Use Cases for Extending the UI of SAP Fiori Apps

Details various options for extending the UI of certain SAP Fiori Apps
Table of Contents
Introduction ............................................................................................................................................. 3
Terminologies .......................................................................................................................................... 3
Functional Scenario – The Report Quality Issue Application ........................................................... 3
Extensibility Use Cases ..................................................................................................................... 6
Setting Up a Custom Application Project ........................................................................................ 6
Implementing Extensibility Use Cases ............................................................................................... 8
  Use case: Hiding a field ..................................................................................................................... 8
  Use case: Adding New Fields at Specified Extension Points ......................................................... 9
  Use case: Adding New Tabs to the Icon Tab Bar ......................................................................... 10
  Use case: Adding a custom list to the S3 screen ......................................................................... 11
  Use Case: Adding a Confirm Button to the Footer of the Q Issue Detail Screen .................. 13
  Use case: Using custom text .......................................................................................................... 15
  Use case: Translation ..................................................................................................................... 16
  Use case: Using custom oData Service URL .............................................................................. 16
  Use case: Replacing an SAP-delivered Standard Application View with a Custom View .......... 16
  Use case: Adding a custom view (Q Issue Task Detail) ................................................................. 23
  Use case: Navigating to the custom view ...................................................................................... 25
  Use case: Manipulating the model of the SAP standard application at predefined hooks in controller 27
Uploading a Custom Application to the Server ................................................................................ 28
Known Limitations of SAP UI5 1.16.4 .............................................................................................. 29
  The i18n resource cannot be accessed directly ............................................................................. 29
Appendix ................................................................................................................................................ 29
  Recommendations on controller extensions .............................................................................. 29
  Overriding functions in classes other than controller classes ................................................... 30
  Reusing the original implementation of a function in an extended controller ......................... 31
Useful References ............................................................................................................................... 32
Introduction
This document describes various extension possibilities for a sample SAP Fiori application. It applies to Transactional apps and to those Analytical apps that use HANA Live and XS OData. The main focus of the document is UI extensibility.

Terminologies

<table>
<thead>
<tr>
<th>Term</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension Point(s)</td>
<td>To facilitate extension of applications on the UI Layer, SAP delivers extension points. These constitute anchor points for extensions. An extension point refers to the SAP UI5 control <code>&lt;core:ExtensionPoint /&gt;</code>. The control is inserted into an-SAP shipped HTML5 application based on an XML type view. Extension points are documented and kept stable. This means that any extensions plugging in are more robust across application updates. More information can be found here: <a href="https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/Customization.html">https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/Customization.html</a></td>
</tr>
<tr>
<td>UI Controller Hooks</td>
<td>A UI controller hook is a certain type of extension point in UI controller code. SAP delivers UI controller hooks to facilitate extending UI controller code in SAP Fiori applications. UI controller hooks are methods in the UI controller code which are documented and kept stable. This means that any extensions plugging in are more robust across application updates. More information can be found here: <a href="https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/Customization.html">https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/Customization.html</a></td>
</tr>
</tbody>
</table>

Functional Scenario – The Report Quality Issue Application
You can use the Report Quality Issue application to capture and view reported quality issues. The application has the following four views:

- Capture a new quality issue
- View the list of reported quality issues
- View the detail of reported quality issues
- View the image associated with a quality issue

You can access examples of SAP-delivered standard applications and custom applications using the following information in your ABAP systems:

<table>
<thead>
<tr>
<th>Application</th>
<th>Application Link</th>
<th>Code Repository</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAP-Delivered Standard Application</td>
<td><code>&lt;your_ABAP_URL&gt;/sap/public/bc/ui5_ui5/demokit/test-resources/sap/ca/appref/noShellIndex.html</code></td>
<td>MIME Repository of ABAP System where SAPUI5 demokit is deployed</td>
</tr>
</tbody>
</table>
**Use Cases for Extending the UI of SAP Fiori Apps**

Run Transaction SE80 → click on “MIME Repository” → SAP → PUBLIC → BC → UI5 → APPLICATIONS → demokit → <SAP_UI5_version_number> → test-resources → sap → ca → appref

**Custom Application**

<table>
<thead>
<tr>
<th>Application</th>
<th>BSP Application Name</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAP-Delivered Standard Application</td>
<td>ZF2UIAPPREF002</td>
<td>i2d.qm.qualityissue.confirm.appref</td>
</tr>
<tr>
<td>Custom Application</td>
<td>ZF2UIAPPREF002E</td>
<td>i2d.qm.qualityissue.confirm.apprefExt</td>
</tr>
</tbody>
</table>

The SAP-delivered standard application has the following four views:

- S2.view.xml
- S3.view.xml
- IssueForm.view.xml
- Viewer.view.xml

These views contain UI elements as shown in the screenshots below:

Replace <your_ABAP_URL> with http://<ABAP server>:<port>

Replace <SAP_UI5_version_number> with the UI 5 version like for example 1.20

**Note:** noShellIndex.html is only used for the sake of the demokit applications. However, this file is not relevant for a productive SAP Fiori app, and it does not need to be set up even for testing extensibility in an SAP Fiori launchpad. The file might be useful only for a non-productive use case of testing an SAP Fiori app without using the launchpad.

In the following example, the BSP name of the SAP-delivered standard application is assumed to be ZF2UIAPPREF002, and the BSP name of the custom application is ZF2UIAPPREF002E.
<table>
<thead>
<tr>
<th>Purpose of the View</th>
<th>View Name</th>
<th>Screen Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capture a new quality issue</td>
<td>IssueForm.view.xml</td>
<td>Q Issue Create Form</td>
</tr>
<tr>
<td>View the list of reported quality issues</td>
<td>S2.view.xml</td>
<td>Q Issue list</td>
</tr>
<tr>
<td>View the detail of reported quality issues</td>
<td>S3.view.xml</td>
<td>Q Issue Detail</td>
</tr>
<tr>
<td>View the images associated with a quality issue</td>
<td>Viewer.view.xml</td>
<td>Q Issue Image Viewer</td>
</tr>
</tbody>
</table>
### Extensibility Use Cases

