HD3C01 – Hello World

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

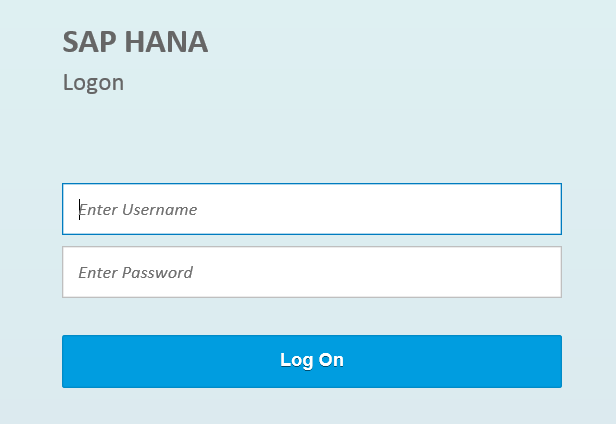
# Hello World MVC App

The traditional first application in any language is the Hello World application. Usually very simple, it’s intended to teach the basic structure of applications and procedures for working with a language. In this case you will create a basic Hello World application and then you’ll use it to practice some basic SAPUI5 concepts.

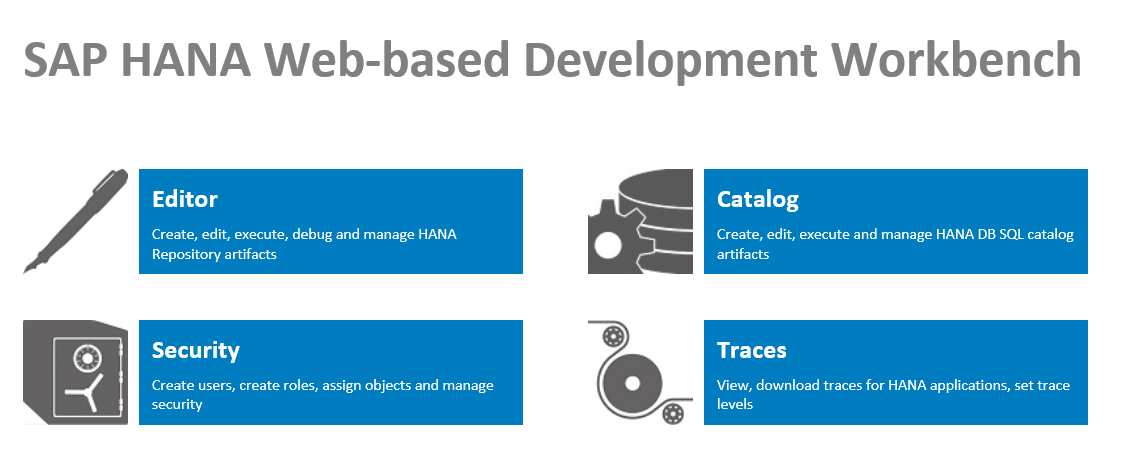
|  |
| --- |
| This series of cases uses HANA’s [Web-Based Development Workbench](https://help.hana.ondemand.com/help/frameset.htm?b8e2e1d42500483caaf0b009a492917f.html) (WDW). As a development environment, the WDW has some limitations. For one thing, as is often the case with online applications, it is only partially compatible with each browser. I normally use Chrome because of [Chrome’s developer tools](https://developer.chrome.com/devtools). However, I’ve found that Chrome does not show all the options on the context menus in the WDW editor. As a result, I use FireFox when using the WDW but often use Chrome’s developer tools to debug applications. FireFox also has an extension called FireBug which is a very capable set of developer tools as well.  In spite of the limitations of the WDW compared to the alternative which is based on Eclipse, the WDW has two advantages. One is that there is no local installation. For many institutions eliminating the logistics of software installations is a significant advantage. The second advantage of using the WDW is that SAP has made it clear the WDW, along with the WebIDE, are their development platforms of the future. While the Eclipse based platform will be supported, more investment will be made in the web-based environments. If you would prefer to work with the Eclipse based platform, there is an alternative set of cases available. |

## Create the Basic Hello World App

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



Log in using the credentials provided by the UCC.



The WDW has the following components:

**Editor** – This is the editor for design time artifacts such as applications, calculation views and core data services files. It’s the editor you will use most of the time.

**Catalog** – This is used to view repository artifacts such as database tables, views and procedures. It has a SQL Console which allows you to execute SQL.

