

DISCOVER OPENUI5 – THE NEW RESPONSIVE WEB UI LIBRARY FROM SAP

OSCON 2014

Exercises / Solutions

Frederic Berg, Andreas Kunz / SAP AG

DJ Adams / Bluefin Solutions



O'REILLY

OScon
OPEN SOURCE CONVENTION



PORTLAND, OR
JULY 21-24, 2014

#oscon

TABLE OF CONTENTS

Exercise 0 – Getting Started	3
Exercise 1 – Resource Model	7
Exercise 2 – Object Controls	9
Exercise 3 – Formatter.....	12
Exercise 4 – Search	16
Exercise 5 – Split App & Shell	18
Exercise 6 – Additional Device Adaptation	21
Exercise 7 – Supplier Tab.....	24
Exercise 8 – Approval Process	26
Exercise 9 – Line Item	29
Exercise 10 – Grouping	33

Exercise 0 – Getting Started

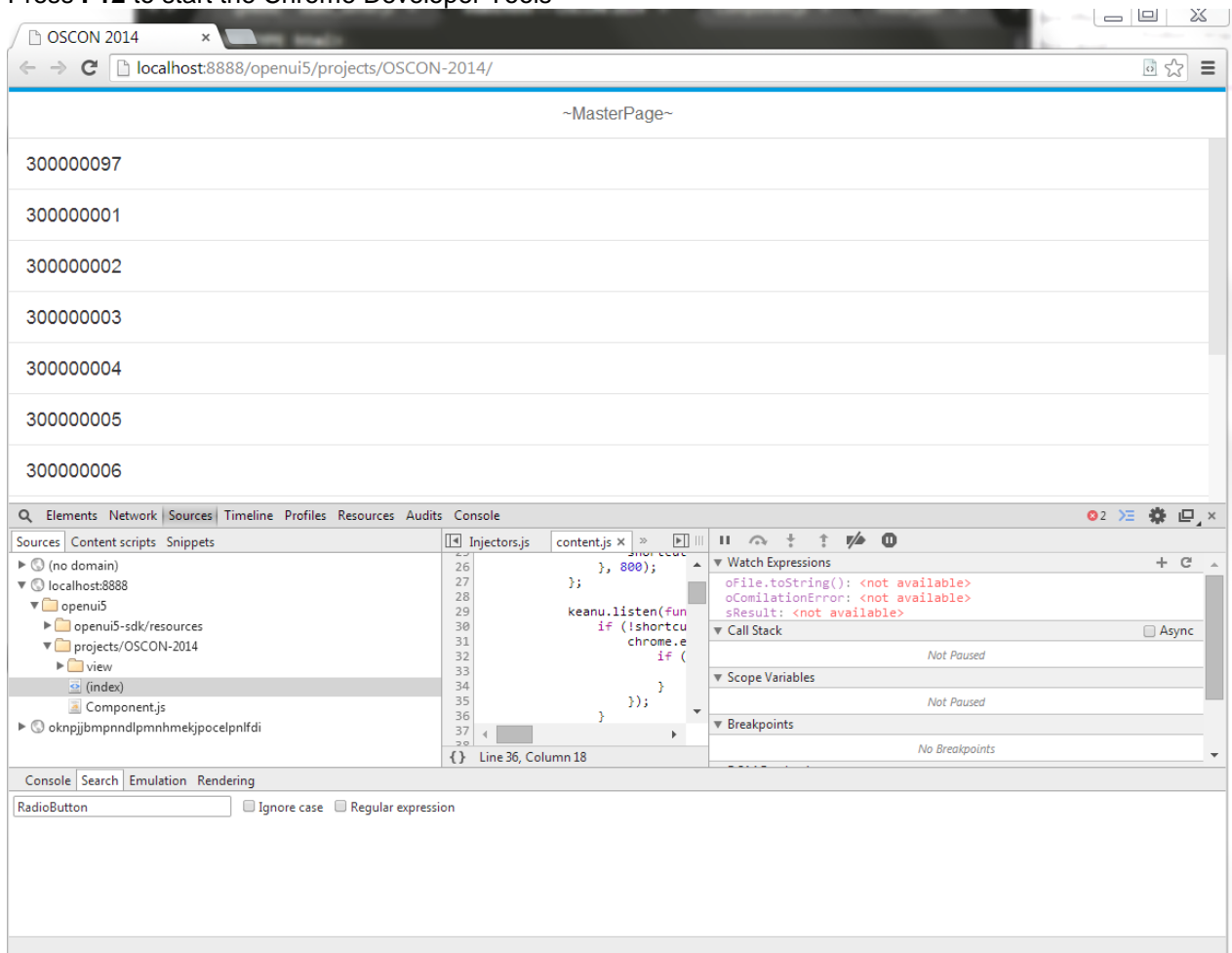
Objective

Set up the development environment:

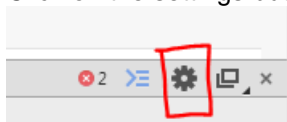
1. Follow the instructions in setup.md to set up your basic development environment.
2. Point your (Chrome) browser to the initial application by opening the following URL:

<http://localhost:8888/openui5/projects/OSCON-2014/>

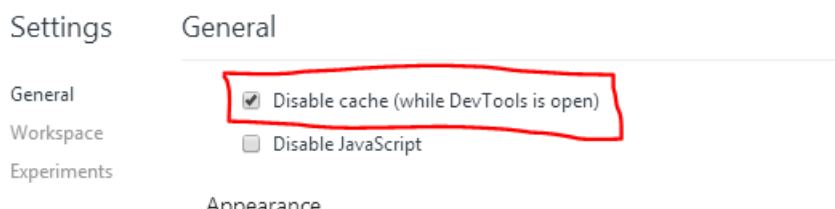
3. Press **F12** to start the Chrome Developer Tools



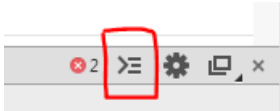
4. Click on the settings button.



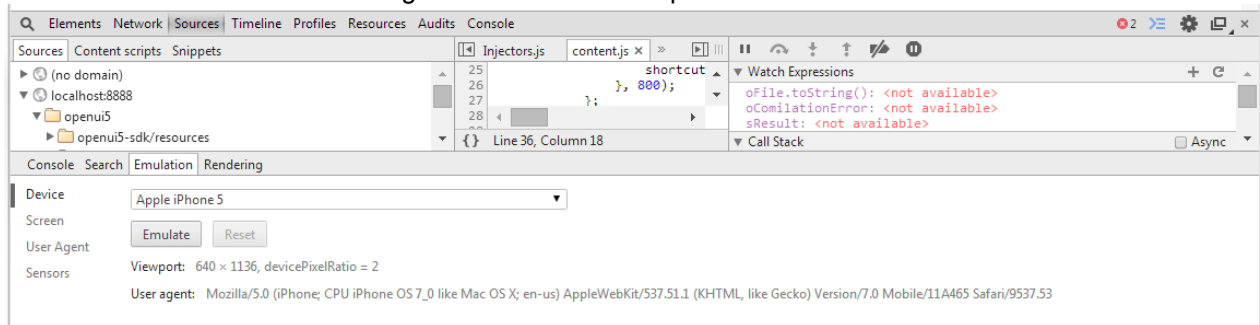
5. Under “Settings -> General” chose the option “Disable cache (while the DevTools is open)”



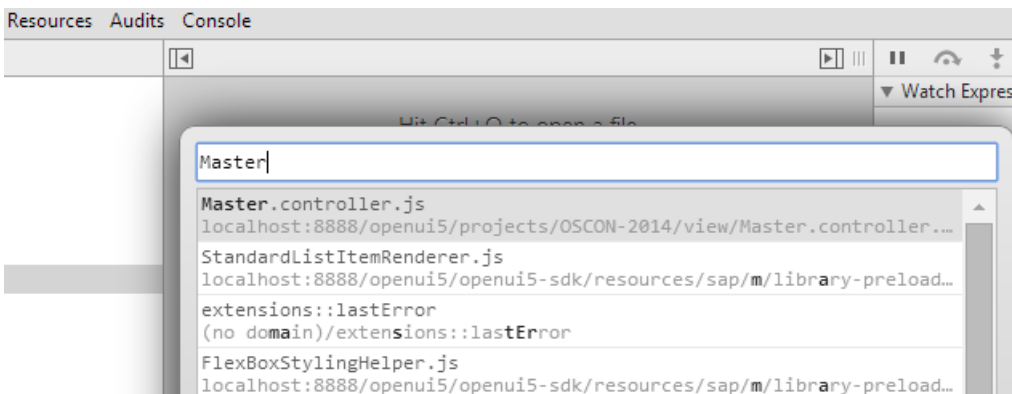
- Make sure the Chrome drawer is shown:



- Under **Emulation**: Set the User Agent to “iPhone 5” and press the “Emulate” button. Close the drawer.