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hiding a field</td>
<td>Since <code>defect</code> is a field that is displayed in the list section, you can hide it on the <code>Q Issue detail screen</code>.</td>
</tr>
<tr>
<td>Adding custom fields at a defined extension point</td>
<td>Add <code>Department Responsible</code> and <code>Coordinator</code> to the information tab of the <code>Q Issue detail screen</code>.</td>
</tr>
<tr>
<td>Adding custom tabs to the icon tab bar</td>
<td>The custom icon tab bar displays the quantities associated with defective items in the <code>Q Issue detail screen</code>.</td>
</tr>
<tr>
<td>Adding a custom list to the <code>Q Issue Detail</code> screen</td>
<td>The custom list contains the list of tasks associated with the quality issue.</td>
</tr>
<tr>
<td>Adding a confirm button to the footer of the <code>Q Issue Detail</code> screen</td>
<td>The custom <code>Confirm</code> button confirms the quality issue. Once the user confirms the quality issue, it can no longer be reopened.</td>
</tr>
<tr>
<td>Using custom texts</td>
<td>A custom text for <code>Department Responsible</code> and <code>Coordinator</code> in the information tab of the <code>Q Issue Detail</code> screen.</td>
</tr>
<tr>
<td>Using custom oData Service</td>
<td>Add model for the custom field <code>Coordinator</code> and <code>Department Responsible</code> in the information tab of the <code>Q Issue Detail</code> screen.</td>
</tr>
<tr>
<td>Replacing an SAP delivered standard application view with a Custom View</td>
<td>You can replace the <code>Q Issue Create Form</code> view with a custom <code>Q Issue Create Form</code> view. This replaces the <code>Description</code> field with a <code>Date</code> field.</td>
</tr>
<tr>
<td>Adding the custom view <code>Q Issue Task Detail</code></td>
<td>The <code>Q Issue Task Detail</code> view displays the details of a task item for a quality issue.</td>
</tr>
<tr>
<td>Navigating to the custom view <code>Q Issue Task Detail</code></td>
<td>Navigate to the custom view when a task item for a quality issue is selected.</td>
</tr>
<tr>
<td>Manipulating the model of SAP standard application at predefined hooks in controller</td>
<td>Change the order of the images in the attachment tab in the <code>Q Issue detail screen</code>.</td>
</tr>
</tbody>
</table>

### Setting Up a Custom Application Project

**Initial Project Setup**

1. In Eclipse, create a new SAPUI5 Application Project by choosing `New → SAPUI5 Application Development` and selecting `SAPUI5 Application Project` from the first page of the wizard.
2. In the New Application Project wizard, enter a name, specify the location, and select the Mobile target device. Deselect the Create an Initial View check box, and then click Finish.

3. Create a view, model, and ‘i18n’ folder, and a Component.js file in the WebContent folder.

   ![WebContent structure]

   Note: Style sheet information is stored in the css folder. You can define formatting logic in the util folder.

Setting up Component.js

1. The Component.js file extends the SAP-delivered standard application component as shown below.

   Note In Eclipse, the BSP application name/project name of the SAP-delivered standard application is ZF2UIAPPREF002.

   ```javascript
   jQuery.sap.declare("i2d.qm.qualityissue.confirm.apprefExt.Component");
   // use the load function for getting the optimized preload file if present
   sap.ui.component.load({
     name: "i2d.qm.qualityissue.confirm.appref",
     url: jQuery.sap.getModulePath("i2d.qm.qualityissue.confirm.apprefExt") + "/../ZF2UIAPPREF002"
   });
   i2d.qm.qualityissue.confirm.appref.Component.extend("i2d.qm.qualityissue.confirm.apprefExt.Component",
```
Use Cases for Extending the UI of SAP Fiori Apps

The namespace for the custom application might differ from the standard SAP application. You may need to adjust this path before uploading the custom application to the ABAP system.

Ensure that the project name of the standard SAP application should be the same as the BSP application service name on the server.

2. Add the configuration, customizing, and routing data to the metadata of the Component.js file.

Implementing Extensibility Use Cases

Use case: Hiding a field
You may want to hide a field on a UI. This use case covers hiding a field.

*Defect* is a field that is displayed on the *Q Issue list* screen as well as on the *Q Issue detail* screen in the SAP-delivered standard application. You can hide the *Defect* field in the *Q Issue detail* screen.

**Prerequisite**

The *Defect* field has an ID associated with it in the SAP-delivered standard application.

1. To hide the field, set the visibility property of the *Defect* field to false.

   The S3.view.xml file contains a field with a *DEFECT* ID associated with it.

   Add the following piece of code to the *metadata* object in the *Component.js* file.

   ```javascript
   metadata: {
     customizing: {
       /*meta data details like configuration, customizing, and routing data*/
     }
   };
   ```
Use Cases for Extending the UI of SAP Fiori Apps

```
"sap.ui.viewModifications": {
  "i2d.qm.qualityissue.confirm.appref.view.S3": {
    "DEFECT": {
      "visible": false,
    }
  }
}
```

Use case: Adding New Fields at Specified Extension Points
This use case covers adding two new fields, Department Responsible and Coordinator, to the Information tab on the Q Issue detail screen.

**Prerequisite:** An extension point is provided in the Information tab of the Q Issue Detail screen in the SAP-delivered standard application.

### SAP-delivered standard application

![SAP-delivered standard application](image1)

### Custom application

![Custom application](image2)

**Steps**

1. Predefine an extension point in the S3.view.xml file of the SAP-delivered standard application.

   ```xml
   <core:ExtensionPoint name="extMoreInfo"/>
   ``

2. Create a new xml file called CustomFrag1.fragment.xml in the View folder of the custom application.

   Note: For more information about adding custom text in the fragment definition, see [Using custom text](#).

   Note: For more information about adding custom field data in the fragment definition, see [Using custom oData Service](#).

3. To begin creating the new fields, you add the following code to the fragment file:

   ```xml
   <core:FragmentDefinition xmlns="sap.m" xmlns:core="sap.ui.core">
   <Label text="{i18n:DEPARTMENT_RESP}"/>
   ```
Use Cases for Extending the UI of SAP Fiori Apps

4. You then add the following code to the Customizing object in the Component.js file:

```javascript
"sap.ui.viewExtensions": {
  "i2d.qm.qualityissue.confirm.appref.view.S3": {
    "extMoreInfo": {
      className: "sap.ui.core.Fragment",
      fragmentName: "i2d.qm.qualityissue.confirm.apprefExt.view.CustomFrag1",
      type: "XML"
    }
  }
}
```

Use case: Adding New Tabs to the Icon Tab Bar
This use case covers adding a new tab. The new tab in the icon tab bar displays the quantities associated with defective items in the Q Issue Detail screen.

**Prerequisite:** An extension point is provided in the icon tab bar of the Q Issue Detail screen in the SAP-delivered standard application.

**Steps**

1. Predefine an extension point in the S3.view.xml file (Q Issue Detail) of the SAP-delivered standard application.

   ```xml
   <core:ExtensionPoint name="extTabQuantities"/>
   ```

2. Create a new xml file called CustomFrag2.fragment.xml in the view folder of the custom application.

   Note: For more information about adding custom text in the fragment definition, see [Using custom text](#).

   Note: For more information about adding custom field data in the fragment definition, see [Using custom oData Service](#).
Use Cases for Extending the UI of SAP Fiori Apps

3. Add the following icon tab bar code to the fragment file:

```xml
<IconTabFilter xmlns="sap.m"
    key="Info"
    icon="sap-icon://example">
    <form SimpleForm xmlns="sap.ui.layout.form"
        id="QUANTITY_FORM"
        minWidth="1024"
        maxContainerCols="2">
        <form:content xmlns="sap.ui.layout.form">
            <Label xmlns="sap.m" text="{i18n>COMPLAINT_QTY}"/>
            <Text xmlns="sap.m" text="{ComplaintQty}"/>
            <Label xmlns="sap.m" text="{i18n>REF_QTY}"/>
            <Text xmlns="sap.m" text="{RefQuantity}"/>
            <Label xmlns="sap.m" text="{i18n>RET_DELIVERY_UNIT}"/>
            <Text xmlns="sap.m" text="{ReturnDeliveryUnit}"/>
            <Label xmlns="sap.m" text="{i18n>RETURNED_ON}"/>
            <Text xmlns="sap.m" text="{path:'ReturnedOn', type:'sap.ca.ui.model.type.Date', formatOptions : { style:'medium'}}"/>
            <Label xmlns="sap.m" text="{i18n>DEF_QTY_INTERNAL}"/>
            <Text xmlns="sap.m" text="{DefiniteQuantityInternal}"/>
            <Label xmlns="sap.m" text="{i18n>DEF_QTY_EXTERNAL}"/>
            <Text xmlns="sap.m" text="{DefiniteQuantityExternal}"/>
        </form:content>
    </form SimpleForm>
</IconTabFilter>
```

4. Add the following code to the Customizing object in the Component.js file:

```javascript
"sap.ui.viewExtensions":{
    "i2d.qm.qualityissue.confirm.appref.view.S3":{
        "extTabQuantities": {
            className: "sap.ui.core.Fragment",
            fragmentName: "i2d.qm.qualityissue.confirm.apprefExt.view.CustomFrag2",
            type: "XML"
        }
    }
}
```

**Use case: Adding a custom list to the S3 screen**

This use case covers adding a custom list to the S3 screen. The list contains the tasks associated with a quality issue. It is displayed below the icon tab bar in the *Q Issue Detail* screen.

**Prerequisite:** An extension point is provided in the *Q Issue Detail* screen in the SAP-delivered standard application.

| SAP-delivered standard application | Custom application |
Use Cases for Extending the UI of SAP Fiori Apps

Steps

1. Predefine an extension point in the S3.view.xml file (Q Issue detail) of the SAP-delivered standard application.
   ```xml
   <core:ExtensionPoint name="extTaskList"/>
   ```

2. Create a new xml file called extList.fragment.xml in the view folder of the custom application.
   Note: For more information about adding custom text in the fragment definition, see Using custom text.
   Note: For more information about adding custom field data in the fragment definition, see Using custom oData Service.

3. Add the following code for the custom list in the fragment file:
   ```xml
   <List xmlns="sap.m" id="lineItemList" class="lineItemList"
       headerText="{i18n>TASK_LIST}" headerDesign="Plain" items="{path:'/Tasks'}">
   <columns xmlns="sap.m" width="12em">
       <Column xmlns="sap.m">
           <header>
               <Label text="{i18n>TASK_TEXT}"/>
           </header>
       </Column>
       <Column xmlns="sap.m" minScreenWidth="Tablet" demandPopin="true" hAlign="Center" id="TaskText">
           <header>
               <Label text="{i18n>TASK_ID}"/>
           </header>
       </Column>
       <Column xmlns="sap.m" minScreenWidth="Tablet" demandPopin="true" hAlign="Center">
           <header>
               <Label text="{i18n>TASK_STATUS}"/>
           </header>
       </Column>
       <Column xmlns="sap.m" hAlign="Right">
           <header>
               <Label text="{i18n>RESPONSIBLE}"/>
           </header>
       </Column>
   </columns>
   <ColumnListItem xmlns="sap.m" id="columnListItem" type="Navigation"
       press="handleTaskItemPress">
       <cells xmlns="sap.m">
           <ObjectIdentifier xmlns="sap.m" title="{TaskText}"/>
           <Text xmlns="sap.m" text="{TaskID}"/>
           <ObjectStatus xmlns="sap.m" text="{TaskStatus}"/>
           <Text xmlns="sap.m" text="{Responsible}"/>
       </cells>
   </ColumnListItem>
   ```
4. Add the following code to the Customizing object in the Component.js file:

```
"sap.ui.viewExtensions":{
  "i2d.qm.qualityissue.confirm.appref.view.S3":{
    "extTaskList": {
      className: "sap.ui.core.Fragment",
      fragmentName: "i2d.qm.qualityissue.confirm.apprefExt.view.extList",
      type : "XML"
    }
   }
}
```

**Use Case: Adding a Confirm Button to the Footer of the Q Issue Detail Screen**

The Confirm button confirms the quality issue. The system prompts the user to confirm the action. Once the user confirms the quality issue, it can no longer be reopened. This use case covers adding this button.

**Prerequisite:** An extension point is provided in the footer of the Q Issue Detail screen in the SAP-delivered standard application.

![SAP-delivered standard application vs Custom application with custom button](image-url)

**Custom application — Click the Submit button**
Steps

1. Predefine an extension point in the S3.view.xml file *(Q Issue detail)* of the SAP-delivered standard application.
   
   ```xml
   <core:ExtensionPoint name="extFooter"/>
   ```

2. Create a new xml file called `extFooter.fragment.xml` in the `view` folder of the custom application.
   
   Note: For more information about adding custom text in the fragment definition, see [Using custom text](#).

3. Add the following code for the custom button to the fragment file:
   
   ```xml
   <Button xmlns="sap.m" text="{i18n>CONFIRM}" press="doConfirm"/>
   ```

4. Create a new file called `extList.controller.js` in the `view` folder.

5. Declare the custom controller as follows:
   
   ```javascript
   sap.ui.controller("i2d.qm.qualityissue.confirm.apprefExt.view.extList", {});
   ```

6. Add the following piece of code to the `extList.controller.js` file:
   
   ```javascript
   sap.ui.controller("i2d.qm.qualityissue.confirm.apprefExt.view.extList", {
   doConfirm : function(e){
       var that = this;
       jQuery.sap.require("sap.ca.ui.dialog.factory");

