HD3C04 – Routing in SAPUI5

|  |  |
| --- | --- |
| **Product and Focus**  HANA Platform/SAPUI5 | **MOTIVATION**  This case uses a simple application to explain routing in SAPUI5 applications.  **PREREQUISITES**  HD3C01 – Hello World HD3C02 – Data Binding |
| **Target Audience**  Undergraduate/Graduate Beginner to Intermediate |
| **Author**  Ross Hightower |
| https://bgoerke.files.wordpress.com/2013/05/section1.png | |

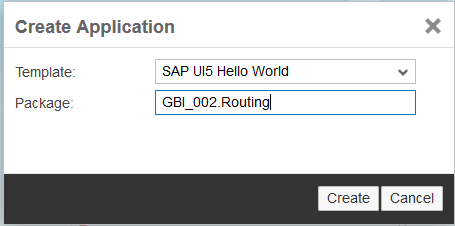
# Routing in SAPUI5

The previous case (HD1C01w) introduced the basic application architecture for SAPUI5 applications used in this curriculum. This case introduces the concept of routing which is one method to navigate between views in SAPUI5 applications.

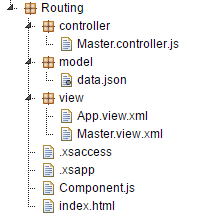
# Create the Application

Log on to the WDW and open the **Editor**. Right-click your package and select **Create Application.**

Select the **SAP UI5 Hello World** template and add .**Routing** to the package name. This will create a package called Routing and create the application in that package.



Now create three packages called **view**, **controller** and **model** and five files called **data.json**, **Component.js**, **App.view.xml**, **Master.view.xml** and **Master.controller.js**.



## Modify the Application

In this section you will rearrange the application into a more robust structure. While this structure is overkill for such a small application, it is the structure used for Fiori applications and is the structure used for application in this curriculum.

### index.html

Replace the code in the index.html file with the code shown below.

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta http-equiv="X-UA-Compatible" content="IE=edge" />  <meta charset="UTF-8">    <title>SAPUI5 Routing</title>  <script id="sap-ui-bootstrap"  src=" /sap/ui5/1/resources/sap-ui-core.js"  data-sap-ui-theme="sap\_bluecrystal"  data-sap-ui-libs="sap.m"  data-sap-ui-resourceroots='{  "routing": "./"  }'>  </script>  <script>  // now create a new, reusable component called ui5 (like our namespace)  sap.ui.getCore().attachInit(function() {  new sap.m.Shell("shell",{  app : new sap.ui.core.ComponentContainer({  height : "100%",  name : "routing"  })  }).placeAt("content"); });  </script>  </head>  <body class="sapUiBody" id="content">  </body>  </html> |

Listing

### Component.js

Add the code shown below.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/UIComponent"  ], function (UIComponent) {  "use strict";  return UIComponent.extend("routing.Component", {  metadata : {  rootView: "routing.view.App"  },    init : function () {  UIComponent.prototype.init.apply(this, arguments);  }  });  }); |

Listing

All the Component does is load the App.view.xml file.

### App.view.xml

Insert the following code in the App.view.xml file so that it matches the code shown below.

|  |
| --- |
| <core:View xmlns:core="sap.ui.core"  xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m"  xmlns:html="http://www.w3.org/1999/xhtml">  <App  id="idAppControl" />  </core:View> |

Listing

The App view is essentially the bootstrap for the rest of the application interface. It has no visual component itself but it creates the application object which manages the behind the scenes functions of the application. The application object, in this case, is an App control which creates a full screen application.

### Master.view.xml

Add the following code:

|  |
| --- |
| <mvc:View controllerName="routing.controller.Master"  xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m" >  <Page title="Master View">  <content>    </content>  </Page>  </mvc:View> |

Listing

### Master.controller.js

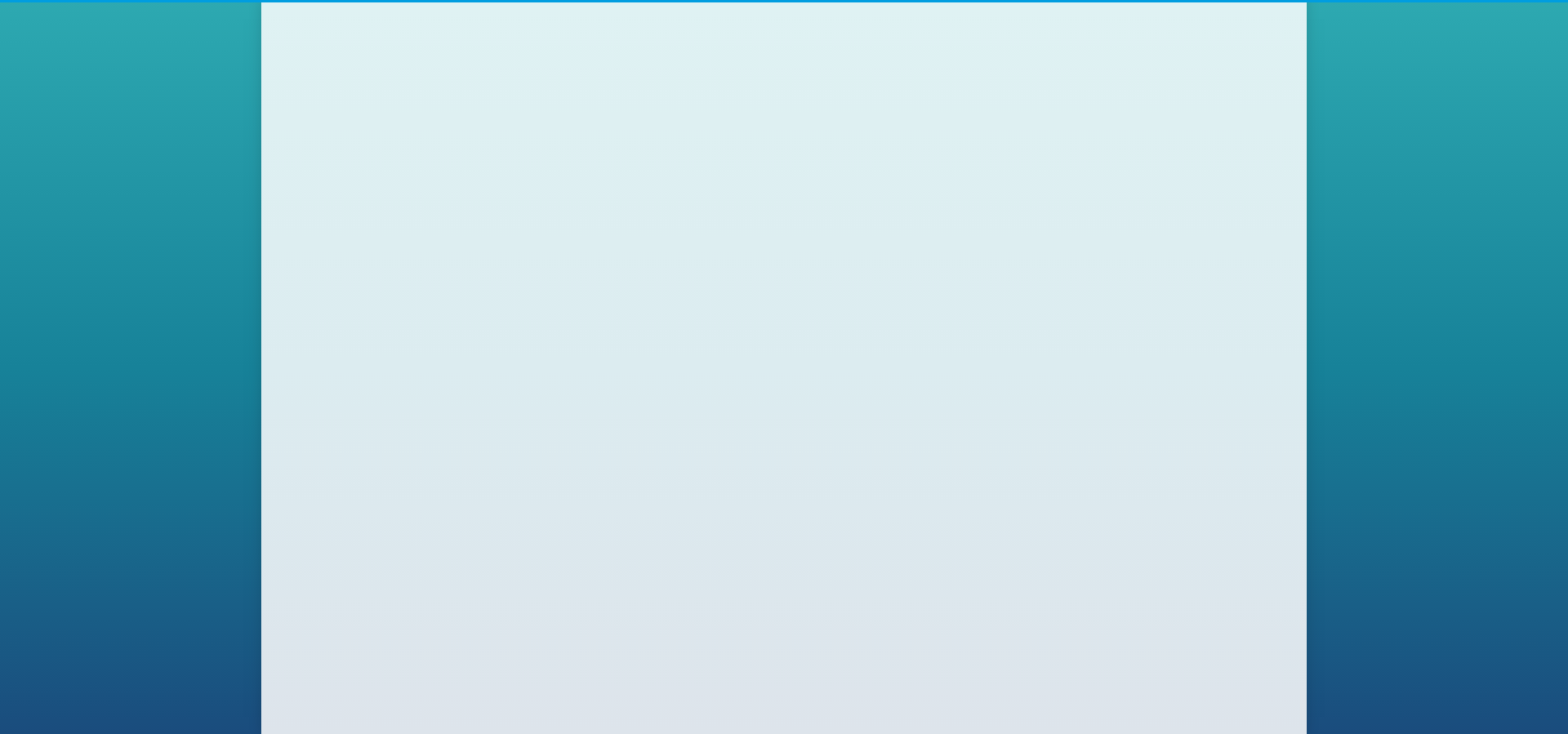
Insert the following code:

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function (Controller) {  "use strict";  return Controller.extend("routing.controller.Master", {    });  }); |

Listing

## Configure the Router and a Default Route

If you run the application now you will get a blank screen showing the Shell control.



That’s because the Component loads the App.view.xml file which has no visual interface. In order to load the Master view into the App control (created in the App view) we have to configure an object called the Router. The Router handles navigation among views in the application. Because it is central to the entire application, it is configured in the Component.js file.

### Component.js

Modify the code as shown below:

|  |
| --- |
| sap.ui.define([  "sap/ui/core/UIComponent"  ], function(UIComponent) {  "use strict";  return UIComponent.extend("routing.Component", {  metadata: {  rootView: "routing.view.App"**,**  **routing: {**  **config: {**  **viewType: "XML",**  **viewPath: "routing.view",**  **transition: "slide",**  **clearTarget: true,**  **targetControl: "idAppControl"**  **},**  **routes: [**  **{**  **pattern: "",**  **name: "Master",**  **view: "Master",**  **targetAggregation: "pages"**  **}**  **]**  }  },  init: function() {  UIComponent.prototype.init.apply(this, arguments);  **this.getRouter().initialize();**  }  });  }); |

The router is initialized with the line this.getRouter().initialize(). The configuration of the routes is added to the metadata object. The config section of the router configuration sets some default values.

**viewType:** This views are defined using XML.

**viewPath:** The location of the view files. Note uses the resource roots prefix defined in the bootstrap of the index.html file.

**transition:** A transition effect when views change.

**clearTarget:** This tells UI5 to delete the currently loaded view before loading the next view.

**targetControl:** The id of the App control created in the App view.

After the config section, the routes are defined. This code only defines one route. The routes have the following parameters:

**Pattern:** Defines the pattern of the URL used to access the route. In this case, the “” indicates this is the default route so it will load when the user enters the base URL.

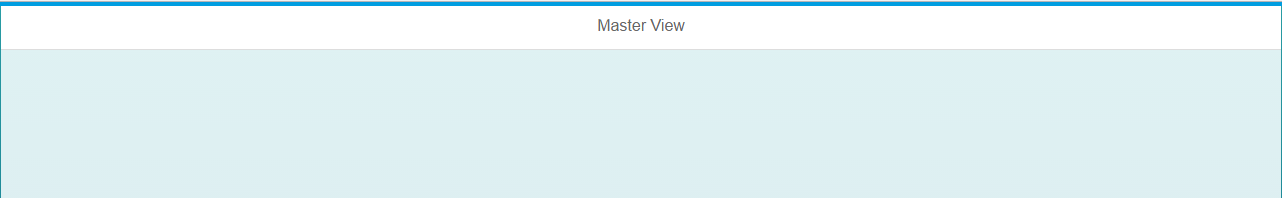


**name:** This is how the route is referenced in JavaScript code.

**view:** The name of the view file. UI5 will look for it in the location specified by viewPath.

**targetAggregation:** The targetControl has one or more aggregation bindings to which views can be bound. Since the App control creates a full screen application, it only has one aggregation called pages. We’ll see an example of a SplitApp control later which has masterPages and detailPages aggregations.

Run the application and you will see the screen show below. The Master view is loaded.



## Add a Second Route

Now let’s add a second route called Detail1. First add two files: **Detail1.view.xml** and **Detail1.controller.js**.

### Detail1.view.xml

Add the following code:

|  |
| --- |
| <mvc:View controllerName="routing.controller.Detail1"  xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m" >  <Page title="Detail1 View">  <content>    </content>  </Page>  </mvc:View> |

Listing

### Detail1.controller.js

Add the following code:

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function (Controller) {  "use strict";  return Controller.extend("routing.controller.Detail1", {    });  }); |

Listing

### Component.js

Now we have to add a route. Update the Component.js file as shown below:

|  |
| --- |
| sap.ui.define([  "sap/ui/core/UIComponent"  ], function(UIComponent) {  "use strict";  return UIComponent.extend("routing.Component", {  metadata: {  rootView: "routing.view.App",  routing: {  config: {  viewType: "XML",  viewPath: "routing.view",  transition: "slide",  clearTarget: true,  targetControl: "idAppControl"  },  routes: [  {  pattern: "",  name: "Master",  view: "Master",  targetAggregation: "pages"  },  **{**  **pattern: "Detail1",**  **name: "Detail1",**  **view: "Detail1",**  **targetAggregation: "pages"**  **}**  ]  }  },  init: function() {  UIComponent.prototype.init.apply(this, arguments);  this.getRouter().initialize();  }  });  }); |

Listing

Finally, we need a way to invoke this route. If you run the application, the default route which is Master, is loaded. We need an interface element that the user can interact with to cause the application to navigate to the Detail1 view. Let’s start by using a simple Button control.

### Master.view.xml

Update the code as shown:

|  |
| --- |
| <mvc:View controllerName="routing.controller.Master"  xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m" >  <Page title="Master View">  <content>  **<Button**  **text="Go"**  **press="go"/>**  </content>  </Page>  </mvc:View> |

Listing

This creates a Button control that has a event handler assigned to the press event called go. This function is implemented in the view’s controller file.

### Master.controller.js

Update the code as shown.

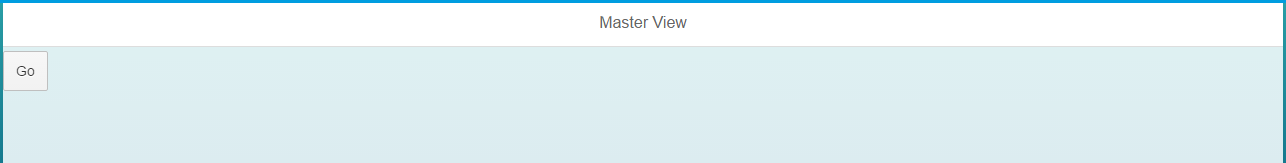
|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function (Controller) {  "use strict";  return Controller.extend("routing.controller.Master", {  **onInit: function(){**  **this.router = sap.ui.core.UIComponent.getRouterFor(this);**  **},**    **go: function(){**  **this.router.navTo("Detail1");**  **}**  });  }); |

Listing

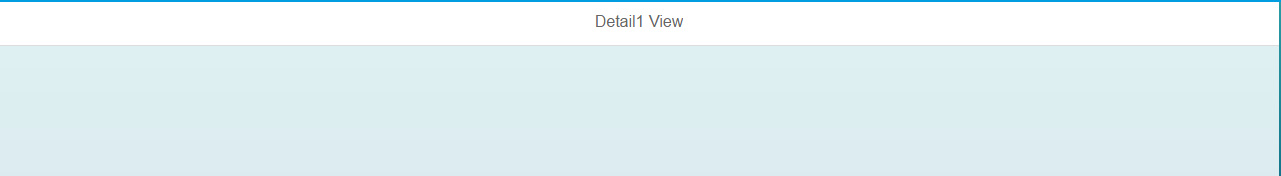
The code in the onInit function (which you will recall runs when the view is first created) retrieves a reference to the router object that was created in the Component.js file.

The code in the go function invokes the navTo function of the router object and passes Detail1 as the target. This is the name property of the route.

Now if you run the application, the Master view has a Button control:



If you click it, the Detail1 view is loaded:



Note also the pattern of the Detail1 route appears in the URL after the # symbol.



## Passing Parameters

We often need to pass parameters in routes. For example, if the user clicks an item on the list to load a view that shows details of the item, we need to pass the identity of the item clicked in the route. This section will illustrate how that is done. We will create a List control on the Master view. When the user clicks and item on the List, the Detail1 view will be loaded showing a List with details of the clicked item.

### data.json

Add the following code to the data.json file.

|  |
| --- |
| {  "collection" : [  {  "prop1" : "A",  "subcol" : [  {  "subProp1" : "One"  },  {  "subProp1" : "Two"  }  ]  },  {  "prop1" : "B",  "subcol" : [  {  "subProp1" : "Three"  },  {  "subProp1" : "Four"  }  ]  }    ]  } |

Listing

This JSON data has a collection called collection. Within each object in collection is another collection called subcol. The items in collection will be shown on the Master list and the items in the subcol of the selected item will be shown on the Detail1 list.

### Master.view.xml

Update the code as shown:

|  |
| --- |
| <mvc:View controllerName="routing.controller.Master"  xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m" >  <Page title="Master View">  <content>  **<List**  **headerText="Master List"**  **items="{/collection}" >**  **<StandardListItem**  **type="Active"**  **press="go"**  **title="{prop1}" />**  **</List>**  </content>  </Page>  </mvc:View> |

Listing

This code replaces the Button control with a List control. The List control has an aggregation binding called items which is bound the collections collection in the model. A StandardListItem control is provided as a template to create each of the items on the list. The title property of the StandardListItem is bound to the prop1 property and the press event is bound to the go function.

### Component.js

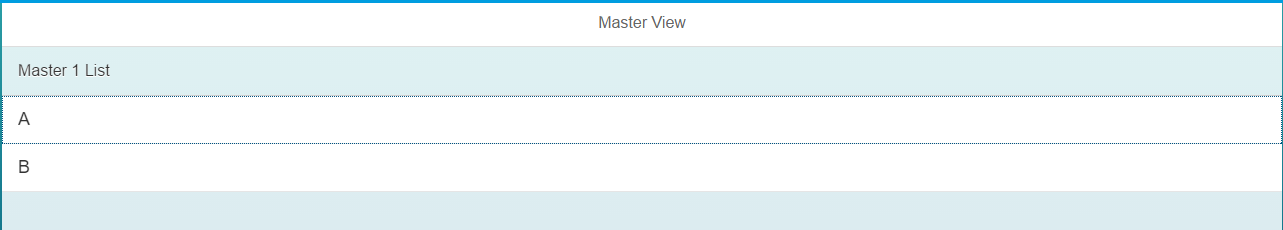
Update the code as shown.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/UIComponent"**,**  **"sap/ui/model/json/JSONModel"**  ], function(UIComponent**, JSONModel**) {  "use strict";  return UIComponent.extend("routing.Component", {  metadata: {  rootView: "routing.view.App",  routing: {  config: {  viewType: "XML",  viewPath: "routing.view",  transition: "slide",  clearTarget: true,  targetControl: "idAppControl"  },  routes: [  {  pattern: "",  name: "Master",  view: "Master",  targetAggregation: "pages"  },  {  pattern: "Detail1",  name: "Detail1",  view: "Detail1",  targetAggregation: "pages"  }  ]  }  },  init: function() {  UIComponent.prototype.init.apply(this, arguments);  this.getRouter().initialize();  **var oModel = new JSONModel("model/data.json");**  **this.setModel(oModel);**  }  });  }); |

Listing

The additions create the model.

Now if you run the application you will see a list with two items.



If you click an item on the List, the app navigates to the Detail1 view but the view is empty.

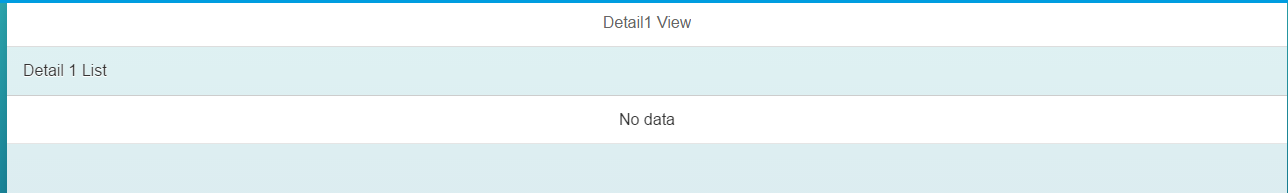
### Detail1.view.xml

Update the code as shown.

|  |
| --- |
| <mvc:View controllerName="routing.controller.Detail1"  xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m" >  <Page title="Detail1 View">  <content>  **<List**  **headerText="Detail 1 List"**  **items="{subcol}" >**  **<StandardListItem**  **type="Active"**  **title="{subProp1}" />**  **</List>**  </content>  </Page>  </mvc:View> |

Listing

This adds a List control but if you run the application and click an item on the Master view list, the Detail1 view is shown but the list is empty.



To allow it to show data we need to pass a parameter so that the Detail1 view knows which item in the collection was pressed.

### Component.js

First, update the route to add a parameter to the Detail1 route. Update the Detail1 route as shown.

|  |
| --- |
| {  pattern: "Detail1**/{parameter}**",  name: "Detail1",  view: "Detail1",  targetAggregation: "pages"  } |

Listing

The parameter, added to the pattern, is enclosed in curly brackets.

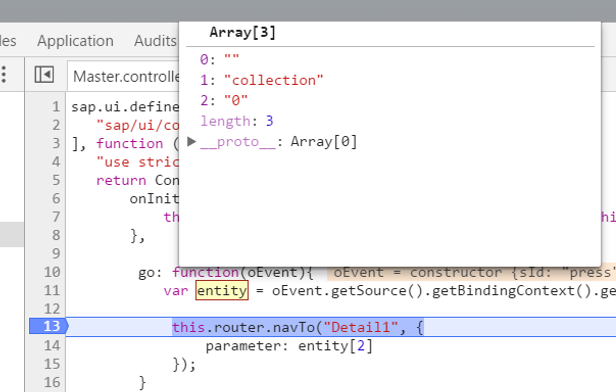
### Master.controller.js

Next, we need to add code to the go function so that it passes the correct parameter.\

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function (Controller) {  "use strict";  return Controller.extend("routing.controller.Master", {  onInit: function(){  this.router = sap.ui.core.UIComponent.getRouterFor(this);  },    **go: function(oEvent){**  **var entity = oEvent.getSource().getBindingContext().getPath().split("/");**  **this.router.navTo("Detail1", {**  **parameter: entity[2]**  **});**  **}**  });  }); |

Listing

Notice there is now an argument in the go function called oEvent. This argument contains information about the press event that triggered the function including the data that is bound to the item that was pressed. The first line of code retrieves the path to the data bound to the list item. For the first list item, this would be /collection/0. The split function parses the path on the / character and creates an array with the parsed values. We can use the source debugger in Chrome’s developer tools to view the results.



The first item on the list was clicked to the path is /collection/0. The entity array has three items “” (before the first /), collection and 0. It’s the 0 that indicates which item was clicked so that is the value we pass the parameter in the route.

The navTo function has been modified to add a second argument which is an object that has one property. The property is the name of the parameter we added to the pattern in the Detail1 route and the value assigned to is the third item in the entity array which contains the index of the item clicked.

### Detail1.controller.js

Finally, we have to create a function in the Detail1 controller which binds the Detail1 view to the item that was clicked.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function(Controller) {  "use strict";  return Controller.extend("routing.controller.Detail1", {  onInit: function() {  **this.router = sap.ui.core.UIComponent.getRouterFor(this);**  **this.router.attachRoutePatternMatched(this.onRouteMatched, this);**  **},**  **onRouteMatched: function(oEvent) {**  **var oParameters = oEvent.getParameters();**  **if (oParameters.name !== "Detail1") {**  **return;**  **}**    **var sEntityPath = "/collection/" + oParameters.arguments.parameter;**  **var oView = this.getView();**  **var oModel = oView.getModel();**  **var context = new sap.ui.model.Context(oModel, sEntityPath);**  **oView.setBindingContext(context);**  **}**  });  }); |

Listing

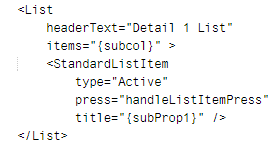
The first line in the onInit function retrieves a reference to the router object. The second line assigns a function called onRouteMatched that will execute when the Detail1 route is invoked.

The first line in onRouteMatched retrieves the parameters for the Detail1 route. The if statement makes sure it’s the Detail1 route that was invoked. This is necessary because the Detail1 route may have subroutes which would cause the onRouteMatched function to execute. If it isn’t the Detail1 route that was invoked, the return statement ends the onRouteMatched function.

Next, a binding path is created using the parameter that was passed with the route. If the first item was clicked on the Master list, sEntityPath would have the value /collection/0, the path to the item we want.

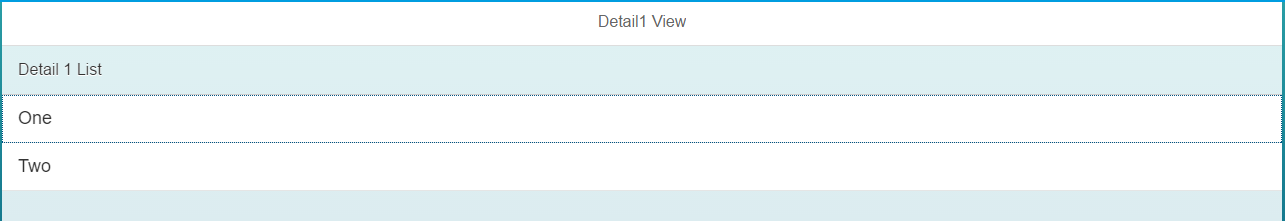
To bind the view to the object in the model we have to get a reference to the Detail1 view then use it to get the model. Since the model was created in the Component.js, it’s available to the view. Next, we use the model and the path to the item we want to create a binding context and, finally, we bind the context to the view.

Now that the view is bound to the item, we can bind the List control’s item property to the subcol collection in the item.



Notice the binding is not preceded by a /. This is because the binding is not relative to the root of the model but relative to the item bound to the Detail1 view.

Now, if you click an item on the Master list you will get the Detail1 view that has a list of the items in the subcol collection of the clicked item.



### Create a Back Button

One final change we can make is the add a Back button to the Detail1 view.

#### Detail1.view.xml

Make the following changes:

|  |
| --- |
| <mvc:View controllerName="routing.controller.Detail1"  xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m" >  <Page title="Detail1 View"  **showNavButton = "true"**  **navButtonPress = "Back"**>  <content>  <List  headerText="Detail 1 List"  items="{subcol}" >  <StandardListItem  type="Active"  title="{subProp1}" />  </List>  </content>  </Page>  </mvc:View> |

This adds a back button the interface which is bound to a function called Back.

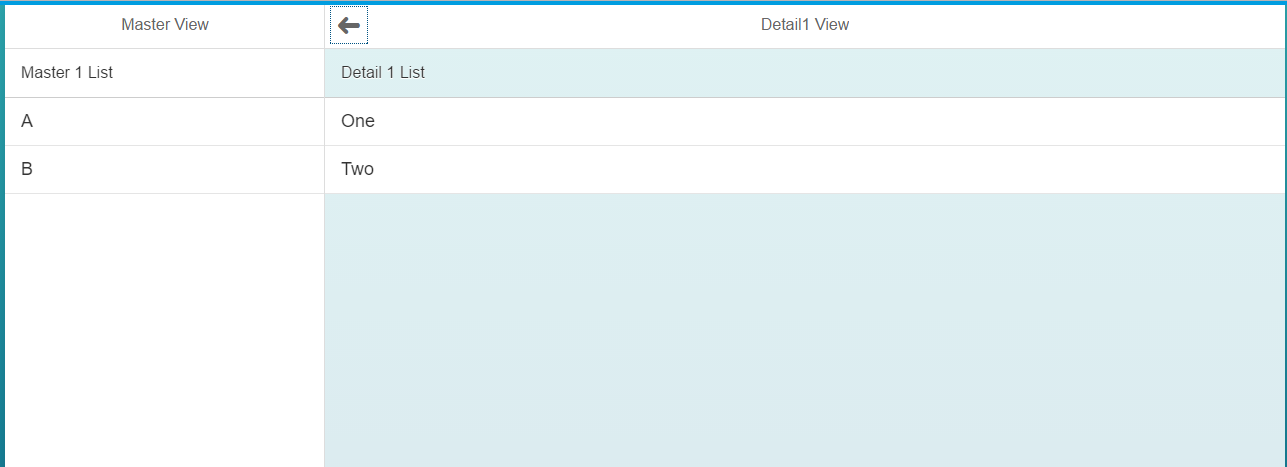
### Detail1.controller.js

Add the Back function:

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function(Controller) {  "use strict";  return Controller.extend("routing.controller.Detail1", {  onInit: function() {  this.router = sap.ui.core.UIComponent.getRouterFor(this);  this.router.attachRoutePatternMatched(this.onRouteMatched, this);  },  onRouteMatched: function(oEvent) {  var oParameters = oEvent.getParameters();  if (oParameters.name !== "Detail1") {  return;  }    var sEntityPath = "/collection/" + oParameters.arguments.parameter;  var oView = this.getView();  var oModel = oView.getModel();  var context = new sap.ui.model.Context(oModel, sEntityPath);  oView.setBindingContext(context);  }**,**    **Back: function(){**  **this.router.navTo('Master');**  **}**  });  }); |

## Create a Master/Detail Application

To create a Master/Detail application you change the App control in the App view to a SplitApp control. The SplitApp control has two aggregations for views. The masterPages aggregation is for views shown in the smaller section on the left side of the screen (see below) and the detailPages aggregation is for views shown on the larger right side of the screen.



### App.view.xml

Update the code as shown.

|  |
| --- |
| <mvc:View  xmlns:mvc="sap.ui.core.mvc"  displayBlock="true"  xmlns="sap.m" >  <**SplitApp** id="idAppControl" />  </mvc:View> |

Listing

### Component.js

Update the routes as shown below.

|  |
| --- |
| routes: [  {  pattern: "",  name: "Master",  view: "Master",  targetAggregation: "**masterPages**"  },  {  pattern: "Detail1/{parameter}",  name: "Detail1",  view: "Detail1",  targetAggregation: "**detailPages**"  }  ] |

Listing

# Exercise

Create a new view called Detail2.view.xml and implement the press event handler to navigate to the Detail2 view in the detailPages aggregation as shown below when an item on the Detail1 list is clicked.

