Links

Thursday, February 18, 2021 6:00 PM

https://www.123techguru.com/courses/concept-and-annotations-in-fiori-elements/	

https://sapui5.hana.ondemand.com/sdk#/topic/03265b0408e2432c9571d6b3feb6b1fd	Official documentation
https://sapui5.hana.ondemand.com/#/api/sap.suite.ui.generic.template.ListReport.extensionAPI	List report extension API

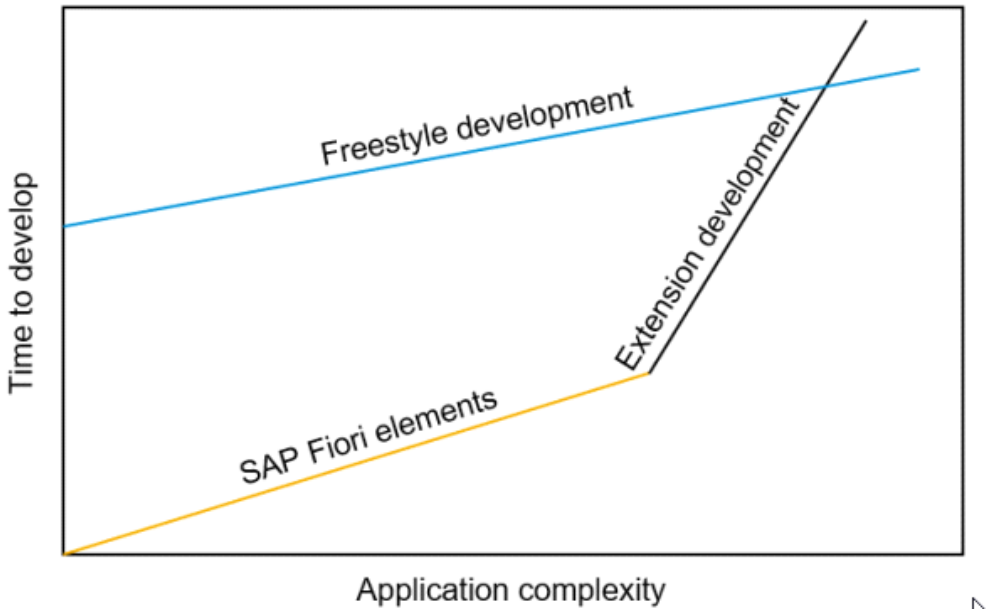
https://blogs.sap.com/2020/05/19/why-choose-sap-fiori-elements-for-your-development/	Why Choose SAP Fiori elements for your development
https://blogs.sap.com/2020/06/12/extending-sap-fiori-elements-applications-what-you-need-to-know/	Extending SAP Fiori elements Applications – What you need to know
https://blogs.sap.com/2020/05/11/fast-track-to-sap-fiori-elements-how-i-got-started/	Fast track to SAP Fiori elements – how I got started
https://blogs.sap.com/2019/12/16/adaptation-project-get-to-know-how-to-extend-a-fiori-elements-app/	Adaptation Project: Get to know how to extend a Fiori Elements app
https://blogs.sap.com/2019/08/15/extending-sap-fiori-elements-apps/	Extending SAP Fiori elements apps
https://d.dam.sap.com/a/21EPJi8/SAP%20Fiori%20elements%20Usage%20Guide%20-%20August%202019.pdf	When to use SAP Fiori elements - USAGE guide
https://blogs.sap.com/2019/12/06/customize-the-list-report-application-generated-by-fiori-elements-to-look-like-a-worklist-application./	Customize the list report application generated by FIORI elements to look like a worklist application.

To address these grey areas, SAP Fiori elements allow developer to extend the application. SAP Fiori elements application extensions are used in unavoidable situations where annotated approach is not able to address the requirement. This is majorly because the extension code belong to the application developers and will need to own the same across design changes or version upgrades,..etc

From <https://blogs.sap.com/2020/06/12/extending-sap-fiori-elements-applications-what-you-need-to-know/>

Use cases	Corresponding SAP Fiori elements Floor Plans
Table / Grid / List	List Report or Worklist
Detail page with CRUD operations	Object Page
Data analysis	Analytical List Page
Dashboard like capabilities	Overview Page

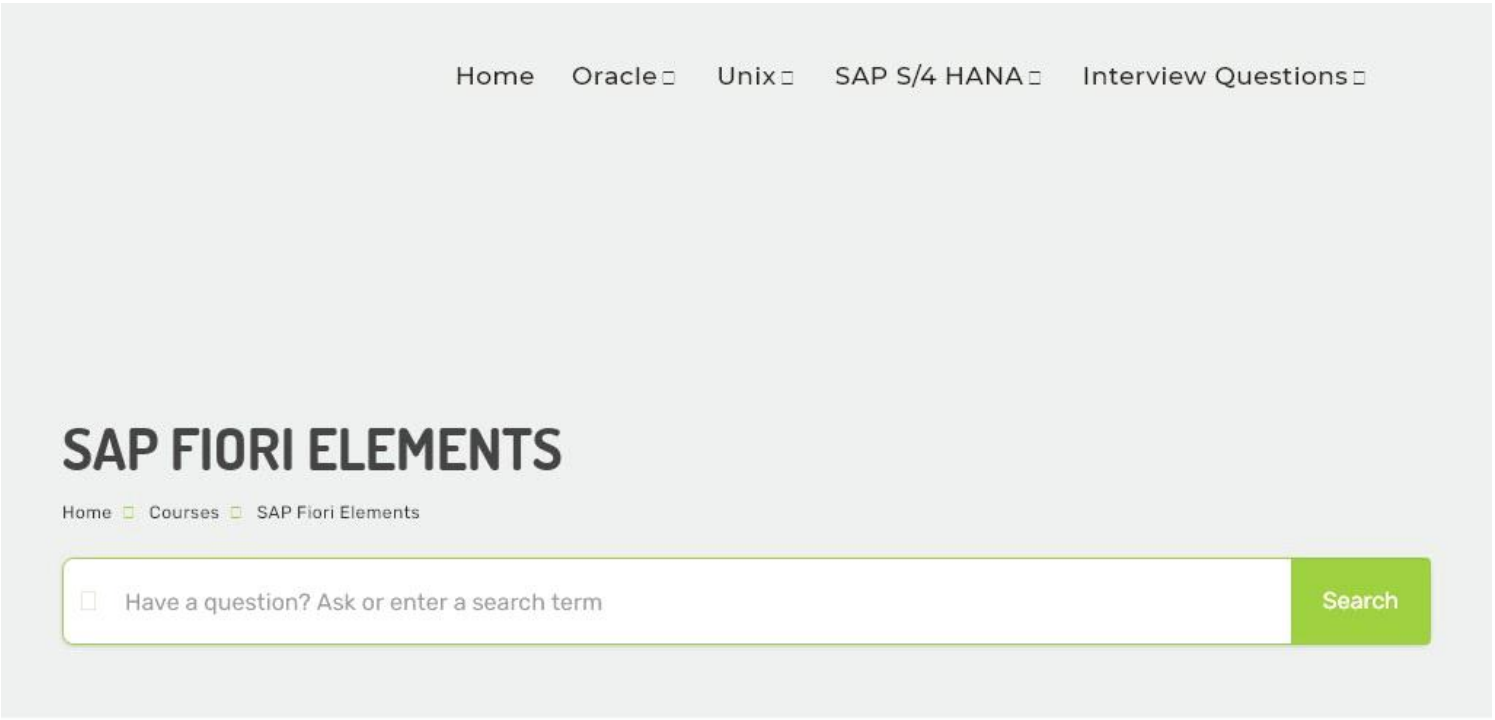
From <https://blogs.sap.com/2020/05/19/why-choose-sap-fiori-elements-for-your-development/>

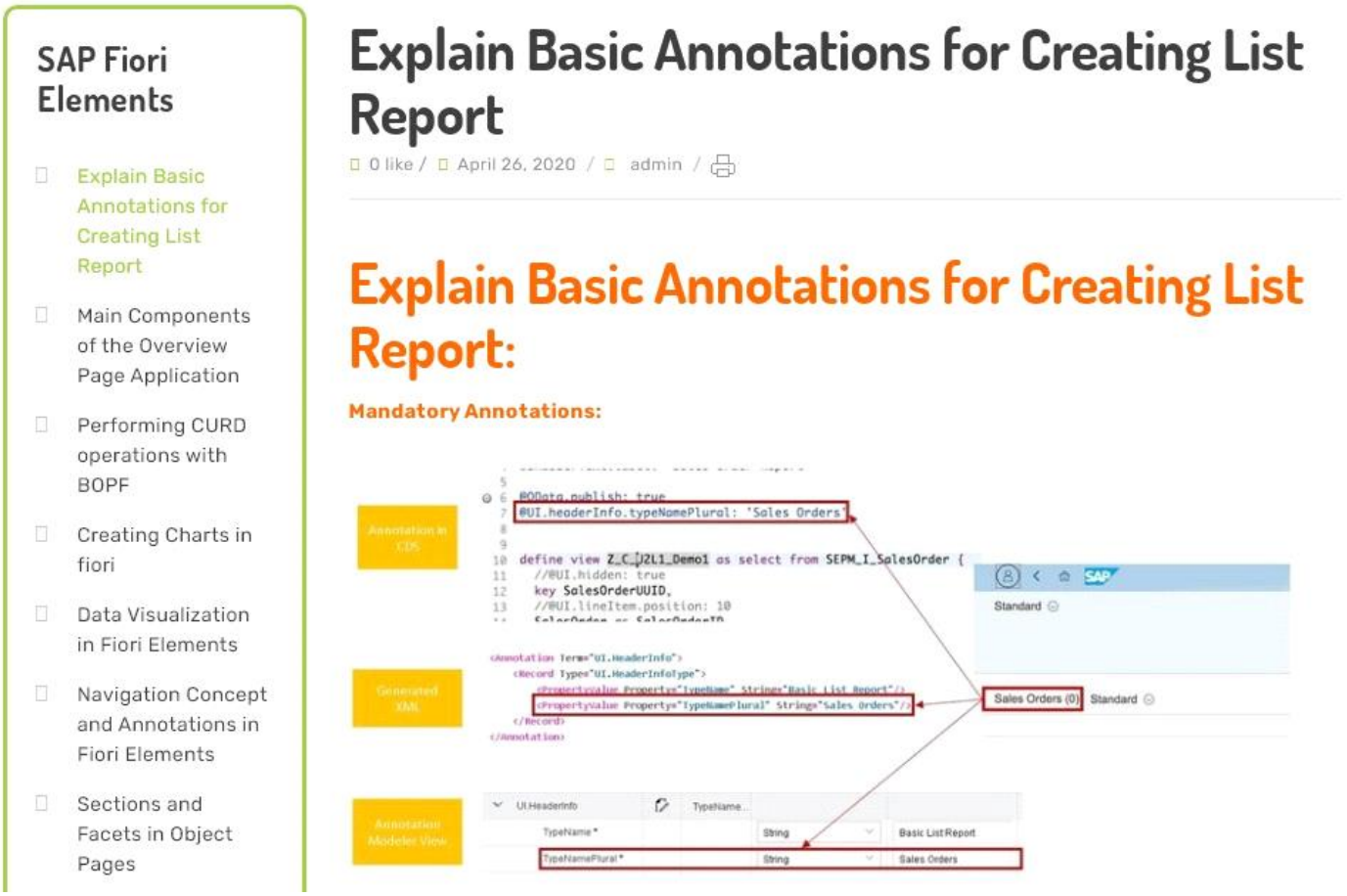

Freestyle development
Extension development
SAP Fiori elements
Time to develop
Application complexity

Explain Basic Annotations For Creating List Report - 123techguru.com | April 26, 2020

Thursday, February 18, 2021 6:02 PM

Clipped from: https://www.123techguru.com/courses/explain-basic-annotations-for-creating-list-report/





The following are typical mandatory annotations:

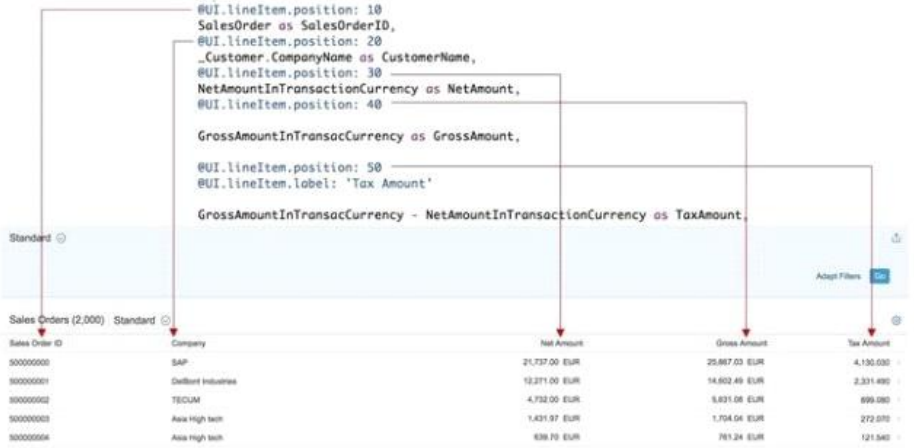
@OData.publish:true

Auto exposures the CDS view as OData Service.

@UI.headerInfo.typeNamePlural

Determines the text displayed on the up-left corner of the list report.

Adding Columns to Report



Adding Columns to Report

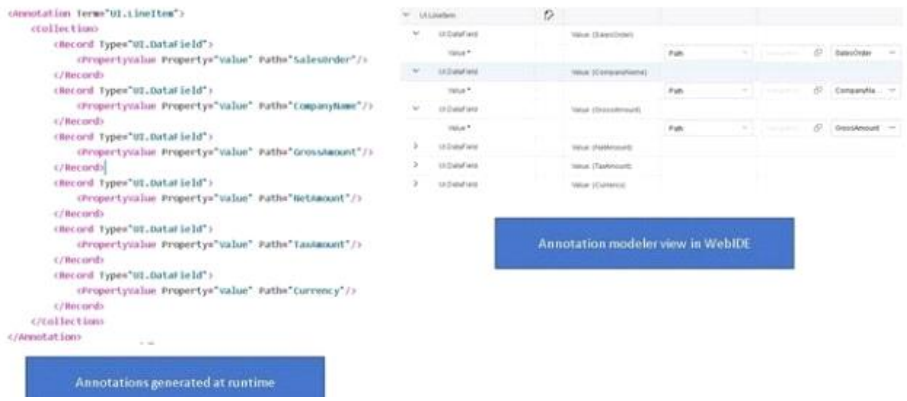
Annotation **@UI.lineItem** is used to set default columns for list report.

The Data types for **@UI.lineItem** is a collection of Data Fields, but in most cases only one **@UI.lineItem** exist for a field.

The mandatory property is position, which determines the sequence of columns. The label for each field comes from field label of data element for a fields.

You can also add calculated field to list report. Since the fields do not reference a data element, you need to either assign it a label by using **@UI.lineItem.lable**, or convert the data type to a data element by using function cast.

Add Columns to Report-XML and Modeler:



- Explaining SAP User Experience Tools and Technologies
- SAP User Experience Strategy Fiori
- Create a List Report using CDS with Annotation
- Overview of the Analytical List Page

Category

- SQL TUTORIAL
- PL SQL TUTORIAL
- Oracle > SQL*Loader
- Oracle > SQL Posts
- Oracle Posts
- ORACLE FORMS TUTORIAL
- Oracle Apps Tutorial
- Oracle > OAF
- Oracle > Oracle Workflow
- Unix > Unix Tutorial
- Unix > Shell Scripting
- UNIX
- JAVA
- SQL INTERVIEW QUESTIONS
- PL SQL INTERVIEW QUESTIONS
- ORACLE FORMS INTERVIEW QUESTIONS
- Oracle Apps Interview Questions
- OAF Interview Questions

The OData annotation generated as XML format looks like the left part of the figure. You can also open it using annotation modeler in WebIDE.

Semantic Information for Amount/Quantity fields

Add semantic information for Amount/Quantity fields

Information about adding semantic Information for Amount/Quantity fields:

- For amount and quantity fields, you should point out where to find its currency/unit fields.
- Fields selecting from DDIC, or other CDS entities may already have this information. If it is defined correctly, the currency/unit will be displayed with the amount/quantity fields.

If it's not displayed correctly, you should add semantic information in your CDS view, using

- @semantics.currencycode/@semantics.unitofmeasure to annotate for currency/unit field.
- @semantics.amount.currencycode/@semantics.quantity.unitofmeasure to annotate amount/quantity field and tell them which field has the information of currencyCode/ unit of measure.

You should always annotate semantics information for calculated fields you declared in your CDS view.

Options for Adjusting the Display of Columns:

Importance of Columns:

Importance of Columns

Controls used in list report is designed for all clients.

Controls used in list report is designed for all clients.

Different clients have different screen width, the fields displayed in a Smart Phone should less then Desktop Browser.

Using **@UI.lineitem.importance** to determine in which clients the field should display:

- #HIGH: Default value, display in all clients
- #MEDIUM: Only display in desktop browser or tablet
- #LOW: Only display in desktop browser

Importance of Columns-XML and Modeler

Importance of Columns-XML and Modeler

The OData annotation generated as XML format looks like the upper part of the figure. You can also open it using annotation modeler in WebIDE.

Hiding Fields

Hiding Fields

Some fields are not suitable for displaying on the report. You can use @UI.hidden to hide them in the table settings dialog.

Please write us here for any queries and concerns.

Was this helpful?

Yes No

Related Articles

- Overview of the Analytical List Page
- Main Components of the Overview Page Application
- Performing CURD operations with BOPF
- Creating Charts in fiori
- Data Visualization in Fiori Elements
- Navigation Concept and Annotations in Fiori Elements
- Sections and Facets in Object Pages

Fast track to SAP Fiori elements – how I got started | SAP Blogs

Wednesday, February 24, 2021 5:04 AM

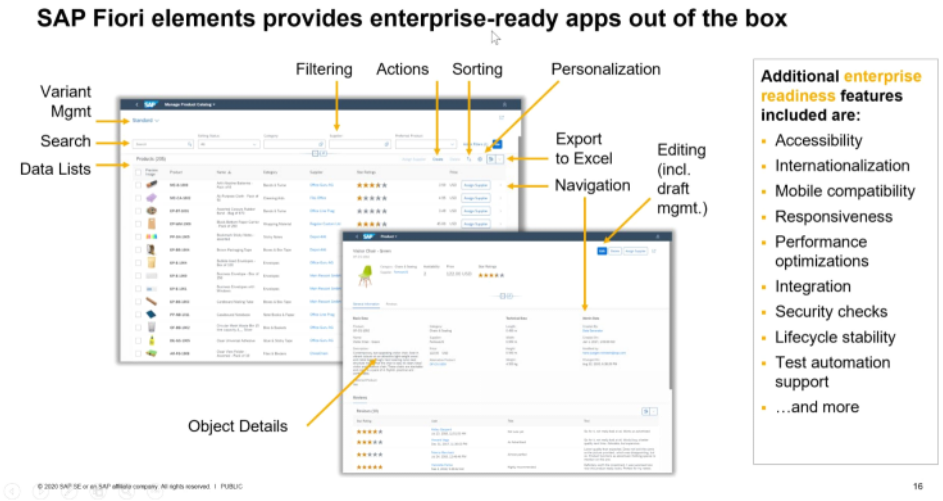
Clipped from: https://blogs.sap.com/2020/05/11/fast-track-to-sap-fiori-elements-how-i-got-started/

Last updated on 9th of December 2020

Have you heard about SAP Fiori elements and how it is supposed to really accelerate development of enterprise-ready web applications with a timeless architecture? I recently started on a journey to learn about SAP Fiori elements and if you are also interested this blog should help you to get on a fast track to ramp up on SAP Fiori elements.

1. What exactly is SAP Fiori elements?

SAP Fiori elements is a **library of common floorplans based on SAPUI5 that allows developers to create SAP Fiori applications with very little – in some cases even no – additional UI code. SAP Fiori elements generates an SAP Fiori app from an existing OData service** with additional metadata using **annotations** that define attributes and relationships of the data. ~80% of the common UI patterns in the enterprise application space can be covered by the supported **page types** (aka floorplans).



Using SAP Fiori elements boosts your developer productivity, ensures UX consistency and significantly reduces your apps' total cost of ownership (TCO) while automatically keeping it up to date with the latest Fiori design guidelines. Essentially the developer selects the relevant floorplan and adds semantic and structural data using metadata annotations. The framework then generates the application screen applying the latest Fiori design guidelines.

I really liked this video series from engineers for engineers to get a good

grip on what SAP Fiori elements actually is, but there are also other sources, e.g. Introduction section of Usage guide, Introduction to SAP Fiori Elements in the Fiori Design guidelines or Introduction in SAPUI5 documentation.

2. Can I build my app leveraging SAP Fiori elements?

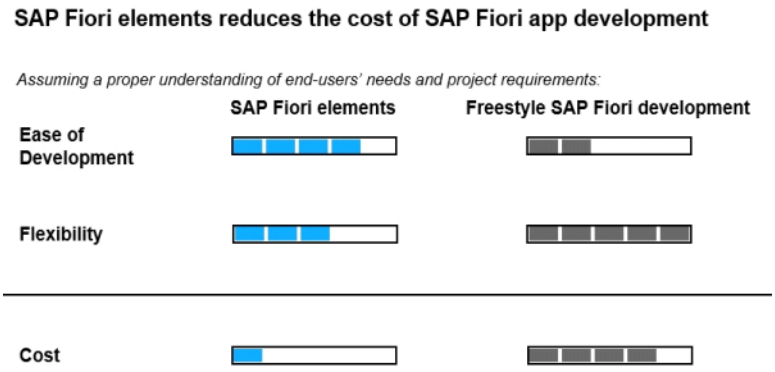
Checking if the app you want to build can be realized leveraging SAP Fiori elements is actually pretty simple – just answer these questions:

1. Are you building a web application (i.e. an application running in a browser, not a native app) based on OData services?
2. Does your target design follow the principle of list + drill-down to detail (=object page) where a list can be a list report, a worklist or an analytical list page? Optionally with a navigation via an overview page?

If your answer was YES to all of these, it is definitely worth your time to take a thorough look at SAP Fiori elements!

3. Benefits and limitations of SAP Fiori elements based apps compared to freestyle apps

In a nutshell, using SAP Fiori elements allows you to lower your costs in terms of development and maintenance efforts while taking care of UX consistency in exchange for full flexibility. Let me emphasize the 'full' here: There is still a good degree of flexibility available if you build an SAP Fiori elements-based app by leveraging our flexible programming model, which allows you to include SAPUI5 freestyle coding into SAP Fiori elements based apps.

