Week 1 Unit 5

Managing Data Models and Internationalization

Please perform the exercises below in your app project as shown in the video.

Table of Contents

[1 Managing Data Models 2](#_Toc449007239)

[2 Internationalization 4](#_Toc449007240)

## Preview

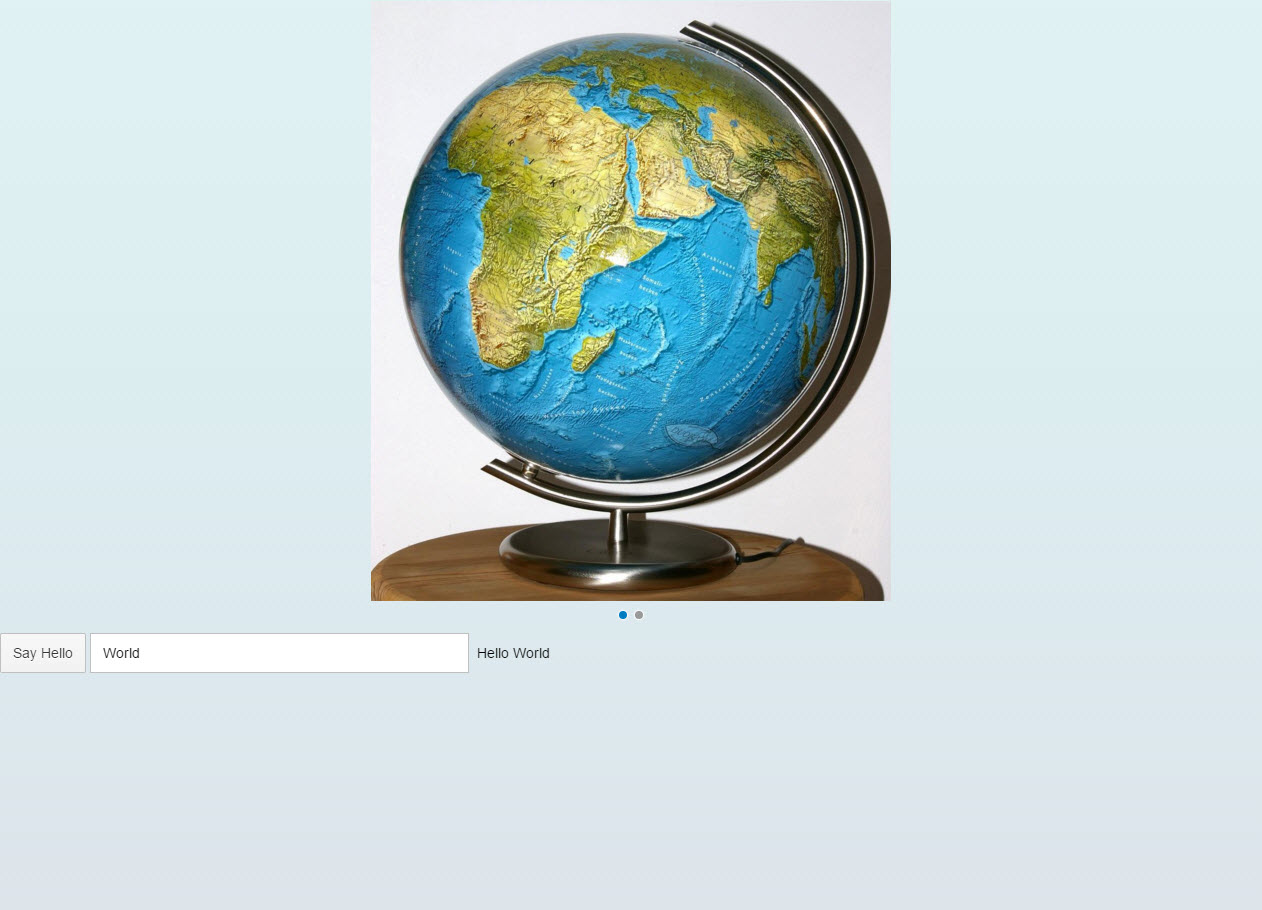


Figure 1 - Preview of the app after doing this unit’s exercises

# Managing Data Models

In this step, we add an input field, bind it to data within our new model and show the user input within the description next to it.

Preview

http://veui5infra.dhcp.wdf.sap.corp:8080/demokit/docs/guide/loioe5a9bb4cb06b4d9c8b37914bf5cd2d13_HiRes.png

Figure 2: An Input with data binding is added

### webapp/model/HelloPanel.json (NEW)

|  |
| --- |
| **{**  **"recipient" : {**  **"name" : "World"**  **}**  **}** |

To achieve this, we will add a JSON model as container for the data on which our application operates. Therefore we create the folder model within the webapp folder of our app, and the file HelloPanel.json inside this folder. As content for the file, we only need one recipient which has an additional property for the name.