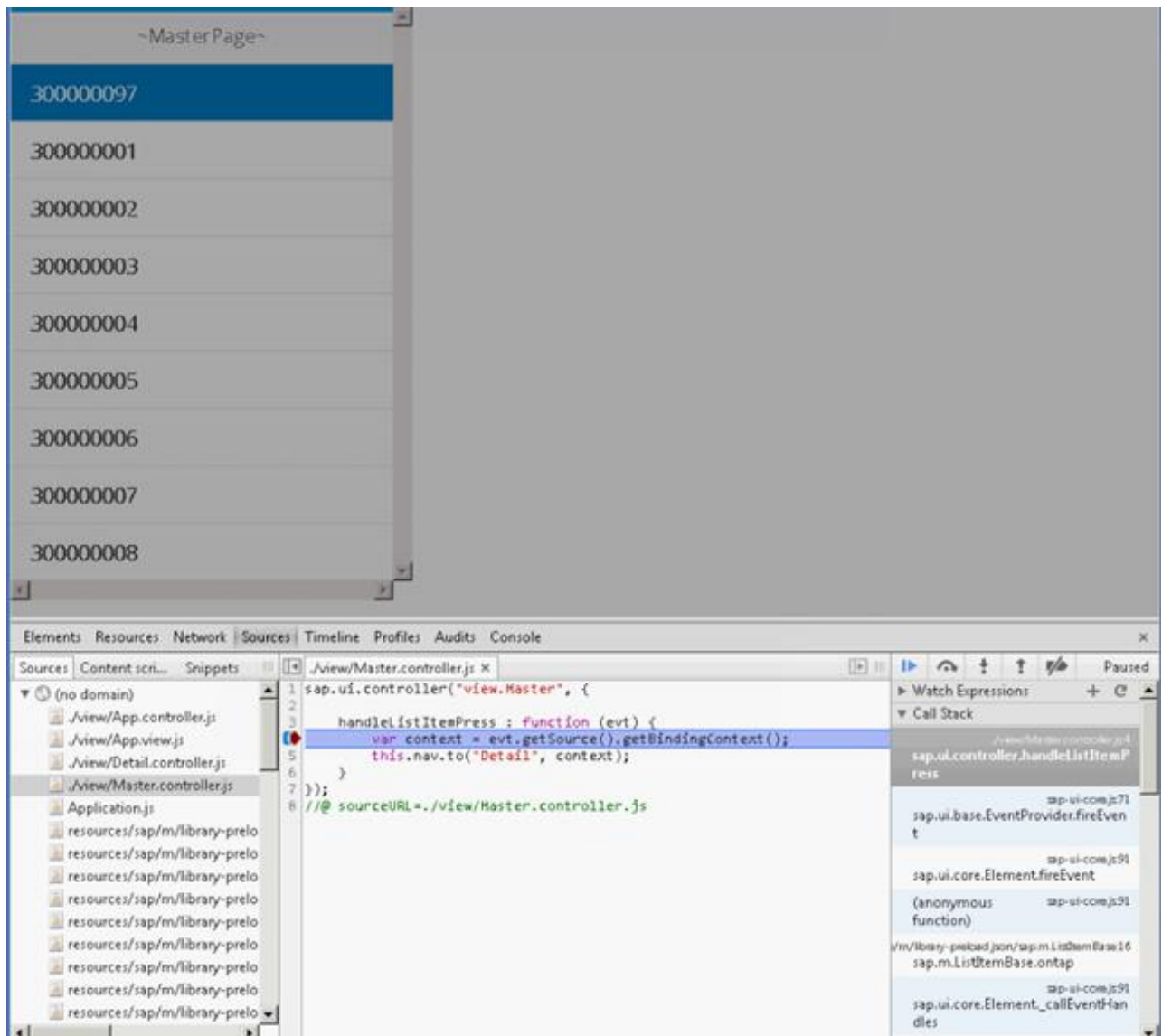
- Open the tab “Sources”. Press **CTRL-O** and enter “Master”. Now the file “Master.controller.js” is selected and you press the Enter key.



- Set a breakpoint in line 4 by clicking on the line number until it is highlighted in blue. Notice the listing of the breakpoint in the right panel.



10. Now click on a line item in the running application. This causes the application to stop at the breakpoint.



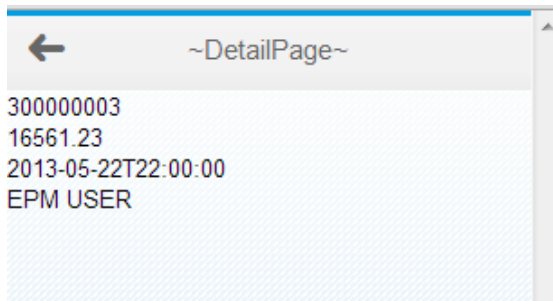
11. Collapse the panel "Call Stack" and open "Scope Variables". Investigate the event parameter **evt** in the right panel. With this you can understand the current state at runtime.



12. Click on the "Play" button (blue) to resume the application execution.



13. The application now displays the **DetailPage**



Exercise 1 – Resource Model

Objective

Set proper titles to master and detail pages by implementing a resource model (aka i18n model, *i18n* stands for *internationalization*).

Preview

Before:

~MasterPage~
300000097
300000001
300000002
300000003
300000004
300000005
300000006
300000007
300000008

After:

Sales Orders
300000097
300000001
300000002
300000003
300000004
300000005
300000006
300000007
300000008

Description

What we're going to do in this exercise is to replace the hardcoded texts in the views with references to texts in a separate properties file. This is done via a resource model, which is used as a wrapper for resource bundles and has a one-time binding mode. Once the texts are abstracted into separate files, they can then be maintained, and translated, independently.

So we'll modify the Master and Detail views, and create the properties file with the text contents.

Changes

i18n/messageBundle.properties (ADD NEW FOLDER i18n > ADD NEW FILE messageBundle.properties)

- Create a new folder named **"i18n"** next to index.html and the "view" and "model" folders
- Add new file **messageBundle.properties** inside this folder "i18n" and put the below content there
- Make sure the file does NOT start with an empty line
- Save the new message bundle file

```
MasterTitle=Sales Orders
DetailTitle=Sales Order
```

Component.js

- The message bundle is loaded with the help of a *ResourceModel*
- The *ResourceModel* is made available as global model under the name **"i18n"**

```
createContent : function() {
    // create root view
    var oView = sap.ui.view({
        id : "app",
        viewName : "oscon2014.view.App",
        type : "JS",
        viewData : { component : this }
    });
}
```

```
});

// set i18n model
var i18nModel = new sap.ui.model.resource.ResourceModel({
    bundleUrl : "i18n/messageBundle.properties"
});
oView.setModel(i18nModel, "i18n");

// set data model on root view
var oModel = new sap.ui.model.json.JSONModel("model/mock.json");
oView.setModel(oModel);

// done
return oView;
}
```

view/Master.view.xml

- Switch the title to point to the “**i18n**” model and there to the text “**MasterTitle**”
- Save the modified **Master.view.xml** file

```
<core:View
    controllerName="oscon2014.view.Master"
    xmlns="sap.m"
    xmlns:core="sap.ui.core" >
    <Page
        title="{i18n>MasterTitle}" >
    ...
```

view/Detail.view.xml

- Also adjust the title of the detail view
- Save the modified **Detail.view.xml** file with shortcut **CTRL+S**

```
<core:View
    controllerName="oscon2014.view.Detail"
    xmlns="sap.m"
    xmlns:core="sap.ui.core" >
    <Page
        title="{i18n>DetailTitle}"
        showNavButton="true"
        navButtonPress="handleNavButtonPress" >
    ...
```

Now reload the page in your browser and check the results.