       // provide your callback function, so that you can get informed if the enduser confirms or cancels the dialog
       var fnClose = function(oResult) {
           if (oResult) {
               console.log("isConfirmed:" + oResult.isConfirmed);
               if (oResult.sNote) {
                   console.log(oResult.sNote);
               }
           }
       }

       //open the confirmation dialog
       sap.ca.ui.dialog.confirmation.open({
           question : this.getView().getModel("i18n").getProperty("CONFIRM_QUESTION"),
           showNote : true,
           title : "Confirm",
           confirmButtonLabel : "Confirm"
       }, fnClose);
   }
   });
   ```

   The custom controller is an extension of the standard SAP UI5 controller. The methods of the custom controller are merged with the methods of the controller in the SAP-delivered standard application.

   For more information about the custom controller, see the [appendix](#).

   For more information, see the [Limitations](#) section.

7. In the dialog confirmation box, i18n is loaded as a model and its properties are accessed using the `getProperty()` method as shown below.

   ```javascript
   sap.ca.ui.dialog.confirmation.open({
       question : this.getView().getModel("i18n").getProperty("CONFIRM_QUESTION"),
   });
   ```
8. Add the following piece of code to the custom button in the footer of the Component.js file:

```javascript
"sap.ui.viewExtensions":{
    "i2d.qm.qualityissue.confirm.appref.view.S3":{
        "extFooter":{
            className: "sap.ui.core.Fragment",
            fragmentName: "i2d.qm.qualityissue.confirm.apprefExt.view.extFooter",
            type: "XML"
        }
    }
}
```

9. Add the following piece of code to the Customizing object for extending the SAP-delivered standard view controller with a custom controller:

```javascript
"sap.ui.controllerExtensions": {
    "i2d.qm.qualityissue.confirm.appref.view.S3":{
        controllerName : "i2d.qm.qualityissue.confirm.apprefExt.view.extList"
    }
}
```

**Use case: Using custom text**

This use case covers adding custom text for Department Responsible and Coordinator in the Information tab of the Q Issue Detail screen.

**Steps**

1. Create a new file called i18n.properties in the i18n folder.
2. Add the following properties and values to the i18n.properties files:

   ```
   COMPLAINT_QTY=Complaint Quantity
   REF_QTY=Reference Quantity
   RET_DELIVERY_UNIT=Return Delivery Unit
   RETURNED_ON=Returned On
   DEF_QTY_INTERNAL=Definite Quantity Internal
   DEF_QTY_EXTERNAL=Definite Quantity External
   DEPARTMENT_RESP=Department Responsible
   COORDINATOR=Coordinator
   TASK_ID=Task ID
   TASK_STATUS=Task Status
   TASK_TEXT=Task Text
   TASK_LIST=Task List
   RESPONSIBLE=Responsible
   CONFIRM=Confirm
   CONFIRM_QUESTION=Are you sure you want to confirm this task?
   DATE=Date
   ENTER_DATE=Enter the date
   CODE_GROUP=Code Group
   TASK_CODE=Task Code
   PLANNED_START=Planned Start
   PLANNED_FINISH=Planned Finish
   COMPLETED_ON=Completed On
   ```

3. Define i18n in the config object of Component.js as follows:

   ```javascript
   config : {
       "sap.ca.i18Nconfigs": {
         "bundleName":"i2d.qm.qualityissue.confirm.apprefExt.i18n.i18n"
       }
   }
   ```
Use case: Translation
If you want the custom application to run in a language not supported by SAP, then you need to create a new property file for that language called `i18n_<language_code>.properties` and place in the in the `i18n` folder.

Please note that the data (for example, customizing for order types) that comes from the backend needs to be translated in the backend systems.

Use case: Using custom oData Service URL
This use case covers adding a model for the custom field `Coordinator & Department Responsible` in the `Information` tab of the `Q Issue Detail` screen.

**Steps**

1. Add the following code to the `config` object of `Component.js`.

   ```javascript
   "sap.ca.serviceConfigs": [{
   name:"Z_I2D_QM_QUALITYISSUE_CONFIRM_SRV",
   serviceUrl:"/sap/opu/odata/sap/PS_MILESTONE_CONFIRM/",
   isDefault:true,
   mockedDataSource:jQuery.sap.getModulePath
   ("i2d.qm.qualityissue.confirm.apprefExt")="/model/metadata.xml"
   }
   ]
   ```

2. Configure the proxy for testing the custom application on a local Fiori Launchpad. `ZF2UIAPPREF002E` is the name of the custom application.

   ```javascript
   "sap.ca.serviceConfigs": [{
   name:"Z_I2D_QM_QUALITYISSUE_CONFIRM_SRV",
   serviceUrl:"/ZF2UIAPPREF002E/proxy.sap/opu/odata/sap/PS_MILESTONE_CONFIRM/",
   isDefault:true,
   mockedDataSource:jQuery.sap.getModulePath
   ("i2d.qm.qualityissue.confirm.apprefExt")="/model/metadata.xml"
   }
   ]
   ```

   Remove proxy configuration before uploading the custom application to server.

   Note that the `name` attribute is the key. In order to override the OData service URL of the SAP-delivered application, use the same `name` attribute as that of the SAP-delivered application and provide the custom OData service URL in the value for the attribute `serviceUrl`. The value of the `name` attribute in the SAP-delivered application can be found in the `Configuration.js` file of the SAP-delivered application.

Use case: Replacing an SAP-delivered Standard Application View with a Custom View
This use case covers replacing the `Q Issue Create Form` view with a custom view. The custom view replaces the `Description` field with a `Date` field. The view controller is replaced by a custom view controller. The custom view controller retrieves the `date` field when submitting an issue.

Since no extension points are present in the `IssueForm.view.xml` file, the only way to achieve the extension is by replacing the view.
Steps

1. Add the following configuration code to the customizing object in Component.js.

   ```javascript
   i2d.qm.qualityissue.confirm.appref.Component.extend("i2d.qm.qualityissue.confirm.apprefExt.Component", {
       metadata: {
           customizing: {
               "sap.ui.viewReplacements": {
                   newHomePage: "i2d.qm.qualityissue.confirm.apprefExt.view.newHomePage",
                   type: "XML"
               }
           }
       }
   });
   ```

2. Create a new file called `newHomePage.view.xml` in the `view` folder.
3. For more information about adding custom text in the custom view, see [Using custom text](#).

4. Define the contents of `newHomePage.view.xml` as follows:

```xml
<view xmlns="sap.m" xmlns:ui="sap.ca.ui" xmlns:core="sap.ui.core" xmlns:layout="sap.ui.layout">
  <page id="page" title="{i18n>CREATION_TITLE}" showNavButton="false">
    <content>
      <layout:Grid hSpacing="1" vSpacing="1">
        <layout:content>
          <Label text="{i18n>PICTURES}" id="attachmentLabel" textAlign="Right" width="100%">
            <layoutData>
              <layout:GridData span="L4 M4 S12"/>
            </layoutData>
          </Label>
          <ui:AddPicture id="addPicture" buttonPageType="Form" pictures="{pictures>/Pictures}" show="onItemPress" maxPictureNumber="6" maxPictureLimitReached="onLimitReached" />
          <ui:pictures>
            <ui:PictureItem source="{pictures>/Source}"/>
          </ui:pictures>
        </layout:content>
        <layout:Grid hSpacing="1" vSpacing="1">
          <layout:content>
            <Label text="{i18n>DESCRIPTION}" id="descriptionLabel" required="true" textAlign="Right" width="100%" padding-top="12px">
              <layoutData>
                <layout:GridData span="L4 M4 S12"/>
              </layoutData>
            </Label>
            <Input value="{creationModel>/Description}" maxLength="40" liveChange="onCheckStatusSubmit" id="descriptionInput">
              <layoutData>
                <layout:GridData span="L5 MS S12"/>
              </layoutData>
            </Input>
          </layout:content>
        </layout:Grid>
        <layout:Grid hSpacing="1" vSpacing="1">
          <layout:content>
            <Label text="{i18n>DATE}" id="dateLabel" textAlign="Right" width="100%">
              <layoutData>
                <layout:GridData span="L4 M4 S12"/>
              </layoutData>
            </Label>
            <DateTimeInput type="DateTime" placeholder="{i18n>ENTER_DATE}" maxLength="246" id="dateTimeInput">
              <layoutData>
                <layout:GridData span="L5 MS S12"/>
              </layoutData>
            </DateTimeInput>
          </layout:content>
        </layout:Grid>
        <layout:Grid hSpacing="1" vSpacing="1">
          <layout:content>
            <Label text="{i18n>DEFECT}" id="defectLabel" required="true" textAlign="Right" width="100%" padding-top="12px">
              <layoutData>
                <layout:GridData span="L4 M4 S12"/>
              </layoutData>
            </Label>
          </layout:content>
        </layout:Grid>
      </layout:Grid>
    </content>
  </page>
</view>
```

```xml
<core:View
  controllerName="i2d.qm.qualityissue.confirm.apprefExt.view.newHomePage"
  xmlns="sap.m"
  xmlns:ui="sap.ca.ui"
  xmlns:layout="sap.ui.layout"
  xmlns:core="sap.ui.core"
/>
```

6. The difference between the controllers in the SAP-delivered standard application and `newHomePage.controller.js` is that the new controller handles the addition of the `Date` field. Other functionalities remain the same. Add the following code to the controller:

```javascript
jQuery.sap.require("sap.ui.core.mvc.Controller");
jQuery.sap.require("sap.m.MessageToast");
jQuery.sap.require("sap.ca.scfld.md.controller.BaseDetailController");
jQuery.sap.require("sap.ca.ui.dialog.factory");
sap.ca.scfld.md.controller.BaseDetailController.extend("i2d.qm.qualityissue.confirm.apprefExt.view.newHomePage", {

/** Called by the UI5 runtime to init this controller */

onInit: function () {
  // Execute onInit for the base class BaseMasterController
  sap.ca.scfld.md.controller.BaseDetailController.prototype.onInit.call(this);

  // Get the application bundle
  this.resourceBundle = this.oApplicationFacade.getResourceBundle();
  // Listen for navigation
  this.isRoot = true;
  var self = this;
  var View = this.getView();

  //...
Use Cases for Extending the UI of SAP Fiori Apps

```javascript
this.oRouter.attachRouteMatched(function(oEvent) {
    if (oEvent.getParameter("name") === "fsIssueForm") {
        this.isRoot = false;
    }
}, this);

// Create the model
this.createModel();
this.getView().setModel(this.creationModel, "creationModel");

// Needed for addPicture component

// get the page
var page = this.getView().byId("page");
util.UiFactory.fillPageHeader(page, this.getView(), util.Title.IMAGE_GALLERY);

// listen to the remove event of the viewer
var bus = sap.ui.getCore().getEventBus();
bus.subscribe("PictureViewer", "RemovePicture", this.onRemovePicture, this);

// get the addPicture control
this.addPicture = this.getView().byId("addPicture");

// Model for addPicture component
this.pictureModel = new sap.ui.model.json.JSONModel({Pictures:[]});
this.getView().setModel(this.pictureModel, "picture");

/**
 * Called by the UI5 runtime to cleanup this controller
 */
onExit : function () {
    // destroy the control if needed
    if (this._defectSelectDialog) {
        this._defectSelectDialog.destroy();
        this._defectSelectDialog = null;
    }
    // destroy the control if needed
    if (this._categorySelectDialog) {
        this._categorySelectDialog.destroy();
        this._categorySelectDialog = null;
    }
},

/**
 * Called by submit button
 */
onSubmit : function () {
    // Handling the OK button
    var fnClose = jQuery.proxy(function(oResult) {
        if (oResult.isConfirmed) {
            // Call your post from there
            // clean the model
            this.cleanModel();
            // Navigate to S2
            this.oRouter.navTo("master");
            // Validation message
            var validationMessage = this.resourceBundle.getText("TXT_VALIDATION");
            sap.m.MessageToast.show(validationMessage, {
                duration: 100000
            });
        }
    }, this);

    // open the confirmation dialog
    sap.ca.ui.dialog.confirmation.open({
        question: this.resourceBundle.getText("SUBMIT_VALIDATION"),
        showNote: false,
        title: this.resourceBundle.getText("SUBMIT"),
        confirmButtonLabel: this.resourceBundle.getText("OK")
    }, fnClose);
},

/**
 * Select defects in a selectDialog control
 */
displayDefect : function (oEvent) {
```
// Create dialog if does not exist
if (!this._defectSelectDialog) {
    this._defectSelectDialog = sap.ui.xmlfragment('i2d.qm.qualityissue.confirm.appref.view.DefectSelectDialog', this);
    this._defectSelectDialog.setModel(oEvent.getSource().getModel('Z_I2D_QM_QUALITYISSUE_CONFIRM_SRV'));
    this._defectSelectDialog.setModel(oEvent.getSource().getModel('i18n'), 'i18n');
}
// Open the dialog
this._defectSelectDialog.open();

