Week 2 Unit 2

working with Expressions and Formatters

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

Table of Contents

[1 Expression Binding 2](#_Toc449108109)

[2 Custom Formatter 4](#_Toc449108110)

## Preview

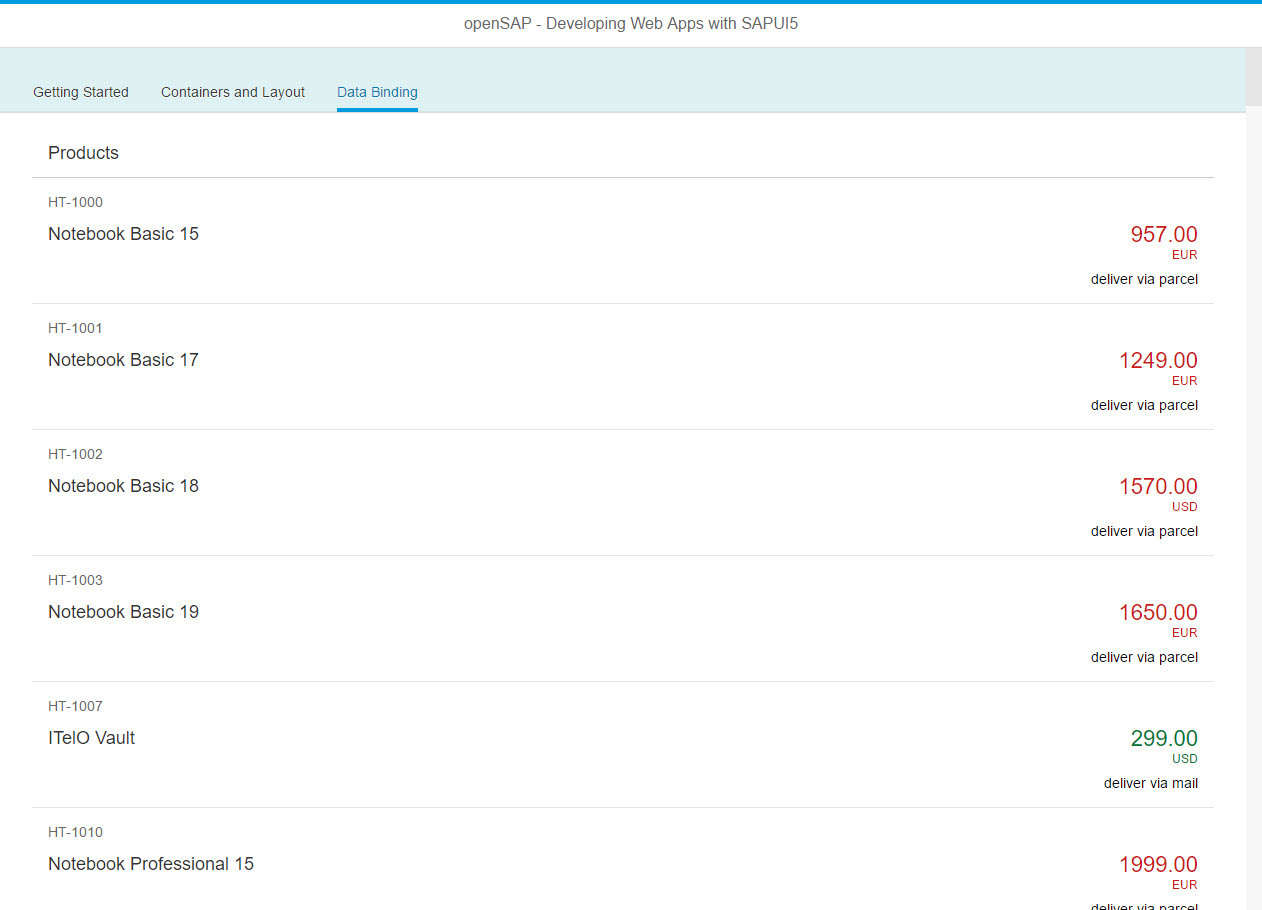


Figure 1 - Preview of the app after doing this unit’s exercises

# Expression Binding

In this step, we use expression binding to add a status to the list items based on the data from the backend service.