You may still leave the phone emulation mode active, we will tell you once things get interesting when run in normal desktop browser mode.

If you still see the old version of the app now or after any of the subsequent exercises (e.g. because you closed the browser's developer tools and the cache is active), try doing Shift-Reload or clearing the browser cache.

Further Reading:

- ModelViewController: <https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/MVC.1.html>
- Component Concept: <https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/Components.html>
- Databinding: <https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/DataBinding.html>
- Localization: <https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/i18NinAppDev.html>
- ResourceModel: <https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/ResourceModel.html>

Exercise 2 – Object Controls

Objective

Make the UI of the master list and the detail page more beautiful by using the SAPUI5 controls ***sap.m.ObjectListItem*** and ***sap.m.ObjectHeader***.

Preview

Before:

Sales Orders
300000097
300000001
300000002
300000003
300000004
300000005
300000006
300000007
300000008

After:

Sales Orders	
300000097	13224.47 EUR
11113.00	P
300000001	12493.73 EUR
10498.94	N
300000002	11666.69 EUR
9803.94	N
300000003	16561.23 EUR
13917.00	P

← Sales Order
300000097
13224.47
2013-05-22T22:00:00
EPM USER

← Sales Order	
300000097	13224.47
	EUR
11113.00	P
EPM USER	
2013-05-22T22:00:00	

Before:

After:

Description

In this exercise we will replace a couple of controls; one in the Master view and the other in the Detail view.

In the Master view, rather than the simple flat list item style presented by the *StandardListItem* control that is in use currently, we'll present the overview of the sales orders in a more readable and useful way by using the *ObjectListItem* control instead.

In the Detail view, we'll make a similar change, replacing the simple layout (currently afforded by the *VBox* control) with a more readable display thanks to the *ObjectHeader* control.

Along the way we'll add a few more properties from the data model, such as *CurrencyCode*.

Changes

view/Master.view.xml

- Replace the **StandardListItem** control with the more powerful **ObjectListItem**
- Attributes and statuses are defined by own objects
- Save the modified **Master.view.xml** file with shortcut **CTRL+S**

```
<core:View
    controllerName="oscon2014.view.Master"
    xmlns="sap.m"
    xmlns:core="sap.ui.core" >
    <Page
        title="{i18n>MasterTitle}" >
        <List
            items="{/SalesOrderCollection}" >
            <ObjectListItem
                type="Active"
                press="handleListItemPress"
                title="{SoId}"
                number="{GrossAmount}"
                numberUnit="{CurrencyCode}" >
                <attributes>
                    <ObjectAttribute text="{BuyerName}" />
                </attributes>
                <firstStatus>
                    <ObjectStatus text="{LifecycleStatus}" />
                </firstStatus>
            </ObjectListItem>
        </List>
    </Page>
</core:View>
```

view/Detail.view.xml

- Replace the VBox holding the texts with the more beautiful **ObjectHeader** control (which has almost the same API as the **ObjectListItem** control but utilizes the space in a different way).
- Save the modified **Detail.view.xml** file with shortcut **CTRL+S**

```
<core:View
    controllerName="oscon2014.view.Detail"
    xmlns="sap.m"
    xmlns:core="sap.ui.core" >
    <Page
        title="Sales Order"
        showNavButton="true"
        navButtonPress="handleNavButtonPress" >
        <ObjectHeader
            title="{SoId}"
            number="{GrossAmount}"
            numberUnit="{CurrencyCode}" >
            <attributes>
                <ObjectAttribute text="{BuyerName}" />
                <ObjectAttribute text="{CreatedByBp}" />
                <ObjectAttribute text="{CreatedAt}" />
            </attributes>
            <firstStatus>
                <ObjectStatus text="{LifecycleStatus}" />
            </firstStatus>
        </ObjectHeader>
    </Page>
</core:View>
```

Further Reading:

- Working with lists: <https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/List.html>
- ObjectHeader API: <https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.ObjectHeader.html>
- ObjectListItem API: <https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.ObjectListItem.html>

Exercise 3 – Formatter

Objective

Format status color and date properly by implementing custom formatters that are used in data binding.

Preview

Before:

Sales Orders	
300000097	13224.47 EUR 11113.00 P
300000001	12493.73 EUR 10498.94 N
300000002	11666.69 EUR 9803.94 N
300000003	16561.23 EUR 13917.00 P

After:

Sales Orders	
300000097	13224.47 EUR 11113.00 In Process
300000001	12493.73 EUR 10498.94 New
300000002	11666.69 EUR 9803.94 New
300000003	16561.23 EUR 13917.00 In Process

Before:

Sales Order	
300000097	13224.47 EUR 11113.00 P EPM USER 2013-05-22T22:00:00

After:

Sales Order	
300000097	13224.47 EUR 11113.00 In Process EPM USER 2013-05-23

Description

In this exercise we will introduce a couple of formatting functions and use them in the application. They are custom functions so we put them in a module file *'Formatter.js'* in a separate folder (in this case we've chosen the folder name 'util'). One of the functions uses a static class of UI5 for date formatting so we specify that requirement (for *sap.ui.core.format.DateFormat*) before defining our functions.

We then use the formatting functions in the *Detail* and *Master* views; in order to do this, we need to *'require'* the new module in the respective controllers. To execute the formatting on the property paths from the data model (such as *'CreatedAt'* or *'LifecycleStatus'*) we need a different binding syntax and for that we have to add a *bindingSyntax* parameter in the SAPUI5 bootstrap.

Changes

i18n/messageBundle.properties

- Add two new texts to the properties file that are used to display the status

```
MasterTitle=Sales Orders
DetailTitle=Sales Order
StatusTextN=New
StatusTextP=In Process
```

util/Formatter.js (ADD NEW FOLDER **util** > ADD NEW FILE **Formatter.js**)

- Create a new folder named “**util**” next to “i18n”, “model” and “view”
- Add new file **Formatter.js** in your folder **util** and put the below content there
- This file contains functions to format dates, status text and status colors.

```
jQuery.sap.declare("oscon2014.util.Formatter");

jQuery.sap.require("sap.ui.core.format.DateFormat");

oscon2014.util.Formatter = {
    _statusStateMap : {
        "P" : "Success",
        "N" : "Warning"
    },

    statusText : function (value) {
        var bundle = this.getModel("i18n").getResourceBundle();
        return bundle.getText("StatusText" + value, "?");
    },

    statusState : function (value) {
        var map = oscon2014.util.Formatter._statusStateMap;
        return (value && map[value]) ? map[value] : "None";
    },

    date : function (value) {
        if (value) {
            var oDateFormat = sap.ui.core.format.DateFormat.getDateTimeInstance({pattern: "yyyy-MM-dd"});
            return oDateFormat.format(new Date(value));
        } else {
            return value;
        }
    }
};
```

index.html

- For the formatting we want to use the “**complex**” binding syntax of SAPUI5. This we enable in the bootstrap script tag.

```
<!DOCTYPE html>
<html>

    ...

    <script
        id="sap-ui-bootstrap"
        src="../../resources/sap-ui-core.js"
        data-sap-ui-theme="sap_bluecrystal"
        data-sap-ui-libs="sap.m"
        data-sap-ui-xx-bindingSyntax="complex"
        data-sap-ui-resourceroots='{
            "oscon2014": "/"
        }' >
```

```

</script>
...
</html>

```

view/Detail.view.xml

- Use a complex binding with a formatter for the **text** field of attribute 'CreatedAt'.
- Use a complex binding with a formatter for the **text** and **state** field (which controls the semantical color of the status text) of status 'LifecycleStatus'.

```

<core:View
    controllerName="oscon2014.view.Detail"
    xmlns="sap.m"
    xmlns:core="sap.ui.core" >
    <Page
        title="{i18n>DetailTitle}"
        showNavButton="true"
        navButtonPress="handleNavButtonPress" >
        <ObjectHeader
            title="{SoId}"
            number="{GrossAmount}"
            numberUnit="{CurrencyCode}" >
            <attributes>
                <ObjectAttribute text="{BuyerName}" />
                <ObjectAttribute text="{CreatedByBp}" />
                <ObjectAttribute text="{
                    path: 'CreatedAt',
                    formatter: 'oscon2014.util.Formatter.date'
                }" />
            </attributes>
            <firstStatus>
                <ObjectStatus
                    text="{
                        path: 'LifecycleStatus',
                        formatter: 'oscon2014.util.Formatter.statusText'
                    }"
                    state="{
                        path: 'LifecycleStatus',
                        formatter: 'oscon2014.util.Formatter.statusState'
                    }" />
            </firstStatus>
        </ObjectHeader>
    </Page>
</core:View>

```

view/Detail.controller.js

- Require the formatter file in the controller of the view

```

jQuery.sap.require("oscon2014.util.Formatter");

sap.ui.controller("oscon2014.view.Detail", {
    ...