/**
 * Select an item in the selectDialog and close it
 */
closeDefectSelectDialog : function (oEvent) {
    //Retrieve the selected item and update the creation model
    var selectedItem = oEvent.getParameter('selectedItem');
    if (selectedItem) {
        this.creationModel.setProperty('/SelectedDefect', selectedItem.getTitle());
        this.onCheckStatusSubmit();
    }
},

/**
 * Display categories in a selectDialog control
 */
displayCategory : function (oEvent) {
    // Create dialog if does not exist
    if (!this._categorySelectDialog) {
        this._categorySelectDialog = sap.ui.xmlfragment('i2d.qm.qualityissue.confirm.appref.view.CategorySelectDialog', this);
        this._categorySelectDialog.setModel(oEvent.getSource().getModel('Z_I2D_QM_QUALITYISSUE_CONFIRM_SRV'));
        this._categorySelectDialog.setModel(oEvent.getSource().getModel('i18n'), 'i18n');
    }
    // Open the dialog
    this._categorySelectDialog.open();
},

/**
 * Select an item in the selectDialog and close it
 */
closeCategorySelectDialog : function (oEvent) {
    //Retrieve the selected item and update the creation model
    var selectedItem = oEvent.getParameter('selectedItem');
    if (selectedItem) {
        this.creationModel.setProperty('/SelectedCategory', selectedItem.getTitle());
    }
},

/**
 * Cancel the creation of issue and navigate to master/detail
 */
onCancel : function () {
    // Confirmation box only if one value has changed
    if (this.hasChanged()){
        // Handling the OK button
        var fnClose = jQuery.proxy(function(oResult) {
            if (oResult.isConfirmed) {
                this.cancelAction();
            }, this);
        }, this);
        //open the confirmation dialog
        sap.ca.ui.dialog.confirmation.open({
            question : this.resourceBundle.getText('SUBMIT_CANCEL'),
            showNote : false,
            title : this.resourceBundle.getText('CANCEL'),
            confirmButtonlabel : this.resourceBundle.getText('OK')
        }, fnClose);
    } else{
        this.cancelAction();
    }
},

/**
 * Reset the model and navigate to master
 */
cancelAction : function () {
    // reset of the model
}
this.cleanModel();
// Navigate to S2
this.oRouter.navTo("master");

/**
 * Check if the submit button should be available
 */
onCheckStatusSubmit : function (oEvent) {
    //Check properties size
    if (!this.creationModel.getProperty("/Description").length == 0 
        && !this.creationModel.getProperty("/SelectedDefect").length == 0 
        && !this.creationModel.getProperty("/Reference").length == 0 
        && !this.creationModel.getProperty("/SelectedCategory").length == 0) {
        //Enable the button when all mandatory fields are not empty
        this.creationModel.setProperty("/toggleSubmit", true);
    }
},

/**
 * Check if the user has entered at least one value
 */
hasChanged : function () {
    var status = false;
    //Check properties size
    if (!this.creationModel.getProperty("/Attachments").length == 0 
        || !this.creationModel.getProperty("/Date").length == 0 
        || !this.creationModel.getProperty("/Description").length == 0 
        || !this.creationModel.getProperty("/SelectedDefect").length == 0 
        || !this.creationModel.getProperty("/Reference").length == 0 
        || !this.creationModel.getProperty("/SelectedCategory").length == 0) {
        //Check if one the value has changed
        status = true;
    }
    return status;
},

/**
 * Create the model
 */
createModel : function () {
    //creating a model for binding data to send to server
    this.creationModel = new sap.ui.model.json.JSONModel({
        "Attachments": "",
        "Description": "",
        "SelectedDefect": "",
        "Reference": "",
        "SelectedCategory": "",
        "Date": "",
        "toggleSubmit": false
    });
},

/**
 * Clean the model
 */
cleanModel : function () {
    //reset the model to initial state
    this.pictureModel.setProperty("/Pictures", null);
    this.creationModel.setProperty("/Attachments", "");
    this.creationModel.setProperty("/Description", "");
    this.creationModel.setProperty("/SelectedDefect", "");
    this.creationModel.setProperty("/Reference", "");
    this.creationModel.setProperty("/SelectedCategory", "");
    this.creationModel.setProperty("/Date", "");
    this.creationModel.setProperty("/toggleSubmit", false);
},

/**
 * Needed for addpicture component : Handler for picture pressed action. Launches the PictureViewer
 * @param event{object} the event data
 */
onItemPress : function (event) {
    var selectedImage = event.mParameters.pictureItem._oImage;
    this.showGallery(selectedImage);
},

/**
 * Needed for addpicture component : Creates the model which is passed to the PictureViewer page after navigation
 * @param selectedImage{object} sap.m.Image - the image that should be shown first
 * @private
 */
Use Cases for Extending the UI of SAP Fiori Apps

showGallery : function (selectedImage) {
    // create model
    var images = [];
    var selectedImageIndex = 0;
    var gallery = this.addPicture_.getPictures();

    // loop through the pictures, add reference to sap.m.Images, get the index of image
    for (var i=0; i<gallery.length; i++) {
        images.push(gallery[i]._oImage);
        if (gallery[i]._oImage === selectedImage)
            selectedImageIndex = i;
    }

    /* Workaround because nav.toRoute does not handle complex object so far */
    this.oData = {
        galleryItems : images,
        selectedIndex : selectedImageIndex
    };

    /* END Workaround*/
    this.oRouter.navTo(this.isRoot ? "Viewer" : "fsViewer");

    /* Workaround because nav.toRoute does not handle complex object so far */
    // raise app event to send the context to the viewer
    sap.ui.getCore().getEventBus().publish("app", "RefreshContext", {
        context: this.oData
    });

    /* END Workaround*/
},

/**
* Needed for addpicture component : Event Handler for remove event sent from PictureViewer
* @param channel {string} the event channel
* @param event {string} the event name
* @param data {object} the event data
*/
onRemovePicture : function (channel, event, data) {
    // remove the picture from the control
    var gallery = this.addPicture;
    gallery.removePicture(data.pictureIndex);
},

/**
* Needed for addpicture component : Handler for limit reached
* @param event {object} the event data
*/
onLimitReached : function (event) {
    sap.ca.ui.message.showToast(
        this.resourceBundle.getText("LIMIT_MESSAGE")
    );
};

The custom controller extends BaseDetailController in the com.sap.ca.sclfd.md library as some useful functionality gets inherited from it. For more information, see the Appendix.

Use case: Adding a custom view (Q Issue Task Detail)
This use case covers adding a custom view, which displays the details of a task item for a quality issue.