### Preview

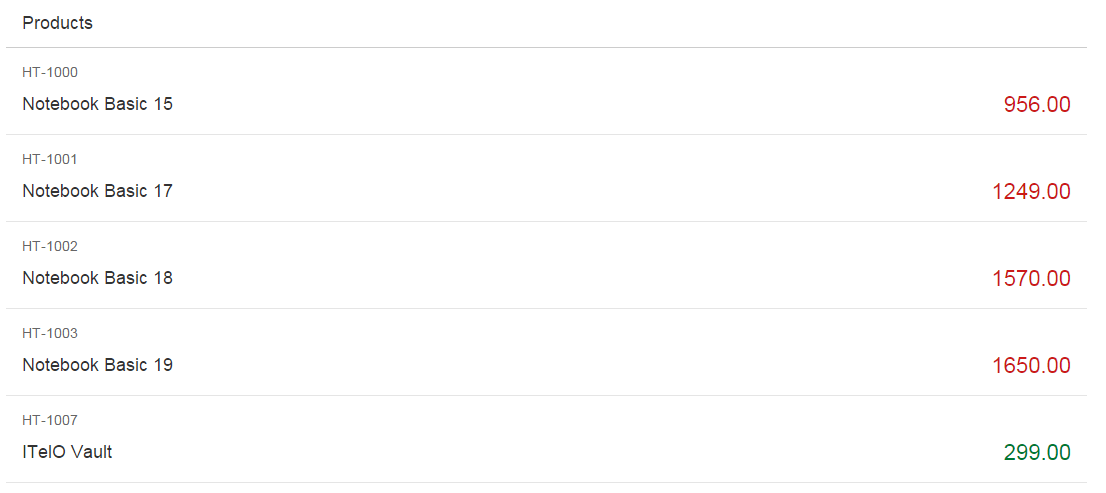


Figure 2: Number state set via expression

webapp/view/App.view.xml

<mvc:View

…

<IconTabFilter  
 text="{i18n>dataBindingFilter}" key="db">  
 <content>  
 <List  
 headerText="{i18n>productListTitle}"  
 items="{/ProductSet}">  
 <items>  
 <ObjectListItem  
 title="{Name}"  
 number="{Price}"  
 **numberState="{= ${Price} > 500 ? 'Error' : 'Success'}"** intro="{ProductID}"**>  
 </ObjectListItem>**  
 </items>  
 </List>  
 </content>  
</IconTabFilter>

…

</mvc:View>

We add the property numberState in our declarative view and introduce a new binding syntax that starts with = inside the brackets. This symbol is used to initiate a new binding syntax, it's called an expression and can do simple calculation logic like the ternary operator shown here.

The condition of the operator is a value from our data model. A model binding inside an expression binding has to be escaped with the $ sign as you can see in the code. We set the state to 'Error' (the number will appear in red) if the price is higher than 500 and to 'Success' (the number will appear in green) otherwise.

Expressions are limited to a particular set of operations that help formatting the data such as Math expression, comparisons, and such. You can look up the possible operations in the documentation.

### Conventions

* Only use expression binding for trivial calculations.

### Related Information

[Expression Binding](https://sapui5.hana.ondemand.com/docs/guide/daf6852a04b44d118963968a1239d2c0.html)

# Custom Formatter

In this step, we add a custom formatter to our list item that can perform more complex conversion and formatting of data.