```

view/Master.view.xml

- Use a complex binding with a formatter for the **text** and **state** field (which controls the semantical color of the status text) of status 'LifecycleStatus'.

```

<core:View
    controllerName="oscon2014.view.Master"
    xmlns="sap.m"
    xmlns:core="sap.ui.core" >
    <Page
        title="{i18n>MasterTitle}" >

```

```

<List
  items="{/SalesOrderCollection}" >
  <ObjectListItem
    type="Active"
    press="handleListItemPress"
    title="{SoId}"
    number="{GrossAmount}"
    numberUnit="{CurrencyCode}" >
    <attributes>
      <ObjectAttribute text="{BuyerName}" />
    </attributes>
    <firstStatus>
      <ObjectStatus
        text="{
          path: 'LifecycleStatus',
          formatter: 'oscon2014.util.Formatter.statusText'
        }"
        state="{
          path: 'LifecycleStatus',
          formatter: 'oscon2014.util.Formatter.statusState'
        }" />
      </firstStatus>
    </ObjectListItem>
  </List>
</Page>
</core:View>

```

view/Master.controller.js

- Require the formatter file in the controller of the view

```
jQuery.sap.require("oscon2014.util.Formatter");
```

```
sap.ui.controller("oscon2014.view.Master", {
```

```
...
```

Further Reading:

- Bootstrap Configuration Options:
<https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/Configuration.html#ListofConfigurationOptions>
- Property Binding and Formatting:
<https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/BindingProperties.html>
- Modularization and Dependency Management (require/declare modules):
<https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/ModularizationConcept.html>


Exercise 4 – Search

Objective

Implement a search on the master list by using ***sap.m.SearchField***

Preview

Sales Orders	
300000097	13224.47 EUR
11113.00	In Process
300000001	12493.73 EUR
10498.94	New
300000002	11666.69 EUR
9803.94	New
300000003	16561.23 EUR
13917.00	In Process

Sales Orders	
Search 	
300000097	13224.47 EUR
11113.00	In Process
300000001	12493.73 EUR
10498.94	New
300000002	11666.69 EUR
9803.94	New
300000003	16561.23 EUR

Before:

After:

Description

Now we're going to add a *SearchField* control to the initial page of the application. We'll add it as a child within the Page's *subHeader* aggregation which expects a *Bar* (*sap.m.Bar*) control.

To handle the search, we'll specify a handler for the search field's *search* event. This handler *handleSearch* is defined in the view's controller, and the search effect is achieved by adding a *contains string* filter to the binding of the List control's items aggregation.

Changes

view/Master.view.xml

- The search field is put to a bar that is placed in the sub header of the page.
- Set the search field to **100%** width to utilize all the space
- Do not forget to add an **"id"** to the list in order to access the list later on in the controller

```
<core:View
  controllerName="oscon2014.view.Master"
  xmlns="sap.m"
  xmlns:core="sap.ui.core" >
  <Page
    title="{i18n>MasterTitle}" >
    <subHeader>
      <Bar>
        <contentLeft>
          <SearchField
            search="handleSearch"
            width="100%" >
          </SearchField>
        </contentLeft>
      </Bar>
    </subHeader>
    <List
      id="list"
      items="{/SalesOrderCollection}" >
```


view/Master.controller.js

- Implement a new handler function on the view controller. Make sure to separate the function from the other handler function with a “,”
- Access the “**query**” as a parameter of the event object
- If the “**query**” is not empty add a **FilterOperator** to the array of filters.
- Access the list instance by calling “**byId**” on the view.
- Apply the filter array on the binding object of the list.

```
jQuery.sap.require("oscon2014.util.Formatter");

sap.ui.controller("oscon2014.view.Master", {

    handleListItemPress : function (evt) {
        var context = evt.getSource().getBindingContext();
        this.nav.to("Detail", context);
    },

    handleSearch : function (evt) {
        // create model filter
        var filters = [];
        var query = evt.getParameter("query");
        if (query && query.length > 0) {
            var filter = new sap.ui.model.Filter("SoId", sap.ui.model.FilterOperator.Contains,
query);
            filters.push(filter);
        }
        // update list binding
        var list = this.getView().byId("list");
        var binding = list.getBinding("items");
        binding.filter(filters);
    }

});
```

Google Chrome browser

Sales Orders	
9	⊗ 🔍
300000097	13224.47
	EUR
11113.00	In Process
300000009	3453.38
	EUR
2902.00	New

In case the search does not work, check whether you forgot adding the id for the List in Master.view.xml.

Further Reading:

- SearchField: <https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.SearchField.html>
- Model Filter: <https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.ui.model.Filter.html>

Exercise 5 – Split App & Shell

Objective

Utilize the additional space by using the **sap.m.SplitApp** control which shows the master and detail view next to each other. Wrap the split app in a shell that fills the remaining space on the desktop.

Preview

Before:

Sales Orders		
Search		
300000097		13224.47 EUR
11113.00		In Process
300000001		12493.73 EUR
10498.94		New

After

Sales Orders	Sales Order
Search	
300000097 13224.47 EUR	300000097 13224.47 EUR
11113.00 In Process	11113.00 EPM USER 2013-05-23
300000001 12493.73 EUR	
10498.94 New	

Description

So far we've had 3 views in our application – *App*, *Master* and *Detail*. *App* is our top-level view, containing the *Master* and *Detail* views. In the *App* view we used an *App* control (yes, the same name) to contain the *Master* and *Detail* views via the *App* control's 'pages' aggregation.

This is a typical scenario for an app designed primarily for a smartphone-sized screen. But if the screen size is larger (e.g. on a tablet or desktop) we want to automatically utilize the extra space and for that we will switch from the *App* control to the *SplitApp* control. Alongside swapping out the control, we'll add new view '*Empty*' which will be shown in the detail part of the *SplitApp* – straightaway, if there is enough space.

Finally, for optimal distribution of space on larger devices such as desktops, we will wrap the whole thing in a *Shell* control.

Changes

view/Empty.view.xml (create NEW XML view **view/Empty.view.xml**)

- This is only a very empty page

```
<core:View
  xmlns="sap.m"
  xmlns:core="sap.ui.core" >
  <Page>
  </Page>
</core:View>
```

view/App.view.js

- Load the empty view **instead of the** detail view

```

sap.ui.jsview("oscon2014.view.App", {
    getControllerName: function () {
        return "oscon2014.view.App";
    },
    createContent: function (oController) {

        // to avoid scroll bars on desktop the root view must be set to block display
        this.setDisplayBlock(true);

        // create app
        this.app = new sap.m.SplitApp();

        // load the master page
        var master = sap.ui.xmlview("Master", "oscon2014.view.Master");
        master.getController().nav = this.getController();
        this.app.addPage(master, true);

        // load the empty page
        var empty = sap.ui.xmlview("Empty", "oscon2014.view.Empty");
        this.app.addPage(empty, false);

        // done
        return this.app;
    }
});