**Security** – Allows the management of security including creating uses, roles and making security assignments.

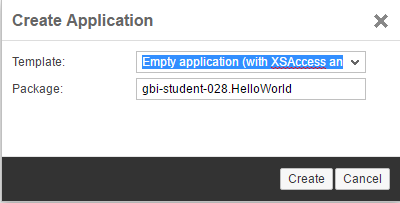
**Traces** – Allows you to view trace files.

|  |
| --- |
| One helpful tip for using the WDW is that the connection to HANA times out and there are times you will not receive an overt message that this has occurred. For example, you may find that a file does not open even when you click it. When in doubt, refresh the browser to refresh your connection to the server. |

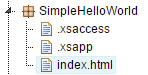
Open the **Editor**. A package has been created for you with a name equal to GBI\_### where ### is the last three digits of your user id. Right-click your package and select **Create Application.**

### Create the Application

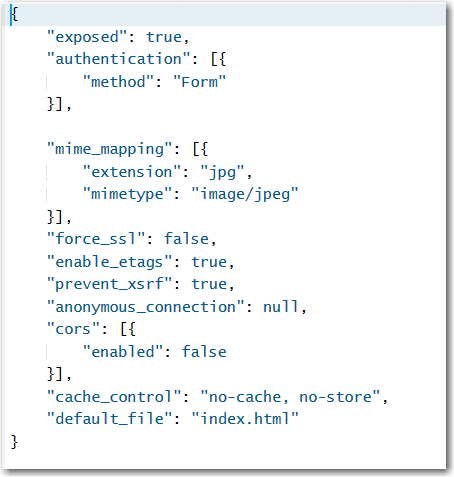
Right click your package and select Create application. Enter .HelloWorld as shown in the image below and click Create.



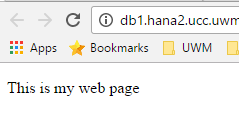
When the application is created, three files are automatically created in the **SimpleHelloWorld** package: **index.html**, **.xsaccess**, and **.xsapp.**



The .xsapp file is essentially empty. Its presence indicates to HANA that the package contains an application. The .xsaccess file contains data that controls access to the application. The key elements for our purposes are the exposed and authentication properties. Exposed determines whether the application is externally accessible and the authentication method determines the types of authentication methods allowed. By default a logon form will be used to authenticate users.



Select the **index.html** fileand click run . All it does it display some text in the browser but we know that HANA accepts this an an application and has granted access to it.



### Index.html

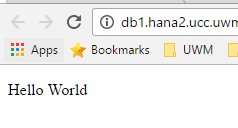
Next, we will build the index.html file which serves the purpose of bootstrapping the application. In a typical UI5 application, the index.html loads the UI5 libraries and then hands off the application to UI5.

First, 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>Hello World</title>  </head>  <body>  <p>Hello World</p>  </body>  </html> |

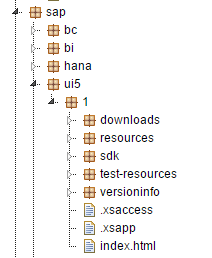


This code doesn’t contain anything that is related to SAPUI5. It’s just simple HTML. You can run the app by clicking the  icon and the result is shown below.



#### Bootstrap

The next step is to add the bootstrap script. This script loads the UI5 libraries which are installed on the HANA system in the sap/ui5/1 package. You can browse to them if you want to see them.



The main library is sap-ui-core.js. Once this is loaded then UI5 can manage all the other libraries that are used. Add the script section highlighted below (or replace all the code) to your index.html.

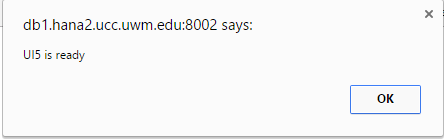
|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <meta charset="utf-8">  <title>Hello World</title>  **<script**  **id="sap-ui-bootstrap"**  **src="/sap/ui5/1/resources/sap-ui-core.js"**  **data-sap-ui-theme="sap\_bluecrystal"**  **data-sap-ui-libs="sap.m"**  **data-sap-ui-compatVersion="edge"**  **data-sap-ui-preload="async" >**  **</script>**  **<script>**  **sap.ui.getCore().attachInit(function () {**  **alert("UI5 is ready");**  **});**  **</script>**  </head>  <body>  <p>Hello World</p>  </body>  </html> |

Note, when you add the code you will get a warning in the border on the left side of the editor. The editor doesn’t like alerts. Don’t worry it will not interfere with the app running.

