HD3C02 – Application Structure

|  |  |
| --- | --- |
| **Product and Focus**  HANA Platform/SAPUI5 | **MOTIVATION**  This case illustrates additional techniques for structuring UI5 applications.  **PREREQUISITES**  HD3C01 – Hello World |
| **Target Audience**  Undergraduate/Graduate Beginner to Intermediate |
| **Author**  Ross Hightower |
| https://bgoerke.files.wordpress.com/2013/05/section1.png | |

## Application Structure

SAPUI5 includes a number of techniques that allow you to structure and modularize applications. These techniques become increasingly important when you create larger, more complex, applications or when you create applications that will be deployed to the Fiori Launchpad.

|  |
| --- |
| This project begins where HD3C01 – Hello World ends. You need the completed Hello World app to complete this project. |

## Structuring the Application

### Components

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 are defined relative to the Component.js file rather than the index.html file.

The basic structure of a component is shown below.

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

Listing

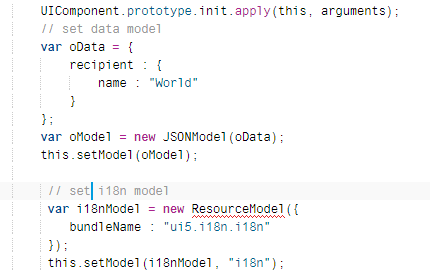
The first two lines initialize the component. Once the component is initialized, function(UIComponent) is invoked which configures the component. In the init function in the code shown above, simply creates a prototypical component. So far there is no configuration that is unique to our application.

We will demonstrate components by extending the Hello World application you created in the previous case. Create a file called Component.js in the webapp package. The code below shows the component code tailored to our application. Insert this code into Component.js.

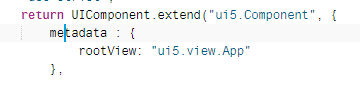
|  |
| --- |
| sap.ui.define([  "sap/ui/core/UIComponent",  "sap/ui/model/json/JSONModel",  "sap/ui/model/resource/ResourceModel"  ], function (UIComponent, JSONModel, ResourceModel) {  "use strict";  return UIComponent.extend("ui5.Component", {  metadata : {  rootView: "ui5.view.App"  },  init : function () {  // call the init function of the parent  UIComponent.prototype.init.apply(this, arguments);  // set data model  var oData = {  recipient : {  name : "World"  }  };  var oModel = new JSONModel(oData);  this.setModel(oModel);  // set i18n model  var i18nModel = new ResourceModel({  bundleName : "ui5.i18n.i18n"  });  this.setModel(i18nModel, "i18n");  }  });  }); |

Listing

In the Hello World app, we defined the models in the App controller. When we do this, the models are only available to the App view. If we want the models to be available throughout our application, we define them in the Component.js file. All of the code to create the JSON and resource models has been moved from the controller file to the Component.js file.



The other thing this file accomplishes is the set the App view as the root view of the application. This will cause UI5 to load the file when the app loads.



Now that we are creating the models in the Component.js file, we can delete that code from the controller. Update the code in the controller file so that it looks like the code shown below.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller",  "sap/m/MessageToast"  ], function (Controller, MessageToast) {  "use strict";  return Controller.extend("ui5.controller.App", {  onShowHello : function () {  // read msg from i18n model  var oBundle = this.getView().getModel("i18n").getResourceBundle();  var sRecipient = this.getView().getModel().getProperty("/recipient/name");  var sMsg = oBundle.getText("helloMsg", [sRecipient]);  // show message  MessageToast.show(sMsg);  }  });  }); |

Listing

Finally, since the Component.js file loads the App view, we must modify the index.html file so that it loads the Component.js file rather than the App view. Update the index.html file as shown below.

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <meta charset="utf-8">  <title>Walkthrough</title>  <script  id="sap-ui-bootstrap"  src="https://openui5.hana.ondemand.com/1.42.6/resources/sap-ui-core.js"  data-sap-ui-theme="sap\_bluecrystal"  data-sap-ui-libs="sap.m"  data-sap-ui-bindingSyntax="complex"  data-sap-ui-compatVersion="edge"  data-sap-ui-preload="async"  data-sap-ui-resourceroots='{  "ui5": "./"  }' >  </script>  <script>  sap.ui.getCore().attachInit(function () {  **new sap.ui.core.ComponentContainer({**  **name : "ui5"**  }).placeAt("content");  });  </script>  </head>  <body class="sapUiBody" id="content">  </body>  </html> |

Listing

If you run the app, it should look and behave the same.



### Application Descriptor

One of the principles of modern application development is the separate configuration data from the application code as much as possible. This makes applications easier to maintain.