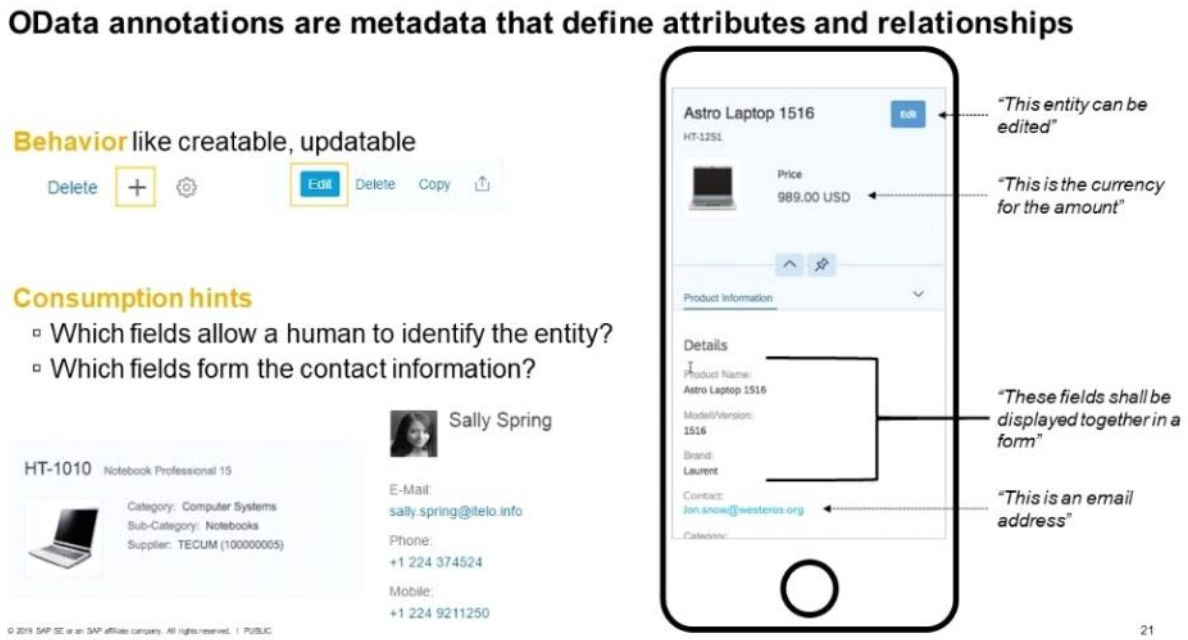


4. A short excursion on annotations

Annotations play a vital role for SAP Fiori elements. I heard about semantic annotations, about capability or 'UI' annotations or behavior. A bit of a jungle at first. What helped me was to look at examples of what they do, i.e. describing:

- the semantics of a field, e.g. 'this is a phone number'
- How fields are related to each other, e.g. 'this is the unit of measure for this value'
- Behaviors of a field, e.g. whether it is editable or read-only

- Groupings of fields, i.e. which set of fields should be displayed together



5. How to get hands-on experience?

Now it's time to try it out. Let's get our hands dirty…

A bit hidden in the depth of the UI5 documentation you can find instructions on How to build an SAP Fiori elements based app using WebIDE
Note that the SAP Business Application Studio has just been released. It is the next generation of the WebIDE. For details please refer to this blog on its release.

Do try out the developer tutorial for SAP Fiori tools (see below)!

6. Which features does SAP Fiori elements support?

Check the SAP Fiori elements feature map to find out if the feature you are looking for is supported. And if it is not, maybe there is a valid alternative that is supported by SAP Fiori elements? Keep in mind that enhancing an SAP Fiori elements app with custom SAPUI5 coding provides great flexibility, but increases the total cost of ownership.

7. Is there any tooling that facilitates SAP Fiori elements app creation?

SAP Fiori tools further eases & accelerates the development of SAP Fiori elements based apps. This collection of productivity tools is available for **SAP Business Application Studio** as well as for **Visual Studio Code** (via the VSC marketplace).

SAP Fiori tools uses **wizard-style** approaches, e.g. to generate the app based on your OData service and the selected template ensuring a **consistent** app structure. It also provides **step-by-step** development instructions via the guided help module. Furthermore it **reduces effort** and

skill level required for creating annotations thanks to the code completion module (aka LSP support). Last but not least it improves **code consistency** and **simplifies** app maintenance.

8. Latest news: SAP Fiori elements now support OData V4

Until recently SAP Fiori elements supported the OData v2 protocol, now we also **OData v4**. Check the announcement for more details.

There is also a great developer tutorial that let's you try the new functionality based on an OData V4 service built on SAP Cloud Application Programming model

As I learn more about SAP Fiori elements, I'll continue to share my findings with you.

Why Choose SAP Fiori elements for your development | SAP Blogs

Wednesday, February 24, 2021 5:13 AM

Clipped from: https://blogs.sap.com/2020/05/19/why-choose-sap-fiori-elements-for-your-development/

If I try to list down most commonly used or developed UI elements, I may think of the most repetitive ones, such as:

- Table/grid or a list displaying all the data
- Then I want my users to navigate to the details of these line items.
- I want my users to perform some actions like create, edit or delete data.
- I want my users to analyse the data.
- I want a dashboard for my users.

If you have any other idea of repeating UI elements or use case, please comment in the comment section

Fortunately, SAP Fiori elements provide all these UI elements and more, as their different floorplans, using which you can render your UI without writing a single line of UI code:

Use cases	**Corresponding SAP Fiori elements Floor Plans**
Table / Grid / List	List Report or Worklist
Detail page with CRUD operations	Object Page
Data analysis	Analytical List Page
Dashboard like capabilities	Overview Page

Let me assure you, SAP Fiori elements do not only support happy flows or hello world use cases, but it supports really exhaustive features and use-cases. Please take a look at the SAP Fiori elements Feature Map which lists all the features supported by SAP Fiori elements. It's not a drag and drop tool which renders your UI, rather it's one of the most powerful and exhaustive metadata-driven UI frameworks available today.

Now with this introduction, let me list down some other benefits of using SAP Fiori elements over conventional code-driven development for custom build applications.

Best Practices

SAP Fiori elements clubs the UX best practices from design and implementation best practices from engineering to provides a seamless and delightful experience for both i.e. the application developers and the business users.

So with SAP Fiori elements, you don't have to worry about where a particular action button should be placed or how your UI code should be structured or about how navigation between two application should be handled etc.

All these best practices and specification are forever-changing like for example in 2019 SAP also introduced Fiori 3 succeeding Fiori 2. Applications built on top of SAP Fiori Elements get these upgrades out of the box without any changes in the application implementation but just by upgrading to the right SAP UI5 version. In the case of custom build applications, the application development team is responsible for adhering to these best practices.

Enterprise Readiness

Non-functional requirements like performance, accessibility, security, responsiveness, adaptability etc. may often be very tricky and will consume more man-hours than the functional requirements itself. For custom build applications, the application development team is responsible for adhering to these non-functional requirements. With changing times, topics like Security and accessibility are no longer just a good to have a feature but also a binding to the applications, for example, the European Accessibility Act.

SAP Fiori elements take care of all these non-functional specifications so

your implementation team can concentrate on a lot more complicated busine ss logic.

UX Consistency

Maintaining UX consistency between different applications and cross-lob, in particular, could be

a very exhaustive process and might require very strict processes and D-Gates.

Failure to achieve a certain degree of UX consistency may not only result in

high user training expenditure, but may also result in lower productivity fro m business users.

All of the application developed with SAP Fiori elements will always be as

consistent as possible, without any effort from the application's engineering or design team.

Rapid Development

As we know SAP Fiori elements is a metadata-driven UI framework, which renders your UI based on your application manifest, service metadata and annotations, rather than application developers writing and maintaining thousands and Kilos line of code for SAP UI5 custom build applications. This development is even escalated more with Web IDE.

So with SAP Fiori elements, you can now productise an application not in months or years but in days and weeks.

Extensibility

SAP Fiori Elements is not rigid, rather it's very flexible and extensible at every layer. It supports application developer extensions for an application developer to incorporate very specific business demands. Fiori Elements supports WYSIWYG based extensibility via the SAP Visual Editor and Key user for customers and vendors. Even It supports personalisation for business users.

Scalability

Let's say you have developed a set of different application and business came back with some suggestions and changes. With SAP Fiori elements, It's quite effortless and quick to incorporate these feedbacks and changes compared to custom-built applications.

Maintenance and Support

Regressions and conflict resolution after upgrades are things of past with SAP Fiori Elements. As since you don't or have very less application-specific code to develop, hence you have very little or no effort for the maintenance.

With the Diagnostic tool, It is as seamless to seek support for SAP Fiori Elements. It collects all the required information which will be useful for the SAP support team to fix your issue. Hence reduce the multiple ping pong between SAP Fiori elements support and application team.

How to get your hands dirty with SAP Fiori elements

Now since you know, that SAP Fiori elements can act as a wormhole to by-pass the implementation effort for many essential and vital requirements like enterprise readiness, UX consistency and other best practices and also help you to productise your application in a much shorter period compared to UI5 freestyle application. You might be interested to dig further into SAP Fiori Elements. I think below links could help you as much as it has helped me

Developing Apps with SAP Fiori Elements Official Documentation

SAP UX Engineering YouTube channel

SAP Fiori Elements Overview Page (OVP) : what and hows

SAP Fiori Elements Analytical List Page (ALP) : what and hows

Hope it helps !!

I often write my blogs under Fiori Tag, Please follow it to get the latest updates from me.

Feedbacks and comments are always welcomed

stay safe !!

UI5ers Buzz #55: Adaptation Project – your one stop tool for extending SAPUI5 Applications | SAP Blogs

Thursday, March 4, 2021 8:50 AM

Clipped from: https://blogs.sap.com/2020/07/15/adaptation-project-your-one-stop-tool-for-extending-sapui5-applications/

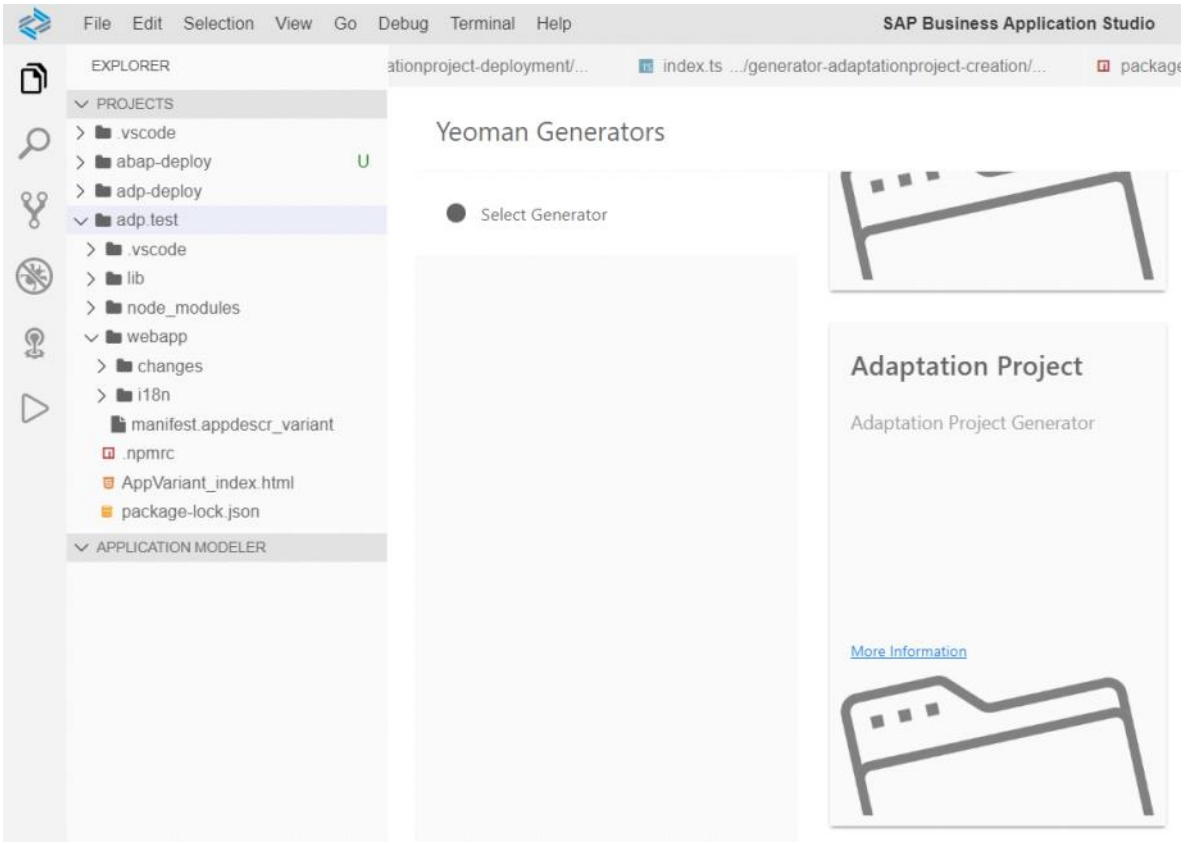
Whether you have already been using Adaptation Project in SAP Web IDE and want some fresh features, or you are trying to find the right tool to adapt applications to customer/partner specific needs, look no further. It's all now available at SAP`s brand new development platform – SAP Business Application Studio. By using it, you are leveraging features of UI5 Flexibility and able to adapt and extend UI5 applications without actually changing the base app, which allows seamless upgrades and lifecycle stability. Adaptation project also comes with an intuitive WYSIWYG tooling and is really fun to use and you are able to add both UI and code changes to your own variant of the application.

Except availability on the new platform, there are a couple of new features available:

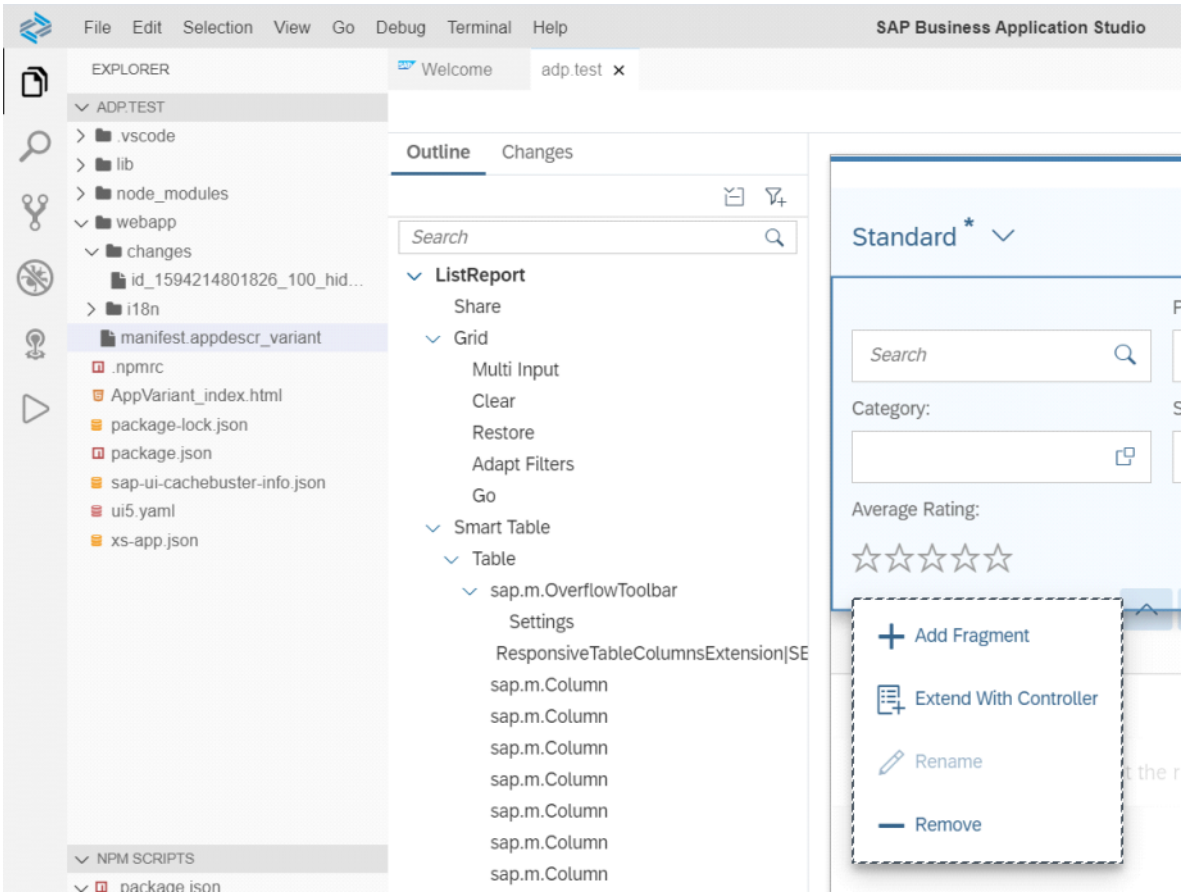
- Support for extension points – The so-called extension points were previously matter of another plugin – Extension Project. Its abilities are being brought to Adaptation Project, so it becomes the unified one and only tool for extension and adaptation. This feature is also now available in SAP Web IDE and SAP Business Application Studio.
- Support of Freestyle applications – Till now, it was possible to extend only Fiori Elements-based applications, but from now on you have the ability to extend also Freestyle applications. This feature is now available in both SAP Web IDE and on SAP Business Application Studio, as well.

Let's go over a simple use case of Adaptation Project on the new platform:

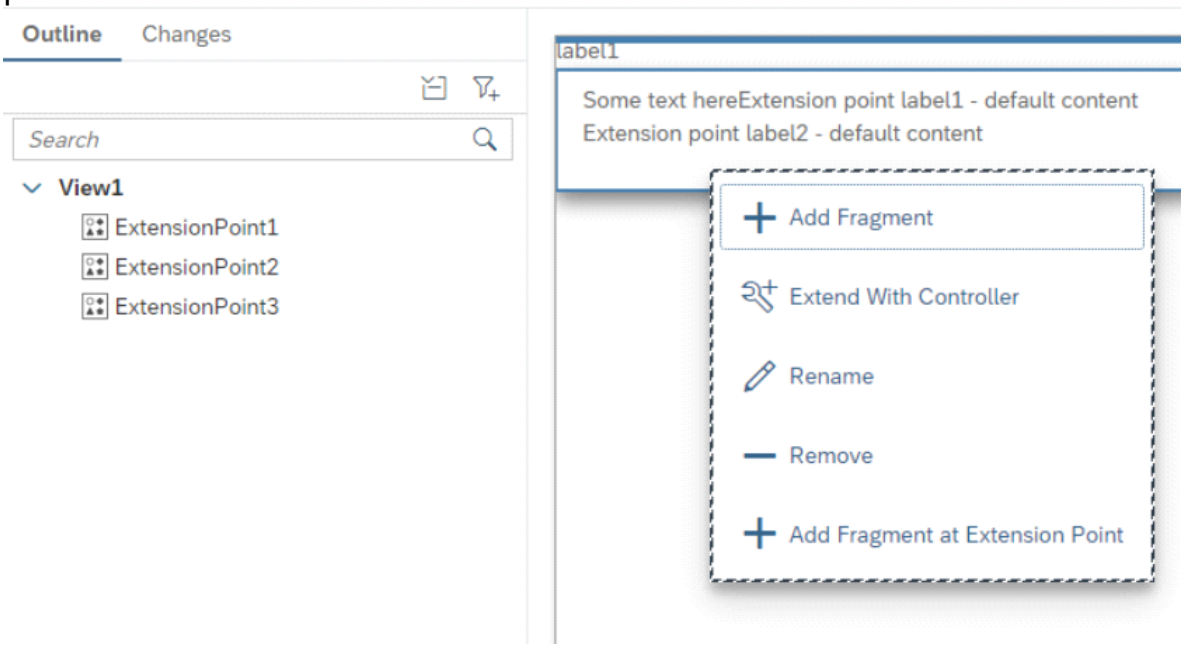
1. Create adaptation project using one of the Yeoman generators available in SAP Business Application Studio. The wizard will guide you through the process and you will be asked to add the initial properties of your project – name, source system, base app, etc.



2. You can start the visual editor which will enable you to both preview and edit the application. The possibilities are the same for both SAP Fiori elements-based and Freestyle applications and you can extend them in the same manner – you can adapt UIs, e.g. add/move/remove fields, rearrange sections or even embed iframes, change control properties, extend i18n texts, add custom content via XML Fragments, etc. You can extend also controllers by using the right-click option "Extend with controller".



3. It's also possible to use extension points (if the base app has such). XML Fragments can be added using the "Add Fragment at Extension Point" option from the context menu, when right-clicked over an extension point.



From now on you can use Adaptation Project to benefit from these great capabilities.

For the complete set of features and technical details you can always refer to the official documentation pages:

Adaptation Project at SAP Web IDE

Adaptation Project at SAP Business Application Studio

Previous Post: UI5ers Buzz #54: I18n with supportedLocales and fallbackLocale configuration

Author

Hristo Tsolev is a product owner in CP Extensions team and the Flexibility area. Always looking to improve the experience of customers

Create an Adaptation Project - SAP Help Portal

Thursday, March 4, 2021 7:04 AM

Create an Adaptation Project - SAP Help Portal

https://help.sap.com/viewer/584e0bcbfd4a4aff91c815cefa0bce2d/Cloud/en-US/9d2028f106d44424a0f4637541b6e25f.html

Create an adaptation project from an existing oData v2 SAP Fiori elements-based and SAPUI5 freestyle applications available in the on-premise...

Adaptation Projects: A Tutorial | SAP Blogs

Thursday, March 4, 2021 8:52 AM

Clipped from: https://blogs.sap.com/2019/01/14/adaptation-projects-a-tutorial/

In prior blogs, I gave a high level overview of Adaptation Projects and presented a video demo. If your interest is piqued, follow along with today's blog to create your own application variant using Adaptation Projects.

Pre-requisites:

When creating an Adaptation Project, you must have access to an ABAP on premise system that contains some applications made with SAP Fiori elements. For this tutorial, you will need a SAP Cloud Platform Cockpit account, access to SAP Web IDE and access an SAP demo system, ES5.

1. Don't have access to Web IDE and SAP Cloud Platform Cockpit? Follow the instructions here to get an account.
2. Next, sign up for a demo account on ES5 here.
3. Next, in SAP Cloud Platform Cockpit you have to create a destination using the following data:

```
#
#Tue Dec 05 14:36:33 UTC 2017
Description=SAP Gateway Demo System
Type=HTTP
TrustAll=true
Authentication=NoAuthentication
WebIDEUsage=odata_abap, bsp_execute_abap, odata_gen, dev_abap
Name=ES5
WebIDEEnabled=true
CloudConnectorVersion=2
URL=https\://sapes5.sapdevcenter.com
ProxyType=Internet
sap-client=002
WebIDESystem=ES5
```

You can simply create a text file that contains the above mentioned data and import the destination as follows:

1. Log in to the SAP Cloud Platform Cockpit.
2. Select your account.
3. Click *Connectivity.*
4. Click *Destinations.*
5. Click *Import Destination.*
6. Select your destination configuration file and click Open.
7. Select Save.