### Preview

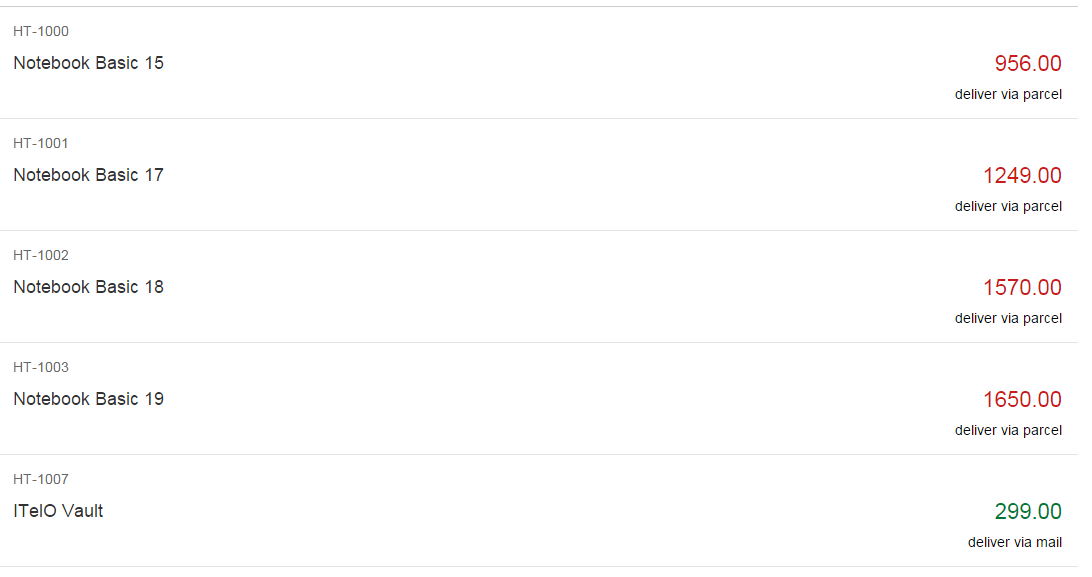


Figure 3: Status added using a custom formatter

**webapp/model/formatter.js (NEW)**

**sap.ui.define([], function() {**

**"use strict";**

**return {**

**delivery: function(sMeasure, iWeight) {**

**var oResourceBundle = this.getView().getModel("i18n").getResourceBundle(),**

**sResult = "";**

**if(sMeasure === "G") {**

**iWeight = iWeight / 1000;**

**}**

**if (iWeight < 0.5) {**

**sResult = oResourceBundle.getText("formatterMailDelivery");**

**} else if (iWeight < 5) {**

**sResult = oResourceBundle.getText("formatterParcelDelivery");**

**} else {**

**sResult = oResourceBundle.getText("formatterCarrierDelivery");**

**}**

**return sResult;**

**}**

**};**

**});**

We create a new file called formatter.js into the model folder and use the already introduced sap.ui.define syntax. This file returns an object with the delivery formatter function that can take several input parameters. We can pass them over from our view later. The formatter function we define takes two parameters and returns a text value from the i18n bundle based on the input parameters.

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

sap.ui.define([

"sap/ui/core/mvc/Controller",

"sap/m/MessageToast"**,**

**"opensap/myapp/model/formatter"**

], function (Controller, MessageToast**, formatter**) {

"use strict";

return Controller.extend("opensap.myapp.controller.App", {

**formatter : formatter,**

onShowHello : function () {

…

}

});

});

To load our formatter functions, we have to add it to the app controller. This controller simply stores the loaded formatter functions in a local property formatter so that we can access them in the view. The formatter reference is loaded as an additional dependency in the sap.ui.define statement.

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

<mvc:View …>

<App>

<pages>

<Page …>

…

<items>

<ObjectListItem …>

**<firstStatus>**

**<ObjectStatus text="{**

**parts: [**

**{path: 'WeightUnit'},**

**{path: 'WeightMeasure'}**

**],**

**formatter : '.formatter.delivery'**

**}"/>**

**</firstStatus>**

</ObjectListItem>

</items>

…

</Page>

</pages>

</App>

</mvc:View>

We add the firstStatus aggregation to our ListItem and use the ObjectStatus control to display the information we provide with our formatter function. For this we use the extended binding syntax with multiple parts. In parts we can pass in the paths to multiple entries in our model to a formatter function. In our case we use the WeightUnit and the WeightMeasure fields from the model as expected by the formatter function. A "."in front of the formatter name means that the function is looked up in the controller of the current view. There we defined a property formatter that holds our formatter functions, so we can access it by .formatter.delivery.

### webapp/i18n/i18n.properties

# App Descriptor

…

# Data Binding Content

productListTitle=Products

**formatterMailDelivery=deliver via mail**

**formatterParcelDelivery=deliver via parcel**

**formatterCarrierDelivery=deliver via carrier**

In a last step we then have to simply add the translatable texts we use in the formatter function to the internationalization file of the application.

### Related Information

[Defining a Formatter](https://sapui5.hana.ondemand.com/docs/guide/52a5ed0a1b304f788aa6dde684693d62.html)