```

index.html

- Wrap the split app in a shell control using the title defined before.
- **Why in the index.html?** This is done outside of the component because if you would plug a component in the SAP Fiori Launchpad this already renders the shell.

```

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="X-UA-Compatible" content="IE=edge" />
        <meta charset="UTF-8">

        <title>OSCON 2014 5</title>

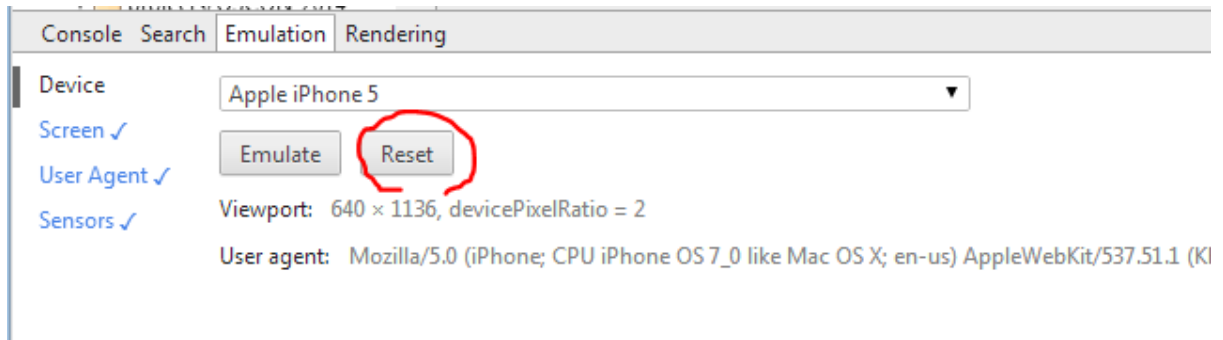
        <script
            id="sap-ui-bootstrap"
            src="../../resources/sap-ui-core.js"
            data-sap-ui-theme="sap_bluecrystal"
            data-sap-ui-libs="sap.m"
            data-sap-ui-xx-bindingSyntax="complex"
            data-sap-ui-resourceroots='{
                "oscon2014": "/"
            }' >
        </script>

        <script>
            new sap.m.Shell({
                app : new sap.ui.core.ComponentContainer({
                    name : "oscon2014"
                })
            }).placeAt("content");
        </script>

    </head>
    <body class="sapUiBody" id="content">
    </body>
</html>

```

- If you have not yet done so, disable the phone simulation mode in the Chrome Developer Tools by pressing the “Reset” button. After doing so, reload the page.
- This will allow the SplitApp control to make the best use of the screen estate.



Further Reading:

- SplitApp control: <https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/SplitApp.html>
- SplitApp API: <https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.SplitApp.html>
- Shell API: <https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.Shell.html>

Exercise 6 – Additional Device Adaptation

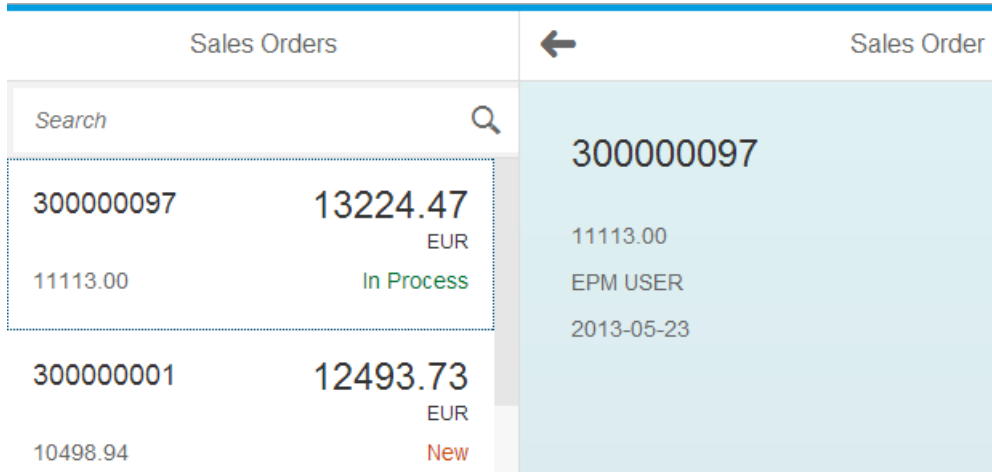
Objective

Adapt the controls to phone/tablet/desktop devices:

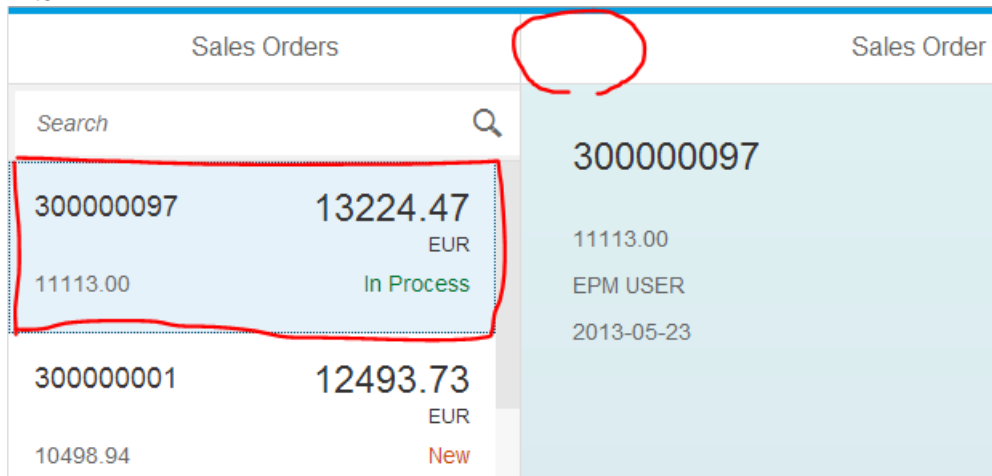
- Show the back button in the detail page only on the phone.
- Switch the list to selection mode on the tablet and desktop.

Preview

Before:



After:



Description

If the user can see both the master and detail section of the *SplitApp* at the same time because, say, they're using a tablet, there's not much point in showing a back button on the detail section – it's only really relevant on smaller screen sizes where either one or the other section is visible. So we will set the visibility of the back button (referred to as the 'navigation button' in the control) to be device dependent.

Also, depending on the device, we will set different list and item selection modes. Notice that we do the device determination up front when the application starts (in *Component.js*) setting the results of the determination in a one-way bound named data model, data from which can then be used in property path bindings in the *Detail* and *Master* views.

Changes

Component.js

- Set a global model named **"device"**
- Set **isPhone**, **listMode** and **listItemType** with the help of the **"device API"**.

```

jQuery.sap.declare("oscon2014.Component");
sap.ui.core.UIComponent.extend("oscon2014.Component", {

    createContent : function() {
        ...

        // set device model
        var deviceModel = new sap.ui.model.json.JSONModel({
            isPhone : jQuery.device.is.phone,
            isNoPhone : ! jQuery.device.is.phone,
            listMode : (jQuery.device.is.phone) ? "None" : "SingleSelectMaster",
            listItemType : (jQuery.device.is.phone) ? "Active" : "Inactive"
        });
        deviceModel.setDefaultBindingMode("OneWay");
        oView.setModel(deviceModel, "device");

        // done
        return oView;
    }
});

```

view/Detail.view.xml

- Bind the **showNavButton** property to the device model

```

<core:View
    controllerName="oscon2014.view.Detail"
    xmlns="sap.m"
    xmlns:core="sap.ui.core" >
    <Page
        title="{i18n>DetailTitle}"
        showNavButton="{device}/isPhone"
        navButtonPress="handleNavButtonPress" >

```

view/Master.view.xml

- Bind the **"list"** mode and the **"list item type"** to the device model
- Add a select event to the list

```

<List
    id="list"
    mode="{device}/listMode"
    select="handleListSelect"
    items="{/SalesOrderCollection}" >
    <ObjectListItem
        type="{device}/listItemType"
        press="handleListItemPress"
        title="{SoId}"
        number="{GrossAmount}"
        numberUnit="{CurrencyCode}" >

```

view/Master.controller.js

- Implement the select event in the view's controller

```

jQuery.sap.require("oscon2014.util.Formatter");

sap.ui.controller("oscon2014.view.Master", {

    handleListItemPress : function (evt) {
        var context = evt.getSource().getBindingContext();
        this.nav.to("Detail", context);
    },

    handleSearch : function (evt) {
        ...
    },

```

```
handleListSelect : function (evt) {  
    var context = evt.getParameter("listItem").getBindingContext();  
    this.nav.to("Detail", context);  
}  
});
```

Exercise 7 – Supplier Tab

Objective

Add an info tab to the detail page that shows a little form with data of the business partner of the sales order.

Preview

Before:

After:

Description

In this exercise we will enhance the display of the sales order detail view with a section showing the supplier name and address.

In the *Detail* view, we'll use an *IconTabBar* control to introduce the information visually, and a *SimpleForm* control to display the information. The *SimpleForm* control is from the *sap.ui.layout* library, so we need to add this to the UI5 bootstrap in the *index.html* too.