Exercise

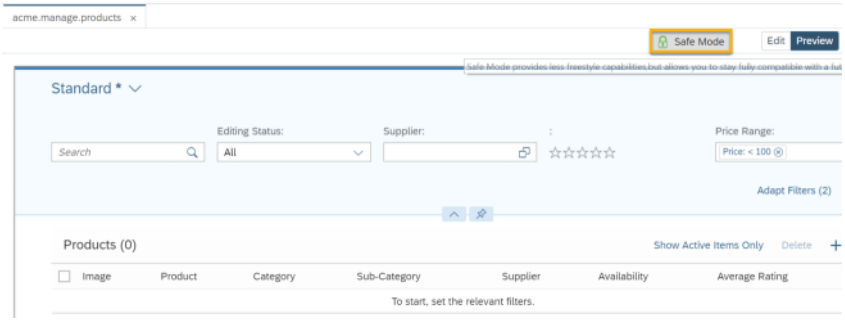
Now that you're configured, let's create an Adaptation Project.

Scenario: you are one of Santa's elves and you had a heck of a time finding the right sized boxes as you were wrapping gifts last Christmas. You'd like to search for products that fit in Small, Medium and Large sized boxes to help organize your packaging when the crush happens next time around.

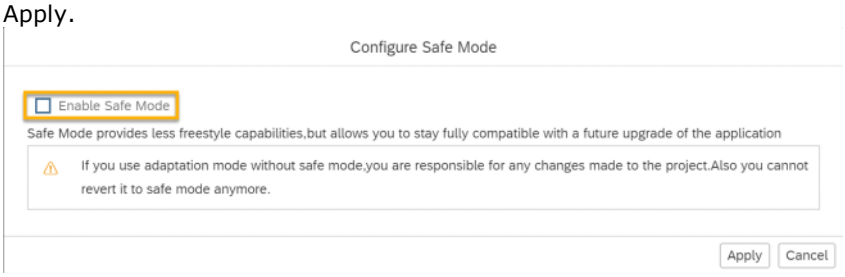
To do this, you are going to extend the Manage Products application that resides on ES5. This application allows users to search for products and look at the details of those products. This application was created using the SAP Fiori elements List Report and Object Page.

When you create an Adaptation Project, you are creating a new variant of an existing application. The app variant refers to the original application but contains a separate set of changes created in the adaptation project. Also, if the original application is changed, the variant will incorporate those changes. A new application ID is defined for the variant and the variant will need to be registered in SAP Fiori launchpad in order for users to access it.

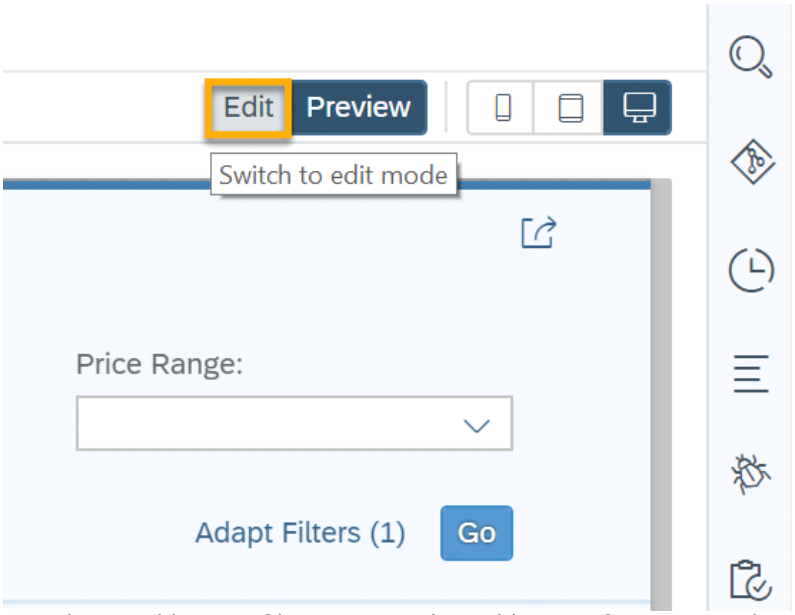
1. Open SAP Web IDE.
2. Create a new adaptation project: File -> New -> Adaptation Project
3. Now you need to enter a project name and application title.
   Project Name: **adapt.listreport.products**
   Application Title: **Adapted Manage Products**
   Namespace: <accept the default value>
4. Select Next.
5. Now we need to select a system and source application upon which to base this variant.Select **SAP Gateway Demo System** (that's ES5). You may need to enter your user name and password for that system.Select the application with the Description **Manage Products** and select Finish.
6. A new project is created and shows you a working application based on the source application. Any changes we make in this variant are not going to change the code of the source application.Next, let's open up this project in the SAPUI5 Visual Editor, where we can preview what the application looks like.Right click on the project and **Select SAPUI5 Visual Editor***Note: You might be prompted for a user name and password. If so, enter your user name and password for the ES5 system, which you created in the 1st pre-requisite step.
7. You see that you can do a search for products based on a number of filters. Do a search where you filter on price range to see how it works.
8. Let's add another filter – one that searches on shipping box sizes.Before we start editing, let's disable Safe Mode.When in Safe Mode, the developer can only make changes that Key Users can make in Adaptation at Runtime, and those are changes that will *always* work when the source application is updated. In Safe Mode you can move sections, rename labels, etc. However, we want to make more changes than that…Select **Safe Mode**



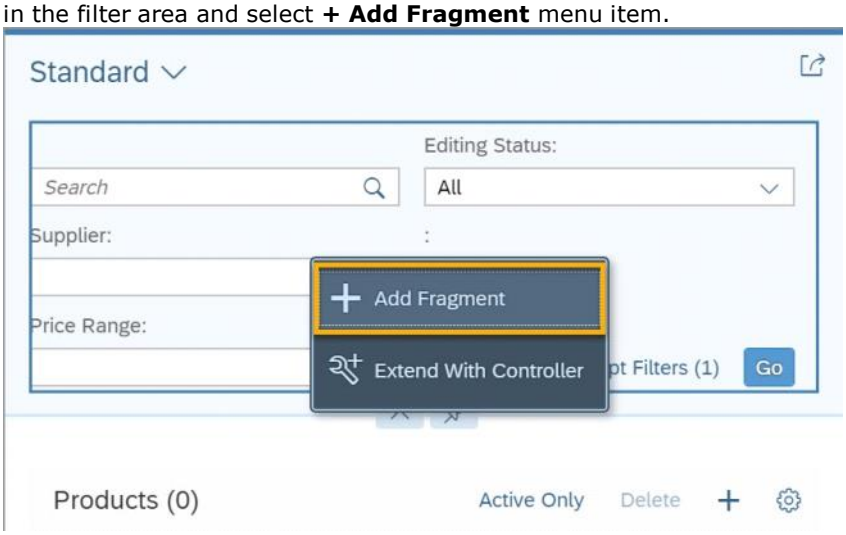
9. De-select the **Enable Safe Mode** checkbox to disable it and select Apply.



10. Go to **Edit** mode.



11. In order to add a new filter, you need to add a new fragment.Right click in the filter area and select **+ Add Fragment** menu item.

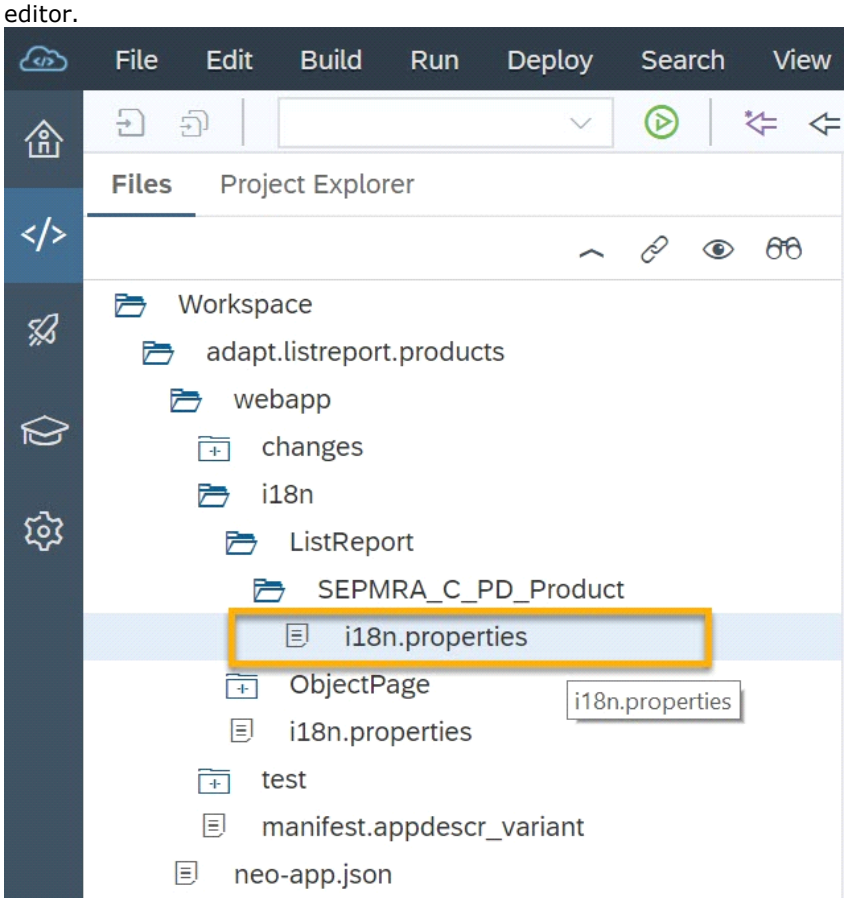


12. Here we are going to create a new Control Configuration named **boxFilter**.
 From the Target Aggregation dropdown, select **controlConfiguration**.Select the **Create new** In the Add Fragment dialog box, enter a Fragment Name of **BoxFilter** and select **Create**.
13. Now we have a new Boxfilter XML file that's blank and we need to add some code. Copy the following code and paste it in to the **boxFilter.fragment.xml** in your SAP Web IDE. Then **Save**.

```
<!-- Use stable and unique id's!-->
<core:FragmentDefinition xmlns:core='sap.ui.core' xmlns='sap.ui.comp.smartfilterbar' xmlns:m='sap.m'>
      <ControlConfiguration
      id="boxFilter"
      key="boxFilter"
      index="20"
      label="{i18n>BOX_FILTER}"
      groupId="_BASIC"
      visibleInAdvancedArea="true"
      filterType="multiple"
      >
            <customControl>
                  <m:ComboBox id="boxTypes">
                        <core:Item id="smallBox" key="small" text="{i18n>SMALL_BOX}" />
                        <core:Item id="mediumBox" key="medium" text="{i18n>MEDIUM_BOX}" />
                        <core:Item id="largeBox" key="large"
```

```
text="{i18n>LARGE_BOX}" />
                </m:ComboBox>
            </customControl>
        </ControlConfiguration>
</core:FragmentDefinition>
```

14. Go back to the Visual Editor tab and accept the message that says changes were made outside the editor and let it reload the editor. You may be asked if you want to save unsaved changes in the editor – if so, select **Save**.
15. To get your new filter configured for localization, you can add keys to the i18n.properties file. In the tree view of Web IDE, double-click on the **i18n.properties** for the ListReport of this project to open it in the editor.



16. Copy the following code and replace the contents of **i18n.properties** and then select the **Save** button: `#Make sure you provide a unique prefix to the newly added keys in this file, to avoid overriding of SAP Fiori application keys.`

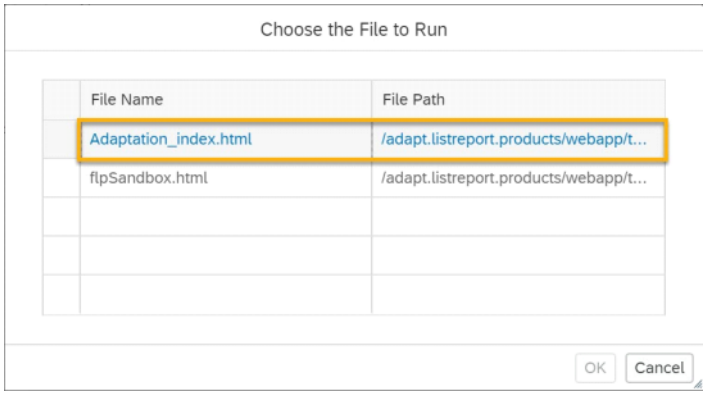
```
BOX_FILTER=Box Size
SMALL_BOX=Small
MEDIUM_BOX=Medium
LARGE_BOX=Large
```

17. Go back to the Visual Editor tab and accept the message that says changes were made outside the editor and let it reload the editor. You'll see the new filter, but there is still no code there to control how it works.
18. Select Edit, right click in the SmartFilter area and select **Extend with Controller**.
19. Enter a name of **ListReportControllerExtension** and select **Extend.**
20. Now you are looking at a new file, ListReportControllerExtension.js. Notice this is *not* an empty file, inside the comments you find documentation about which methods you can extend. There is a lot of documentation on controller extensions in the SAPUI5 documentation.For now, replace the contents with the following code and select the **Save** button: `/***
@controller`

```
Name:sap.suite.ui.generic.template.ListReport.view.ListReport,
*@viewId:nw.epm.refapps.st.prod.manage::sap.suite.ui.generic.templ
ate.ListReport.view.ListReport::SEPMRA_C_PD_Product
*/
sap.ui.define([
            "sap/ui/core/mvc/ControllerExtension",
            "sap/ui/model/Filter",
            "sap/ui/model/FilterOperator"
      ],
      function (
            ControllerExtension,
            Filter,
            FilterOperator
      ) {
            "use strict";

            //hard coded list of shipping box sizes:
            var BOX_SIZES = {
                  small: new Filter({
                        filters: [
                              new Filter({
                                    path: "Height",
                                    operator: FilterOperator.BT,
                                    value1: 0,
                                    value2: 10
                              }),
                              new Filter({
                                    path: "Width",
                                    operator: FilterOperator.BT,
                                    value1: 0,
                                    value2: 10
                              }),
                              new Filter({
                                    path: "Depth",
                                    operator: FilterOperator.BT,
                                    value1: 0,
                                    value2: 10
                              })
                        ],
                        and: true
                  }),
                  medium: new Filter({
                        filters: [
                              new Filter({
                                    path: "Height",
                                    operator: FilterOperator.BT,
                                    value1: 10,
                                    value2: 20
                              }),
                              new Filter({
                                    path: "Width",
                                    operator: FilterOperator.BT,
                                    value1: 10,
                                    value2: 20
                              }),
                              new Filter({
                                    path: "Depth",
                                    operator: FilterOperator.BT,
                                    value1: 10,
                                    value2: 20
                              })
                        ],
                        and: true
                  }),
                  large: new Filter({
                        filters: [
                              new Filter({
                                    path: "Height",
                                    operator: FilterOperator.GT,
                                    value1: 20
                              }),
```

```
                              new Filter({
                                    path: "Width",
                                    operator: FilterOperator.GT,
                                    value1: 20
                              }),
                              new Filter({
                                    path: "Depth",
                                    operator: FilterOperator.GT,
                                    value1: 20
                              })
                        ],
                        and: true
                  })
            };

            return
ControllerExtension.extend("customer.adapt.listreport.products.Lis
tReportControllerExtension", {

                  // this section allows to extend lifecycle hooks
or override public methods of the base controller
                  override: {
                        //    override public methods of the
ListReport controller and its members like templateBaseExtension
                        templateBaseExtension: {
                              /**
                               * Can be used to add filters. They
will be combined via AND with all other filters
                               * sControlId is the ID of the control
on which extension logic to be applied.
                               * For each filter the extension must
call fnAddFilter(oControllerExtension, oFilter)
                               * oControllerExtension must be the
ControllerExtension instance which adds the filter
                               * oFilter must be an instance of
sap.ui.model.Filter
                               */
                              addFilters: function (fnAddFilter,
sControlId) {
                                    var oComboBox =
this.byId("boxTypes");
                                    var sSelectedBoxType =
oComboBox.getSelectedKey();
                                    //lookup the filters from the
definition
                                    var oFilter =
BOX_SIZES[sSelectedBoxType];
                                    if (oFilter) {
                                          fnAddFilter(this, oFilter);
                                    }
                              }
                        }
                  }
            });
      });
```

21. Now go ahead and preview as a web application. **Select Run-> Run as Web Application.** If asked which configuration to use, select the **Adaptation_index.html**.



Congratulations, now you're running your application! Try filtering on the box sizes and note the different results when you select Small, Medium or Large boxes. If you like, you can customize the table to show the dimension columns to see whether the products will really fit in to the box sizes indicated.

Your variant of the Manage Products application is now complete and ready to deploy! Unfortunately, the ES5 system does not allow the general public to deploy to it, so I cannot give you instructions for doing that part of the project. However, if you are curious about how to deploy a SAPUI5 application to SAP Fiori launchpad, here is an excellent tutorial that details the steps.

If you found this tutorial helpful and interesting, keep an eye out for a new OpenSAP course on SAPUI5 in early 2019. The new course will include some lessons on Adaptation Projects and other flexibility functionality.

Helpful Links:

SAPUI5 Visual Editor

Adaptation Projects for Fiori Elements Applications

Extending Delivered Apps Using Adaptation Extensions

Adaptation Extension Example: Adding a Button to the Table Toolbar in List Report

About the authors Jessica Merz and Sebastian Wennemers

Jessica is the Product Manager for SAPUI5 flexibility functionality and is always looking for ways to improve the user experience of SAPUI5 based applications. With a background of nearly 20 years of user research and UX design, she enjoys helping people make their applications intuitive and delightful.

As Development Architect for SAPUI5 flexibility functionality Sebastian enables several people, from end users to developers, to get the best out of existing SAPUI5 apps. With the work of his team they can adapt these apps with easy to use tooling in a lifecycle stable and modification-free way, so that the apps fit best to their processes and needs.

How to create a Component-preload.js file in a custom site template | SAP Blogs

Thursday, March 11, 2021 9:23 AM

Clipped from: https://blogs.sap.com/2019/05/28/how-to-create-a-component-preload.js-file-in-a-custom-site-template/

To improve the performance of a custom site template and speed up it's initial loading, it is recommended to create a Component-preload.js file.

In the following procedure, we explain how to create such a file for your project.

Steps:

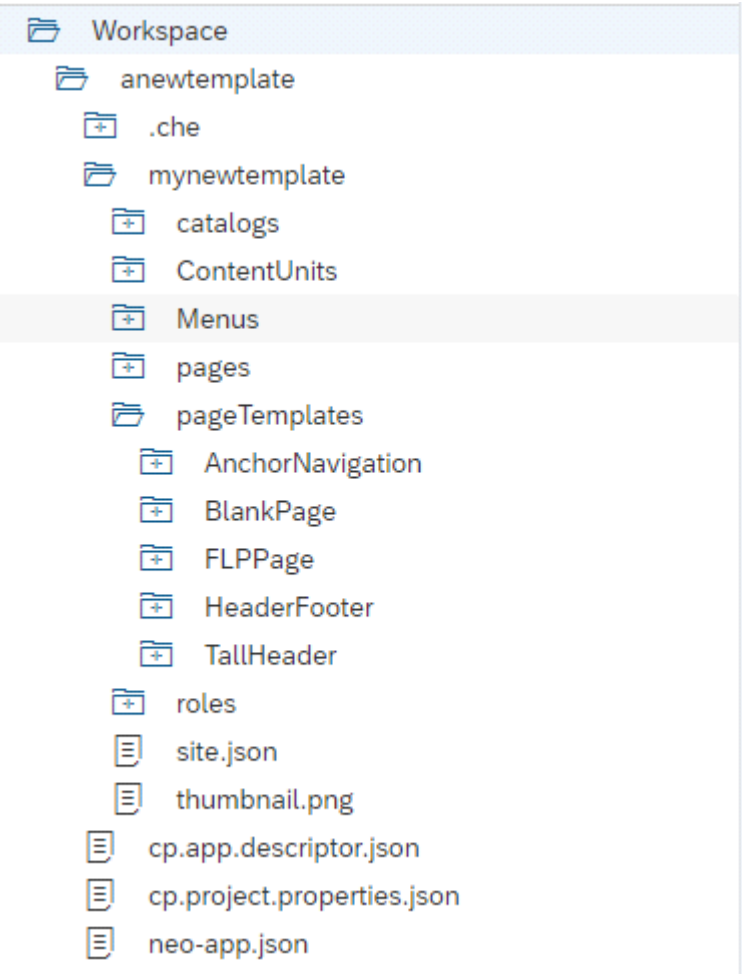
1. In SAP Web IDE, create a new site template, based on the Basic Layout template.

Project name: *anewtemplate*

Site template name: *mynewtemplate*

For more information about how to create a site template, see Create a Site Template.

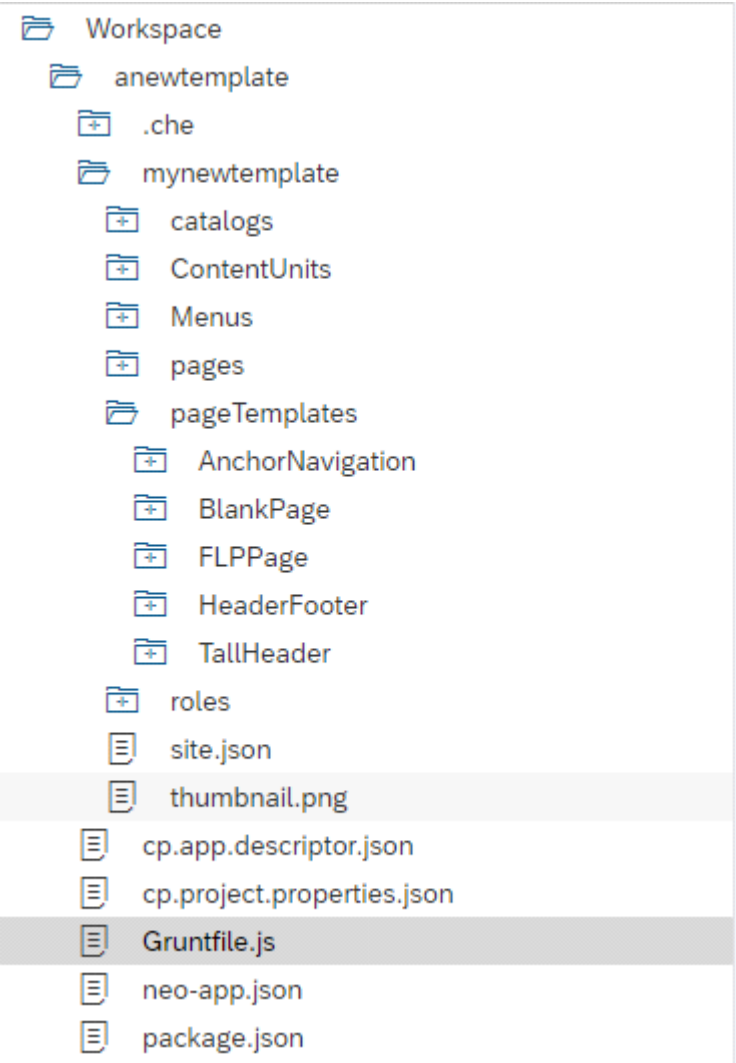
You should get the following project structure:



2. In your project folder, create the following files:

– package.json

– Gruntfile.js



3. Open the package.json file and add the following code:

```
{
  "name": "anewtemplate",
  "version": "0.0.1",
  "description": "",
  "private": true,
  "devDependencies": {
    "grunt": "1.0.3",
    "grunt-openui5": "^0.14.0"
  },
  "scripts": {}
}
```

NOTE: The *name* attribute must be the name of the project.

