Week 3 Unit 6

Implementing Dialogs

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

Table of Contents

[1 Preparation: Metadata and Control 2](#_Toc449113487)

[2 Create the Dialog 3](#_Toc449113488)

[3 Call the Dialog 4](#_Toc449113489)

[4 Add the Necessary Texts to the Properties File. 5](#_Toc449113490)

[5 Create a Fragment 6](#_Toc449113491)

[6 Adapt the Worklist View and Controller 7](#_Toc449113492)

[7 Adapt the Object View and Controller 8](#_Toc449113493)

## Preview

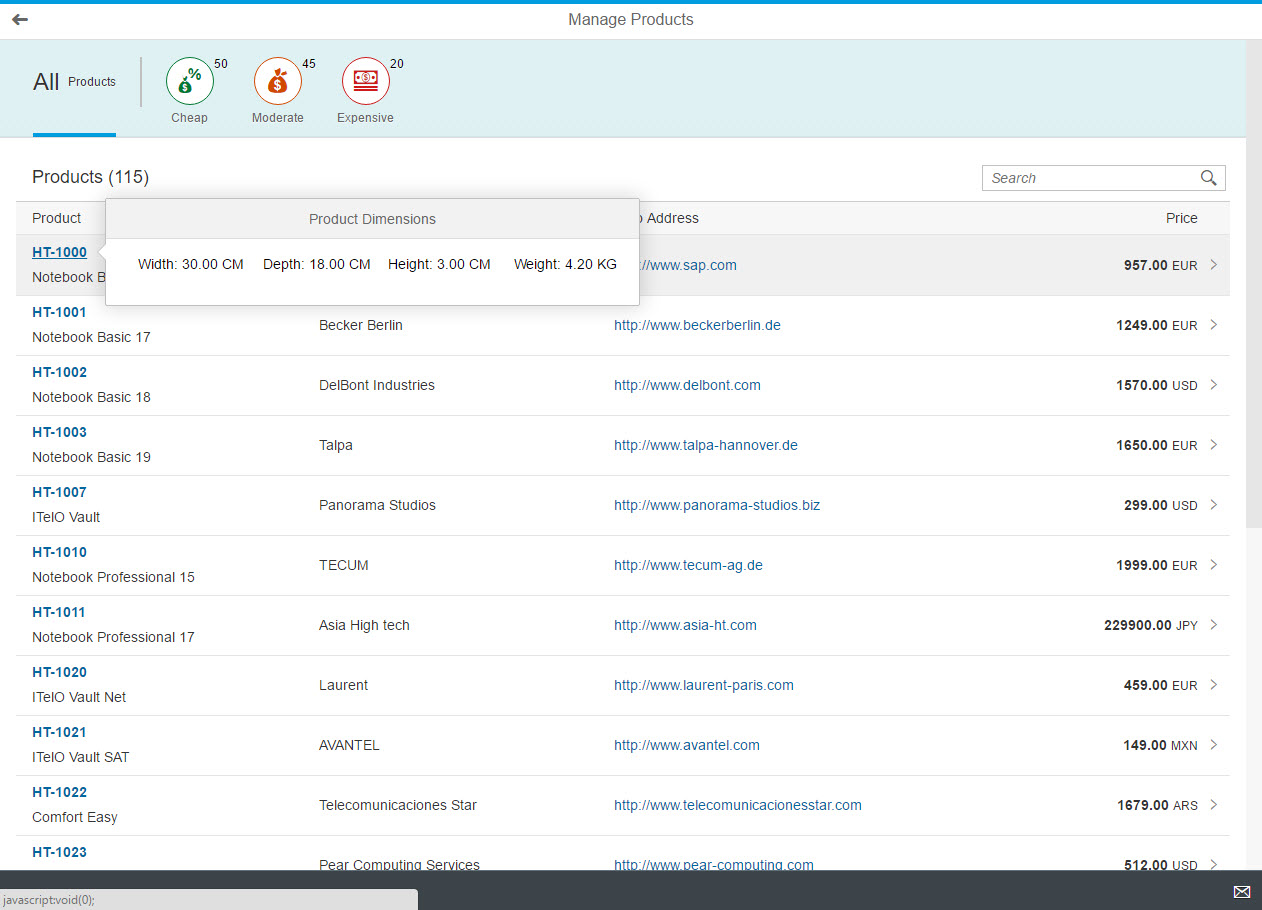
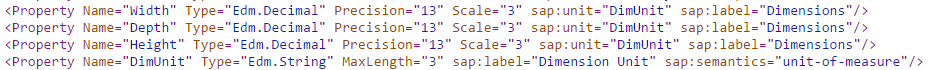