Create a file called manifest.json in the webapp package and insert the code shown below.

|  |
| --- |
| {  "\_version": "1.1.0",  "sap.app": {  "\_version": "1.1.0",  "id": "sap.ui.demo.wt",  "type": "application",  "i18n": "i18n/i18n.properties",  "title": "{{appTitle}}",  "description": "{{appDescription}}",  "applicationVersion": {  "version": "1.0.0"  }  },  "sap.ui": {  "\_version": "1.1.0",  "technology": "UI5",  "deviceTypes": {  "desktop": true,  "tablet": true,  "phone": true  },  "supportedThemes": [  "sap\_bluecrystal"  ]  },  "sap.ui5": {  "\_version": "1.1.0",  "rootView": "ui5.view.App",  "dependencies": {  "minUI5Version": "1.30",  "libs": {  "sap.m": {}  }  },  "models": {  "i18n": {  "type": "sap.ui.model.resource.ResourceModel",  "settings": {  "bundleName": "ui5.i18n.i18n"  }  }  }  }  } |

Listing

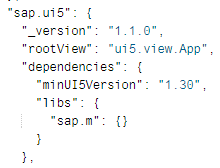
There are various sections of this file which organize the configuration information into categories. The sap.app section contains general application parameters.



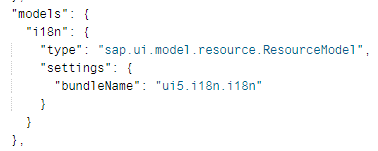
The sap.ui section contain parameters relevant for the user interface.



The sap.ui5 section contains parameters relevant for UI5. One of these parameters is rootView which defines the view that loads initially when the application runs. Note that this was previously configured in the Component.js file.



The models section configures models and is used, in this case, to configure he resource model. We can’t configure the JSON model because it relies on data we create in the component.



Note in the sap.app section of the manifest we created two variables appTitle and appDescription. The values for these are defined in the properties files. Update the i18n.properties file as shown below. Note the use of comments that begin with #.

|  |
| --- |
| # App Descriptor  appTitle=Hello World  appDescription=A simple walkthrough app that explains the most important concepts of SAPUI5  # Hello Panel  showHelloButtonText=Say Hello  helloMsg=Hello {0} |

Listing

Since we moved the configuration of the rootView and resource model to the manifest.json file, we have to remove that configuration from the Component.js file. We also have to configure the component to look for the manifest.json file. Update the Component.js file as shown below.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/UIComponent",  "sap/ui/model/json/JSONModel"  ], function (UIComponent, JSONModel) {  "use strict";  return UIComponent.extend("ui5.Component", {  **metadata : {**  **manifest: "json"**  **},**  init : function () {  // call the init function of the parent  UIComponent.prototype.init.apply(this, arguments);  // set data model  var oData = {  recipient : {  name : "World"  }  };  var oModel = new JSONModel(oData);  this.setModel(oModel);  }  });  }); |

Listing

The app still looks the same but it is now properly organized according to SAPUI5 standards.



## Structuring the Interface

In this section, we’ll add some structure and styling to the interface.

### Pages and Panels

The Pages and Panels controls, which are included in the sap.m library, are both used to add structure to an application’s interface. Update the code in the App.view.xml file as shown below.

|  |
| --- |
| <mvc:View  controllerName="ui5.controller.App"  xmlns="sap.m"  xmlns:mvc="sap.ui.core.mvc"  **displayBlock="true">**  **<App>**  **<pages>**  **<Page title="{i18n>homePageTitle}">**  **<content>**  **<Panel**  **headerText="{i18n>helloPanelTitle}">**  **<content>**  <Button  text="{i18n>showHelloButtonText}"  press="onShowHello"/>  <Input  value="{/recipient/name}"  description="Hello {/recipient/name}"  valueLiveUpdate="true"  width="60%"/>  **</content>**  **</Panel>**  **</content>**  **</Page>**  **</pages>**  **</App>**  </mvc:View> |

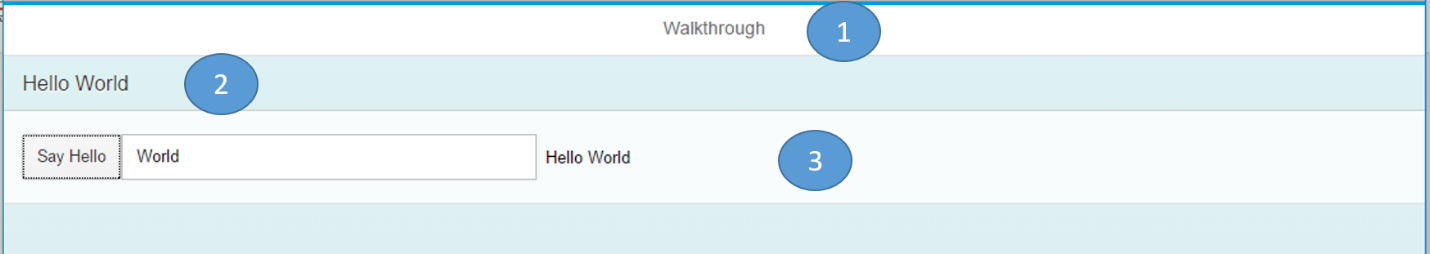
Listing

We have additional text fields in the App view for the Page and Panel headers so update the properties file as shown below.