4. Open the Gruntfile.js file and add the following code:

```
module.exports = function (grunt) {
    grunt.initConfig({
            "openui5_preload": {
            }
    });
      grunt.loadNpmTasks('grunt-openui5');
    grunt.registerTask('default', ['openui5_preload']);
}
```

5. Inside the “openui5_preload” attribute, add a script that will create the Component-preload.

In the following example, a Component-preload is added to the AnchorNavigation page which is located under the pageTemplates folder:

```
module.exports = function (grunt) {
    grunt.initConfig({
      "openui5_preload": {
           "AnchorNavigation": {
                options: {
                  resources: {
                    cwd:
'mynewtemplate/pageTemplates/AnchorNavigation',
                    prefix: 'cpv2/templates/AnchorNavigationV2',
                    src:[
                        '**/*.js',
                        '**/*.fragment.json',
                        '**/*.fragment.xml',
                        '**/*.view.xml',
                        '**/*.properties'
                     ]
                    },
                    dest:
"mynewtemplate/pageTemplates/AnchorNavigation"
                },
                components: '**'
            }
           }
    });
      grunt.loadNpmTasks('grunt-openui5');
      grunt.registerTask('default', ['openui5_preload']);
};
```

Note:

- The string that appears next to *cwd* and *dest* is a relative path to the

site template that you've created. The path consists of "<your-template-name>/pageTemplates/<the-page-name>".

- The string that appears next to *prefix* can be found in the Component.js file of the chosen page. For example, this is the prefix for the AnchorNavigation page, taken from the AnchorNavigation folder -> Component.js file:

```
1  // define a root UIComponent which exposes the main view
2  /*global jQuery, sap */
3  jQuery.sap.declare("cpv2.templates.AnchorNavigationV2.Component");
4  jQuery.sap.require("sap.ui.core.UIComponent");
5  jQuery.sap.require("sap.ui.core.routing.Router");
6
7  // new Component
8  sap.ui.core.UIComponent.extend("cpv2.templates.AnchorNavigationV2.Component", {
9
10     oMainView: null,
11
12     // use inline declaration instead of component.json to save 1 round trip
13     metadata: {
14         manifest: "json"
15     },
16
17     createContent: function () {
18         "use strict";
19         this.oMainView = sap.ui.view({
20             type: sap.ui.core.mvc.ViewType.XML,
21             viewName: "cpv2.templates.AnchorNavigationV2.Template",
   Strings must use doublequote. (quotes) [ESLINT: (quotes)]
24
25         return this.oMainView;
26     }
27 });
```

  When you copy this prefix from the Component.js file, make sure to replace the dots with slashes and remove the word "Component".
- The rest of the code in the Component-preload example should be copied as-is.

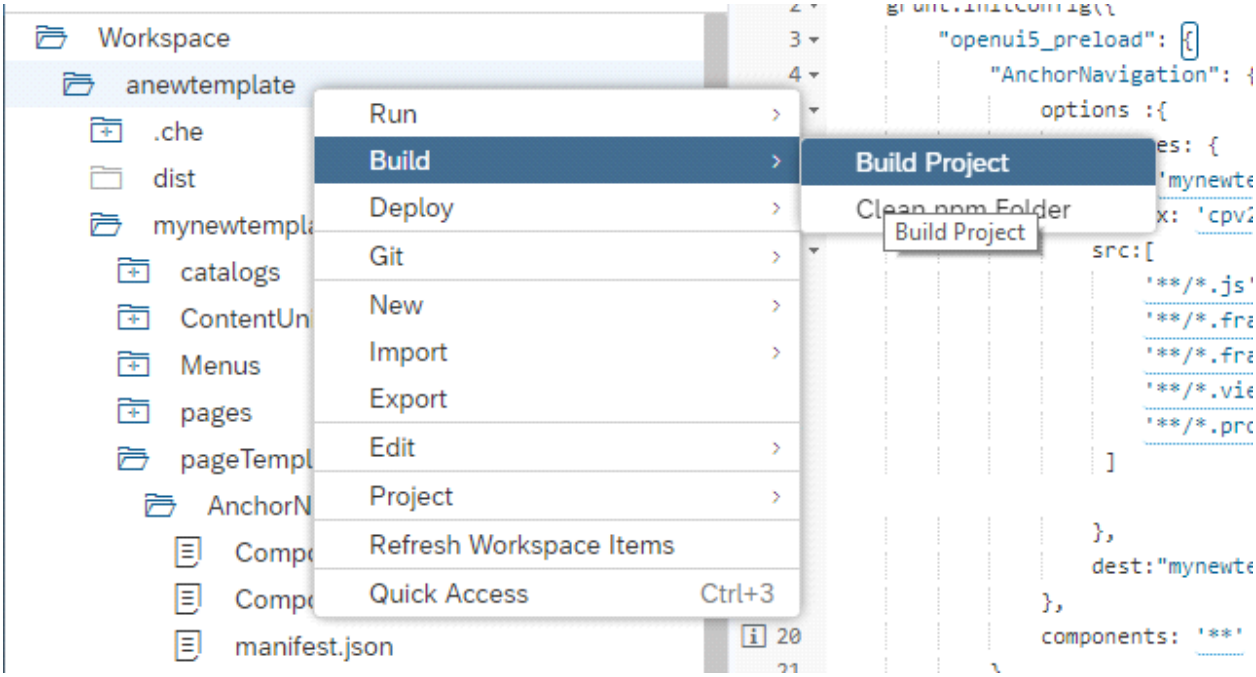
6. To add a Component-preload script to another page in the site template (in this example, the BlankPage), add another script block under the "openui5_preload" section, separated by a comma.

```
module.exports = function (grunt) {
    grunt.initConfig({
        "openui5_preload": {
            "AnchorNavigation": {
            // Here goes the previous code
            },
            "BlankPage": {
                options :{
                  resources: {
```

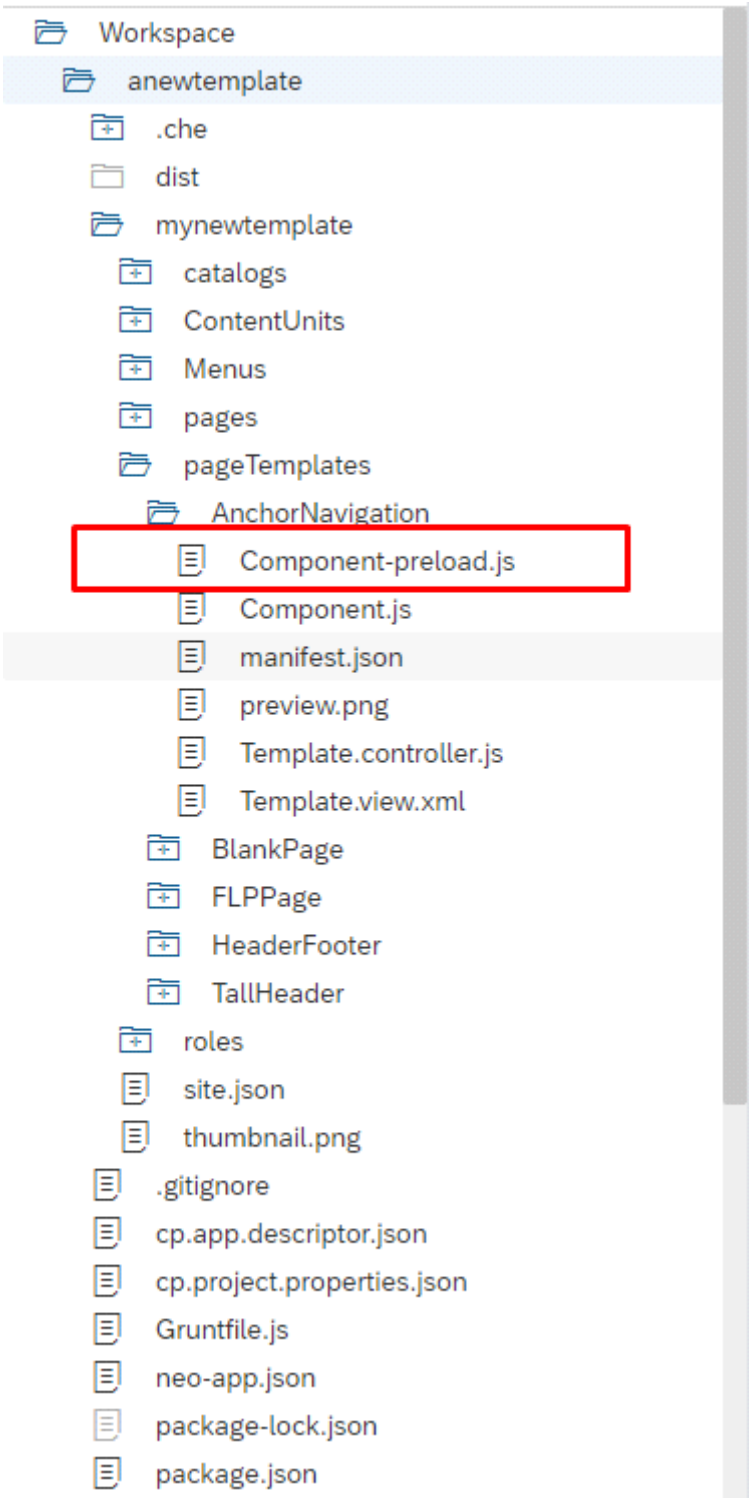
```
                cwd: 'mynewtemplate/pageTemplates/BlankPage',
                prefix: 'cpv2/templates/Blank',
                src:[
                    '**/*.js',
                    '**/*.fragment.json',
                    '**/*.fragment.xml',
                    '**/*.view.xml',
                    '**/*.properties'
                 ]
                },
                dest:"mynewtemplate/pageTemplates/BlankPage"
            } ,
            components: '**'
        }
    }
});
```

7. Right-click the project -> Build -> Build Project.

The build might take a few minutes to complete.

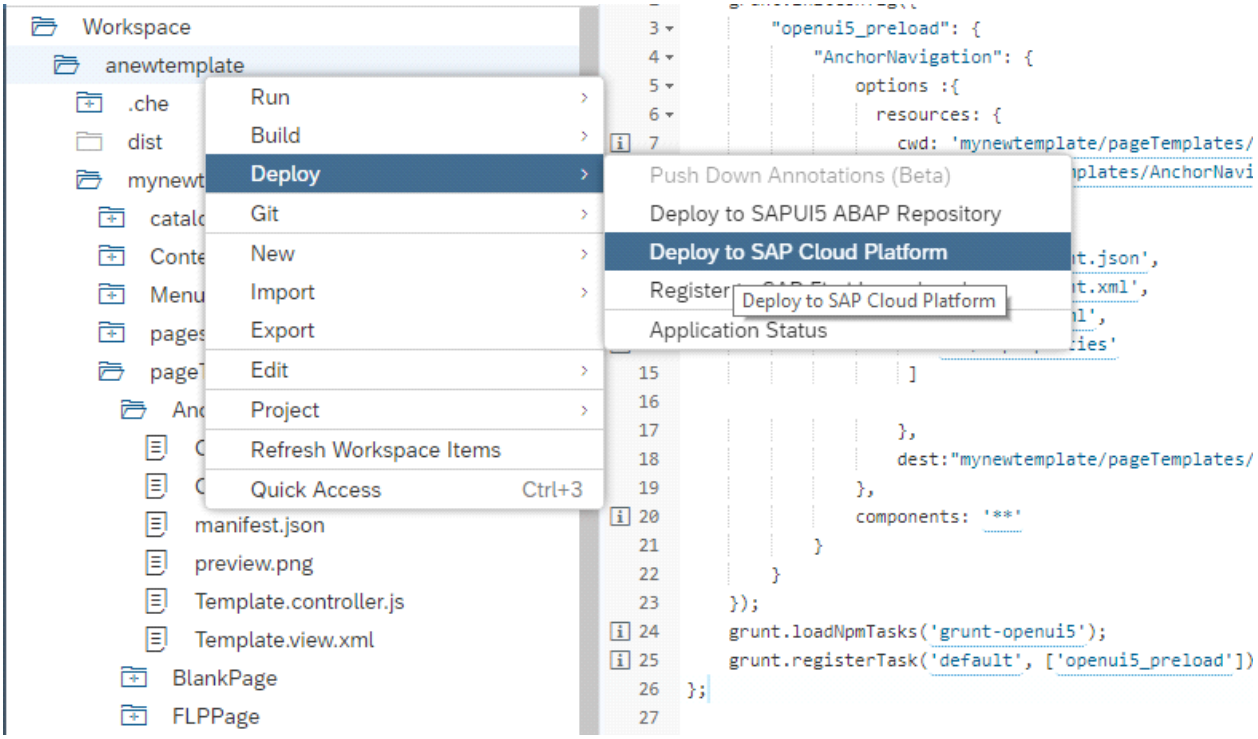


After the build finishes successfully, you can find the Component-preload.js file under the pageTemplate folder.



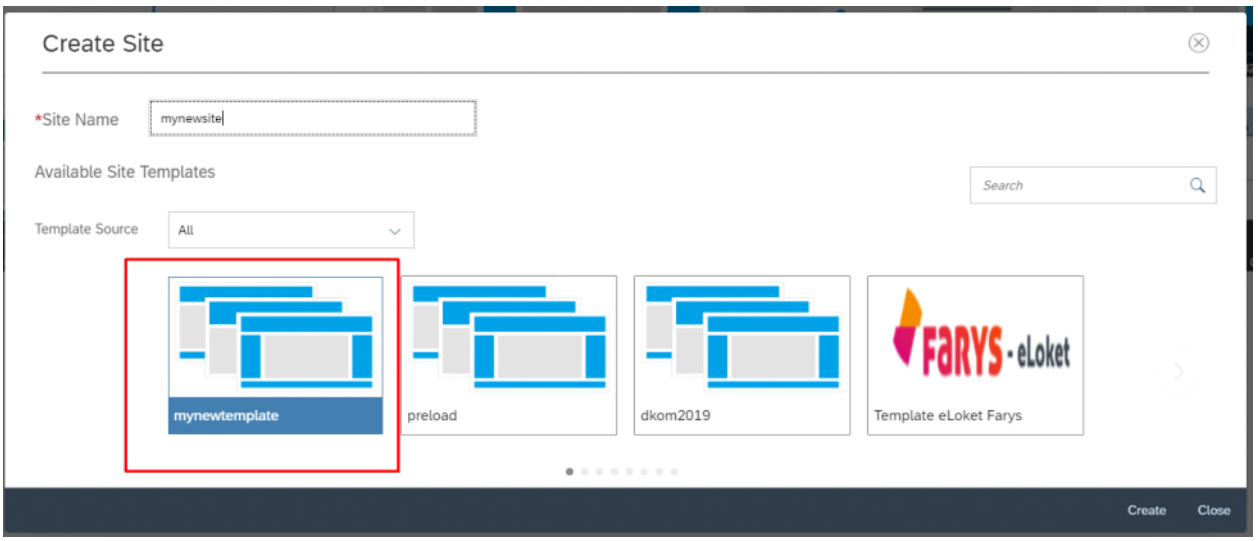
8. Before deploying the project, you must delete the package.json file (and package-lock.json file, if exists).

9. Right-click the project-> Deploy -> Deploy to SAP Cloud Platform.



10. After the deployment finishes successfully, you can test your flow.

Open the Site Directory and create a site using the site template that you've created.



The new site template will load the Component-preload.js instead of Component.js and you should notice a faster loading!

In conclusion, we know how important performance is to our sites. The lack of Component-preload files in our custom site templates can harm the performance and slow things down. In this blog post, we've seen how easy it is to add Component-preload files to our custom site templates and rapidly increase the performance.

Hope this will help you make better sites with better performance!

UI5 Tooling 2.0: How to develop and run SAP Fiori elements locally! | SAP Blogs

Friday, April 17, 2020 3:52 PM

Peter Muessig

Posted on April 15, 2020 7 minute read

UI5ers Buzz #51: UI5 Tooling 2.0: How to develop and run SAP Fiori elements locally!

Introduction

On April 1st I talked about the *"UI5 Tooling – a modern CLI-based development experience"* in a SAP Community call. A week later, on April 7th I published the blog post *"UI5 Tooling: a modern development experience for UI5!"* which demonstrates the capabilities of the UI5 Tooling how we can consume UI5 library dependencies locally and how we can use custom middlewares to improve the overall UI5 development experience. This time, I want to focus on the availability of SAPUI5 on NPM and how we can utilize this to develop and run an SAP Fiori elements application locally.

SAPUI5 from NPM!

The biggest benefit of the UI5 Tooling 2.0 is the consumption of the SAPUI5 library dependencies from NPM. Merlin Beutlberger explains the new features of the UI5 Tooling 2.0 in his blog post *"The UI5 Tooling and SAPUI5 – The Next Step"*. Thanks to the awesome work of the UI5 Tooling team, we can now develop SAPUI5 applications locally. We can consume the *sap.suite.ui.generic.template* library to build and develop SAP Fiori elements applications locally and also consume the *sap.ushell* library to make the application running in the SAP Fiori launchpad sandbox. Within this blog post, I will focus on creating an SAP Fiori elements application in SAP Web IDE and run it locally with the UI5 Tooling.

Prerequisites

As the UI5 Tooling is based on Node.js. We need to ensure that at least Node.js version 10 is installed and configured properly.

Step 1: Create an SAP Fiori elements application

In the first step, we create an SAP Fiori elements application based on the List Report Application template. This has to be done in SAP Web IDE by using the template wizard.

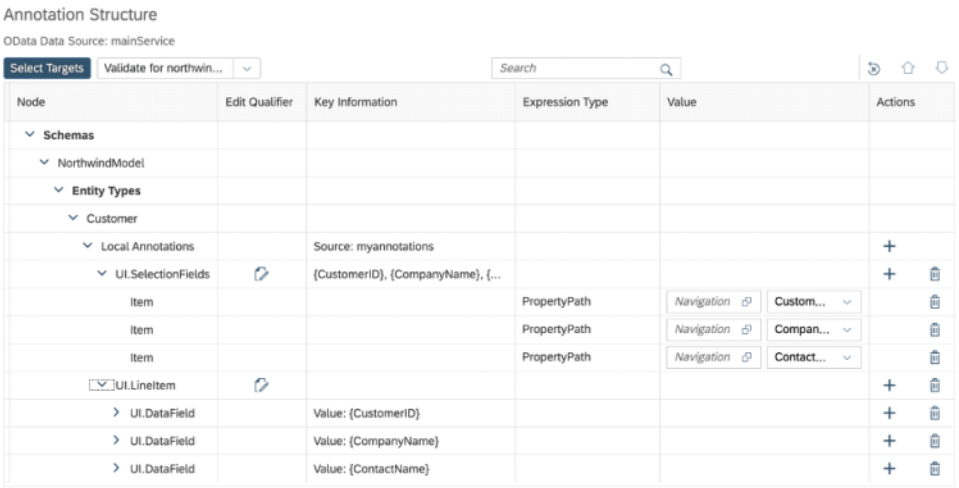
Follow these steps to create a List Report Application:

1. Open SAP Web IDE
2. "File" > "New" > "Project from template"
3. Select environment "Neo" (as this affects the datasource URL relevant for the proxy)
4. Select category "SAP Fiori Elements"
5. Select template "List Report Application"
6. Enter your project name, title and namespace
7. In the data connection step (prerequisite: setup a destination to Northwind OData Services first), select the Northwind OData Services as "Service URL", use the "Relative URL": */V2/Northwind/Northwind.svc/*) and press "Test".
8. Skip the annotation selection step, just press "Next"
9. In the template customization step, select the Object Collection: *Customers*, OData Navigation: *Orders*, OData Sub Navigation: *Order_Details*.
10. Finish

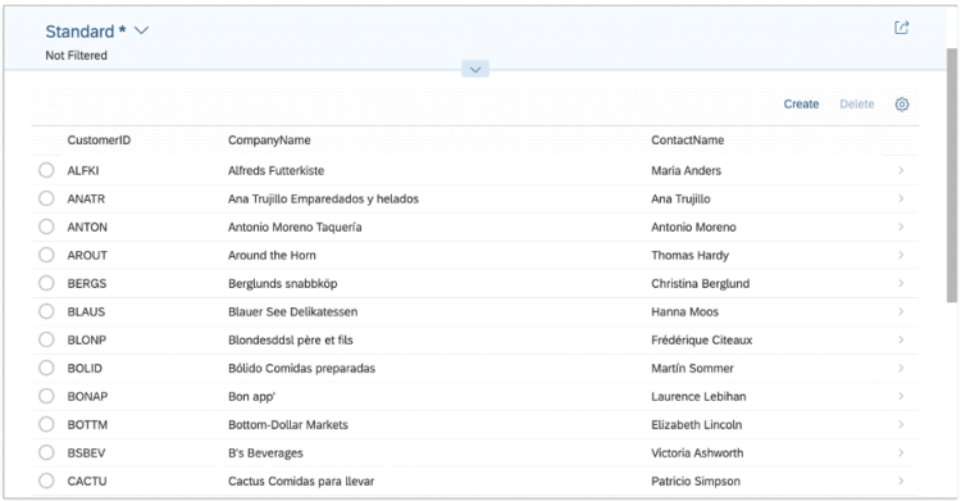
Step 2: Create the annotation file

As second step, we need to create a simple annotation file for our SAP Fiori elements application. Therefore, follow these steps:

1. Create the folder *annotations* in the *webapp* folder
2. Use the context menu on the *annotations* folder and select "New" > "Annotation File"
3. Just keep the default settings for the annotation file and press "Next" and "Finish"
4. Create a *LineItem* annotation with the *DataField*s: *CustomerID*, *CompanyName* and *ContactName* and the *SelectionFields* annotation with the *Items: CustomerID*, *CompanyName* and *ContactName* Feel free to add more annotations. 😳 Finally, the simple annotation file should look like this:



We can test the annotations file by running the SAP Fiori elements application with the SAPUI5 Visual Editor. The application should look like this (select the project in the project explorer and use the context menu on the project folder to launch the SAPUI5 Visual Editor):



That's all we need to do in the SAP Web IDE. Let's go local…

Step 3: Download and prepare the SAP Fiori elements application

After the SAP Fiori elements application has been created, select it in the project explorer, export it from SAP Web IDE and unzip it locally in your file system. As a first step locally, we validate whether we need to update the project dependencies of the created UI5 application. To update the project dependencies, we can use the open-source tool npm-check-updates. This tool can be installed via:

```
npm install -g npm-check-updates
```

After the installation of the tool, we can run:

```
ncu -u
```

to upgrade the project dependencies in the *package.json*. Once the project dependencies are upgraded, run:

```
npm install
```

to prepare the project for a first run. The project can be started with the command:

```
npx ui5 serve -o test/flpSandbox.html
```

Although the development server starts technically, there are a few things missing to make the SAP Fiori elements application to run properly (e.g. the SAPUI5 resources).