### webapp/manifest.json

|  |
| --- |
| {  ...  "sap.ui5": {  …  **"models": {**  **"helloPanel": {**  **"type": "sap.ui.model.json.JSONModel",**  **"uri": "model/HelloPanel.json"**  **}**  **},**  …  }  } |

Now we have to define the model within the manifest.json. We add a new model helloPanel to the sap.ui5 section of the descriptor. We use a JSON model, so we set the type to sap.ui.model.json.JSONModel. The uri property holds the path to our test data relative to the component. With this little configuration, our component automatically instantiates a new JSON model which loads its data from the HelloPanel.json file. Finally, the instantiated JSON model is made available to the component as a named model helloPanel.

### webapp/view/App.view.xml

|  |
| --- |
| <mvc:View …>  …  <Button  text="Say Hello!"  press="onShowHello"/>  **<Input**  **value="{helloPanel>/recipient/name}"**  **description="Hello {helloPanel>/recipient/name}"**  **valueLiveUpdate="true"**  **width="60%"/>**  </mvc:View> |

The curly brackets enclosing a binding path (binding syntax) are automatically interpreted as a data binding. These binding instances are called property bindings: The control's value property is bound to the recipient name property at the root of our named model, which is stated first. The slash (/) at the beginning of the binding path denotes an absolute binding path.

The description uses a so-called complex binding syntax, as it is a combination of text and data binding. The complex syntax is not enabled by default, so we have to explicitly enable it within our app. This can be done within the index.html with parameter data-sap-ui-compatVersion="edge" on the SAPUI5 bootstrap tag.

### webapp/index.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  …  <script  id="sap-ui-bootstrap"  src="https://sapui5.hana.ondemand.com/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='{  "opensap.myapp": "./"  }'>  </script>  …  </head>  …  </html> |

Finally, your app should look like the preview picture above. When you change something within the input field, the label next to it is automatically updated. This is because

* we have valueLiveUpdate enabled on the input field,
* both controls’ properties are bound to the same property within the JSON model,
* And the model uses a two-way binding.

# Internationalization

Within this step, we will prepare for internationalization (i18n).

Preview

http://veui5infra.dhcp.wdf.sap.corp:8080/demokit/docs/guide/loioe5a9bb4cb06b4d9c8b37914bf5cd2d13_HiRes.png

Figure 3: A message toast displays the "Hello World" message which comes from the resource bundle

### webapp/i18n/i18n.properties (NEW)

|  |
| --- |
| **# Hello Panel**  **showHelloButtonText=Say Hello**  **helloMsg=Hello {0}** |

The implementation we just did was overly simplistic as we stored language-specific text directly in a JSON model object. Generally speaking, unless language-specific text is derived directly from a back-end system which already takes care of translation, it is not considered good programming practice to place translatable texts directly into a model. So let's correct this situation by placing all translatable texts (such as field labels) into a translatable resource bundle.

Create an i18n folder within the webapp folder. Inside this new folder, create a new file called i18n.properties. Within this file, we put the text as name-value pairs. If you need parameters in the text, put a number (starting with 0) within a curly bracket as placeholder for each parameter in the appropriate position of the text. Never concatenate strings that are translated, as the order of words may differ between different languages, so concatenation might lead to unexpected results in certain languages.

Right now we only created the default i18n file, which is used if you do not provide a specific language file for a language that is used with your app. In a productive app, provide another i18n file for each language you support, for example for English i18n\_en.properties.

**Note: SAP Translation Hub**

If you need an initial translation to a foreign language of the applications texts, you can use the translation workflow of SAP Translation Hub (available as a beta version on the trial landscape of SAP HANA Cloud Platform). To see the translation workflow in action, check out the video – [Translating HTML5 Apps](https://www.youtube.com/watch?v=nQt5euCU288). Or try it out by following the steps in this SCN Blog: [Translation Hub Tutorial](http://scn.sap.com/docs/DOC-72415)

A technical prerequisite for this service is that your i18n property files are stored in the git repository of SAP HANA Cloud Platform (HCP). The process of deploying apps to HCP will be explained in week 3 unit 1 of this course. All files of your project are automatically added to a git repository during deployment.