|  |
| --- |
| # App Descriptor  appTitle=Hello World  appDescription=A simple walkthrough app that explains the most important concepts of SAPUI5  # Hello Panel  showHelloButtonText=Say Hello  helloMsg=Hello {0}  **homePageTitle=Walkthrough**  **helloPanelTitle=Hello World** |

Listing

When you run the app now it has some structure.



This code adds a Page control which has the general structure:

<Page>

<content>

</content>

</Page>

The Page control adds the header (1 in the figure). It has an aggregation called content which can contain other controls. Aggregations, in UI5, allow you to assign or bind multiple items to them. In this case, the page contains a Panel control.

The Panel control has a header (2 in the figure) and a body. The body of the Panel control has an aggregation called content to which we can assign other controls.

The Page control is contained within an App control. The App control provides some configuration that will enable the app to run well on mobile devices and provides animation on navigation.

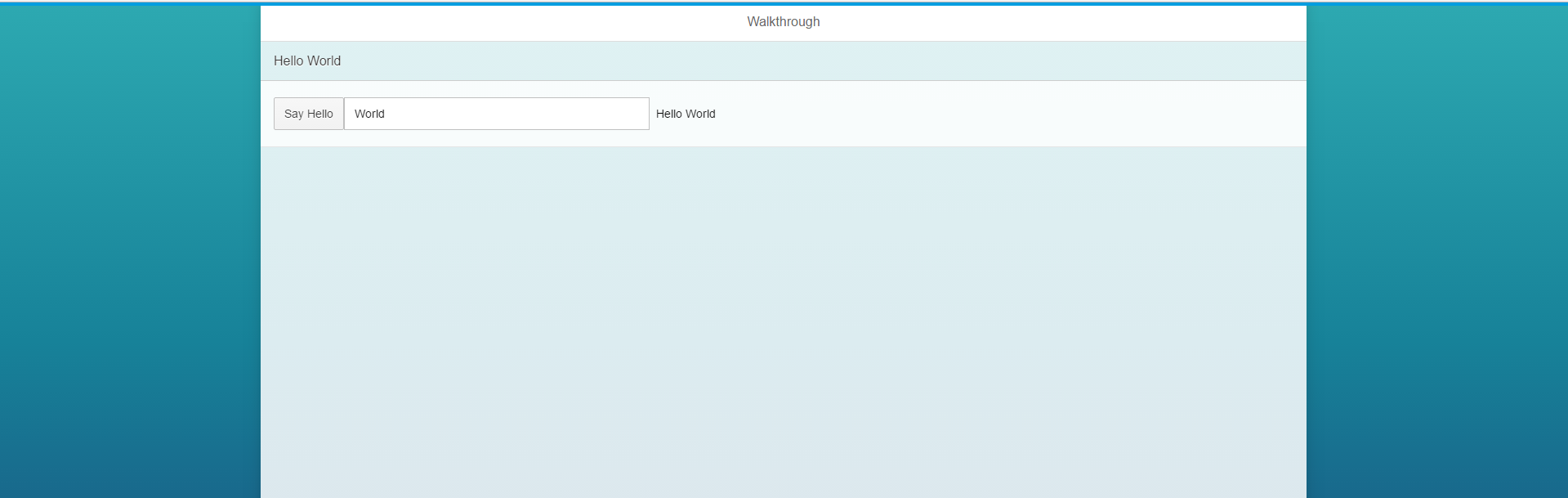
### Shell Control as Container

A Shell control services as a container for the application and handles some of the tasks of adapting the app to different screen sizes. To use the Shell control, update the code in the index.html file as shown below.

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <meta charset="utf-8">  <title>Walkthrough</title>  <script  id="sap-ui-bootstrap"  src="https://openui5.hana.ondemand.com/1.42.6/resources/sap-ui-core.js"  data-sap-ui-theme="sap\_bluecrystal"  data-sap-ui-libs="sap.m"  data-sap-ui-bindingSyntax="complex"  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

The Shell control adds alleys on either side of the app on large screens.



## Styling

Next, we’ll add some styling the application using built in UI5 styling as well as custom CSS.