Figure 1 - Preview of the app after doing this unit’s exercises

# Preparation: Metadata and Control

In this exercise, we want to display some additional product data within a dialog. We will be using the dimensions we can find at each product in the service metadata:



Inspect the service metadata by looking at the Product entity: <https://sapes4.sapdevcenter.com/sap/opu/odata/IWBEP/GWSAMPLE_BASIC/$metadata>

As we want our app to be as responsive as possible, we will use a sap.m.ResponsivePopover as dialog. (See [Responsive Popover Explored Sample](https://sapui5.hana.ondemand.com/explored.html#/entity/sap.m.ResponsivePopover/samples))

# Create the Dialog

We will start with a dialog created from within a single view. Dialogs can be added to views similarly to other controls. However, as they do not get part of the view’s own control tree, we have to add them to a particular aggregation called dependents. In this case, we are adding the dialog to the dependents of the semantic page control. As we will be using a sap.ui.layout.Grid within the popup, we also need to add the namespace for at the top of the view:

## webapp/view/Worklist.view.xml

<mvc:View

controllerName="opensap.manageproducts.controller.WorklistWithDependent"

xmlns="sap.m"

xmlns:core="sap.ui.core"

xmlns:mvc="sap.ui.core.mvc"

xmlns:semantic="sap.m.semantic"

**xmlns:l="sap.ui.layout"**>

<semantic:FullscreenPage

id="page"

navButtonPress="onNavBack"

showNavButton="true"

title="{i18n>worklistViewTitle}">

…

</semantic:content>

**<semantic:dependents>**

**<ResponsivePopover title="{i18n>dimensionsTitle}"**

**class="sapUiPopupWithPadding"**

**id="dimensionsPopover">**

**<content>**

**<l:Grid>**

**<l:content>**

**<Text text="{i18n>productWidth}: {**

**path: 'Width',**

**formatter: '.formatter.numberUnit'**

**} {DimUnit}" />**

**<Text text="{i18n>productDepth}: {**

**path: 'Depth',**

**formatter: '. formatter.numberUnit'**

**} {DimUnit}" />**

**<Text text="{i18n>productHeight}: {**

**path: 'Height',**

**formatter: '. formatter.numberUnit'**

**} {DimUnit}" />**

**<Text text="{i18n>productWeight}: {**

**path: 'WeightMeasure',**

**formatter: '. formatter.numberUnit'**

**} {WeightUnit}" />**

**</l:content>**

**</l:Grid>**

**</content>**

**</ResponsivePopover>**

**</semantic:dependents>**

</semantic:FullscreenPage>

</mvc:View>

In this dialog, we use new texts from the resource model. We will add the actual texts to the resource model later. The actual values and the number unit come from the model and are displayed in a sap.m.Text control each. We will also reuse the formatter function numberUnit from one of the previous units. Remember that the formatter is available in the controller at the formatter object.

Note that the binding paths for the Text controls showing the dimensions are relative, so they will resolve relatively to the currently picked product later on.

# Call the Dialog

We also need to adapt the List in the view in order to open our Popover from somewhere. For this, we will set the ObjectIdentifier title into an active state, so it can be clicked, and assign an event handler which shall open the Popover for us:

## webapp/view/Worklist.view.xml

<items>

<ColumnListItem

type="Navigation"

press="onPress">

<cells>

<ObjectIdentifier

title="{ProductID}"

text="{Name}"

**titleActive="true"**

**titlePress="onShowDetailPopover"/**>

<Text text="{SupplierName}"/>

<Link

text="{ToSupplier/WebAddress}"

href="{ToSupplier/WebAddress}"

target="\_blank"/>

<ObjectNumber

number="{

path: 'Price',

formatter: '.formatter.numberUnit'

}"

unit="{CurrencyCode}"/>

</cells>

</ColumnListItem>

</items>

Now, we need to implement this event handler. Let us go to the Worklist controller and add the corresponding function, right above the section where the private methods begin:

## webapp/controller/Worklist.controller.js

**…**

**onRefresh : function () {**

**this.\_oTable.getBinding("items").refresh();**

**},**

**/\*\***

**\* Event handler for press event on object identifier.**

**\* opens detail popover to show product dimensions.**

**\* @public**

**\*/**

**onShowDetailPopover : function (oEvent) {**

**var oPopover = this.byId("dimensionsPopover");**

**var oSource = oEvent.getSource();**

**oPopover.bindElement(oSource.getBindingContext().getPath());**

**// open dialog**

**oPopover.openBy(oEvent.getParameter("domRef"));**

**},**

**/\* =========================================================== \*/**

**/\* internal methods \*/**

**/\* =========================================================== \*/**

**…**

Here, we retrieve the Popover by its id from the view. Next, we ask the event object oEvent, which automatically gets passed into our event handler, for the event source. The source is the particular control which triggered the event when clicked.