Changes

index.html

- Load the additional UI library “**sap.ui.layout**”

```
<!DOCTYPE html>
...

<script
  id="sap-ui-bootstrap"
  src="../../resources/sap-ui-core.js"
  data-sap-ui-theme="sap_bluecrystal"
  data-sap-ui-libs="sap.m, sap.ui.layout"
  data-sap-ui-xx-bindingSyntax="complex"
  data-sap-ui-resourceroots='{
    "oscon2014": "../../"
  }' >
</script>
...
```

view/Detail.view.xml

- Set xml namespaces for the new package (form)

- Implement a **sap.m.IconTabBar**
- Implement a **sap.ui.layout.SimpleForm** and bind the data. The data source will be connected in the next step.

```
<core:View
    controllerName="oscon2014.view.Detail"
    xmlns="sap.m"
    xmlns:form="sap.ui.layout.form"
    xmlns:core="sap.ui.core" >
    <Page
        title="{i18n>DetailTitle}"
        showNavButton="{device>/isPhone}"
        navButtonPress="handleNavButtonPress" >
        <ObjectHeader
            ""
        >
        </ObjectHeader>
        <IconTabBar
            expanded="{device>/isNoPhone}" >
            <items>
                <IconTabFilter
                    icon="sap-icon://supplier">
                    <form:SimpleForm
                        id="SupplierForm"
                        minWidth="1024" >
                        <core:Title text="Address" />
                        <Label text="Name" />
                        <Text text="{CompanyName}" />
                        <Label text="City" />
                        <Text text="{City}, {PostalCode}" />
                        <Label text="Street" />
                        <Text text="{Street}" />
                    </form:SimpleForm>
                </IconTabFilter>
            </items>
        </IconTabBar>
    </Page>
</core:View>
```

- **Bind the supplier form we just created to the data of the structure “BusinessPartner”**

```
sap.ui.controller("oscon2014.view.Detail", {
    handleNavButtonPress : function (evt) {
        this.nav.back("Master");
    },
    onBeforeRendering:function(){
        this.byId("SupplierForm").bindElement("BusinessPartner");
    }
});
```

Further Reading:

- Icon Tab Bar API: <https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.IconTabBar.html>
- Simple Form API: <https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.ui.layout.form.SimpleForm.html>

Exercise 8 – Approval Process



Objective

Add a button to the footer of the detail page to trigger the approval of a sales order. When the user presses the button a confirmation dialog is shown. If the user confirms the dialog, the sales order is deleted from the model and a confirmation message is shown.



Disclaimer: The server is not really called.


Preview

Before:

Sales Orders	Sales Order								
<div>Search </div> <tr> <td>300000097</td><td>13224.47 EUR 11113.00 In Process</td></tr> <tr> <td>300000001</td><td>12493.73 EUR 10498.94 New</td></tr> <tr> <td>300000002</td><td>11666.69 EUR 9803.94 New</td></tr> <tr> <td>300000003</td><td>16561.23 EUR 13917.00 In Process</td></tr>	300000097	13224.47 EUR 11113.00 In Process	300000001	12493.73 EUR 10498.94 New	300000002	11666.69 EUR 9803.94 New	300000003	16561.23 EUR 13917.00 In Process	<div>300000003</div> <div>16561.23 EUR In Process</div> <div>13917.00 EPM USER 2013-05-23</div> <div></div> <div>Address</div> <div>Name: AVANTEL</div> <div>City: Mexiko City, 17000</div> <div>Street: Bosque de Duraznos</div>
300000097	13224.47 EUR 11113.00 In Process								
300000001	12493.73 EUR 10498.94 New								
300000002	11666.69 EUR 9803.94 New								
300000003	16561.23 EUR 13917.00 In Process								

After:

Sales Orders	Sales Order								
<div>Search </div> <tr> <td>300000097</td><td>13224.47 EUR 11113.00 In Process</td></tr> <tr> <td>300000001</td><td>12493.73 EUR 10498.94 New</td></tr> <tr> <td>300000002</td><td>11666.69 EUR 9803.94 New</td></tr> <tr> <td>300000003</td><td>16561.23 EUR 13917.00 In Process</td></tr>	300000097	13224.47 EUR 11113.00 In Process	300000001	12493.73 EUR 10498.94 New	300000002	11666.69 EUR 9803.94 New	300000003	16561.23 EUR 13917.00 In Process	<div>300000097</div> <div>13224.47 EUR In Process</div> <div>11113.00 EPM USER 2013-05-23</div> <div></div> <div>Address</div> <div>Name: SAP AG</div> <div>City: Walldorf, 69190</div> <div>Street: Dietmar-Hopp-Allee</div>
300000097	13224.47 EUR 11113.00 In Process								
300000001	12493.73 EUR 10498.94 New								
300000002	11666.69 EUR 9803.94 New								
300000003	16561.23 EUR 13917.00 In Process								

 Approve

Description

To achieve the aim of this exercise, we'll be making small changes to lots of the files in the project.

We need to add a footer bar (a *Bar* control within the footer aggregation of the *Page*) to each of the views (*Detail*, *Empty* and *Master*) to keep things visually nice and consistent.

We'll add a *Button* control to the right side of the footer bar in the *Detail* view, and in the corresponding controller we'll define the function to be called (*handleApprove*) when the *Button*'s *press* event is fired. We'll just simulate the approval process by displaying a *MessageBox* popup control and then showing a *MessageToast*. For this we'll need to show some texts, so we'll add them to the same properties file we set up earlier in relation to the resource model.

Changes

i18n/messageBundle.properties

- Add more texts for the approve button and dialog

```
MasterTitle=Sales Orders
DetailTitle=Sales Order
StatusTextN=New
StatusTextP=In Process
ApproveButtonText=Approve
ApproveDialogTitle=Approve Sales Order
ApproveDialogMsg=Do you want to approve this sales order now?
ApproveDialogSuccessMsg=The sales order has been approved
```

view/Detail.view.xml

- Add a footer to the *Detail* page which holds the button to trigger the approval

```
< IconTabBar >
...
</IconTabBar>
<footer>
  <Bar>
    <contentRight>
      <Button
        text="{i18n>ApproveButtonText}"
        type="Accept"
        icon="sap-icon://accept"
        press="handleApprove" />
    </contentRight>
  </Bar>
</footer>
</Page>
</core:View>
```

view/Detail.controller.js

- First we need to register one more class because it is not a control, but just a helper class (*MessageBox*)
- On handling the approve event we first show a confirmation dialog (*MessageBox*)
- If the user confirms we only show a success message (*MessageToast*). **Calling a real service is not part of this exercise.**

```
jQuery.sap.require("oscon2014.util.Formatter");
jQuery.sap.require("sap.m.MessageBox");

sap.ui.controller("oscon2014.view.Detail", {
  handleNavButtonPress : function (evt) {
    this.nav.back("Master");
  },
  handleApprove : function (evt) {
    // show confirmation dialog
```

```

        var bundle = this.getView().getModel("i18n").getResourceBundle();
        sap.m.MessageBox.confirm(
            bundle.getText("ApproveDialogMsg"),
            function (oAction) {
                if (sap.m.MessageBox.Action.OK === oAction) {
                    // notify user
                    var successMsg = bundle.getText("ApproveDialogSuccessMsg");
                    sap.m.MessageToast.show(successMsg);
                    // TODO call proper service method and update model (not part of this
session)
                }
            },
            bundle.getText("ApproveDialogTitle")
        );
    },
};

```

```

onBeforeRendering:function(){
    this.byId("SupplierAddress").bindElement("BusinessPartner");
}
});

```

view/Empty.view.xml

- We now need footers in all pages for symmetry

```

<core:View
    xmlns="sap.m"
    xmlns:core="sap.ui.core" >
    <Page>
        <footer>
            <Bar>
            </Bar>
        </footer>
    </Page>
</core:View>

```

view/Master.view.xml

- We now need footers in all pages for symmetry

```

        ...
        </ObjectListItem>
    </List>
    <footer>
        <Bar>
        </Bar>
    </footer>
</Page>
</core:View>

```

Further Reading:

- Page API: <https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.Page.html>
- Modularization and Dependency Management (require/declare modules):
<https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/ModularizationConcept.html>

Exercise 9 – Line Items

Objective

Extend the detail page with a table that shows the line items of the sales order. The rows are active and allow navigating to the new line item page.