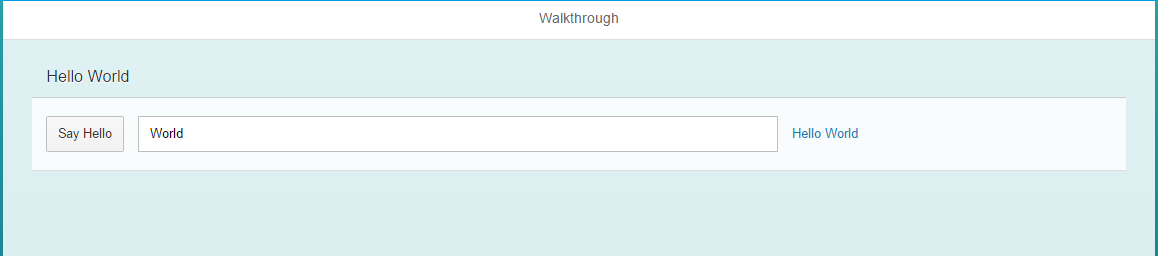
### Builtin UI5 Styles

UI5 includes several classes you can use for styling. These are an alternative to writing your own custom CSS and are preferable to custom CSS because they are designed to work well with different screen sizes and create a consistent look-and-feel. Update the code in the App.view.xml file as shown below.

|  |
| --- |
| <mvc:View  controllerName="ui5.controller.App"  xmlns="sap.m"  xmlns:mvc="sap.ui.core.mvc"  displayBlock="true">  <App>  <pages>  <Page title="{i18n>homePageTitle}">  <content>  <Panel  headerText="{i18n>helloPanelTitle}"  **class="sapUiResponsiveMargin"**  **width="auto"**>  <content>  <Button  text="{i18n>showHelloButtonText}"  press="onShowHello"  class="sapUiSmallMarginEnd"/>  <Input  value="{/recipient/name}"  valueLiveUpdate="true"  width="60%"/>  <Text  text="Hello {/recipient/name}"  **class="sapUiSmallMargin sapThemeHighlight-asColor"**/>  </content>  </Panel>  </content>  </Page>  </pages>  </App>  </mvc:View> |

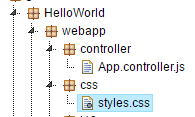
Listing

The result is shown below. There is now, is more appealing.



### Custom Styles and CSS

Finally, we will add some custom styles. Create a new package in the webapp package called css and create a file in the new package called styles.css.



Insert the code shown below into the new file.

|  |
| --- |
| .myAppDemoWT .myCustomButton.sapMBtn {  margin-right: 0.125rem  }  html[dir="rtl"] .myAppDemoWT .myCustomButton.sapMBtn {  margin-left: 0.125rem;  margin-right: 0  }  .myAppDemoWT .myCustomText {  font-weight: bold;  }  .panelHeader {  background-color: magenta;  } |

Listing

To enable the application to find the styles.css file, add the highlighted portion of the code below to the manifest.json file.

|  |
| --- |
| {  "\_version": "1.1.0",  "sap.app": {  "\_version": "1.1.0",  "id": "ui5",  "type": "application",  "i18n": "i18n/i18n.properties",  "title": "{{appTitle}}",  "description": "{{appDescription}}",  "applicationVersion": {  "version": "1.0.0"  }  },  "sap.ui": {  "\_version": "1.1.0",  "technology": "UI5",  "deviceTypes": {  "desktop": true,  "tablet": true,  "phone": true  },  "supportedThemes": [  "sap\_bluecrystal"  ]  },  "sap.ui5": {  "\_version": "1.1.0",  "rootView": "ui5.view.App",  "dependencies": {  "minUI5Version": "1.30",  "libs": {  "sap.m": {}  }  },  "models": {  "i18n": {  "type": "sap.ui.model.resource.ResourceModel",  "settings": {  "bundleName": "ui5.i18n.i18n"  }  }  **},**  **"resources": {**  **"css": [**  **{**  **"uri": "css/styles.css"**  **}**  **]**  **}**  }  } |

Listing

Now update the code in the App.view.xml file to assign the styles as shown below.

|  |
| --- |
| <mvc:View  controllerName="ui5.controller.App"  xmlns="sap.m"  xmlns:mvc="sap.ui.core.mvc"  displayBlock="true">  <App **class="myAppDemoWT"**>  <pages>  <Page title="{i18n>homePageTitle}">  <content>  <Panel  headerText="{i18n>helloPanelTitle}"  class="sapUiResponsiveMargin **panelHeader**"  width="auto">  <content>  <Button  text="{i18n>showHelloButtonText}"  press="onShowHello"  **class="myCustomButton"**/>  <Input  value="{/recipient/name}"  valueLiveUpdate="true"  width="60%"/>  <Text  text="Hello {/recipient/name}"  class ="sapUiSmallMargin sapThemeHighlight-asColor **myCustomText"**/>  </content>  </Panel>  </content>  </Page>  </pages>  </App>  </mvc:View> |

Listing

The Button is now closer to the Input control and the Panel and Panel header are ugly colors.