This code does a number of things:

* It loads sap-ui-core.js
* It loads a styling theme
* It loads the sap.m library which contains many of the main controls (buttons, lists, etc.) used by apps
* It sets a variable that enables the always troublesome Microsoft browsers to work
* It sets a variable that allows the app to load files asynchronously which makes the application appear more responsive to the user

In the second script section the application is initialized and, once the initialization is complete, it loads an alert message. If you run (or refresh the app if it is still open in a tab), you will see an alert pop up.

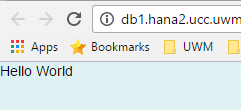


#### Controls

Now that we have UI5 loaded and initialized, we can load some UI5 controls. Update the code with the highlighted portions shown below.

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <meta charset="utf-8">  <title>Hello World!</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" >  </script>  <script>  sap.ui.getCore().attachInit(function () {  **new sap.m.Text({**  **text : "Hello World"**  **}).placeAt("content");**  });  </script>  </head>  **<body class="sapUiBody" id="content">**  </body>  </html> |

Now, rather than showing an alert, the code creates a Text control, sets the text property to “Hello World” and inserts it into the HTML element that has an id of content. The id of content is added to the HTML body and the sapUiBody class is added to the HTML BODY tag for styling purposes. The result is:



We haven’t done much yet but we have laid the groundwork for creating a UI5 application. The SAPUI5 bootstrap libraries with some basic configuration. We could continue to build the app in this format but SAPUI5 is built on a architecture called Model-View-Controller and before we go any further, we will arrange the application according to that architecture.

## Model – View – Controller

Modern web applications are structured to make them easier to create and maintain. For a small application like this, it’s hard to see the benefit but applications don’t have to be very large before they become hard to manage. UI5 follows a structure called Model-View-Control or MVC for short. The model-view-controller or MVC convention divides the application components according to their function:

**Model** – The model accesses and manages the data. In SAPUI5 the data in models can be bound to controls in views so that the flow of data between the model and the interface is handled automatically by SAPUI5. You can define models based on XML, JSON or oData data sources in SAPUI5.

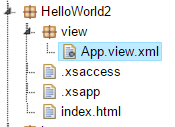
**View** – The view is used to create the user interface. SAPUI5 allows you to define views in JavaScript, HTML, XML or JSON. The recommended method is to use XML because it reinforces the separation of the application logic in the controller and the user interface in the view. Also, the WYSIWIG editor under development works with XML views. Fiori apps use XML views.

**Controller** – The controller is used to define the application logic. In SAPUI5, the controllers are defined in JavaScript and are tied to specific views.

#### Views

Create a package called view inside your webapp package and cut-and-paste the index.html file inside the webapp package..

Create a file called App.view.xml inside the view package.



|  |
| --- |
| Note the small dot next to the Component.js file in the image below. This indicates the file has not been *activated.* When you save files in the WDW, the design time file is saved and then HANA attempts to activate it in the repository. If a file is not activated, it is not available when you run the application. At times you save a file and it is not activated and you will see the small dot as in the image. If this occurs you can attempt to activate it by right-clicking the file and selecting Activate. |

Paste the code shown below into that file.

|  |
| --- |
| <mvc:View  xmlns="sap.m"  xmlns:mvc="sap.ui.core.mvc">  <Text text="Hello World"/>  </mvc:View> |

The code is in XML format. You can also create views in JavaScript, JSON and HTML format, however, XML has become the standard method.

XML views are encapsulated in the <mvc:View></mvc:View> element. The opening View tag also references two namespaces (or UI5 libraries) that are required for this view: sap.m and sap.ui.core.mvc.

The reference **xmlns:mvc = “sap.ui.core.mvc”** defines mvc as a shortcut to the sap.ui.core.mvc namespace. Notice that View in the opening and closing tags is prefixed with mvc:. This indicates that SAPUI5 will find the definition of the View element in the sap.ui.core.mvc namespace. The other elements in this view are found in the sap.m namespace. The reference xmlns="sap.m" indicates that the sap.m namespace will be the default namespace for the controls in the view because there is nothing after xmlns.

Notice that the Text control does not have a prefix indicating that it can be found in the sap.m namespace.