Step 4: Consume SAPUI5 library dependencies from NPM

Now, we are configuring the application to consume the SAPUI5 library dependencies from NPM. Therefore, we modify the *ui5.yaml* to use the *specVersion: 2.0*:

```
specVersion: '2.0'
metadata:
  name: mylistreport
type: application
[…]
```

As next step, we add the framework dependency to SAPUI5 by running the following command:

```
npx ui5 use sapui5@latest
```

Once the framework dependency is added, we can include the required SAPUI5 libraries with the following command (make sure to grab the whole line => it's long 😉):

```
npx ui5 add sap.ui.core sap.m sap.ushell sap.collaboration sap.ui.generic.app
sap.suite.ui.generic.template themelib_sap_fiori_3
```

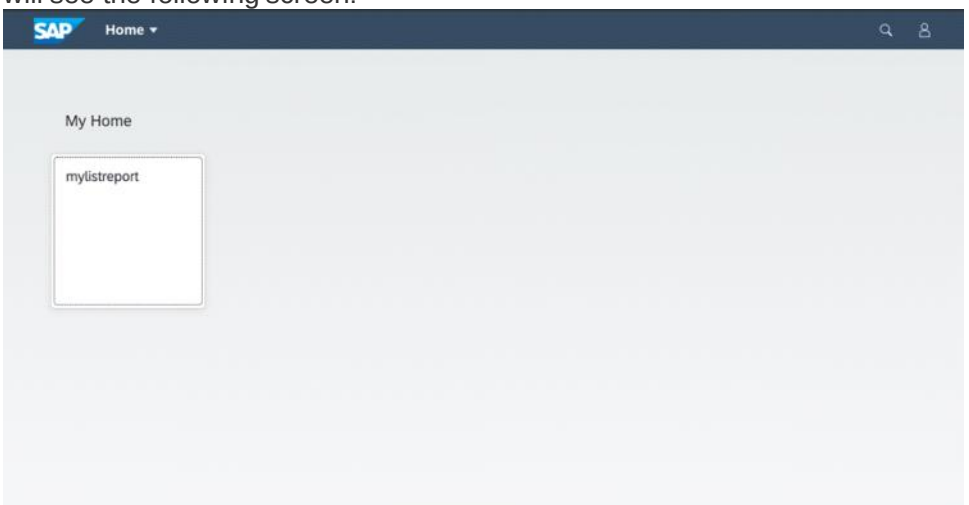
This will add the *sap.ui.core, sap.m, sap.ushell, sap.collaboration, sap.ui.generic.app, sap.suite.ui.generic.template and the themelib_sap_fiori_3* as UI5 library dependencies to the project and the two commands above will create the following section in the ui5.yaml:

```
framework:
  name: SAPUI5
  version: "1.76.1"
  libraries:
    - name: sap.collaboration
    - name: sap.m
    - name: sap.suite.ui.generic.template
    - name: sap.ui.core
    - name: sap.ui.generic.app
    - name: sap.ushell
    - name: themelib_sap_fiori_3
```

After starting the development server with:

```
npx ui5 serve -o test/flpSandbox.html
```

the SAPUI5 resources are loaded from the framework dependencies configured in the ui5.yaml. We will see the following screen:



Now the SAP Fiori elements application is starting up and you can go ahead to configure the proxy to access the Northwind OData Service.

Step 5: Setting up a proxy to tunnel the Northwind OData Service

To access the Northwind OData Service in the SAP Fiori elements application, we can simply use

the simple proxy middleware from the UI5 Ecosystem Showcase. It can be installed as a dev dependency by running the following command:

```
npm install ui5-middleware-simpleproxy --save-dev
```

Afterwards, we need to add the *ui5-middleware-simpleproxy* as *ui5/dependencies* in the *package.json*:

```
 "ui5": {
   "dependencies": [
     "@sap/ui5-builder-webide-extension",
     "ui5-middleware-simpleproxy"
   ]
 },
```

Now, we need to open the ui5.yaml and add the following configuration before the framework section:

```
[…]
server:
  customMiddleware:
  - name: ui5-middleware-simpleproxy
    mountPath: /Northwind/V2/Northwind/Northwind.svc/
    afterMiddleware: compression
    configuration:
      baseUri: https://services.odata.org/V2/Northwind/Northwind.svc/
framework:
  name: SAPUI5
  version: "1.76.1"
[…]
```

This configuration ensures that requests to */Northwind/V2/Northwind/Northwind.svc/* are proxied to *https://services.odata.org/V2/Northwind/Northwind.svc/*.

Step 6: Validate that the SAP Fiori elements application runs locally

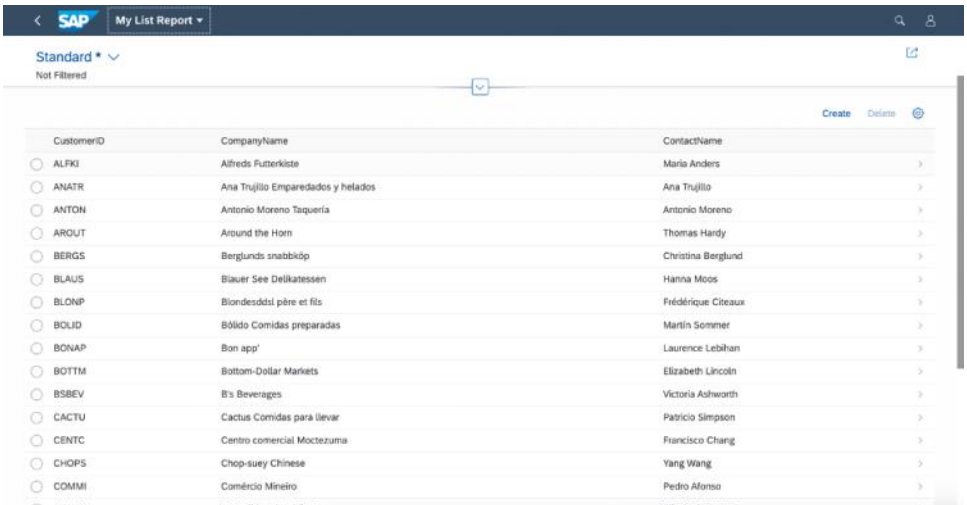
The next step is to validate that the SAP Fiori elements application runs locally. Therefore, we first ensure that all project dependencies are installed by running:

```
npm install
```

Afterwards, we can start the development server by executing:

```
npx ui5 serve -o test/flpSandbox.html
```

This will open our preferred browser and the SAP Fiori launchpad will come up with a tile for our SAP Fiori elements application. If we click on this tile, our application will start, and the data will be loaded from the Northwind OData Service.



Step 7: Add the livereload middleware

With the livereload middleware you can achieve a click and refresh behavior while developing. The middleware is watching the changes of the sources in the file system. Once a source file has been modified the livereload middleware refreshes the application in the browser.

Add the livereload middleware with the following command:

```
npm install ui5-middleware-livereload --save-dev
```

Afterwards in the package.json ensure to list the ui5-middleware-livereload in the ui5/dependencies:

```
  "ui5": {
    "dependencies": [
      "@sap/ui5-builder-webide-extension",
      "ui5-middleware-simpleproxy",
      "ui5-middleware-livereload"
    ]
  },
```

Afterwards, in the ui5.yaml just include the livereload middleware configuration next to the simpleproxy middleware configuration to the *server/customMiddleware* section:

```
[…]
server:
  customMiddleware:
  - name: ui5-middleware-simpleproxy
    mountPath: /Northwind/V2/Northwind/Northwind.svc/
    afterMiddleware: compression
    configuration:
      baseUri: https://services.odata.org/V2/Northwind/Northwind.svc/
  - name: ui5-middleware-livereload
    afterMiddleware: compression
    configuration:
      ext: "xml,json,properties"
      path: "webapp"
framework:
  name: SAPUI5
  version: "1.76.1"
[…]
```

The next time when you start the development server by running the following command:

```
npx ui5 serve -o test/flpSandbox.html
```

the livereload middleware is watching your file system and in case of changes it will refresh your browser (remark: in case you face issues with livereload, maybe your html page was cached before, just ensure to reload the html page properly).

Conclusion

As we can see, we can develop an SAP Fiori elements application locally with the UI5 Tooling. The SAPUI5 library dependencies are available locally and can be consumed. However, local SAP Fiori elements application development still feels a bit bumpy! Within SAP Web IDE, due to the availability of the Annotation Modeler and the SAPUI5 Visual Editor we have quite nice visual tools which support us to develop SAP Fiori elements applications. For local development these visual tools really missing. But for the purists among us who like to develop directly in the annotation files with the angle brackets, this could be nonetheless an interesting alternative. Enjoy…

Related Links

- SAPUI5 Tools for Eclipse – Now is the Time to Look for Alternatives (by Matthias Schmalz)
- End-To-End setup of local development environment with UI5 Tooling (by Jakob Marius Kjaer)
- Utilize ui5-tooling's extension capabilities for an improved development experience (by Volker

Buzek)
- Introducing UI5 Tooling for SAPUI5 Projects (by Andreas Ecker)
- UI5ers Buzz #49: The UI5 Tooling and SAPUI5 – The Next Step (by Merlin Beutlberger)

Previous Post: UI5ers Buzz #50: The Loading Evolution: “Get the most out of your UI5 app!”

Peter is one of the initiators who started the Phoenix project (a.k.a. OpenUI5/SAPUI5) grown into the role of the UI5 Chief Architect mainly focusing on UI innovation in the area of the UI framework (UI5 Evolution), controls (UI5 Web Components), tooling (UI5 Tooling).

- Alert Moderator

Assigned tags
- SAPUI5
- JavaScript
- Open Source
- SAP Fiori Elements
- SAP Fiori Launchpad
- SAP Web IDE
- User Interface
- #OpenUI5
- #SAPUI5
- UI5 Evolution
- UI5 Tooling
- UI5ers buzz

View more...

Related Blog Posts

Related Questions

11 Comments

You must be Logged on to comment or reply to a post.

Rahul Magdum
April 16, 2020 at 3:03 am

I must say…. That was a good tutorial… Eager to try it out myself. Thanks Peter!

- Like(1)

Shai Sinai
April 16, 2020 at 10:42 am
Thanks for this guide, a very detailed one.
However, isn't it a little paradoxical to use Web IDE for a local SAPUI5 application?

- Like(1)

Peter Muessig Post author
April 16, 2020 at 10:50 am
Hi Shai,
thanks for your feedback. SAPUI5 applications can also be created using the SAPUI5 Yeoman generator. I explained that in one of my other posts: https://blogs.sap.com/2020/04/07/ui5-tooling-a-modern-development-experience-for-ui5/. The SAP Fiori elements applications do not yet have a Yeoman generator available (at least nothing is known to me here).
BR,
Peter

- Like(0)

Shai Sinai
April 16, 2020 at 11:42 am
Thanks.
Yet, using non-local IDE for a local SAPUI5 application sounds a little paradoxical to me.
Is there any pragmatic use case for it?

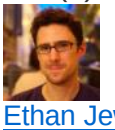
- Like(0)

Peter Muessig Post author
April 16, 2020 at 1:55 pm
Basically, there are good arguments for both – having the IDE either local or remote. The main **advantage of an online IDE** is that your projects are available on any machine, there is no setup and configuration required to connect to your cloud systems and there is no hardware limitation since all resources are stored in your workspace in the **cloud**.

- Like(0)

Shai Sinai
April 16, 2020 at 2:19 pm
Yes,
These are all valid points, of course.
My question was if there is any use-case for creating an SAPUI5 application via the Web IDE and then run it locally?

- Like(0)



Ethan Jewett
April 16, 2020 at 4:50 pm
Yes, of course. 2 or more developers working on the same application, one using WebIDE and one using a local development toolchain.

- Like(0)

Peter Muessig Post author
April 16, 2020 at 5:53 pm
A centrally managed IDE has some limitations on which tools can be used and in case of you want to extend this IDE with additional plugins to improve your development experience then this can only be done on your local IDE. There you have everything under control and you can plugin different tools as you want and as you need.

- Like(0)

Pascal Gutheil
April 16, 2020 at 6:37 pm
Cool! Thought about that when the 2.0 tooling was released, but a small caveat is that so far only the version 1.76 of the SAPUI5 libraries are released via the npm channel. A lot of customers run 1.71 due to the announced maintenance duration, but I guess older versions are not brought to npm?

- Like(0)

Peter Muessig Post author
April 17, 2020 at 8:23 am
Hi Pascal,
thanks for your feedback. Unfortunately, supporting 1.71 would be a lot of work. But as of today, you can either use a proxy middleware e.g. ui5-middleware-simpleproxy or use the ui5-middleware-servestatic and download a 1.71 runtime from https://tools.hana.ondemand.com/#sapui5.
HTH and BR,
Peter

- Like(0)

Robert Eijpe
April 17, 2020 at 5:34 pm
Thanks Peter, a very helpful blog

- Like(0)

Find us on

- Privacy
- Terms of Use
- Legal Disclosure
- Copyright
- Trademark
- Cookie Preferences
- Newsletter
- Support

Inserted from <https://blogs.sap.com/2020/04/15/ui5ers-buzz-51-ui5-tooling-2.0-how-to-develop-and-run-sap-fiori-elements-locally/>

UI5 Tooling: a modern development experience for UI5! | SAP Blogs

Friday, April 17, 2020 3:54 PM

Peter Muessig

Posted on April 7, 2020 8 minute read

UI5 Tooling: a modern development experience for UI5!

Introduction

On April 1st I talked about the *"UI5 Tooling – a modern CLI-based development experience"* in a SAP Community call. Besides explaining the UI5 Tooling in general, I showed how to grab a UI5 application from SAP Web IDE and how to configure it to run it locally with the UI5 Tooling.

The slides can be found here: https://www.slideshare.net/PeterMuessig1/ui5-tooling-ecosystem

This blog explains the individual steps to you to make the same experience.

Background

Approximately one year ago, the SAP Web IDE switched to use the UI5 Tooling to build UI5 projects. Today, all UI5 projects created with the SAP Web IDE include the proper configuration to use the UI5 Tooling for the build process. As the SAP Web IDE comes with an integrated preview based on the HTML5 runtime, the development server of the UI5 Tooling is not required here. But when grabbing projects from SAP Web IDE and running them locally, the development server is

mandatory. Compared to the HTML5 runtime, the UI5 Tooling just provides access to the UI5 resources and doesn't provide a proxy to access backend services. But without using mockdata, such a proxy is needed.

Extensibility

Last year, at the UI5con at SAP, Volker Buzek approached me and asked for a proxy as part of the UI5 Tooling. I asked Volker what kind of proxy he needs: a simple proxy, an AppRouter, a HTML5 runtime router, …? All proxies are somehow specific to their platform, coming with different configurations and IMO the UI5 Tooling team will never satisfy all demands. But I offered Volker to help him to write a proxy and to get started with the extensibility of the UI5 Tooling demonstrating the task and middleware extensibility. This was the kick-start for the UI5 Ecosystem Showcase repository – a repository to demonstrate how easy the UI5 Tooling can be enhanced with own custom tasks and middlewares.

UI5 Ecosystem Showcase

The UI5 Ecosystem Showcase repository contains several ideas to improve the UI5 development experience by leveraging open-source tools. The extensibility concept of the UI5 Tooling allows to integrate those tools easily. So, the repository step by step grow to integrate custom tasks to e.g. transpile ES6 to ES5 code by using Babel or to provide custom middlewares to get a livereload of the browser in case of changes in the project sources as well as a simple HTTP proxy to tunnel backend services. All those features can be combined and used for UI5 development to come to a modern development experience!

A real-life example: Local UI5 development with the UI5 Tooling

Now, let's start with some hands-on and let's make our hands dirty. Every UI5 developer should make this experience to feel how good local UI5 development can be already today. Keep in mind, the SAP Web IDE still provides a lot of value and benefits with its wizards, tools and integration into various SAP platforms. But IMO it should be seamlessly possible to either develop a UI5 application in SAP Web IDE and locally in e.g. Visual Studio Code together with the UI5 Tooling.

Prerequisites

Before we start, we need to ensure that our computer is setup properly. Node.js 10+ should be installed and configured properly.

Step 1: Create a UI5 application

In the first step we create a UI5 application based on the SAP Fiori Worklist Application template. This can be done either in SAP Web IDE by using the template wizard or with the Yeoman generator for SAPUI5 templates.

Option A) Using the SAP Web IDE template wizard

Follow these steps to create an SAP Fiori Worklist application:

1. Open SAP Web IDE
2. File > New > Project from template
3. Select environment "Neo" (as this affects the datasource URL relevant for the proxy)
4. Select category "SAP Fiori Application"
5. Select template "SAP Fiori Worklist Application"
6. Enter your project name, title and namespace
7. In the data connection step (prerequisite: setup a destination to Northwind OData Services first), select the Northwind OData Services as "Service URL", use the "Relative URL": /V2/Northwind/Northwind.svc/) and press "Test".
8. In the template customization step, select the "Standalone App" application type and Object

Collection: Customers, Object Collection ID: CustomerID, Object Title: CompanyName.

9. Finish
   After the UI5 application has been created, select it in the project explorer, export it from SAP Web IDE and unzip it locally in your file system.

Option B) Using the Yeoman generator for SAPUI5 templates

Follow these steps to create an SAP Fiori Worklist application:

1. Install the Yeoman generator via:
   `npm install -g yo @sapui5/generator-sapui5-templates`
2. Create an empty folder and switch to it
3. Run:
   `yo @sapui5/sapui5-templates`
4. Enter module name and module namespace
5. Select template “SAP Fiori Worklist Application”
6. Enter title and description
7. Select “Standalone App”
8. As Service Base URI enter: https://services.odata.org/V2/Northwind/Northwind.svc/
9. As Datasource URL enter: /svc/
10. In the following steps enter Object Collection: Customers, Object Collection ID: CustomerID, Object Title: CompanyName, the rest leave empty.
    Now your UI5 application has been created in your local file system.

Step 2: Updating the project dependencies

After following one of the options of the previous step, we should validate whether we need to update the project dependencies of the created UI5 application. To update the project dependencies, we can use the open-source tool npm-check-updates. This tool can be installed via:

`npm install -g npm-check-updates`
After the installation of the tool, we can run:

`ncu -u`
to upgrade the project dependencies in the package.json. Once the project dependencies are upgraded run:

`npm install`
to prepare the project for a first run. The project can be started with the command:

`npx ui5 serve -o index.html`
Although the development server starts technically, there are a few things missing to make the UI5 application to run properly (e.g. the SAPUI5 resources).

Step 3: Setting up a proxy to access the SAPUI5 resources

To access the SAPUI5 resources in the UI5 application we can simply use the simple proxy middleware from the UI5 Ecosystem Showcase. I can be installed as a dev dependency by running the following command:

`npm install ui5-middleware-simpleproxy --save-dev`
(if the project has been generated with Yeoman the simple proxy middleware is already installed).

Option A) UI5 application created by SAP Web IDE

For the project created with the SAP Web IDE, we need to add the ui5-middleware-simpleproxy as ui5/dependencies in the package.json:

```
{
  […]
  "ui5": {
    "dependencies": [
      "ui5-middleware-simpleproxy"
    ]
  },
  […]
}
```

Afterwards we need to open the ui5.yaml and add the following configuration at the end:

```
server:
  customMiddleware:
  - name: ui5-middleware-simpleproxy
    mountPath: /resources/
    afterMiddleware: compression
    configuration:
      baseUri: https://sapui5.hana.ondemand.com/resources/
```

This configuration will ensure that all requests to /resources/ will be proxied to https://sapui5.hana.ondemand.com/resources/.

Option B) UI5 application created by Yeoman generator

If the project has been created with the Yeoman template, we need to add the following configuration to the server/customMiddleware section in the ui5.yaml (just include the first middleware entry and put it before the already existing ui5-middleware-simpleproxy entry):

```
server:
  customMiddleware:
  - name: ui5-middleware-simpleproxy
    mountPath: /resources/
    afterMiddleware: compression
    configuration:
      baseUri: https://sapui5.hana.ondemand.com/resources/
  - name: ui5-middleware-simpleproxy
    mountPath: /proxy/
```

But be careful with yaml – the indentation is pretty important.

Now the application should start properly and load the SAPUI5 resources via the proxy middleware. But still the UI5 application created with SAP Web IDE will fail to load its data since there is no proxy configured which tunnels the requests to the Northwind OData Service.

Step 4: Setting up a proxy to tunnel the Northwind OData Service

This step is only needed for the project created by SAP Web IDE. The project created with the Yeoman generator already includes the configuration to connect to the Northwind OData Service.

In the project created with SAP Web IDE open the ui5.yaml and add the following configuration (just include the second entry for the OData service into the ui5.yaml):

```
server:
  customMiddleware:
  - name: ui5-middleware-simpleproxy
    mountPath: /resources/
    afterMiddleware: compression
    configuration:
      baseUri: https://sapui5.hana.ondemand.com/resources/
  - name: ui5-middleware-simpleproxy
    mountPath: /Northwind/V2/Northwind/Northwind.svc/
    afterMiddleware: compression
    configuration:
      baseUri: https://services.odata.org/V2/Northwind/Northwind.svc/
```

This configuration ensures to proxy the request to /Northwind/V2/Northwind/Northwind.svc/ to https://services.odata.org/V2/Northwind/Northwind.svc/.

Step 5: Validate that the project runs locally

Now it's the time to validate that the project starts locally. To ensure that all project dependencies are installed, just run:

```
npm install
```

in the UI5 application again. Now you can start the development server by running:

```
npx ui5 serve -o index.html
```

Now you should see the application running and the data being loaded from the Northwind OData Service.

Step 6: Add the livereload middleware

The livereload middleware is watching to changes of the sources in the file system. Once a source file has been modified the livereload middleware reloads the browser. Add the livereload middleware with the following command:

```
npm install ui5-middleware-livereload --save-dev
```

Afterwards in the package.json ensure to list the ui5-middleware-livereload in the ui5/dependencies (just add the last entry):

```
{
  […]
  "ui5": {
    "dependencies": [
      "@sap/ui5-builder-webide-extension",
      "ui5-middleware-simpleproxy",
      "ui5-middleware-livereload"
    ]
  },
  […]
}
```

The first entry @sapui5/ui5-builder-webide-extension is only present when you started to create the project with the SAP Web IDE. Afterwards, in the ui5.yaml just include the livereload middleware:

```
server:
  customMiddleware:
  […]
  - name: ui5-middleware-livereload
    afterMiddleware: compression
```

```
    configuration:
      ext: "xml,json,properties"
      path: "webapp"
```
Now you can start your development server again with the command:

```
npx ui5 serve -o index.html
```
When you modify your sources and immediately get an update in your running application (remark: in case you face issues with livereload, maybe your html page was cached before, just ensure to reload the html page properly).

Step 7: Use the UI5 Tooling 2.0 to consume SAPUI5 from NPM

With the availability of SAPUI5 on NPM and the UI5 Tooling 2.0 we finally can consume SAPUI5 as dependencies. Therefore, the UI5 Tooling implemented an own frontend package manager to overcome the versioning issue in the SAPUI5 distribution layer. The details and reasoning behind this decision can be read in the [UI5ers Buzz #49: The UI5 Tooling and SAPUI5 – The Next Step](#).