### webapp/manifest.json

|  |
| --- |
| {  "\_version": "1.3.0",  "sap.app": {  "\_version": "1.3.0",  "id": "opensap.myapp",  "type": "application",  **"i18n": "i18n/i18n.properties",**  ...  "sap.ui5": {  …  "models": {  **"i18n": {**  **"type": "sap.ui.model.resource.ResourceModel",**  **"settings": {**  **"bundleName": "opensap.myapp.i18n.i18n"**  **}**  **},**  "helloPanel": {  "type": "sap.ui.model.json.JSONModel",  "uri": "model/HelloPanel.json"  }  },  ...  }  } |

Now we use the i18n file we just created. Therefore we define the ResourceModel in manifest.json and state the location of the i18n file in bundleName. The bundle name consists of the application namespace (the application root as defined in the index.html), the folder name i18n, and finally the file name i18n without extension. The SAPUI5 runtime calculates the correct path to the resource, in this case the path to our i18n.properties file. Next, the model instance is set on the view as model named i18n.

### webapp/controller/App.controller.js

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

Now we want to use an i18n text. The resource bundle can be accessed via the getResourceBundle method of a ResourceModel. Rather than concatenating translatable texts manually, we can use the second parameter of getText to replace parts of the text with dynamic data. During runtime, SAPUI5 tries to load the correct i18n\_\*.properties file based on the current language of the user. In SAPUI5 applications started via their own HTML file, this language depends on your browser settings and your locale. When your app runs in the SAP Fiori launchpad, there are some more aspects that influence the effective language. In our case we have only created the default i18n.properties file to keep this unit simple. However, you can see in the network trace of your browser’s developer tools that SAPUI5 tries to load one or more i18n\_\*.properties files before falling back to the default i18n.properties file. In the onShowHello event handler, we access the i18n model to get the text from the message bundle file and replace the placeholder {0} with the recipient name from our data model. The getProperty method can be called in any model and takes the path in the model as an argument. In addition, the resource bundle has a specific getText method that expects an array of strings as second argument, which replace placeholders in the translated text.

### webapp/view/App.view.xml

|  |
| --- |
| <mvc:View …>  …  <Button  **text="{i18n>showHelloButtonText}"**  press="onShowHello"/>  <Input  value="{helloPanel>/recipient/name}"  description="Hello {helloPanel>/recipient/name}"  valueLiveUpdate="true"  width="60%"/>  </mvc:View> |

As last step, we use data binding to connect the button text to the sayHelloButtonText property in the i18n model. A resource bundle is a flat structure, therefore the leading slash (/) can be omitted from the path. Your app should look like before, but as we saw in the debugger, the text is not hardcoded anymore, but language-dependent. When you enter something before in the input field and press the button, you see the information within the MessageToast.

**Note: Special translation features for developers in SAP Web IDE**

Save time when entering texts with the SAP Translation Hub’s suggestion service:

• Auto completion of texts that you enter in the view file based on a central text repository

• Automatic connection of the button text to the property in the i18n.properties file

To call the suggestion service, simply hit Ctrl + Space in the code editor. For more details, check out [this blog on SCN](http://scn.sap.com/community/globalization/blog/2016/02/22/save-time-with-sap-translation-hub-in-sap-web-ide).

### webapp/i18n/i18n.properties

|  |
| --- |
| **# App Descriptor**  **appTitle=My demo app**  **appDescription=A simple demo app**  # Hello Panel  showHelloButtonText=Say Hello  helloMsg=Hello {0} |

Additionally, we now add the app title and app description to the i18n.properties file, which we are use in the manifest.json.

**Note: On the fly translation service in SAP Web IDE**

After adding all required texts to your i18n.properties file, you can translate your texts on the fly using SAP Translation Hub’s translation service. Simply right click the i18n.properties file in your app project and choose Generate translation files. The translation service translates the texts and creates language specific files (i18n\_<language>.properties) in the i18n folder.

### Conventions

* The resource model for internationalization is registered with name i18n.
* The default filename is i18n.properties.
* Resource bundle keys are written in (lower) camelCase.
* Resource bundle values can contain parameters like {0}, {1}, {2}, …
* Never concatenate strings that are translated, always use placeholders.
* Use Unicode escape sequences for special characters.

### Related Information

[Data Binding and Backend Services (OData)](https://sapui5.hana.ondemand.com/docs/guide/90e370925add47fdba3cbd01ca7b9ab3.html)

[Instantiating a JSON Model](https://sapui5.hana.ondemand.com/docs/guide/91f107eb6f4d1014b6dd926db0e91070.html)

[Instantiating a Resource Model](https://sapui5.hana.ondemand.com/docs/guide/91f122a36f4d1014b6dd926db0e91070.html)

[SAP Translation Hub - SAP Hana Cloud Platform service](https://hcp.sap.com/capabilities/dev-ops/translation-hub.html)

[SCN Blog - Translation Hub Tutorial](http://scn.sap.com/docs/DOC-72415)