We get the instance of the ObjectIdentifier back, and can ask for its binding context to get information about which product was picked. We can then ask for the path at this binding context, and use it to bind the Popover to the correct entity in the model. We use an element binding at the Popover here, so the binding paths in the Popover will be resolved relatively to the correct product.

To open the Popover here, we call its openBy method and pass the event parameter domRef, so that the Popover is positioned very close to the right border of the actual link that was clicked. If we passed only the ObjectIdentifier instance directly, the Popover would have been opened at the right border of the whole ObjectIdentifier, which is quite far away from the link.

# Add the Necessary Texts to the Properties File.

All that is left to do now is to create the new texts in the properties files of the application.

Add the texts below:

## webapp/i18n/i18n.properties

**#~~~ Worklist Popover Fragment ~~~~~~~~~~~~**

**#XTIT: The title of the popover for product dimensions**

**dimensionsTitle=Product Dimensions**

**#XTIT: The label for the product width dimension**

**productWidth=Width**

**#XTIT: The label for the product height dimension**

**productHeight=Height**

**#XTIT: The label for the product depth dimension**

**productDepth=Depth**

**#XTIT: The label for the product weight**

**productWeight=Weight**

When you now run the app, you should see the ResponsivePopover with the corresponding dimensions appear when you click on an object ID in the Worklist view.

# Create a Fragment

## webapp/view/ResponsivePopover.fragment.xml (NEW)

**<core:FragmentDefinition**

**xmlns="sap.m"**

**xmlns:l="sap.ui.layout"**

**xmlns:core="sap.ui.core">**

**<ResponsivePopover title="{i18n>dimensionsTitle}"**

**class="sapUiPopupWithPadding">**

**<content>**

**<l:Grid>**

**<l:content>**

**<Text text="{i18n>productWidth}: {**

**path: 'Width',**

**formatter: '.formatter.numberUnit'**

**} {DimUnit}" />**

**<Text text="{i18n>productDepth}: {**

**path: 'Depth',**

**formatter: '.formatter.numberUnit'**

**} {DimUnit}" />**

**<Text text="{i18n>productHeight}: {**

**path: 'Height',**

**formatter: '.formatter.numberUnit'**

**} {DimUnit}" />**

**<Text text="{i18n>productWeight}: {**

**path: 'WeightMeasure',**

**formatter: '.formatter.numberUnit'**

**} {WeightUnit}" />**

**</l:content>**

**</l:Grid>**

**</content>**

**</ResponsivePopover>**

**</core:FragmentDefinition>**

We now want to make our ResponsivePopover better reusable. To achieve this, we will create a fragment containing the code from the Popover.

We can simply **cut** and paste the ResponsivePopover code from the Worklist view into our new fragment. Just don’t forget to wrap it in the fragment definition and **do not forget to remove the code from the Worklist view!**

**webapp/view/Worklist.view.xml**

**…**

**</IconTabBar>**

**</semantic:content>**

**<semantic:sendEmailAction>**

**<semantic:SendEmailAction id="shareEmail" press="onShareEmailPress"/>**

**</semantic:sendEmailAction>**

**~~<semantic:dependents>~~**

**~~<ResponsivePopover title="{i18n>dimensionsTitle}"~~**

**~~class="sapUiPopupWithPadding"~~**

**~~id="dimensionsPopover">~~**

**~~<content>~~**

**~~<l:Grid>~~**

**~~<l:content>~~**

**~~<Text text="{i18n>productWidth}: {~~**

**~~path: 'Width',~~**

**~~formatter: '.formatter.numberUnit'~~**

**~~} {DimUnit}" />~~**

**~~<Text text="{i18n>productDepth}: {~~**

**~~path: 'Depth',~~**

**~~formatter: '.formatter.numberUnit'~~**

**~~} {DimUnit}" />~~**

**~~<Text text="{i18n>productHeight}: {~~**

**~~path: 'Height',~~**

**~~formatter: '.formatter.numberUnit'~~**

**~~} {DimUnit}" />~~**

**~~<Text text="{i18n>productWeight}: {~~**

**~~path: 'WeightMeasure',~~**

**~~formatter: '.formatter.numberUnit'~~**

**~~} {WeightUnit}" />~~**

**~~</l:content>~~**

**~~</l:Grid>~~**

**~~</content>~~**

**~~</ResponsivePopover>~~**

**~~</semantic:dependents>~~**

**</semantic:FullscreenPage>**

# Adapt the Worklist View and Controller

In the Worklist view, you should now have no code left that belongs to the ResponsivePopover. Don’t forget to also remove the unused namespace declaration for the sap.ui.layout library, as it is not required in this view anymore.

