  this.router.navTo("Lists", {  from: "ListDetail"  });  }  });  }); |

Listing 5

Objects in a collection are accessed via a path variable that includes each of the levels from the root of the collection to the individual object. The object itself is represented by an index. The first object of the Products collection is shown below:



The BOTL1000 product is accessed using the path /Products(‘BOTL1000’). The code below from the ListDetail controller constructs that path from the entity which is passed from the Lists view then creates a binding context with the gbi model and the path. Binding this context to the view makes the properties of the selected object available to the controls in the view.



### Component.js

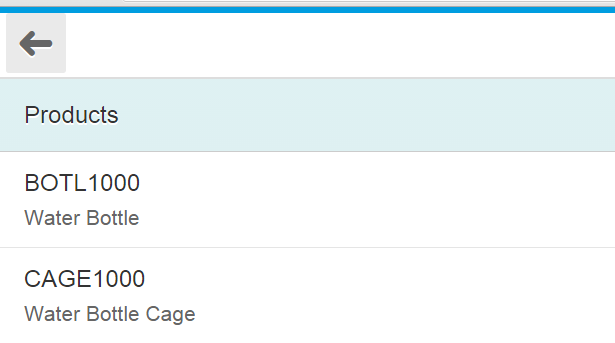
Add the following route for the ListDetail view.

|  |
| --- |
| {  pattern: "ListDetail/{entity}",  name: "ListDetail",  view: "ListDetail",  targetAggregation: "pages",  targetControl : "idAppControl"  } |

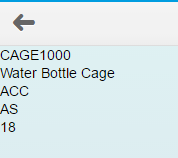
Listing 6

Note the entity parameter passed from the Lists controller will be inserted into the {entity} placeholder.

Run the application. When you click it, you will see a page with the list of products:



And the details:



## Enhance the List

The SAPUI5 list control has a lot more capabilities this so let’s dress it up a little by using an ObjectListItem rather than the StandardListItem.

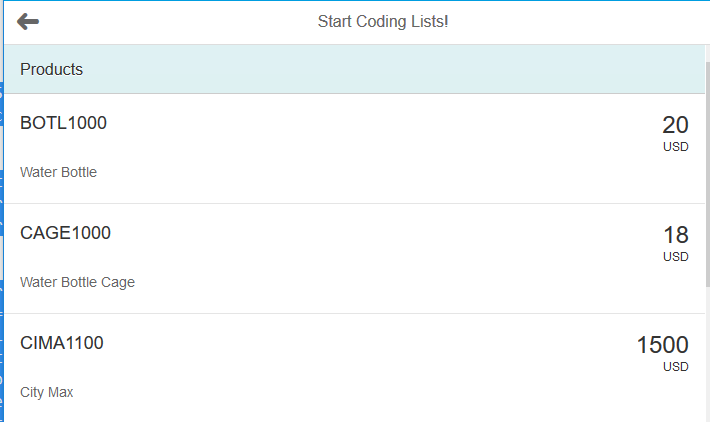
### Lists.view.xml

Replace the code in Lists.view.xml with the code shown below:

|  |
| --- |
| <mvc:View  controllerName="ui5.controller.Lists"  xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m">  <Page  showNavButton = "true"  navButtonPress = "handleNavButtonPress"  title="Start Coding Lists!" >    <List  id="ShortProductList"  headerText="Products"  items="{gbi>/Products}" >  <ObjectListItem  type="Active"  press="handleListItemPress"  title="{gbi>ID}"  number="{gbi>Price}"  numberUnit = "USD">  <attributes>  <ObjectAttribute text ="{gbi>ProductName}" />  </attributes>  <firstStatus>  <ObjectStatus text = "{gbi>ProductGroup}" />  </firstStatus>  </ObjectListItem>  </List>  </Page>  </mvc:View> |

Listing 7

This list uses an ObjectListItem rather than a StandardListItem.



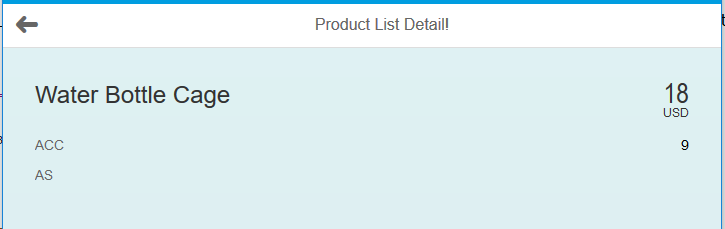
### ListDetail.view.xml

However the detail screen is not very pretty so let’s dress that up a bit. Replace the code in ListDetail.view.xml with the code below:

|  |
| --- |
| <mvc:View  controllerName="ui5.controller.ListDetail"  xmlns:mvc="sap.ui.core.mvc"  xmlns:commons="sap.suite.ui.commons"  xmlns="sap.m">  <Page  showNavButton = "true"  navButtonPress = "handleNavButtonPress"  title="Product List Detail!" >  <ObjectHeader  title="{gbi>ProductName}"  number = "{gbi>Price}"  numberUnit = "USD" >  <attributes>  <ObjectAttribute text = "{gbi>ProductCategory}" />  <ObjectAttribute text = "{gbi>ProductGroup}" />  <ObjectAttribute text = "{gbi>Division}" />  </attributes>  <firstStatus>  <ObjectStatus text = "{gbi>InternalPrice}" />  </firstStatus>  </ObjectHeader>  </Page>  </mvc:View> |

Listing 8

The details view now looks a little better:



## Add a Search Field

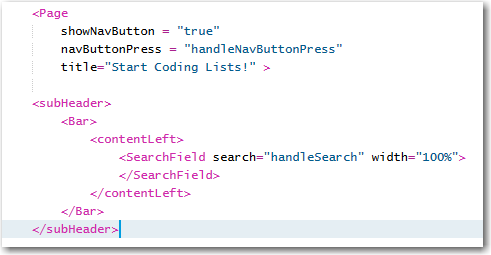
Finally, we’ll add a search field to the list.