Let's experience it live – we now remove the ui5-middleware-simpleproxy and migrate to specVersion: '2.0' in the ui5.yaml:

```
specVersion: '2.0'
[…]
server:
  customMiddleware:
  - name: ui5-middleware-simpleproxy
    mountPath: /Northwind/V2/Northwind/Northwind.svc/
    afterMiddleware: compression
    configuration:
      baseUri: https://services.odata.org/V2/Northwind/Northwind.svc/
  - name: ui5-middleware-livereload
    afterMiddleware: compression
    configuration:
      ext: "xml,json,properties"
      path: "webapp"
```
Next we add a framework dependency to SAPUI5 by running the following command:

```
npx ui5 use sapui5@latest
```
Now we can include the required libraries with the following command:

```
npx ui5 add sap.ui.core sap.m sap.f themelib_sap_fiori_3
```
This will create the following section at the end of the ui5.yaml:

```
specVersion: '2.0'
[…]
framework:
  name: SAPUI5
  version: "1.76.1"
  libraries:
    - name: sap.f
    - name: sap.m
    - name: sap.ui.core
    - name: themelib_sap_fiori_3
```
In case of the project has been created with the Yeoman template, open the index.html and change the bootstrap URL from "https://sapui5.hana.ondemand.com/resources/sap-ui-core.js" to "resources/sap-ui-core.js". After starting the development server with:

`npx ui5 serve -o index.html`
the UI5 resources are loaded from the local framework dependencies configured in the ui5.yaml.

Step 8: Explore the UI5 Ecosystem Showcase repository and add more middlewares or tasks

Now you can go ahead and take a look into the UI5 Ecosystem Showcase repository to include additional tasks and middlewares into the ui5.yaml of your project. If you want to use modern JavaScript language features, I would recommend you to take a look into the transpile task and the livetranspile middleware. If you have ideas for additional custom tasks or custom middlewares please consider to share them with the UI5 community.

What now

Get familar with the UI5 Tooling 2.0 and the new capabilities to consume SAPUI5 dependencies from NPM. Play around with the available tasks and middlewares and try to combine them for your local UI5 development.

Help us to spread the word and share with other UI5 developers that you can also have a great and modern development experience for UI5 development.

Related links

- UI5 Evolution: The Build and Development Tooling
- Introducing the UI5 Tooling for SAPUI5 Projects
- The UI5 Tooling and SAPUI5
- Q/A for the SAP Community call about the UI5 Tooling

- Alert Moderator

Assigned tags

- SAPUI5
- Open Source
- SAP Fiori
- SAP Web IDE
- User Interface
- #sapcommunitycall
- openui5
- ui5tooling

View more...
Related Blog Posts
Related Questions
6 Comments
You must be Logged on to comment or reply to a post.


Fatih Pense
April 12, 2020 at 2:57 am
Hello Peter,
I have followed this guide with the Yeoman option and committed the changes when the step is complete. So if anyone finds it useful to check, the Git commits are here:
https://github.com/fatihpense/ui5-tooling-example/commits/master
Also, it may be a good idea to include a default .gitignore file in the Yeoman templates.
I have just watched the video and I think I have an ancient idea for an ui5-task-*
I want to contribute it to the community repo in the hope that I will get a cut/sip from each donated drink. I will also benefit from your code review!
I still remember my excitement and increased confidence that SAP can adapt to modern technologies when OpenUI5 was first announced as open-source. These tooling improvements reminded me of those times. I want to thank you for making things more open.
Regards,
Fatih

- Like(1)


Peter Muessig Post author
April 12, 2020 at 6:19 am
Hi Fatih,
thanks for creating the GitHub repository – this is really great for all to follow the steps. I also had in mind to create it but finally I missed it. Big thanks for your work! 😳
I will add the .gitignore to the Yeoman templates. This is indeed a good idea for the standalone usage. In case of the usage in SAP Business Application Studio, it takes care. to create the .gitignore for us.
If you are interested in a contribution to the ui5-ecosystem-showcase, I would be more than happy to share some sips with you… 😉 Just create a PR and assign me as reviewer – and please don't forget to add yourself as one of the listed people in the derived beerware license statement in the README.md of you task…
Thanks for your nice words – I will make sure that the UI5 developers will receive them! I am more than happy if people enjoy working with UI5 and also benefit from all the things happening around in the JavaScript community.
Best regards,
Peter

- Like(0)


umit duran
April 13, 2020 at 5:41 pm
Great!

- Like(0)


Ludo Noens
April 16, 2020 at 1:12 am
If you initially created the project in SAP Web IDE Full-Stack and followed the tutorial for creating the destination, then the simple proxy middleware configuration to access the data source should reflect the name of the destination you've created. In this case, Peter used a destination with the name

‘northwind_odata_services’.
mountPath: /northwind_odata_services/V2/Northwind/Northwind.svc/
If you’ve followed the tutorial, this should be
mountPath: /Northwind/V2/Northwind/Northwind.svc/
Cheers,
Ludo

- Like(1)

Peter Muessig Post author
April 16, 2020 at 5:59 am
Thanks, Ludo for sharing this. 😊
Yes, there is indeed a difference for which environment you create the template. I used “Neo” in the tutorial above. I will add a hint to that.

- Like(0)

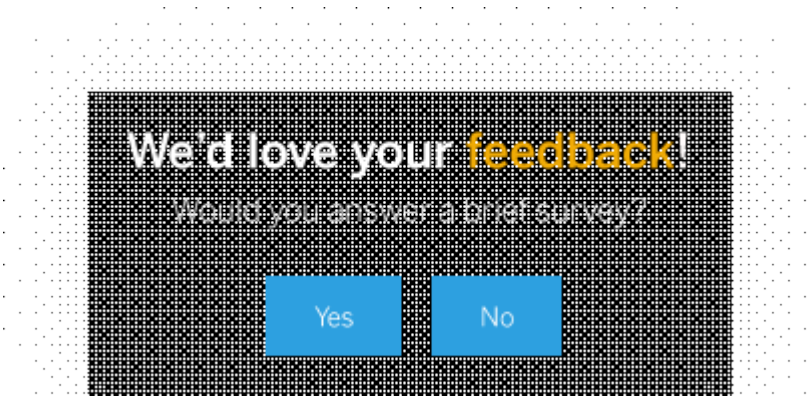
Peter Muessig Post author
April 17, 2020 at 8:42 am
Ahh – got it – I somehow missed that the tutorial I am referring to for the creation of the Northwind service uses “Northwind” as destination name and not “northwind_odata_services”. I fixed this now in the blog post.
Thx, Ludo!

- Like(0)

Find us on

- Privacy
- Terms of Use
- Legal Disclosure
- Copyright
- Trademark
- Cookie Preferences
- Newsletter
- Support

Visual Text
Paragraph



Inserted from <https://blogs.sap.com/2020/04/07/ui5-tooling-a-modern-development-experience-for-ui5/>

Test ODATA service

Friday, April 17, 2020 4:00 PM

https://services.odata.org/V2/Northwind/Northwind.svc/


services.odata.org/V2/Northwind/Northwind.svc/
Click entity for XPath. Double-click to collapse/expand. Enter XPath or XML string then clic
Tree-ify, respectively.
<collection href="Alphabetical_list_of_products">
<atom:title>Alphabetical_list_of_products</atom:title>
</collection>
<collection href="Category_Sales_for_1997">
<atom:title>Category_Sales_for_1997</atom:title>
</collection>
<collection href="Current_Product_Lists">
<atom:title>Current_Product_Lists</atom:title>
</collection>
<collection href="Customer_and_Suppliers_by_Cities">
<atom:title>Customer_and_Suppliers_by_Cities</atom:title>
</collection>
<collection href="Invoices">
<atom:title>Invoices</atom:title>
</collection>
<collection href="Order_Details_Extendeds">
<atom:title>Order_Details_Extendeds</atom:title>
</collection>
<collection href="Order_Subtotals">
<atom:title>Order_Subtotals</atom:title>
</collection>
<collection href="Orders_Qries">
<atom:title>Orders_Qries</atom:title>
</collection>
<collection href="Product_Sales_for_1997">
<atom:title>Product_Sales_for_1997</atom:title>
</collection>
<collection href="Products_Above_Average_Prices">
<atom:title>Products_Above_Average_Prices</atom:title>
</collection>
<collection href="Products_by_Categories">
<atom:title>Products_by_Categories</atom:title>
</collection>
<collection href="Sales_by_Categories">
<atom:title>Sales_by_Categories</atom:title>
</collection>
<collection href="Sales_Totals_by_Amounts">
<atom:title>Sales_Totals_by_Amounts</atom:title>
</collection>
<collection href="Summary_of_Sales_by_Quarters">
<atom:title>Summary_of_Sales_by_Quarters</atom:title>

SAP Fiori Tutorial. Part 8 – How to Extend Standard Fiori App – both UI & OData? |

Wednesday, March 10, 2021 8:16 AM

Clipped from: https://sapyard.com/sap-fiori-tutorial-part-8-how-to-extend-standard-fiori-app-both-ui-odata/amp/

Karan Shaheri

In June 2017 we prepared a flow chart showing the *steps to follow if we need to enhance any standard SAP Fiori Application*. We assumed ABAPers have learned the trade of Fiori development and we did not pursue to post the detailed tutorial. But in the last few weeks, we started getting requests for the step by step process to extend any standard Fiori Application both at front-end and back-end sides.

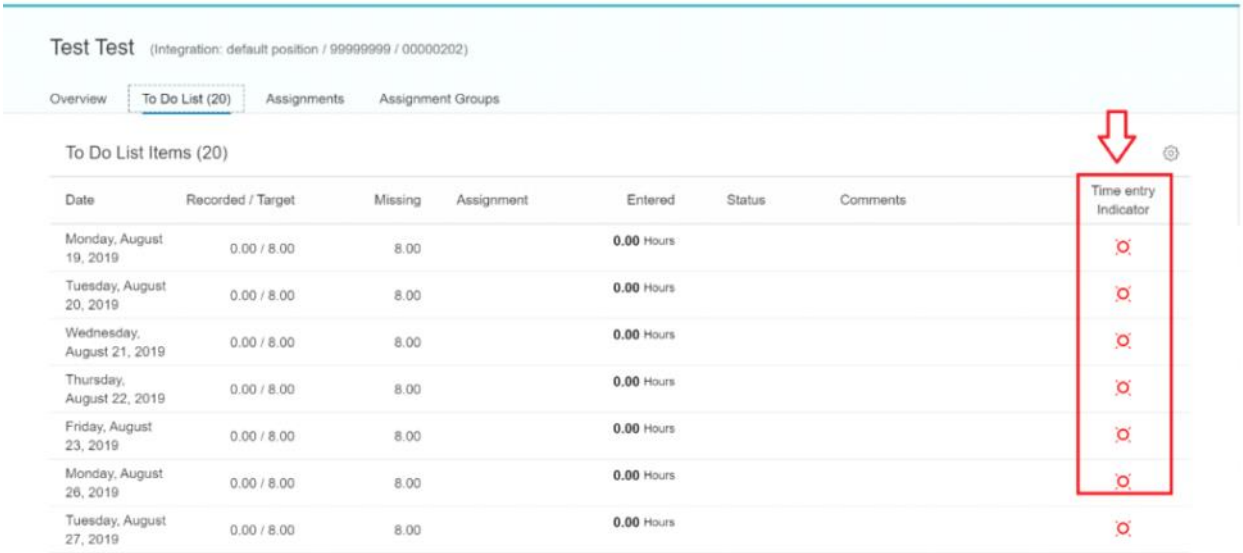
So here we come. In this article, we would see the steps to extend a standard Fiori application. There may be times when process for a particular customer can deviate from the process used in Standard Fiori application. Hence, to suffice the customer requirements, we may need to extend standard Fiori Application.

Disclaimer – The idea for featured image **Fiori** to **Yeti** is inspired from the blog "PART 2: SAP-WHAT'S NEXT AFTER S/4HANA" written by James Olcott.

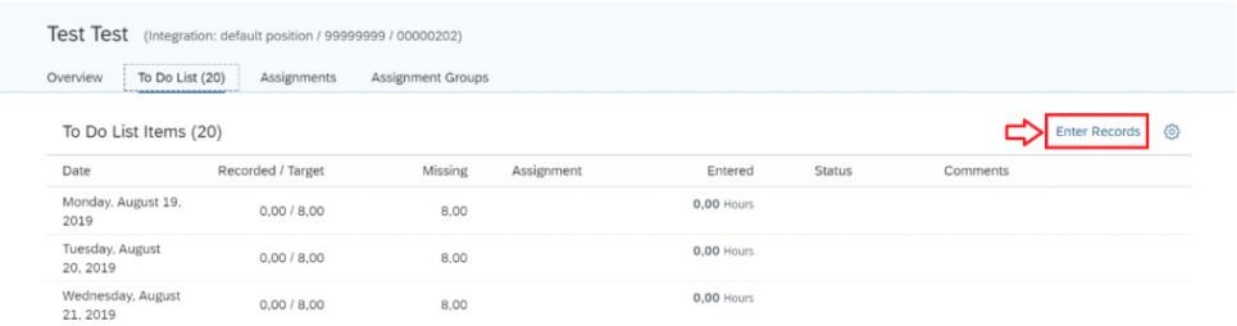
We understand, standard app will not suffice all business process of the clients. But clients should also consider that if we try to put all custom business logic to the Fiori App, eventually it will turn into a monster. *We should always try to follow the 5 basic principles of Fiori App i.e. Role-based, Simple, Delightful, Coherent and Responsive*.

Having said that, let come back to our tutorial. We will be extending **My Timesheet Fiori** application. Our goal today is to meet the below two requirements.

1. Add a **New Column** called "**Time Entry Indicator**" to the "**To Do List**" table which will indicate a **red** icon *if the Actual Entered Hours are less than the Target hours expected*.



2. We will **remove** "**Enter Records**" button from "**To Do List**" table toolbar



If you are an ABAP developer working on Fiori project or the UI developer in the Fiori project, you need to understand that to meet the above two goals, we need to **enhance backend OData** to pull the information and calculate and also we need to **extend the frontend UI screen** to display the output.

Following steps will be carried out to make the two changes mentioned above.

1. *We will **add** custom field "**ZZICON**"* to Include structure "**HCMFAB_S_TIMESHEET_WCALE_INCL**" which is present in the structure "**HCMFAB_S_TIMESHEET_WORKCALE**" as this structure is used to create **OData** entity "**ActionItem**" for **To Do List** table.
2. ***Redefine OData** service "**HCMFAB_TIMESHEET_MAINT_SRV**"* to include custom field in OData entity "**ActionItem**".
3. ***Enhance** the Standard Data provider class* and implement **Overwrite-Exit** of method "**ACTION_ITEMS_GET_DEEP_ENTITY**" to include custom field as there is no BADI available to add custom field logic.
4. ***Extend** standard "**My Timesheet**"* fiori application "**HCMFAB_MY_TIME**" to include the custom field on the UI and hide the "**Enter Records**" button.

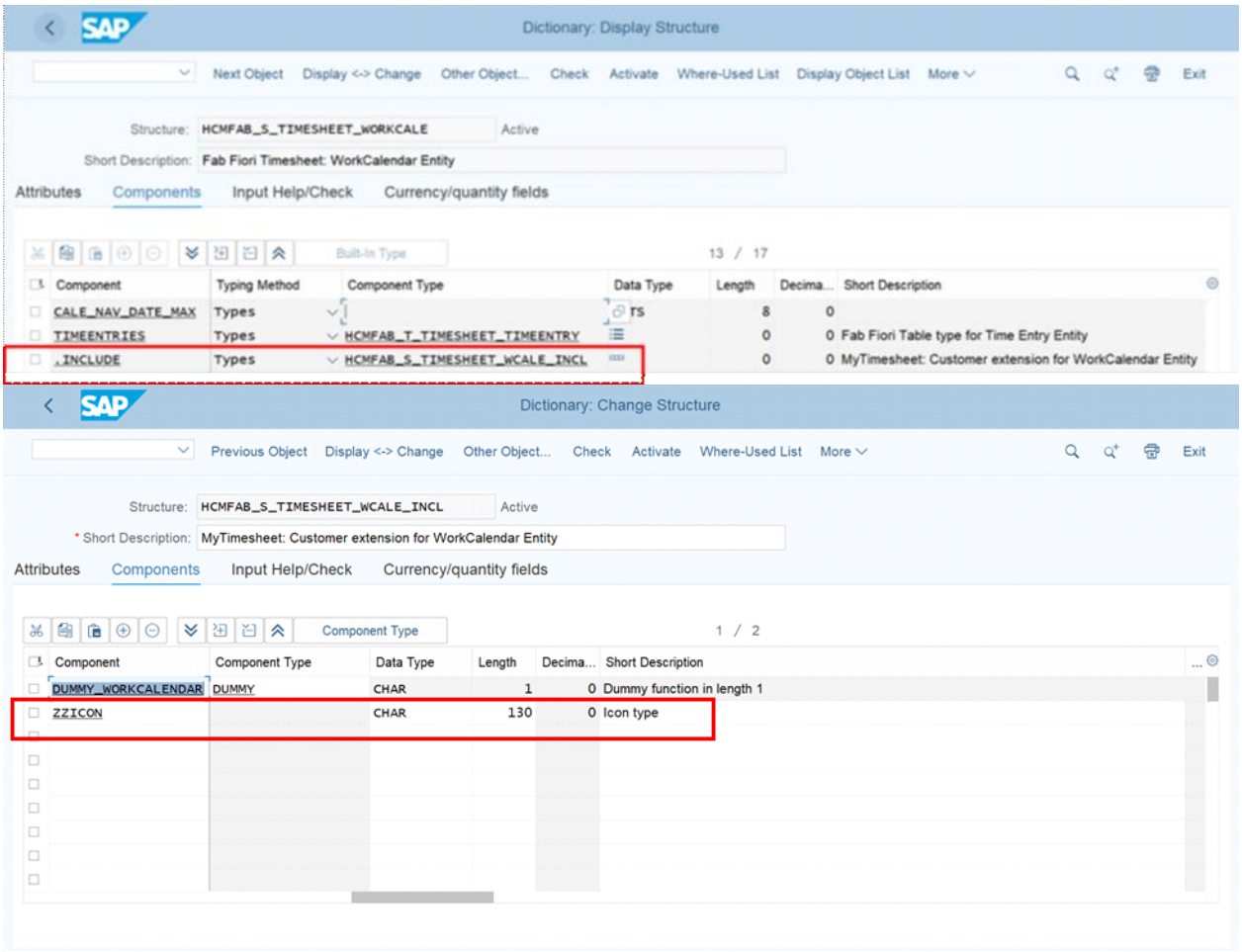
Also Read – SAP Fiori Tutorial. Part I. System Check, Installation and Configuration

Let's start our journey of Extending the standard Fiori App *or should we say, evolution of Fiori to Big Foot, Yeti*? 😉

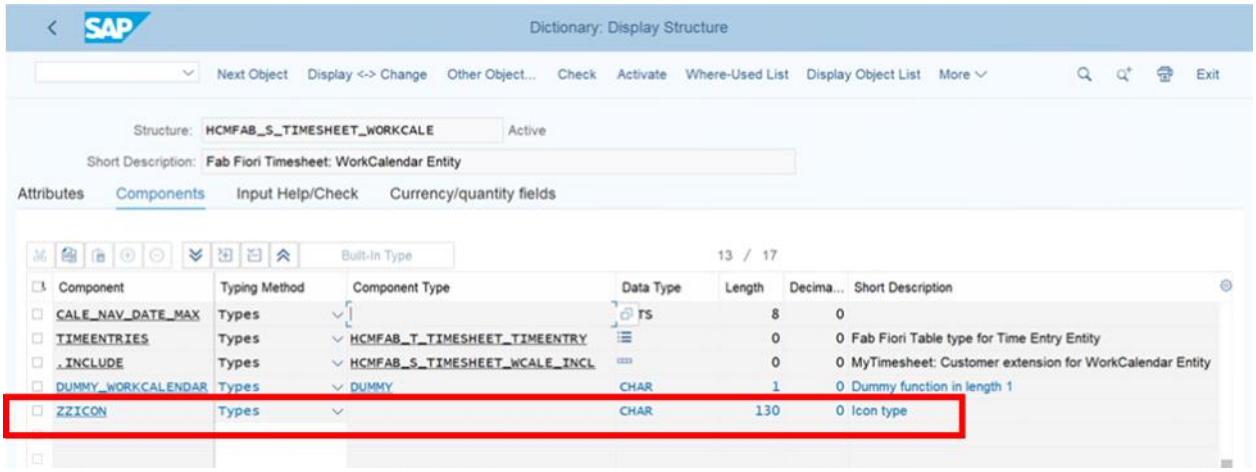
Part 1. Steps to Enhance ABAP Backend OData Service

Step 1. Enhance Structure with New Custom Field(s)

Add custom field "**ZZICON**" to Include structure "**HCMFAB_S_TIMESHEET_WCALE_INCL**" which is present in the structure "**HCMFAB_S_TIMESHEET_WORKCALE**" as shown in below screenshot.



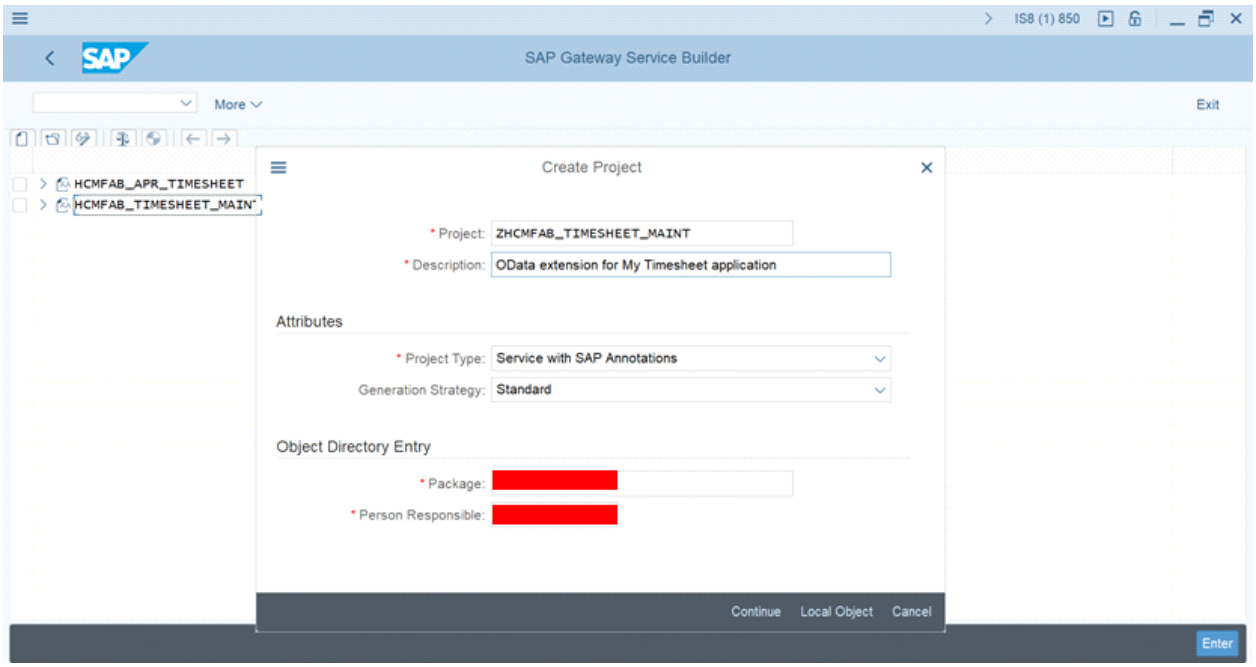
We can see now in below screenshot that custom field "**ZZICON**" is available in the structure **HCMFAB_S_TIMESHEET_WORKCALE**.