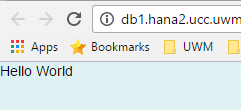
Update the code in index.html with highlighted portions of the code shown below.

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <meta charset="utf-8">  <title>Hello World!</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 () {  **sap.ui.xmlview({**  **viewName : "ui5.view.App"**  **}).placeAt("content");**  });  </script>  </head>  <body class="sapUiBody" id="content">  </body>  </html> |

We’ve added code to the bootstrap section that assigns a tag (ui5) to the root of our application (the webapp package). This will be used to allow UI5 to find our project files.

The code that runs once UI5 is initialized now loads the App view file rather than creating the Text control directly. Note the use of the ui5 tag in the code ui5.view.App. This means UI5 can find the App view file in the view package which is located in the webapp package.

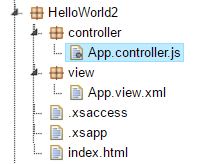
When you run the app, it looks the same but it has a more robust structure.



#### Controllers

Controllers provide the application logic and are located in a package called controller. Create the controller package in your webapp package.

Create a file called App.controller.js in the controller package.



The first thing we have to do is modify the App.view.xml file to indicate that it is associated with the App.controller.js controller and to add a Button control. Each view can be associated with one controller and the standard is that the view and controller have the same name and the names begin with uppercase letters. Update the App.view.xml code with the highlighted portions shown below.

|  |
| --- |
| <mvc:View  **controllerName="ui5.controller.App"**  xmlns="sap.m"  xmlns:mvc="sap.ui.core.mvc">  **<Button**  **text="Say Hello"**  **press="onShowHello"/>**  </mvc:View> |

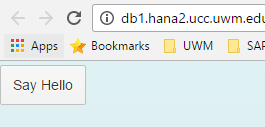
The controller file is identified at the top of the code. Note the use of the ui5 tag again.

The Button control has a text property used to configure the text displayed on the button. The press property identifies a function that will be executed when the button is pressed or clicked. This function becomes the press event’s event handler. This function is implemented in the controller file using JavaScript.

Insert the code shown below into the App.controller.js file.

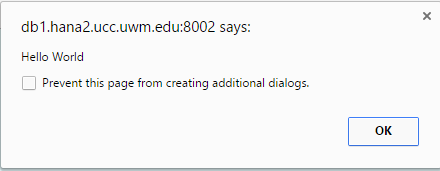
|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function (Controller) {  "use strict";  return Controller.extend("ui5.controller.App", {  });  }); |

The first two lines initialize this as a controller. Once the initialization process is complete the function (function(Controller)) is executed. In this function we will define our custom code. At this point, there is none. If you run the app now, you will see a button but, since we haven’t created the onShowHello function, nothing will happen if you press the button.



Modify the controller code as shown below. Now, when you click the button, an alert will appear.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function (Controller) {  "use strict";  return Controller.extend("ui5.controller.App", {  **onShowHello : function () {**  **// show a native JavaScript alert**  **alert("Hello World");**  **}**  });  }); |



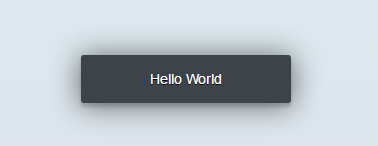
##### Modules

In UI5, libraries are called modules. In this section we’ll illustrate how to load a module by replacing the alert with a toast. Update the code in the controller file as 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 () {  MessageToast.show("Hello World");  }  });  }); |

First, the MessageToast module is loaded (sap/m/MessageToad) and then it is injected into the controller (function(Controller, MessageToast). Now we can access the module control in our onShowHello function.

Now, when you click the button you will see a toast message.



#### Models

Next, we’ll add a model. Update the controller code with the code shown below.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller",  "sap/m/MessageToast",  "sap/ui/model/json/JSONModel"  ], function (Controller, MessageToast, **JSONModel**) {  "use strict";  return Controller.extend("ui5.controller.App", {  **onInit : function () {**  **// set data model on view**  **var oData = {**  **recipient : {**  **name : "World"**  **}**  **};**  **var oModel = new JSONModel(oData);**  **this.getView().setModel(oModel);**  **},**  onShowHello : function () {  MessageToast.show("Hello World");  }  });  }); |

First, the module for creating models for data in JSON format is loaded and injected. Next, we add a new function called onInit. This function is one of the lifecycle functions. Lifecycle functions are invoked automatically at certain points in the view’s lifecycle. The onInit function is called when the controller is first initialized. It’s used to initialize resources like models that will be used in the controller.