Prerequisite: Use case Adding a custom list in S3 screen is implemented.
Custom application

Steps

1. Define a new view called newView.view.xml as newView.view.xml in the view folder and define a new controller called newView.controller.js for the same.

   ```xml
   <core:View xmlns="sap.m" xmlns:ca="sap.ca.ui" xmlns:layout="sap.ui.layout"
   xmlns:core="sap.ui.core" xmlns:form="sap.ui.layout.form">
   <controllerName="i2d.qm.qualityissue.confirm.apprefExt.view.newView">
   <Page id="page" navButtonPress="goToDetailPage">
   <content>
   <form:SimpleForm id="TASK_DETAILS_FORM" minWidth="1024">
   <form:content>
   <form:content>
   <label text="{i18n:TASK_ID}" />
   </form:content>
   </form:content>
   </form:SimpleForm>
   </content>
   </Page>
   </controllerName>
   </core:View>
   ```

Note: For more information about adding custom text in the fragment definition, see Using custom text.

Note: For more information about adding custom field data in the fragment definition, see Using custom oData Service.

2. Define the contents of newView.view.xml as follows:
Use Cases for Extending the UI of SAP Fiori Apps

Use case: Navigating to the custom view
This covers navigating to the custom view when a task item for a quality issue is selected.

Prerequisite:
Section Adding a custom view (Q Issue Task detail) is implemented.

For more information, see https://sapui5.netweaver.ondemand.com/sdk/#docs/guide/Navigation.html

Custom application: Click on a Task Item

Note: For more information about navigating to the newly added view, see Navigating to the custom view.

Custom application: Navigation to Custom View
Use Cases for Extending the UI of SAP Fiori Apps

Steps

1. Configure the route in the `routing` object of `component.js` as follows:

   ```javascript
   routing : {
     "routes" : {
       //Defined in routes of SAP delivered standard application
       "masterDetail" : {
         //Defined in routes of SAP delivered standard application
         "subroutes" : {
           "master" : {
             "subroutes" : {
               //Navigation details added
               "taskDetail" : {
                 "pattern": "detail/ReportedIssueCollection/taskDetail/{contextPath}",
                 "viewPath": "i2d.qm.qualityissue.confirm.apprefExt.view",
                 "targetAggregation": "detailPages",
                 "view": "newView",
                 "viewLevel": 2,
               }
             }
           }
         }
       }
     }
   }
   
   Note: The following is a view of the routing definition in `ComponentBase.js` of the `com.sap.ca.scfld.md` library:

   ```javascript
   "masterDetail" : {
     "view" : "MainSplitContainer",
     "name" : "masterDetail",
     "viewPath" : "sap.ca.scfld.md.view",
     "targetControl" : " FioriContent", // This is the control in which the new view will be placed
     "targetAggregation" : "pages", // This is the aggregation in which the new view will be placed
     "subroutes" : {
       "master" : {
         "pattern" : "", // will be the uri and from has to be provided in the data
         "view" : "MyView",
         "viewLevel" : 1,
         "subroutes" : {
           "detail" : {
             "pattern": "detail/{contextPath}",
             "view" : "MyView",
             "viewLevel" : 1
           },
           "noData" : {
             "pattern": "noData/{viewTitle}/(languageKey)",
             "viewPath": "sap.ca.scfld.md.view",
             "view": "MyView",
             "viewLevel" : 1
           }
         }
       }
     }
   },
   ```

2. Define a controller for the fragment `extList.fragment.xml`. For more information, see [Adding a custom list in S3 screen](#).

3. Navigate to the new view in the `handleTaskItemPress` event handler in `extList.controller.js` as follows:

   ```javascript
   handleTaskItemPress : function(e){
     //Navigating to a new view.
   }
   ```
Use Cases for Extending the UI of SAP Fiori Apps

```javascript
this.oRouter.navTo("taskDetail", {
  contextPath : e.getSource().getBindingContext().sPath.substr(1),
  TaskID : e.getSource().getCells()[1].getText(),
});
}

4. Add the routing configuration in `newView.controller.js` as shown in the code below. For more information, see Adding a custom view [Q Issue Task detail].

   ```javascript
   sap.ca.scfld.md.controller.BaseDetailController.extend("i2d.qm.qualityissue.confirm.apprefExt.view.newView", {
     onInit : function(){
       var view = this.getView();
       this.oRouter.attachRouteMatched(function(oEvent) {
         if (oEvent.getParameter("name") === "taskDetail") {
           var context = new sap.ui.model.Context(view.getModel(), '/' +
             oEvent.getParameter("arguments").contextPath);
             view.setBindingContext(context);
         }
       }, this); 
     };
   })
   ```

The custom controller extends `BaseDetailController` in the `com.sap.ca.scfld.md` library as some useful functionality gets inherited from it. For more information, see the Appendix.

Use case: Manipulating the model of the SAP standard application at predefined hooks in controller

**Requirement**
This use case covers changing the order of images in the Attachment tab in the Detail screen.

**Prerequisite:** `S3.controller.js` (Q Issue Detail) in the SAP-delivered standard application has an explicit hook as shown below.

In the Attachments tab of the Q Issue detail screen of the SAP-delivered standard application, there are five images as shown below.
Use Cases for Extending the UI of SAP Fiori Apps

SAP-delivered standard application

Custom application: The last image is changed to the second last image.

Steps

1. In Component.js add the following piece of code to the customizing object:
   
   ```javascript
   "sap.ui.controllerExtensions": {
     "I2D.QM.QUALITYISSUE.CONFIRM.APREF.VIEW.S3": {
       controllerName: "I2D.QM.QUALITYISSUE.CONFIRM.APREFExt.VIEW.EXTLIST"
     }
   }
   ```

2. Create a new controller called extList.controller.js in the view folder.

3. Add the following piece of code to the extList.controller.js file. It sets the last image to be the same as the second-last image in the model.

   ```javascript
   // Unique Name (HOOK) defined in the SAP delivered standard application
   onDataReceived: function(oDataFromS3) {
     if (oDataFromS3) {
       // Model changed via hook
       oDataFromS3.oData.Pictures[4]=oDataFromS3.oData.Pictures[3];
     }
   } // no need to return anything as in this example the original object is modified
   ```

   For more information, see the appendix.