Step 2. Redefine Standard OData Service

Now we will Re-define the standard OData service "**HCMFAB_TIMESHEET_MAINT_SRV**" to include custom field in OData entity "**ActionItem**".

Create a new project using SEGW with details as provided in below screenshot.



Right click on "**Data Model**" folder. Select Redefine and then select "**OData Service (SAP GW)**" to select Standard OData service to be re-defined.

Click on F4 Help on "**Technical Service Name**" field to select the OData service "**HCMFAB_TIMESHEET_MAINT_SRV**" and its version as shown in below screenshots.

Please Note – You can easily find the backend OData Service name and frontend UI Service name from the **Fiori Library**. **Check this End to End Fiori Tutorial for Beginners to learn more about Fiori Library**

Once the service is selected, click on "**Next**" button to select the entities of the standard OData service.

Select all the entities and click on "**Finish**".

Important: *Here we will have to select all the entities, but we will change only "**ActionItem**" entity to add our custom field.*

Right click on "**ActionItem**" entity. Select "**Properties**" under the "**Import**" option.

Now we can see our custom field "**ZZICON**" that we added to the custom include of standard structure "**HCMFAB_S_TIMESHEET_WORKCALE**". Select the custom field "**ZZICON**" and press "**Next**".

Now change the property name of our custom field to "**Icon**" from automatically suggested property name by the tool as shown in below screenshot.

We changed to Icon, as we want to show it. If you do not want Icon, you may leave it as it is.

We can see in the below screenshot that custom property "**Icon**" is added to "**ActionItem**" entity highlighted in yellow.

Click the button to generate the runtime objects of OData service. Below pop-up will be come suggesting data provider class names, model name, service name. Tick the "**Overwrite Base/Extended Service**" checkbox and click continue to generate runtime objects.

Also Watch – If you are new to OData, take this 2 Hours Free Video Course on SAP OData Services for Beginners.

Step 3. Enhance Standard DPC_EXT Class for Custom Field Logic

We will now enhance the standard class "**CL_HCMFAB_TIMESHEET_CR_DPC_EXT**" to include logic for custom field *as **no** BADI is available to add logic for custom field*. Click on "**Enhance**" button to create an Enhancement implementation

Create **Overwrite-Exit** for method "**ACTION_ITEMS_GET_DEEP_ENTITY**" to add logic for custom field

We will add the **error icon** if missing hours are greater than 0 else, we will add the **success icon** as shown in below screenshot.

Step 4. Register the Custom Enhanced OData Service

Backend development for adding custom field is now completed. *We will now register our customer service using transaction* "**/IWFND/MAINT_SERVICE**". If "**Centralized Hub**" architecture is used in

the project, *login to the Frontend Server and register the service* else for "**Embedded System**" register the service in the same server.

Go to transaction code "**/IWFND/MAINT_SERVICE**". Click on "**Add Service**" button highlighted in yellow as shown in the screenshot.

Select the system alias pointing to backend server and click on "**Get Services**" button to get all the backend services which are not registered.

Filter on our custom service which we earlier created and click on "**Add Selected Services**" button to register in the frontend server.

Popup will be opened with the service details. By default, Technical service name, Technical Model name will be auto-populated. You can modify these values if you want and press OK to register the service.

Once service is registered successfully, we will get the message that *service is created and metadata loaded successfully* as shown in below screenshot.

We can see that service is now shown as registered with backend system alias.

Also Check – SAP Fiori Tutorial. Part II. End to End Implementation of Fiori App

Part 2. Steps to Enhance Frontend UI Service

OData extension is completed. We will start with extending UI5 application "**HCMFAB_MY_TIME**". How did we find out the UI Service name? Fiori Library is the Bible.

Step 1. Enhance the Fiori UI Service using Extensibility Pane

Login to **WebIDE** on SAP Cloud platform. Click on "**New**" option from "**File menu**" and select "**Extension Project**" as shown in below screenshot.

Click on "**Select Application**". It will show 2 options. "**SAPUI5 ABAP Repository**" and "**SAP Cloud Platform**". *For our scenario as application resides in "**SAPUI5 ABAP Repository**", this option is selected*.

Now search for the UI5 application "**HCMFAB_MY_TIME**" from the ABAP repository to be extended.

Provide a name for extension project. *Tick the checkbox if you want the extended SAPUI5 application to hav*e the same version as SAPUI5 ABAP Repository version.

Select the option "**Open extension project in extensibility pane**" and click on **Finish**. New project which we created will now be opened in extensibility pane.

Step 2. Check Fiori Library for Extension Points

We now must add the custom column to the **To Do List** table. We will now check in Fiori App Library, if extension points are available to add custom column to the standard application. *As shown in below screenshot, we do have the extension available highlighted in yellow to add the custom column.*

We will now find the extension point in the extensibility pane and select that extension. Now select the "**Extend View/Fragment**" option inside the "**Extend**" option.

Click on the "**Open Extension Code**" option to see the extended code in the new project we created.

There is a new entry with "**sap.ui.viewExtensions**" in **manifest.json** file for the extension point as shown in below screenshot.

We will now add a custom column called "**Time Entry Indicator**" in the fragment which was created with extension.

Step 3. Extend i18n File

Text for column name is maintained in **i18n file.** *As we are extending standard application, we will now have to extend i18n model of standard application to add our column name text in i18n file*. Below screenshots shows the process to extend i18n model of standard Fiori application.

Select the extension project and click on **File -> New-> Extension**.

Keep the project name as it is. Click on "**Next**" .

Select "**i18n Resource Text Customization**" option and click on "**Finish**".

We can now add text of column name in i18n file.

We can now see standard application is now extended and custom column is added to the application.

We now must add the **custom cell** to row of the **To Do List** table. We will now check in Fiori App Library, if extension points are available to add cells to the row of To Do List table in the standard application. *As shown in below screenshot, we do have the extension available highlighted in yellow to add the custom column.*

We will now find the extension point in the extensibility pane and select that extension. Now select the "**Extend View/Fragment**" option inside the "**Extend**" option.

Application is now extended to add custom Cell to the row of **To Do List** table. Click on "**Open Extension Code**" to see the extended code.

We can now see in **manifest.json** file that a new extension is added to "**sap.ui.viewExtensions**" section as shown in below screenshot.

Add a "**Icon**" control from "**sap.ui.core**" library to the column cell. *Bind the property "**Icon**" of OData entity "**ActionItem**" to the **src** attribute of Icon control to bind the icon name*.

Also Learn – SAP Fiori Tutorial. Part VI. How to Troubleshoot SAP Fiori Errors?

Step 4. Extending Controller to Add Logic for Custom Field

We will first check if model binding on UI will **automatically display** the value in custom column. *This may depend on how the code has been written in the controller of the application.* For our case, the value for the custom field doesn't pull up on the screen. We should now check for **controller extension** in the Fiori App Library. These are called as **Hook Methods** in SAPUI5 terminology.

We don't have any hook methods for this application. Only option we now have is **replacing the controller with our custom controller** and **add additional logic in the relevant method**. *We will now replace the Worklist Controller with our Custom Controller to add the logic for "Icon" field.* After having a look at controller code, we found a method "**getToDoList**" and adding our OData custom property to the "**TodoList**" model which we bound to the XML view in the previous step will solve our purpose.

Now let's see how to extend the controller.

We will now find the **Worklist Controller** in the **extensibility** pane. Select the "**Extend with a Copy of Original Controller**" option inside the "**Extend**" option.

Worklist Controller is now extended to add custom our custom field. Click on "**Open Extension Code**" to see the extended code.

Caution: As shown in popup, there are implications in replacing the original controller. Future upgrades to the original controller will not automatically reflect in our custom controller. We suggest keeping Controller replacement as the last resort if no extensible methods are available in standard controller.

We can now see in **manifest.json** file that a new extension is added to "**sap.ui.controllerExtensions**" section as shown in below screenshot.

As discussed earlier, we found a method "**getToDoList"** which will be modified to add our custom property to the model. We will now assign "**Icon**" attribute of OData entity to the newly created "**Icon**" property in the "**ToDoList**" model as shown below highlighted in yellow.

Step 5. Replace the Standard Service with Custom Extended OData Service

Next step is to replace the standard service with the new service which we have created earlier. Click on **File->New->Extension**. Popup will be shown.

Keep the project name as it is and click on "**Next**".

Select "**Replace Service**" option and click on "**Next**".

Search for the new service created and select the service. Click on "**Finish**".

"**dataSources**" section is now added in manifest.json with our custom OData service.

We can now run the application. Icon's are now displayed in the "**Time Entry Indicator**" column and are displayed in "**Red**" color as "**Missing hours**" are greater than 0.

Step 6. How to Hide any Control in Standard Fiori App?

We will now see how to hide any control of the standard Fiori application. For our scenario, we will hide "**Enter Records**" button in the toolbar of "**To Do List**" table.

Pre-requisite to hide any control of standard Fiori application is that **controls should be associated with id's.**

We can see from below screenshot that "**Enter Records**" button has id as "**editButtonTodo**" in Worklist XML view.

Click on **File->New->Extension**. Popup will be shown.

Keep the Project name as it is and click on "**Next**".

Select the "**Hide Control**" option and click on "**Next**"

Select the view where control has to be hidden. In our case, we will select "**Worklist**" as view and UI control id as "**editButtonTodo**" which we identified earlier and click on "**Finish**".

We can now see a new entry "**sap.ui.viewModifications**" is added in **manifest.json** file with the control that needs to be hidden.

With this we have hidden the button "**Enter Records**" in "**To Do List**" table. We can now see that button is hidden from the toolbar of "**To Do List**" table.

I have tried to show all the tricks you need to extend/enhance any Fiori App. *Most of the Fiori extension requirements would be to show new field on the header or new column in the table. You might also need to hide or display some texts or buttons.* Usually, client would not come up to you and say, give me a brand new Tab or completely new Screen in the standard Fiori App. *If they have that requirement, probably then it is a good candidate for custom SAPUI5 App development.*

This is my first tutorial at SAPYard and I thoroughly enjoyed this process of compiling the tutorial and sharing the little knowledge I have gained over the period to the SAP Community. *Your suggestions and criticism would help me come up with better topic.* So please let the comments flow in.

Comments & Questions Please

Do join 5490+ SAP Technical Consultants in this **Telegram SAP Technical Discuss Group.** Ask, Answer and Learn is our Motto. You need to install Telegram App first in your mobile or desktop and then click the joining link.

Please SUBSCRIBE to SAPYard's Youtube Channel for Free End to End SAP Video Course and Training.

Step by Step Tutorials on SAP Fiori

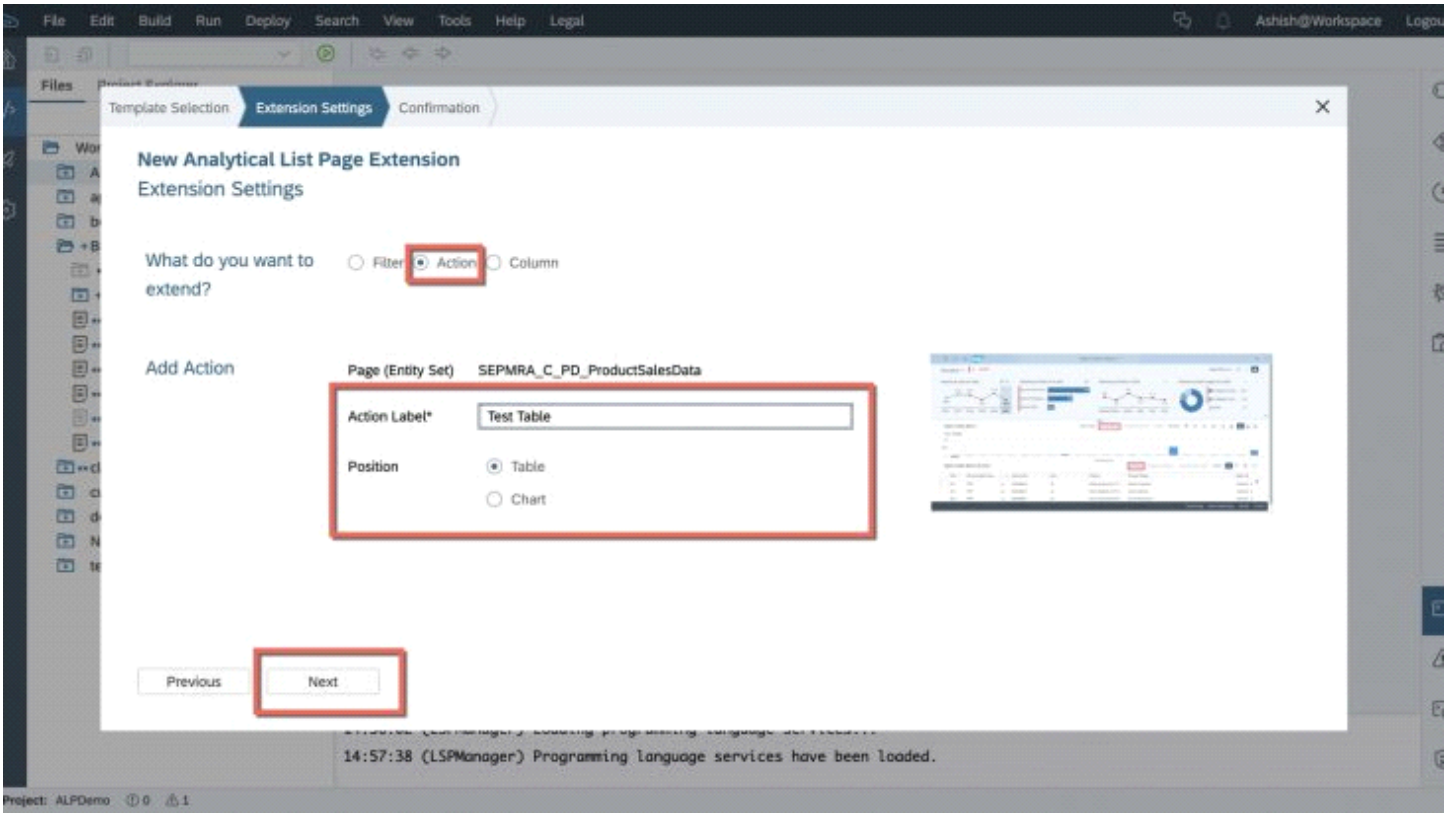
Analytical List Page

Thursday, March 11, 2021 9:42 AM

Analytical List Page (ALP) developer extension - SAPSPOT

Thursday, March 11, 2021 9:42 AM

Clipped from: http://www.sapspot.com/analytical-list-page-alp-developer-extension/



In this blog, I'll discuss various extension points made available to the application developers by ALP for example custom actions, custom filters, table and chart extensions.

Read More: **SAP Fiori System Administration Certification Preparation Guide**

Prerequisites

1. You should have a count in SCP to access Web IDE
2. Register **here** **to access the backend OData Service.**
3. Add the above-registered system as a destination in your SCP account. You can use below destination configuration for your reference

```
#
#Fri Aug 09 05:54:52 UTC 2019
Description=SAP Gateway Demo System
Type=HTTP
TrustAll=true
Authentication=NoAuthentication
```

```
WebIDEUsage=odata_abap,bsp_execute_abap,odata_gen,odata_abap,ui5
_execute_abap,dev_abap
Name=ES5
WebIDEEnabled=true
URL=https\://sapes5.sapdevcenter.com
ProxyType=Internet
sap-client=002
WebIDESystem=ES5
```

4. For reference, you can clone the GIT repo for this sample application which we are building.

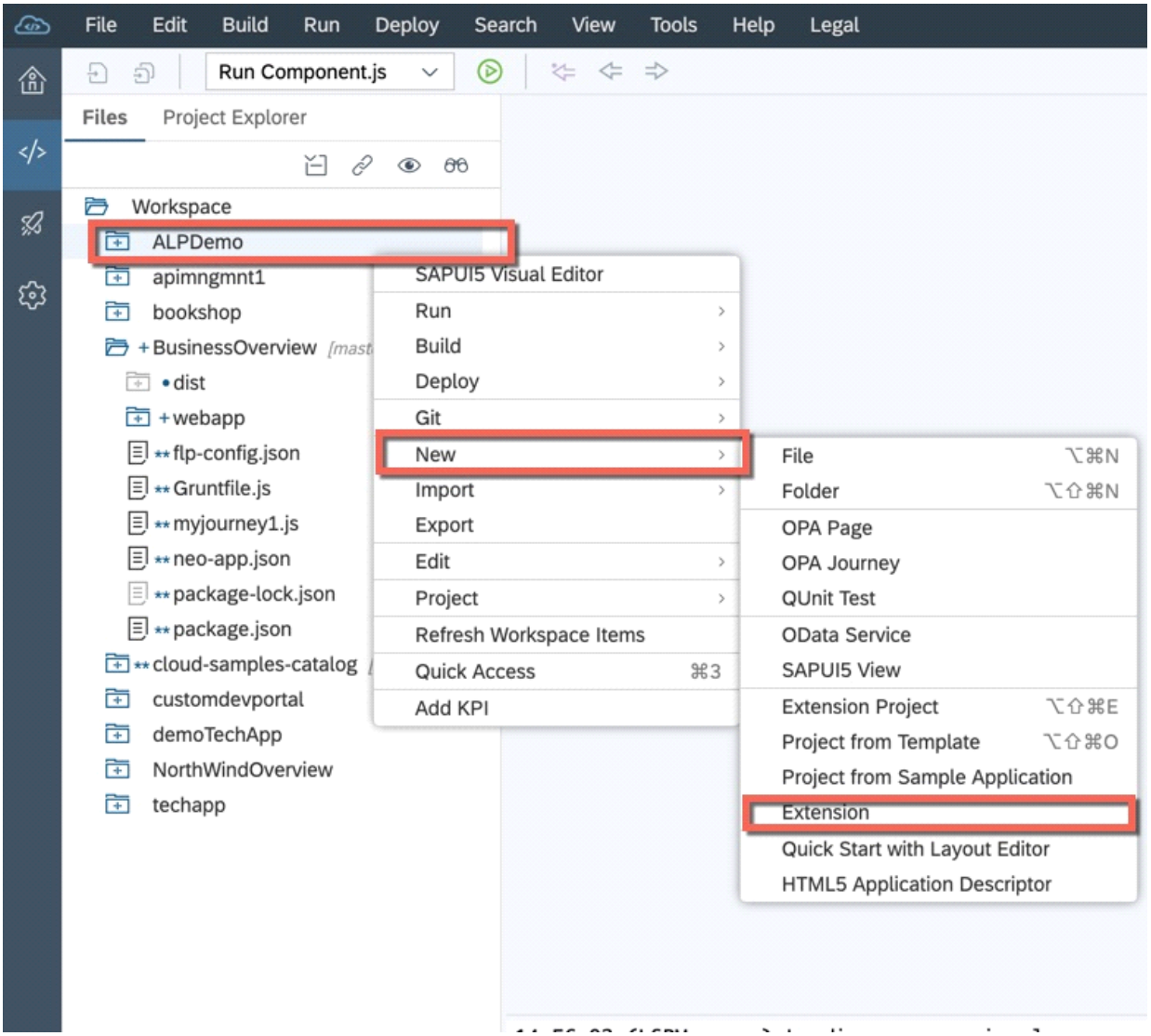
ALP Application structure

As any other Fiori elements application, an ALP application is also contained inside a shell bar. Majorly an ALP application can be broadly divided into 3 components:

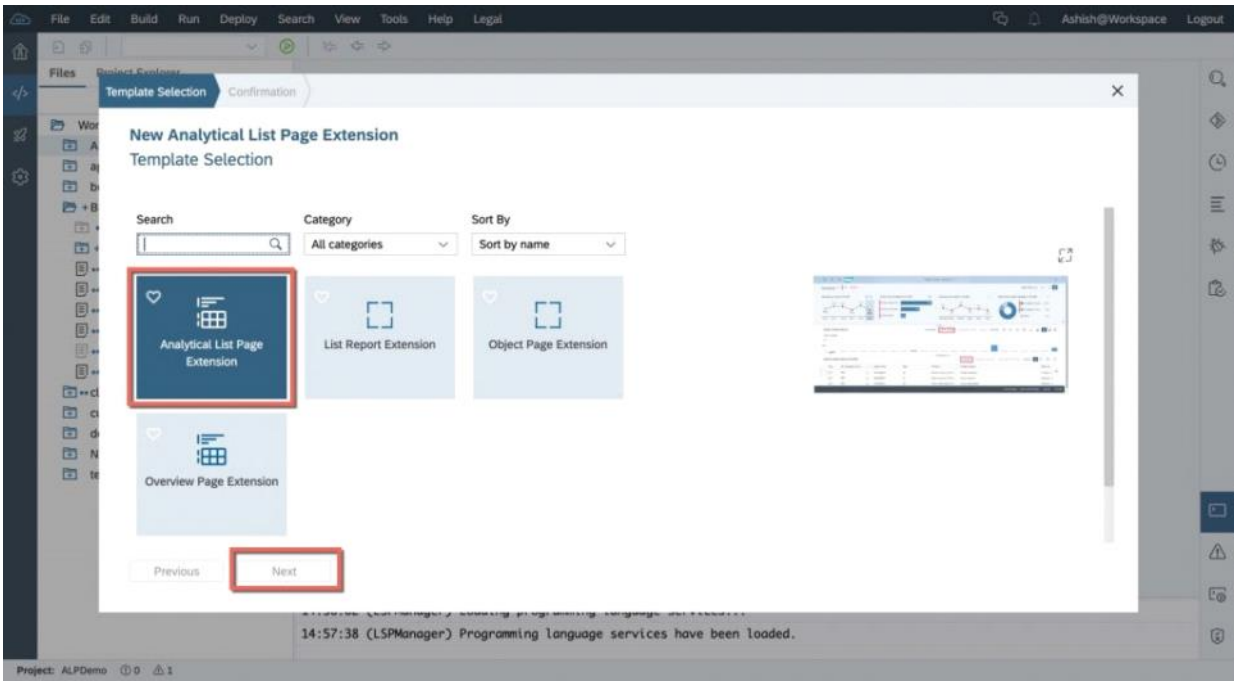
1. **Title area:** The title area of an ALP application may contain variant management, Global KPIs, Visual/compact filter switch button and share menu.
2. **Filter area:** The Filter area of an ALP application contains filters bar. ALP in addition to a smart filter bar (known as the compact filter in ALP) also provides an option to represent the filters as charts, known as the visual filter. The filter area also contains the adapt filter and go button.
3. **Content area:** The content area of an ALP application may also contain filterable KPIs. The content area of ALP application may contain a chart subsection and a table subsection or both. The content area also has action buttons along with other toolbar buttons at chart and table toolbar.