## webapp/controller/Worklist.controller.js

/\*\*

\* Event handler for press event on object identifier.

\* opens detail popup from component to show product dimensions.

\* @public

\*/

onShowDetailPopover : function (oEvent) {

var oPopover = this.\_getPopover();

var oSource = oEvent.getSource();

oPopover.bindElement(oSource.getBindingContext().getPath());

// open dialog

oPopover.openBy(oEvent.getParameter("domRef"));

},

/\* =========================================================== \*/

/\* internal methods \*/

/\* =========================================================== \*/

**\_getPopover : function () {**

**// create dialog lazily**

**if (!this.\_oPopover) {**

**// create popover via fragment factory**

**this.\_oPopover = sap.ui.xmlfragment(**

**"opensap.manageproducts.view.ResponsivePopover", this);**

**this.getView().addDependent(this.\_oPopover);**

**}**

**return this.\_oPopover;**

**},**

In the controller of the Worklist view, we now have to slightly adapt the function opening the popover.

We need to create an instance of the ResponsivePopover before we can bind and open it, as it is now not automatically instantiated when the view gets parsed. In order to only instantiate the popover when it is needed, and to avoid instantiating it multiple times, we create a new private function \_getPopover. This function will create an instance of the popover from the fragment and assign it to a private property of the controller, but only if this instance is not already there.

In our event handler, we now replace the line retrieving the popover by its id with the call to the \_getPopover method.

One more adaptation we need now is to add the ResponsivePopover instance to the view by using addDependent. This will allow the popover to get access to the model the view has inherited from the component, and it will also make it take part in the view’s lifecycle.

# Adapt the Object View and Controller

We can now reuse the ResponsivePopover easily in our Object view. Let us make the Object ID here clickable as well. We will adapt the ObjectHeader in this view a little:

## webapp/view/Object.view.xml

<semantic:content>

<ObjectHeader

id="objectHeader"

title="{ProductID}"

**titleActive="true"**

**titlePress="onShowDetailPopover"**

responsive="true"

number="{

path: 'Price',

formatter: '.formatter.numberUnit'

}"

numberUnit="{CurrencyCode}">

</ObjectHeader>

</semantic:content>

</semantic:FullscreenPage>

## webapp/controller/Object.controller.js

**…**

**onNavBack : function() {**

**var sPreviousHash = History.getInstance().getPreviousHash();**

**if (sPreviousHash !== undefined) {**

**history.go(-1);**

**} else {**

**this.getRouter().navTo("worklist", {}, true);**

**}**

**},**

**/\*\***

**\* Event handler for press event on object identifier.**

**\* opens detail popup from component to show product dimensions.**

**\* @public**

**\*/**

**onShowDetailPopover : function (oEvent) {**

**var oPopover = this.\_getPopover();**

**var oSource = oEvent.getSource();**

**oPopover.bindElement(oSource.getBindingContext().getPath());**

**// open dialog**

**oPopover.openBy(oEvent.getParameter("domRef"));**

**},**

**/\* =========================================================== \*/**

**/\* internal methods \*/**

**/\* =========================================================== \*/**

**\_getPopover : function () {**

**// create dialog lazily**

**if (!this.\_oPopover) {**

**// create popover via fragment factory**

**this.\_oPopover = sap.ui.xmlfragment(**

**"opensap.manageproducts.view.ResponsivePopover", this);**

**this.getView().addDependent(this.\_oPopover);**

**}**

**return this.\_oPopover;**

**},**

Last, we simply copy over the two methods \_getPopover and onShowDetailPopover from the Worklist.controller.js to the Object.controller.js.

When you now re-run the app and switch to the Object view for one product, you should be able to click on the product Id to make the Popover also appear in this view.

**Coding Samples**

Any software coding or code lines/strings (“Code”) provided in this documentation are only examples and are not intended for use in a productive system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules for certain SAP coding. SAP does not warrant the correctness or completeness of the Code provided herein and SAP shall not be liable for errors or damages cause by use of the Code, except where such damages were caused by SAP with intent or with gross negligence.