Preview

Before:

Sales Orders	Sales Order																								
<div>Search</div> <table> <tr> <td>300000097</td><td>13224.47 EUR</td></tr> <tr> <td>11113.00</td><td>In Process</td></tr> <tr> <td>300000001</td><td>12493.73 EUR</td></tr> <tr> <td>10498.94</td><td>New</td></tr> <tr> <td>300000002</td><td>11666.69 EUR</td></tr> <tr> <td>9803.94</td><td>New</td></tr> <tr> <td>300000003</td><td>16561.23 EUR</td></tr> <tr> <td>13917.00</td><td>In Process</td></tr> <tr> <td>300000004</td><td>12515.23 EUR</td></tr> <tr> <td>10517.00</td><td>In Process</td></tr> <tr> <td>300000005</td><td>8368.08 EUR</td></tr> <tr> <td>7032.00</td><td>New</td></tr> </table>	300000097	13224.47 EUR	11113.00	In Process	300000001	12493.73 EUR	10498.94	New	300000002	11666.69 EUR	9803.94	New	300000003	16561.23 EUR	13917.00	In Process	300000004	12515.23 EUR	10517.00	In Process	300000005	8368.08 EUR	7032.00	New	<div>300000097</div> <div>13224.47 EUR</div> <div>11113.00</div> <div>EPM USER</div> <div>2013-05-23</div> <div> </div> <div>Address</div> <div> Name: SAP AG City: Walldorf, 69190 Street: Dietmar-Hopp-Allee </div> <div> </div>
300000097	13224.47 EUR																								
11113.00	In Process																								
300000001	12493.73 EUR																								
10498.94	New																								
300000002	11666.69 EUR																								
9803.94	New																								
300000003	16561.23 EUR																								
13917.00	In Process																								
300000004	12515.23 EUR																								
10517.00	In Process																								
300000005	8368.08 EUR																								
7032.00	New																								

After:

Sales Orders

Search

300000097

13224.47

EUR

11113.00

In Process

300000001

12493.73

EUR

10498.94

New

300000002

11666.69

EUR

9803.94

New

300000003

16561.23

EUR

13917.00

In Process

300000004

12515.23

EUR

10517.00

In Process

300000005

8368.08

EUR

7032.00

New

Sales Order

300000097

13224.47

EUR

11113.00

EPM USER

2013-05-23

Address

Name: SAP AG

City: Walldorf, 69190

Street: Dietmar-Hopp-Allee

Products

Product	Delivery Date	Quantity	Price
HT-1000	2013-05-30	1	1137.64 EUR >
HT-1091	2013-05-30	2	61.88 EUR >
HT-6100	2013-05-30	2	1116.22 EUR >

Approve

Description

In this exercise we're going to add some more details to the existing *Detail* view, specifically a new *Table* control containing the line items from the selected order. We'll put the *Table* control underneath the *IconTabBar* that we introduced in an earlier exercise.

To format each order item's quantity, we'll add a further function called 'quantity' to the *Formatter.js* module we already have. This will then be used in the complex binding definition of the respective 'quantity' text in the table *ColumnListItem*'s cells aggregation.

We'll handle the selection of a line in the line items table with a 'handleLineItemsPress' function in the *Detail* view's controller. This is bound to the press event of the *Table*'s *ColumnListItem* as you can see in the *Detail* view XML below. On selection, we want to navigate to a new view, *LineItem*, passing the context of the selected item.

So we'll create a new *LineItem* view, also containing a *Page* control with a *Bar* in the footer aggregation, like all the other views, and display line item details. When the navigation button is pressed we transition back to the *Detail* view with a simple handler 'handleNavBack' in the *LineItem* controller.

Changes

i18n/messageBundle.properties

- Add more message texts

```
MasterTitle=Sales Orders
DetailTitle=Sales Order
StatusTextN=New
StatusTextP=In Process
ApproveButtonText=Approve
ApproveDialogTitle=Approve Sales Order
ApproveDialogMsg=Do you want to approve this sales order now?
ApproveDialogSuccessMsg=The sales order has been approved
LineItemTableHeader=Products
LineItemTitle=Product
```

util/Formatter.js

- We need a new formatter for quantities that removes the trailing zeroes from the number

```
jQuery.sap.declare("oscon2014.util.Formatter");
jQuery.sap.require("sap.ui.core.format.DateFormat");

oscon2014.util.Formatter = {
    ...
},
    quantity : function (value) {
        try {
            return (value) ? parseFloat(value).toFixed(0) : value;
        } catch (err) {
            return "Not-A-Number";
        }
    }
};
```

view/Detail.view.xml

- We set a CSS class on the page control that will set proper margins on the table control in this page.
- There is quite a bit of change to implement the table with the help of a list

```
<core:View
    controllerName="oscon2014.view.Detail"
```

```

xmlns="sap.m"
xmlns:form="sap.ui.layout.form"
xmlns:core="sap.ui.core" >
<Page
    title="{i18n>DetailTitle}"
    class="sapUiFioriObjectPage"
    showNavButton="{device>/isPhone}"
    navButtonPress="handleNavButtonPress" >

...

</IconTabBar>
<Table
    headerText="{i18n>LineItemTableHeader}"
    items="{LineItems}" >
    <columns>
        <Column>
            <header><Label text="Product" /></header>
        </Column>
        <Column>
            minScreenWidth="Tablet"
            demandPopin="true"
            hAlign="Center" >
            <header><Label text="Delivery Date" /></header>
        </Column>
        <Column>
            minScreenWidth="Tablet"
            demandPopin="true"
            hAlign="Center" >
            <header><Label text="Quantity" /></header>
        </Column>
        <Column>
            hAlign="Right" >
            <header><Label text="Price" /></header>
        </Column>
    </columns>
    <ColumnListItem
        type="Navigation"
        press="handleLineItemPress" >
        <cells>
            <ObjectIdentifier
                title="{ProductId}" />
            <Text
                text="{
                    path: 'DeliveryDate',
                    formatter: 'oscon2014.util.Formatter.date'
                }"/>
            <Text
                text="{
                    path: 'Quantity',
                    formatter: 'oscon2014.util.Formatter.quantity'
                }"/>
            <ObjectNumber
                number="{GrossAmount}"
                numberUnit="{CurrencyCode}" />
        </cells>
    </ColumnListItem>
</Table>
<footer>
    ...
</footer>
</Page>
</core:View>

```

view/Detail.controller.js

- When a line item is pressed, we navigate to the new line item page

...

```

        handleApprove : function (evt) {
...
        },
        handleLineItemPress : function (evt) {
            var context = evt.getSource().getBindingContext();
            this.nav.to("LineItem", context);
        },

        onBeforeRendering:function(){
            this.byId("SupplierAddress").bindElement("BusinessPartner");
        }
    });

```

view/LineItem.view.xml (ADD NEW FILE, content should be as below)

- For the sake of simplicity we only put an object header to the line item page.

```

<core:View
    controllerName="oscon2014.view.LineItem"
    xmlns="sap.m"
    xmlns:core="sap.ui.core" >
    <Page
        id="page"
        title="{i18n>LineItemTitle}"
        showNavButton="true"
        navButtonPress="handleNavBack" >
        <footer>
            <Bar>
            </Bar>
        </footer>
        <content>
            <ObjectHeader
                title="{ProductId}"
                number="{GrossAmount}"
                numberUnit="{CurrencyCode}" >
                <attributes>
                    <ObjectAttribute text="{
                        path: 'DeliveryDate',
                        formatter: 'oscon2014.util.Formatter.date'
                    }" />
                    <ObjectAttribute text="{
                        path: 'Quantity',
                        formatter: 'oscon2014.util.Formatter.quantity'
                    }" />
                </attributes>
            </ObjectHeader>
        </content>
    </Page>
</core:View>

```

view/LineItem.controller.js (ADD NEW FILE, content should be as below)