### Lists.view.xml

Add the code shown below the Lists.view.xml file. The code creates a **subHeader** with a **SearchField** control. This code should be inserted immediately after the **Page** tag.

|  |
| --- |
| <subHeader>  <Bar>  <contentLeft>  <SearchField search=*"handleSearch"* width=*"100%"*>  </SearchField>  </contentLeft>  </Bar>  </subHeader> |

Listing 9



The search attribute of SearchField references a handler which we will not add to the List.controller.js file.

### Lists.controller.js

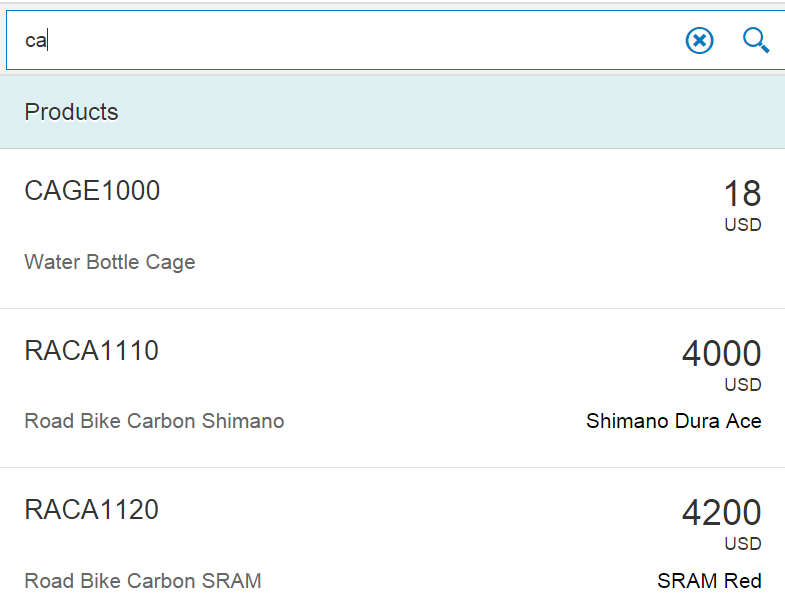
Add the function shown below to the Lists.controller.js file. Remember to place a comma between this and the other functions in the controller.

|  |
| --- |
| handleSearch : function (evt) {  var filters = [];  var query = evt.getParameter("query");  if (query && query.length > 0){  var filter = new sap.ui.model.Filter("ID", sap.ui.model.FilterOperator.Contains, query);  filters.push(filter);  }  var list = this.getView().byId('ShortProductList');  var binding = list.getBinding("items");  binding.filter(filters);  } |

Listing 10

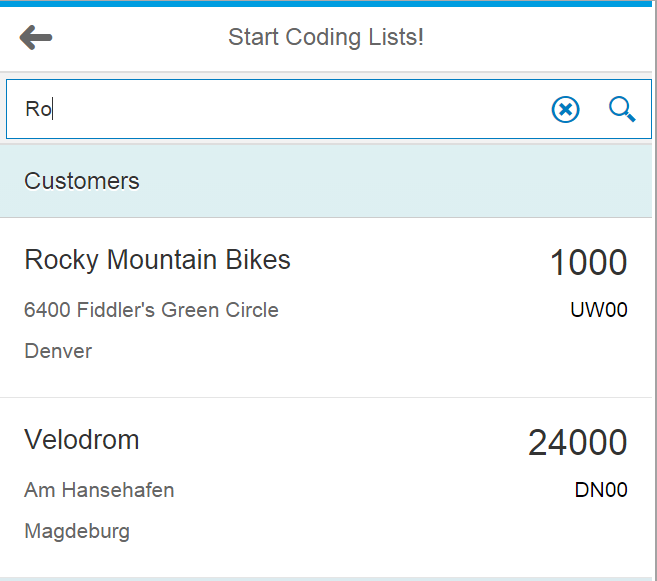
The code works by filtering the data in the list. It first creates an array to contain filters then retrieves the contents of the SearchField and places it in a variable called query. If there is a search term then a filter is created based on the PRODUCT field. The operator, Contains, means that the string appears somewhere in the PRODUCT fields.

Once the filter is added to the filters array the data binding of the list has to be updated. To do this a reference the list is obtained using the list’s id and then the binding is retrieved and the filter is added to it. You can now filter the list.



# Exercise

Create a view with a list that uses the Customers collection. The list should look like the image below. It has a search field that allows you to search on the company name. Hint: You can add multiple ObjectAttributes to an object list item.



The detail view looks like this image:

