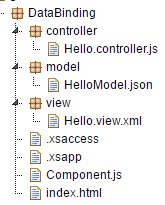
HD3C03 – Data Binding

|  |  |
| --- | --- |
| **Product and Focus**  HANA Platform/SAPUI5 | **MOTIVATION**  This case uses a simple application to explain data binding in SAPUI5 development.  **PREREQUISITES**  None |
| **Target Audience**  Undergraduate/Graduate Beginner to Intermediate |
| **Author**  Ross Hightower |
| https://bgoerke.files.wordpress.com/2013/05/section1.png | |



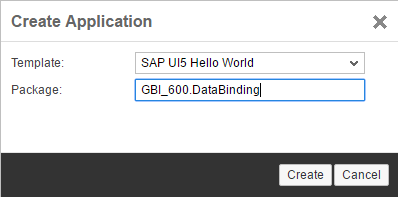
# Data Binding in SAPUI5

The link between the data in the model and controls on the view is accomplished via data binding. This allows the model to manage the flow of data between the view and model. Create the Basic Hello World App

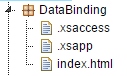
Open a browser and navigate to the URL provided by your UCC to access the WDW.

Open the **Editor**. A package has been created for you with a name equal to GBI\_### where ### is the last three digits of your user id. Right-click your package and select **Create Application.**

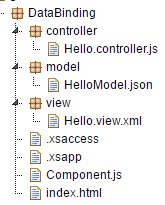
Select the **SAP UI5 Hello World** template and add .**DataBinding** to the package name. This will create a package called HelloWorld and create the application in that package.



When the application is created, three files are automatically created in the **DataBinding** package: **index.html**, **.xsaccess**, and **.xsapp.**



Use the context menu (accessed by right-clicking a package) to create three new packages: **view**, **controller** and **model** and four new files: **HelloModel.json**, **Hello.controller.js**, **Hello.view.xml** and **Component.js.**



Note, that when you first create a file, it will have a small dot next to the name. This indicates that the file hasn’t been activated in the system. Once you add code and save the file, it is activated. That completes the basic structure.

## Add the Code

Now we will add the code.

### 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>Data Binding</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-compatVersion="edge"  data-sap-ui-preload="async"  data-sap-ui-resourceroots='{  "ui5": "./"  }' >  </script>  <script>  sap.ui.getCore().attachInit(function () {  new sap.m.Shell({  app : new sap.ui.core.ComponentContainer({  name : "ui5",  height : "100%"  })  }).placeAt("content");  });  </script>  </head>  <body class="sapUiBody" id="content">  </body>  </html> |

Listing

This second script element creates a Shell control. A Shell control services as a container for the application and handles some of the tasks of adapting the app to different screen sizes. Within the Shell control, a Component is created. Components are independent and reusable parts used in SAPUI5 applications. Components are required for applications deployed to a Fiori Launchpad. When you use components, all resources, such as models, are defined relative to the Component.js file rather than the index.html file.

### Component.js

Insert the following code into the Component.js file.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/UIComponent"  ], function (UIComponent) {  "use strict";  return UIComponent.extend("ui5.Component", {  metadata : {  rootView: "ui5.view.Hello"  },  init : function () {  // call the init function of the parent  UIComponent.prototype.init.apply(this, arguments);    }  });  }); |

Listing

### Hello.view.xml

Insert the code below into Hello.view.xml.

|  |
| --- |
| <mvc:View controllerName="ui5.controller.Hello"  xmlns="sap.m" xmlns:mvc="sap.ui.core.mvc">  <StandardTile id="tileId" press="doIt" icon="sap-icon://world"  title="Hello World"  info="English" />    </mvc:View> |

Listing

This code implements a single StandardTile.

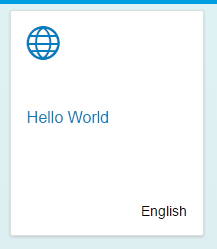
### Hello.controller.js

Finally, insert the code shown below in the Hello.controller.js file. This implements the doIt function. When the button is clicked or tapped the doIt function will show a toast message that includes the Button’s id. The oEvent parameter is an object that contains information about the event that invoked the function. The function oEvent.getSource() returns the control that initiated the event and getId() returns the control’s id.

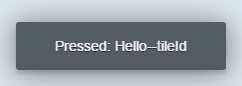
|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller",  "sap/m/MessageToast"  ], function (Controller, MessageToast) {  "use strict";  return Controller.extend("ui5.controller.Hello", {  doIt : function (oEvent) {  MessageToast.show("Pressed: " + oEvent.getSource().getId());  }  });  }); |

Listing

Save all your files and then run the application by selecting the index.html file and clicking the run icon.



If you click the tile, the toast message is displayed. The id is constructed from the name of the view and the id assigned to the Button control.



# Add a Model

We’ve demonstrated the overall architecture of SAPUI5 applications and the functions of views and controllers. In this section we’ll demonstrate the use of models.

### HelloModel.json

Add the following code to the **HelloModel.json** file in the model package.

|  |
| --- |
| {    "Language" : "English",  "Greeting" : "Hello World!"    } |

Listing

This is standard JSON (JavaScript Object Notation) code. It has a single object with two properties.

### Component.js

Add the model definition to the Component.js file as shown below in the highlighted code.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/UIComponent"**,**  **"sap/ui/model/json/JSONModel"**  ], function (UIComponent**, JSONModel**) {  "use strict";  return UIComponent.extend("ui5.Component", {  metadata : {  rootView: "ui5.view.Hello"  },  init : function () {  // call the init function of the parent  UIComponent.prototype.init.apply(this, arguments);    **var oModel = new JSONModel("model/HelloModel.json");**  **this.setModel(oModel, "hello");**    }  });  }); |

Listing

This creates a JSON model that references the HelloModel.json file. It then adds it to the component and names the model hello.

### Hello.view.xml

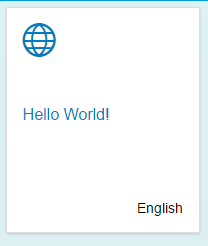
Update the StandardTile control as shown in the highlighted code below.

|  |
| --- |
| <mvc:View controllerName="ui5.controller.Hello"  xmlns="sap.m" xmlns:mvc="sap.ui.core.mvc">  <StandardTile id="tileId" press="doIt" icon="sap-icon://world"  title="**{hello>/Greeting}**"  info="**{hello>/Language}**" />    </mvc:View> |

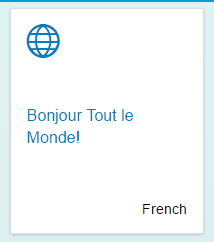
Listing

In UI5, curly brackets, { }, are used to indicate a binding to a model. If the model is given a name as our is, hello, then the name of the model followed by a > is used to indicate the model that is the source of the binding. The / indicates that UI5 should begin looking for the data to bind at the root of the data model. In our case, the data only has one object so the root of the model is that one object. The /Greeting then means to bind to the Greeting property in that data.

If you run the application again, you will see that is hasn’t changed. That’s because we used the same values for the properties that we used for the properties before.



However, if you change the values in the HelloModel.json file and refresh the application, the changes are reflected in the tile without having to change the code in the view.



## A More Complex Model

Now, let’s increase the complexity of the data in the model.

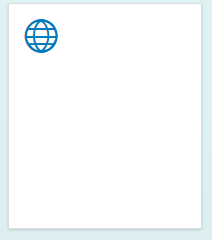
### HelloModel.json

Replace the code with the code shown below.

|  |
| --- |
| {  "HelloCollection": [  {  "Language" : "English",  "Greeting" : "Hello World!"  },  {  "Language" : "français",  "Greeting" : "Bonjour le monde!"  }  ]  } |

Listing 9

This code illustrates how collections of objects are modeled in JSON. It consists of a collection (a JavaScript array) called HelloCollection that contains two objects. If you refresh the application now, you will see this.



The problem is SAPUI5 doesn’t know which of the two objects to bind to so we have to tell it which one.

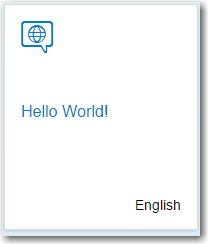
### Hello.view.xml

Update the highlighted portions of the code as shown below.

|  |
| --- |
| <mvc:View controllerName="ui5.controller.Hello"  xmlns="sap.m" xmlns:mvc="sap.ui.core.mvc">  <StandardTile id="tileId" press="doIt" icon="sap-icon://world"  title="{hello>/HelloCollection/0/Greeting}"  info="{hello>/HelloCollection/0/Language}" />  </mvc:View> |

Listing

In binding syntax in SAPUI5 represents array indices with a slash, /, followed by the index of the object. JavaScript arrays are zero based (the first object has an index of 0), so the first item in the HelloCollection array in JavaScript would be HelloCollection[0] whereas with SAPUI5 binding it would be HelloCollection/0/. If you run the application now, the tile is bound to the properties in the first object.



Change the binding to **hello>/HelloCollection/1** and you will see this:



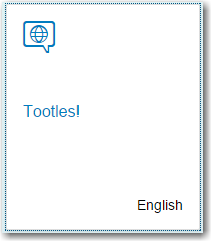
## An Even More Complex Model

Now, replace the code in the JSON file with this code:

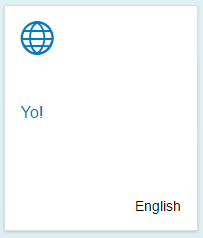
|  |
| --- |
| {  "HelloCollection": [  {  "Language" : "English",  "Greeting" : "Hello World!",  "Variations" : [  {  "Language" : "English",  "Greeting" : "Tootles!"  },  {  "Language" : "English",  "Greeting" : "Yo!"  }  ]  },  {  "Language" : "français",  "Greeting" : "Bonjour le monde!"  }  ]  } |

Listing

We still have the HelloCollection array but now we’ve added a property called Variations and it’s also a collection. In JavaScript, the English variation of the first object in HelloCollection would be HelloCollection[0].Variations[0]. The binding path to the Language property of the first English variation would be /HelloCollection/0/Variations/0/Language.



How would you bind to the second English variation?



## Aggregation Binding

The previous example bound a single object to a control. Some controls have aggregations that allow you to bind multiple objects to the controls. Examples are the TileContainer, List and Table controls. Let’s see how that works.

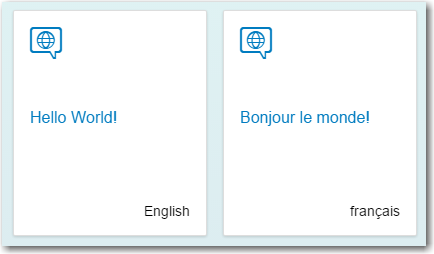
### Hello.view.xml

Replace the StandardTile control with the code shown below.

|  |
| --- |
| <mvc:View height="100%" controllerName="ui5.controller.Hello"  xmlns="sap.m" xmlns:mvc="sap.ui.core.mvc">  <TileContainer tiles="{hello>/HelloCollection}">  <StandardTile id="tileId" press="doIt" icon="sap-icon://world"  title="{hello>Greeting}"  info="{hello>Language}" />  </TileContainer>  </mvc:View> |

Listing

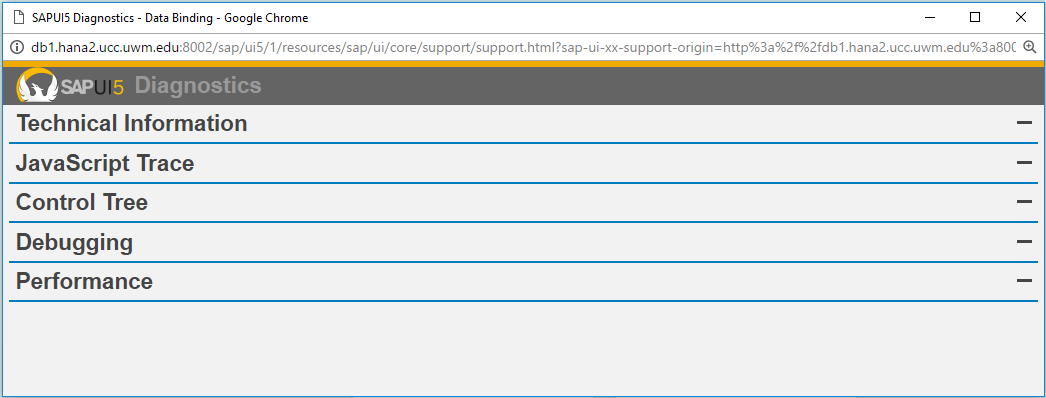
The TileContainer control has an aggregation called tiles. We can bind the HelloCollection array of the hello model to that aggregation in the view code. Then we supply the StandardTile as a template for depicting each object in the HelloCollection array. SAPUI5 will create a StandardTile for each object in the HelloCollection array. Note that the bindings in the StandardTile control omit the leading slash. This is because these bindings are not relative to root of the data model but are relative the object being processed by the TileContainer control.



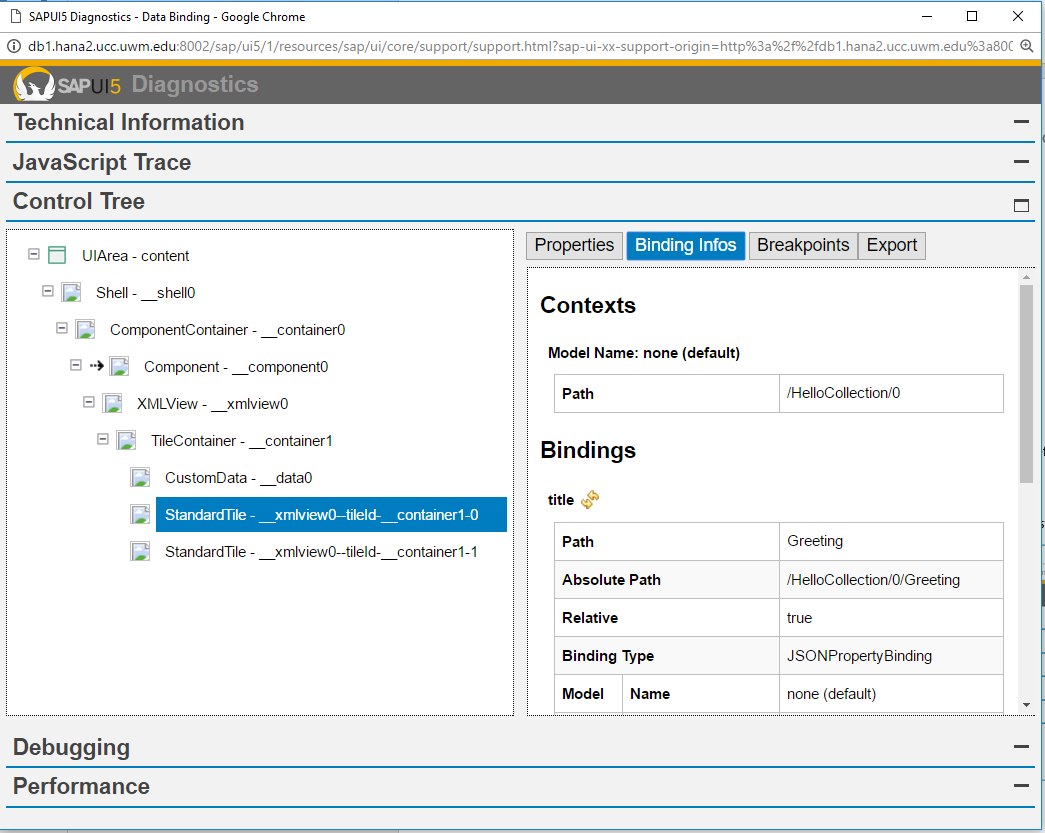
Managing data binding is one of the more challenging aspects of SAPUI5 to master but hopefully this has given you the beginnings of a deeper understanding.

## Diagnostics

Make sure the application is running and click CTRL-SHIFT-ALT-S. This opens the UI5 diagnostic tool.



Drill into the Control Tree (you have to click the – on the right side) and you can get information about the control bindings.



This can be very helpful when you are troubleshooting binding issues.

# Exercise

How would you bind the TileContainer to the English variations?