Adding extensions to ALP

To add any available extension to your ALP application, right-click on your ALP project –> New –> Extensions



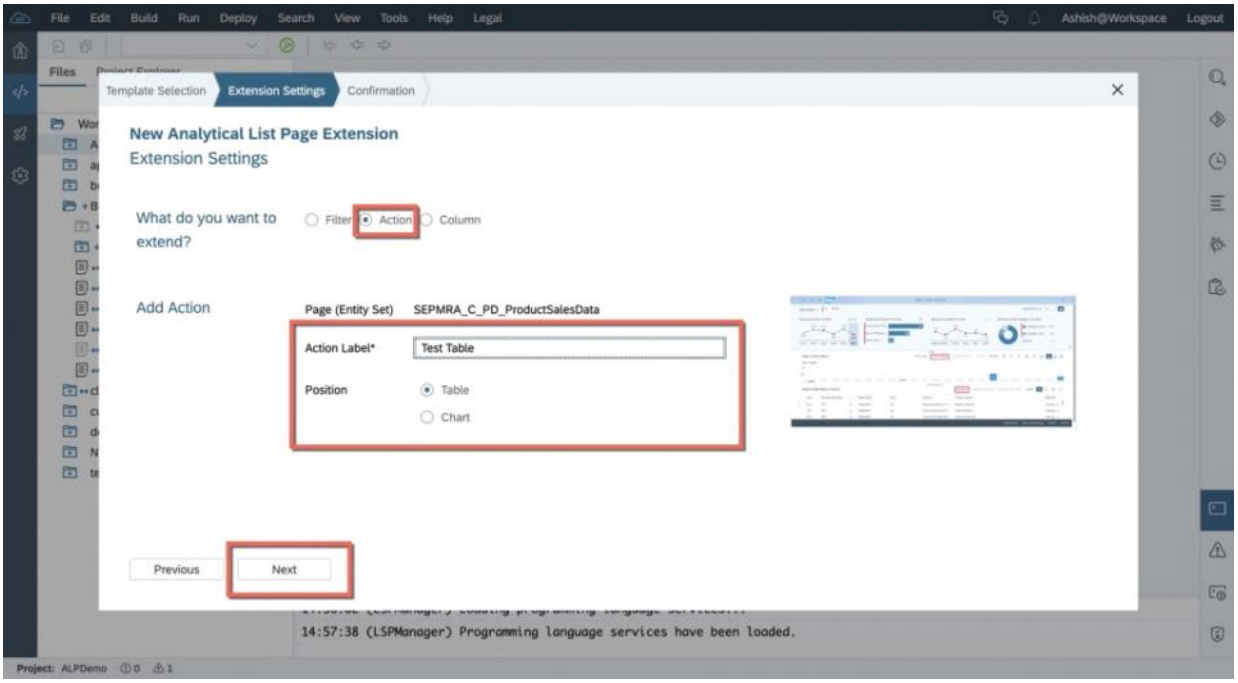
This shall bring up the template selection screen, where you should select Analytical list application and click Next.



This shall present you all the possible extension point for ALP application, which I'll discuss one by one in subsequent sections of this blog.

Adding Custom Actions

Application developers can add custom action buttons to chart or table toolbar of an ALP application. To do this select "Action" radio button in your extension selection screen, provide an appropriate label for the action button, select the position i.e. table or chart and click on Next.

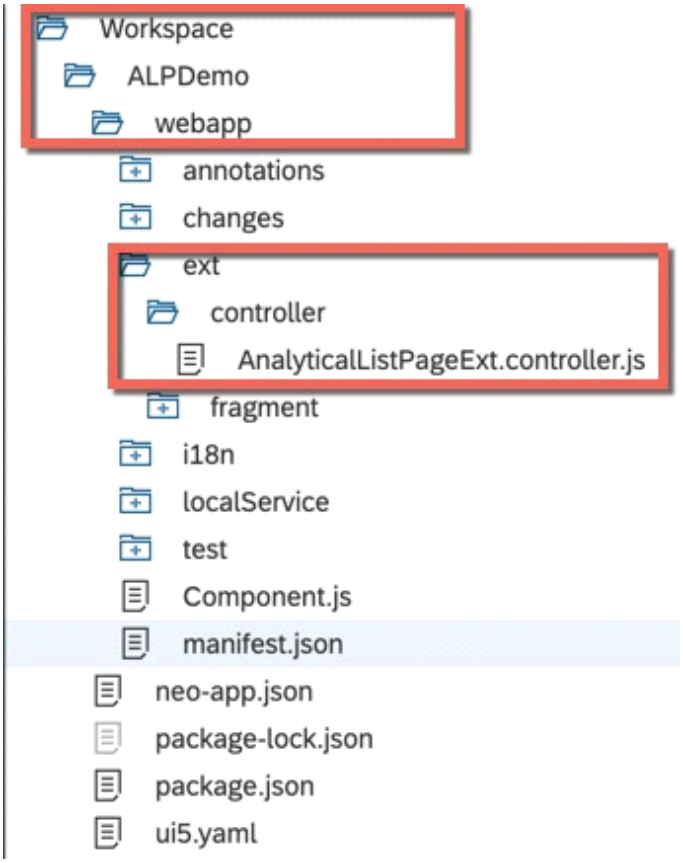


Once you have done the above steps, you will notice the following sections (sap.ui.controllerExtension) getting added to your application manifest:

```
"extends": {
                "extensions": {
                     "sap.ui.controllerExtensions": {

     "sap.suite.ui.generic.template.AnalyticalListPage.view.Analytical
ListPage": {
                                  "controllerName":
"demo.alp.ALPDemo.ext.controller.AnalyticalListPageExt",
                                  "sap.ui.generic.app": {
                                       "SEPMRA_C_PD_ProductSalesData":
{
                                            "EntitySet":
"SEPMRA_C_PD_ProductSalesData",
                                            "Actions": {

     "ActionSEPMRA_C_PD_ProductSalesData1": {
                                                 "id":
"ActionSEPMRA_C_PD_ProductSalesData1button",
                                                 "text":
"{{ActionSEPMRA_C_PD_ProductSalesData1}}",
```

```
                                                    "press":
"onClickActionSEPMRA_C_PD_ProductSalesData1"
                                                },

     "ActionSEPMRA_C_PD_ProductSalesData2": {
                                                    "id":
"ActionSEPMRA_C_PD_ProductSalesData2button",
                                                    "text":
"{{ActionSEPMRA_C_PD_ProductSalesData2}}",
                                                    "press":
"onClickActionSEPMRA_C_PD_ProductSalesData2"
                                                },

     "ActionSEPMRA_C_PD_ProductSalesData3": {
                                                    "id":
"ActionSEPMRA_C_PD_ProductSalesData3button",
                                                    "text":
"{{ActionSEPMRA_C_PD_ProductSalesData3}}",
                                                    "press":
"onClickActionSEPMRA_C_PD_ProductSalesData3",
                                                    "filter":
"chart"
                                                }
                                            }
                                        }
                                    }
                                }
                            },
```

what is says is that a new extension controller (AnalyticalListPageExt) is added to this project. and all the actions are also mentioned with associated text labels, id and click handler methods which you need to implement in your generated extension controller file. You can find this extension controller under path webapp –> ext –> controller.



So let's go ahead and implement these click event handlers:

```
onClickActionSEPMRA_C_PD_ProductSalesData1: function (oEvent) {
            alert("SEPMRA_C_PD_ProductSalesData1 is clicked");
      },
      onClickActionSEPMRA_C_PD_ProductSalesData2: function (oEvent) {
            alert("SEPMRA_C_PD_ProductSalesData2 is clicked");
      },
      onClickActionSEPMRA_C_PD_ProductSalesData3: function (oEvent) {
            alert("SEPMRA_C_PD_ProductSalesData3 is clicked");
      },
```

Adding custom filter

Application developers can add a custom filter to ALP's smart filter bar. To do this select "Filter" radio button in your extension selection screen and click on Next. This will add the following sections (sap.ui.viewExtension) getting added to your application manifest:

```
"sap.ui.viewExtensions": {

      "sap.suite.ui.generic.template.AnalyticalListPage.view.Analytical
ListPage": {

      "SmartFilterBarControlConfigurationExtension|SEPMRA_C_PD_ProductS
alesData": {
                                          "className":
"sap.ui.core.Fragment",
```

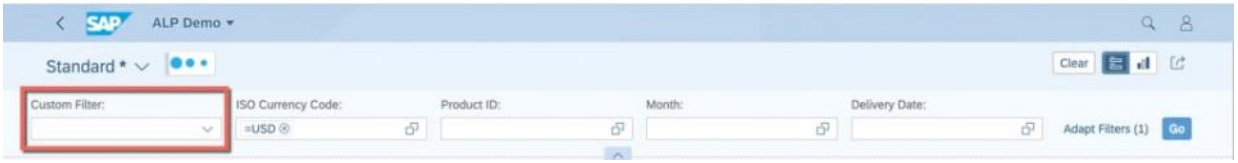
```
                                        "fragmentName":
"demo.alp.ALPDemo.ext.fragment.Customfilter",
                                        "type": "XML"
                                    }
                                }
```

The above view extension links the application to the autogenerated XML fragment file (Customfilter.fragment.xml) which contain the view definition for your custom filters. You can find this file under path webapp –> Ext –> fragment. Let's go ahead and define a custom filter.

```
<core:FragmentDefinition xmlns="sap.m"
xmlns:smartfilterbar="sap.ui.comp.smartfilterbar"
xmlns:core="sap.ui.core">
     <smartfilterbar:ControlConfiguration key="pricerange" index="1"
label="Custom Filter" groupId="_BASIC">
          <smartfilterbar:customControl>
               <ComboBox id="id">
                    <core:Item key="0" text="Item1"/>
                    <core:Item key="1" text="Item2"/>
                    <core:Item key="2" text="Item3"/>
               </ComboBox>
          </smartfilterbar:customControl>
     </smartfilterbar:ControlConfiguration>
</core:FragmentDefinition>
```

Likewise you can add as many custom filter by adding new controls to this XML fragment file. Let's go ahead and run the application. We can see one combo box with 3 options getting added to the compact filters of the application.

Please note that the custom filters added via developer extension only gets added to the ALP's compact filter and not to the visual filter.



onInitSmartFilterBarExtension

The extension method onInitSmartFilterBarExtension (as the name suggests) is called when the smart filter bar of an ALP application is initialised. This extension method can be used to add some event handler to our custom filter:

```
onInitSmartFilterBarExtension: function(oEvent) {
       var oSmartFilterBar = oEvent.getSource();

oSmartFilterBar.getControlByKey("pricerange").attachSelectionChange(fu
nction(oChangeEvent){
```

```
            alert(oChangeEvent.getParameter("selectedItem").getText() +
" selected");
        },this);
    },
```

getCustomAppStateDataExtension

The values of this custom filter should be stored in the app state, so that it doesn't get lost while navigation and also for the URL sharing functionality to work properly. ALP's extension method getCustomAppStateDataExtension should be used to store the custom filter value to the app state.

```
/*
       * Content of the custom field shall be stored in the app state,
so that it can be restored later again e.g. after a back navigation.
       * @param oCustomData  : referance to the custome data.
       */
      getCustomAppStateDataExtension: function (oCustomAppData) {

            var oCustomField1 = this.oView.byId("customFilterComboBox");
            if (oCustomField1) {
                  oCustomAppData.SampleFilterFieldID =
oCustomField1.getValue();
            }
            return oCustomAppData;

      },
```

restoreCustomAppStateDataExtension

The value of the custom filters which was stored in app state should also be restored while inbound navigation. ALP's extension method restoreCustomAppStateDataExtension should be used to fill the value from app state to the custom control:

```
/*
       * In order to restore content of the custom field in the
filterbar e.g. after a back navigation.
       * @param oCustomData  : referance to the custome data.
       */
      restoreCustomAppStateDataExtension: function (oCustomAppData) {

            if (oCustomAppData.SampleFilterFieldID !== undefined) {
                  if ( this.oView.byId("customFilterComboBox") ) {

      this.oView.byId("SampleFilterFieldID").setSelectedKey(oCustomAppD
ata.SampleFilterFieldID);
                  }
            }

      }
```

onBeforeRebindChartExtension

The filter value selected in custom filters should also change the data visualized via chart. ALP's method extension onBeforeRebindChartExtension should be used to sync your ALP's chart data to your custom filter values. This extension methods get triggers before the chart gets bound to the model or the binding gets refreshed.

```
onBeforeRebindChartExtension: function (oEvent) {
            alert("onBeforeRebindChartExtension called!");
      },
```

OnBeforeRebindTableExtension

The table data should also respect the custom filter value. ALP's method extension onBeforeRebindTableExtension should be used to sync your ALP's table data to your custom filter values. This extension methods get triggers before the table gets bound to the model or the binding gets refreshed.

```
onBeforeRebindTableExtension: function (oEvent) {
            alert("onBeforeRebindTableExtension called!");
      }
```

onBeforeRebindFilterableKPIExtension

onBeforeRebindFilterableKPIExtension should be used to sync your ALP's filterable KPI tag value to your custom filter values.

```
onBeforeRebindFilterableKPIExtension: function (oSelectionVariant,
sEntityType, sKPIId) {
            if (sKPIId ===

      "demo.alp.ALPDemo::sap.suite.ui.generic.template.AnalyticalListPa
ge.view.AnalyticalListPage::SEPMRA_C_PD_ProductSalesData--
template::KPITag::kpi::KPIRevenue1"
            ) {
                  oSelectionVariant.addSelectOption("Item", "I", "EQ",
"item 1", null);
            }
      }
```

onClearFilterExtension

The values of your custom filters should also get clear when the end-user clicks on the "clear" button of the filter bar. ALP's extension method "onClearFilterExtension" should be use to sync your custom filters with the clear button.

```
onClearFilterExtension: function (oEvent) {
            if (this.byId("customFilterComboBox")) {

      this.byId("customFilterComboBox").setSelectedKey(null);
```

```
            }
      }
```

I haven't discussed how to enable this clear button in your ALP application yet, I'll discuss this in my next blog of this blog series when I'll discuss how to use the visual editor to customize your ALP application.

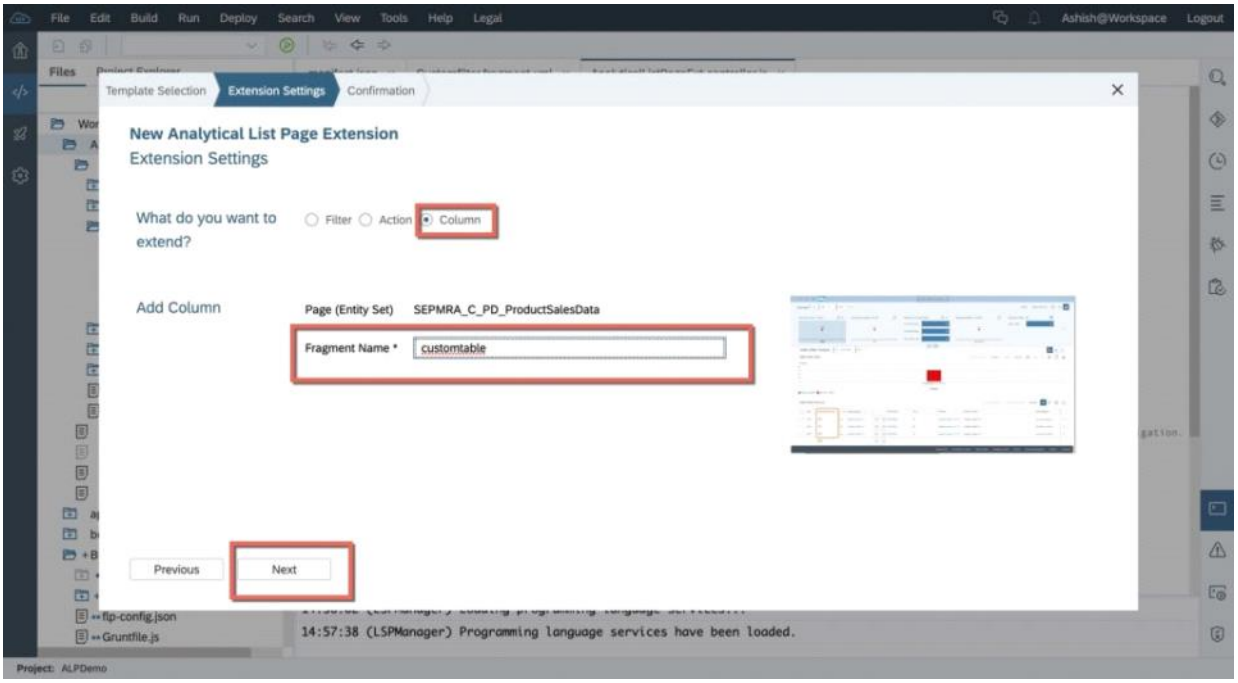
Customizing visual filters

onBeforeRebindVisualFilterExtension extension method can be used to change the properties of your visual filters. But one another very commonly used and important use case of this extension method is to provide a batch id to your visual filters so that the data fetch call for the visual filters having same batch id should be combined within the same batch call. By default, data fetch call for all visual filters goes as one BATCH call. This functionality is very much to performance optimization of your ALP application.

```
onBeforeRebindVisualFilterExtension: function (sEntityType,
sDimension, sMeasure, oContext) {
            switch (sDimension) {
            case "Product":
                  oContext.groupId = "Group1";
                  break;
            case "Currency":
                  oContext.groupId = "Group3";
                  break;
            case "Product1":
                  oContext.groupId = "Group2";
                  break;
            default:
                  oContext.groupId = "default";
            }
      },
```

Adding custom table columns

In order to add a custom column to your ALP's table select "Column" radio button in your extension selection screen and provide an appropriate name for the view fragment and click on Next.



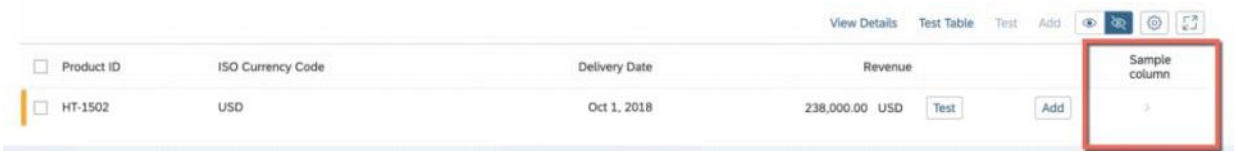
This will add the following sections (sap.ui.viewExtension) getting added to your application manifest:

```
"ResponsiveTableColumnsExtension|SEPMRA_C_PD_ProductSalesData": {
                                        "type": "XML",
                                        "className":
"sap.ui.core.Fragment",
                                        "fragmentName":
"demo.alp.ALPDemo.ext.fragment.customtable"
                                    }
```

The above view extension links the application to the autogenerated XML fragment file (customtable.fragment.xml) which contain the view definition for your custom filters. You can find this file under path webapp –> Ext –> fragment. Let’s go ahead and add a custom column to your table.

```
<core:FragmentDefinition xmlns:core="sap.ui.core" xmlns="sap.m">
     <Column id="ExtensionWizard::ColumnBreakout">
          <Text text="Sample column"/>
          <customData>
               <core:CustomData key="p13nData" value='\{"columnKey":
"Test", "columnIndex" : "101"}'/>
          </customData>
     </Column>
</core:FragmentDefinition>
```

The binding of this table can be controlled via extension method onBeforeRebindTableExtension which is discussed above.



Custom table row navigation

extension method onListNavigationExtension can be used to define navigation based on the selected rows of the table:

```
onListNavigationExtension: function(oEvent) {
    var oNavigationController =
this.extensionAPI.getNavigationController();
    var oBindingContext = oEvent.getSource().getBindingContext();
    var oObject = oBindingContext.getObject();
    // for notebooks we trigger external navigation for all others we
use internal navigation
    if (oObject.CostCenter == "300-1000") {

oNavigationController.navigateExternal("ActualCostsKPIDetails");
    } else {
        // return false to trigger the default internal navigation
        return false;
    }
    // return true is necessary to prevent further default navigation
    return true;
 }
```

SAP Fiori Elements Analytical List Page (ALP) : what and hows | SAP Blogs

Thursday, March 11, 2021 9:43 AM

Clipped from: https://blogs.sap.com/2019/08/10/sap-fiori-elements-analytical-list-page-alp-what-and-hows/

An analytical List report or ALP is another Fiori Elements floorplan which displays your data in both analytical and list forms allowing the user to analyse the data from different dimensions and at various layers.

In this blog series, I'll try to discuss Analytical List Page or ALP in detail. As ALP works with OData based on metadata and metadata annotations, In this blog series apart from ALP configurations, we will discuss different annotation supported by ALP in details.

What are we building

We will create an ALP application using SAP Web IDE. I'll be using publicly available OData service from SAP and will add annotations using web IDE annotation modeller.

Prerequisite

1. You should have a count in SCP to access Web IDE
2. Register here to access the backend OData Service.
3. Add the above-registered system as a destination in your SCP account. You can use below destination configuration for your reference #

```
#Fri Aug 09 05:54:52 UTC 2019
Description=SAP Gateway Demo System
Type=HTTP
TrustAll=true
Authentication=NoAuthentication
WebIDEUsage=odata_abap,bsp_execute_abap,odata_gen,odata_abap,ui5
_execute_abap,dev_abap
Name=ES5
WebIDEEnabled=true
URL=https\://sapes5.sapdevcenter.com
ProxyType=Internet
sap-client=002
WebIDESystem=ES5
```

4. For reference, you can clone the GIT repo for this sample application which we are building.

ALP Application structure

As any other Fiori elements application, an ALP application is also contained inside a shell bar. Majorly an ALP application can be broadly divided into 3

components:

1. **Title area:**The title area of an ALP application may contain variant management, Global KPIs, Visual/compact filter switch button and share menu.
2. **Filter area**The Filter area of an ALP application contains filters bar. ALP in addition to a smart filter bar (known as the compact filter in ALP) also provides an option to represent the filters as charts, known as the visual filter. The filter area also contains the adapt filter and go button.
3. **Content area**The content area of an ALP application may also contain filterable KPIs. The content area of ALP application may contain a chart subsection and a table subsection or both. The content area also has action buttons along with other toolbar buttons at chart and table toolbar.

Next Blogs

In the forthcoming blogs of this blog series, I'll discuss how to create an ALP application using SAP web IDE, discuss different manifest settings of an ALP application, how to annotate your service using annotation modeller, discuss the different configuration of global and local filters, visual and compact filters, chart and table area and navigations and action button configuration. I'll also discuss different developer extension possible within an ALP application. so stay connected:

Creating a bare-bone Analytical List Page (ALP) application

Configuring filters area of an Analytical List Page (ALP) application

Configuring chart content area of an Analytical List Page (ALP) application

Configuring table area of an Analytical List Page (ALP) application

Configuring KPI tags of an Analytical List Page (ALP) application

Analytical List Page (ALP) developer extension

Customising an Analytical List Page (ALP) application with Visual Editor

SAP Fiori elements Analytical List Page (ALP) : Performance optimisation

I'll keep updating and adding new features and developments as and when they come in future. Happy reading !! ?

Feedbacks, questions and comments are most welcome!!