Uploading a Custom Application to the Server

Steps

1. Remove the proxy configuration from the oData Service URL as defined in Using custom oData Service.

   ```javascript
   "sap.ca.serviceConfigs": [{
     name: "Z_I2D_QM.QUALITYISSUE_CONFIRM_SRV",
     serviceUrl: "/sap/opu/odata/sap/PS_MILESTONE_CONFIRM/"
   }]
   ```
Use Cases for Extending the UI of SAP Fiori Apps

```javascript
mockedDataSource: jQuery.sap.getModulePath
("i2d.qm.qualityissue.confirm.apprefExt") + "/model/metadata.xml"
}
```

2. Remove the proxy configuration from `/WEB-INF/web.xml`. For more information, see **Use a SimpleProxyServlet for Testing to Avoid Cross-domain Requests** in the Developer Guide for UI [https://websmp102.sap-ag.de/~sapidb/011000358700000944892012E](https://websmp102.sap-ag.de/~sapidb/011000358700000944892012E).

   ```xml
   <servlet> <servlet-name>SimpleProxyServlet</servlet-name> <servlet-class>com.sap.ui5.proxy.SimpleProxyServlet</servlet-class> </servlet>
   <servlet-mapping> <servlet-name>SimpleProxyServlet</servlet-name> <url-pattern>/proxy/*</url-pattern> </servlet-mapping>
   ```

3. Change the path to SAP standard application in `Component.js`.

   ```javascript
   jQuery.sap.registerModulePath("i2d.qm.qualityissue.confirm.appref", "/SAP-Namespace:/ZF2UIAPPREF002");
   ```

   Note: For more information, see **Sharing the SAPUI5 Application Project with the SAPUI5 Repository** in the Developer Guide for UI [https://websmp102.sap-ag.de/~sapidb/011000358700000944892012E](https://websmp102.sap-ag.de/~sapidb/011000358700000944892012E) for uploading custom applications to the server.

**Known Limitations of SAP UI5 1.16.4**

The i18n resource cannot be accessed directly

The strings in the i18n resource bundle cannot be accessed using `resourceBundle.getProperty()`. It is recommended to access the strings in i18n model as shown below:

```javascript
this.getView().getModel("i18n").getProperty("CONFIRM_QUESTION")
```

It is recommended to use this approach for the following use cases:

- Where a button is added to the footer
- Where a title in the header is changed

**Appendix**

**Recommendations on controller extensions**

**sap.ca.scfld.md.controller.BaseDetailController**

Library: com.sap.ca.scfld.md

`BaseDetailController` provides the following functionalities for its extended controllers:

1. Initializations of the router object. You can access the router object using `this.oRouter`. 
Use Cases for Extending the UI of SAP Fiori Apps

2. Setting up the application’s default model. You can access the default model using `this.getModel()`.

We recommend that you use a controller that extends `sap.ca.scfld.md.controller.BaseDetailController` for the following use cases:

- Where you create a new view
- Where you replace an SAP view with a custom view

We recommend that you use a controller of type `sap.ui.controller` for the following use cases:

- Where you want to extend the SAP controller methods (by hooks or by adding customer methods)

Overriding functions in classes other than controller classes

Please note while overriding such functions is technically feasible, it is not guaranteed to be upgrade-stable. This means that after an SAP upgrade or fix, you may have to adjust the coding yourself. Only extensions performed using controller hooks are guaranteed to be stable even after upgrades.

There might be use cases where an SAP-delivered Standard formatter functions in `utils` folder needs to be replaced with a customer formatter.

For example, let’s say the following is the code in SAP-delivered Standard formatter class and in the custom formatter class you want to override the behavior of “only” the function `formatIssueCode()`

```javascript
jQuery.sap.declare("cus.sd.sofulfil.monitor.utils.Formatter");

cus.sd.sofulfil.monitor.utils.Formatter = {
    formatIssueCode : function(issueCode) {
        return "Hello SAP";
    },
};
```

Below are the steps you need to follow

1. Insert the following piece of code in the Component.js file of the extension project
   ```javascript
   jQuery.sap.require("cust.sd.sofulfil.monitor.utils.Formatter");
   ```

2. Create a new `Formatter.js` file in the `utils` folder of the extension project and insert the following lines of code
   ```javascript
   jQuery.sap.require("sap.sd.sofulfil.monitor.utils.Formatter");
   jQuery.sap.declare("cust.sd.sofulfil.monitor.utils.Formatter");
   ```
Use Cases for Extending the UI of SAP Fiori Apps

sap.sd.sofulfil.monitor.utils.Formatter.formatIssueCode = function(issueCode) {
    return "Hello Customer";
};

By doing as mentioned above,

- Functions (other than formatIssueCode()) in the SAP Formatter class continue to remain with the same implementation as that of SAP-delivered Standard code
- the SAP-delivered Standard views using the formatIssueCode() will reflect the custom code without changing the SAP-delivered Standard views
- The fixes/enhancements delivered by SAP in SAP-delivered formatter functions (other than the ones you have overridden) will be reflected in your application

Reusing the original implementation of a function in an extended controller

There might be use cases where you want to retain the SAP-delivered implementation, not override the SAP-delivered implementation and then your implementation for a given function.

For example, in the below SAP-delivered standard controller, you want to create a controller that overrides the doSomething() function but call the SAP’s original implementation of doSomething() before calling your code in the custom doSomething()

```javascript
sap.ui.controller("samples.components.ext.sap.Main", {
    onInit : function () {
        console.log("samples.components.ext.sap.Main - onInit");
    },
    doSomething: function() {
        alert("this is an original standard action");
    },
    doSomeStandardAction: function() {
        alert("this is another original standard action");
    }
});
```

Below in how you can do: by using standard JavaScript mechanisms to copy the "doSomething" method into a local variable in the custom Controller and override "doSomething" as described below.

```javascript
sap.ui.controller("samples.components.ext.customer.CustomMain", {
    onInit : function () {
        console.log("samples.components.ext.customer.CustomMain - onInit");
        // save original doSomething as super and overwrite it locally
```
```javascript
this.fnSuper_doSomething = samples.components.ext.sap.Main.prototype.doSomething;

doSomething: function() {
    // call original doSomething as super
    this.fnSuper_doSomething();

    // then do custom things
    alert("this is a customer action");
},

doSomeCustomAction: function() {
    alert("this is another customer action");
}
});
```

**Useful References**

An example SAP UI5 application built using XML views can be found here: [https://sapui5.netweaver.ondemand.com/sdk/test-resources/sap/m/demokit/poa/index.html](https://sapui5.netweaver.ondemand.com/sdk/test-resources/sap/m/demokit/poa/index.html)

Documentation on building SAP UI5 application using XML view type can be found at [https://openui5.hana.ondemand.com/docs/guide/MVC.html](https://openui5.hana.ondemand.com/docs/guide/MVC.html)