- We only need to handle the back navigation to the **Detail** page

```

sap.ui.controller("oscon2014.view.LineItem", {
    handleNavBack : function (evt) {
        this.nav.back("Detail");
    }
});

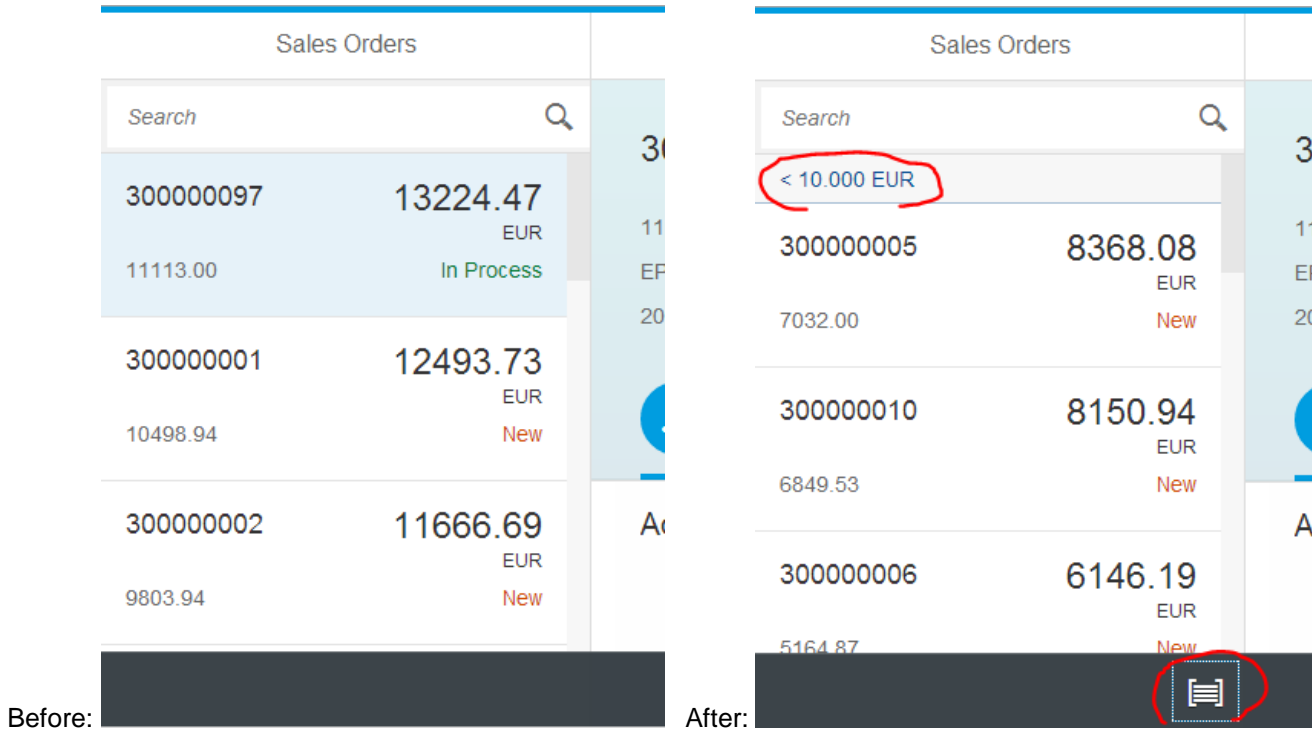
```


Exercise 10 – Grouping

Objective

Add a **“Select”** to the master list that lets the user select a grouping. Handle the user selection and apply the grouping to the data binding.

Preview



Description

We're almost there. In this last exercise we're going to add grouping features that can be applied when aggregation bindings are sorted. In this case the binding is the one between the sales orders in the data model and the items aggregation in the *List* control in the *Master* view.

We'll create a new file in the *'util'* folder, containing two custom grouping functions. We'll add a *Select* control to the *Bar* in the Page footer in the *Master* view, and in the corresponding controller, we will handle the button press with a function *'handleGroup'* that updates the data binding of the list.

Changes

i18n/messageBundle.properties

- Add more message texts

```
MasterTitle=Sales Orders
DetailTitle=Sales Order
StatusTextN=New
StatusTextP=In Process
ApproveButtonText=Approve
ApproveDialogTitle=Approve Sales Order
ApproveDialogMsg=Do you want to approve this sales order now?
ApproveDialogSuccessMsg=The sales order has been approved
LineItemTableHeader=Products
LineItemTitle=Product
MasterGroupNone=None
MasterGroupStatus=Status
MasterGroupAmount=Amount
```

util/Grouper.js (ADD NEW file Grouper.js)

- This new file contains two functions that implement the logic to group sales orders by
 - **Status** (simple string comparison)
 - **Amount** (a little bit more sophisticated price checks)

```
jQuery.sap.declare("oscon2014.util.Grouper");

oscon2014.util.Grouper = {

    bundle : null, // somebody has to set this

    LifecycleStatus : function (oContext) {
        var status = oContext.getProperty("LifecycleStatus");
        var text = oscon2014.util.Grouper.bundle.getText("StatusText" + status, "?");
        return {
            key: status,
            text: text
        };
    },

    GrossAmount : function (oContext) {
        var price = oContext.getProperty("GrossAmount");
        var currency = oContext.getProperty("CurrencyCode");
        var key = null,
            text = null;
        if (price <= 5000) {
            key = "LE10";
            text = "< 5000 " + currency;
        } else if (price > 5000 && price <= 10000) {
            key = "LE100";
            text = "< 10.000 " + currency;
        } else if (price > 10000) {
            key = "GT100";
            text = "> 10.000 " + currency;
        }
        return {
            key: key,
            text: text
        };
    }
};
```

view/Master.view.xml

- Add a select control to the footer of the master page to choose a criteria for grouping

```
<core:View
    controllerName="oscon2014.view.Master"
    xmlns="sap.m"
    xmlns:core="sap.ui.core" >
    ...
    <footer>
        <Bar>
            <contentRight>
                <Select
                    id="groupSelect"
                    change="handleGroup"
                    icon="sap-icon://group-2"
                    type="IconOnly"
                    selectedKey="None"
                    autoAdjustWidth="true" >
                    <core:Item
                        key="None"
                        text="{i18n>MasterGroupNone}"/>
                    <core:Item
                        key="GrossAmount"
                        text="{i18n>MasterGroupAmount}"/>
                    <core:Item
```

```

                                key="LifecycleStatus"
                                text="{i18n>MasterGroupStatus}"/>
                            </Select>
                        </contentRight>
                    </Bar>
                </footer>
            </Page>
        </core:View>

```

view/Master.controller.js

- Require the new “**Grouper.js**” file
- Implement the “**handleGroup**” function
 - Compute the sorter object that will perform the grouping
 - Apply the grouping to the data binding

```

jQuery.sap.require("oscon2014.util.Formatter");
jQuery.sap.require("oscon2014.util.Grouper");

sap.ui.controller("oscon2014.view.Master", {

    handleListItemPress : function (evt) {
        var context = evt.getSource().getBindingContext();
        this.nav.to("Detail", context);
    },

    handleSearch : function (evt) {

        // create model filter
        var filters = [];
        var query = evt.getParameter("query");
        if (query && query.length > 0) {
            var filter = new sap.ui.model.Filter("SoId", sap.ui.model.FilterOperator.Contains,
query);
            filters.push(filter);
        }

        // update list binding
        var list = this.getView().byId("list");
        var binding = list.getBinding("items");
        binding.filter(filters);
    },

    handleListSelect : function (evt) {
        var context = evt.getParameter("listItem").getBindingContext();
        this.nav.to("Detail", context);
    },

    handleGroup : function (evt) {

        // compute sorters
        var sorters = [];
        var item = evt.getParameter("selectedItem");
        var key = (item) ? item.getKey() : null;
        if ("GrossAmount" === key || "LifecycleStatus" === key) {
            oscon2014.util.Grouper.bundle = this.getView().getModel("i18n").getResourceBundle();
            var grouper = oscon2014.util.Grouper[key];
            sorters.push(new sap.ui.model.Sorter(key, true, grouper));
        }

        // update binding
        var list = this.getView().byId("list");
        var oBinding = list.getBinding("items");
        oBinding.sort(sorters);
    }
});

```

Further Reading:

- Select API: <https://sapui5.netweaver.ondemand.com/sdk/#docs/api/symbols/sap.m.Select.html>

© 2014 by SAP AG or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG.

The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

Please see

<http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark>

for additional trademark information and notices.



The Best-Run Businesses Run SAP™