The code in the onInit function creates a variable called oData and sets it equal to a JavaScript object. The next line initializes a JSON model using the oData object as it’s source of data then assigns the model to the App view.

Now that the model has been created and assigned to the view, we can access it in the view file. Update the view file using the code below.

|  |
| --- |
| <mvc:View  controllerName="ui5.controller.App"  xmlns="sap.m"  xmlns:mvc="sap.ui.core.mvc">  <Button  text="Say Hello"  press="onShowHello"/>  <Input  value="{/recipient/name}"  valueLiveUpdate="true"  description="Hello {/recipient/name}"  width="20%"/>  </mvc:View> |

An Input control has been added. The attributes of the control are bound to properties in the model. The curly brackets ({}) tell UI5 to bind, or link, the attribute to the indicated value in the model.

Now, when you run the application the value for the model’s name property is inserted in the value and description attributes of the Input control.



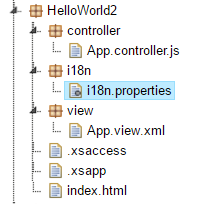
Note, that the binding is two-way by default so if you edit the value in the Input control, the description is updated and click the Button, the changes are reflected in the toast message.



## Internationalization (i18n)

Part of the process of adapting apps to different parts of the world is translating texts. The method used in UI5 is called i18n (the first letter of internationalization, the number of letters in the work internationalization and the last letter in internationalization). It relies on property files which contain the text used in the app’s interface. You would have a separate property file for each language you wished to support. UI5 will attempt to find a properties file that matches the language configured in the user’s browser.

The property files in UI5 are located in a package called i18n. Create the i18n package in the webapp package. Then create a file called i18n.properties in the new package.



Paste the code shown below into that file.

|  |
| --- |
| showHelloButtonText=Say Hello  helloMsg=Hello {0} |

This code creates labels (on the left side of the equals sign) that we can used to look up the text assigned to them. If we have multiple property files we use the same labels in all of them but change the text.

Now, update the controller file with the highlighted portions of the code shown below.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller",  "sap/m/MessageToast",  "sap/ui/model/json/JSONModel"**,**  **"sap/ui/model/resource/ResourceModel"**  ], function (Controller, MessageToast, JSONModel, **ResourceModel**) {  "use strict";  return Controller.extend("ui5.controller.App", {  onInit : function () {  // set data model on view  var oData = {  recipient : {  name : "World"  }  };  var oModel = new JSONModel(oData);  this.getView().setModel(oModel);  // set i18n model on view  **var i18nModel = new ResourceModel({**  **bundleName: "ui5.i18n.i18n"**  **});**  **this.getView().setModel(i18nModel, "i18n");**  },  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);**  }  });  }); |

This code loads and injects a module that is used to manage resource models. Property files are considered an application resource so they are managed with resource models. The new code in the onInit function creates a new resource model which references the i18n.properties file. It then set’s the model as a model used by the view and assigns it the name ‘i18n’.

The new code in the onShowHello function retrieves the resource model. It then retrieves the name property from the JSON model. Next, it creates a variable called sMsg and assigns a string which is constructed from the helloMsg label in the i18n.properties file and the name property from the JSON model. Finally, it displays the message using a toast.

Finally, 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">  <Button  text="{i18n>showHelloButtonText}"  press="onShowHello"/>  <Input  value="{/recipient/name}"  description="Hello {/recipient/name}"  valueLiveUpdate="true"  width="20%"/>  </mvc:View> |

This code binds the showHelloButtonText from the i18n.properties file to the text property of the button. If you run the app, it looks the same.



### Provide a Translation

Create a new file in the i18n package called **i18n\_fr.properties**. Insert the code shown below (You can use another language if you want but make sure you use the correct language code. You can find the language codes here: <http://www.w3.org/International/O-charset-lang.html>.):

Insert the code shown below into i18n\_fr.properties.

|  |
| --- |
| showHelloButtonText=Dis Bonjour  helloMsg=Bonjour {0} |

If you a change the language settings of your browser to French, UI5 will choose the i18n\_fr.properties files for translations.



## Exercise

Notice in the image above, you still have an English word in the description. Update the app so that it looks like this:



You will have to add another label to the i18n files and update the code in the App.view.xml file.