Technical Articles

Peter Jirsak
October 15, 2019 26 minute read

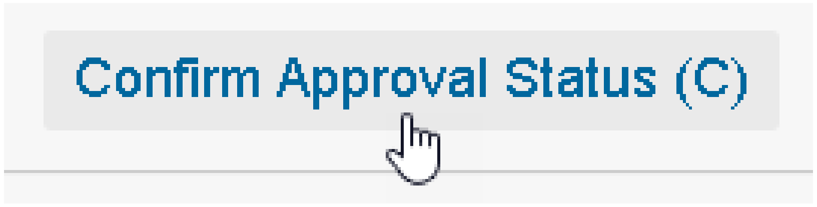
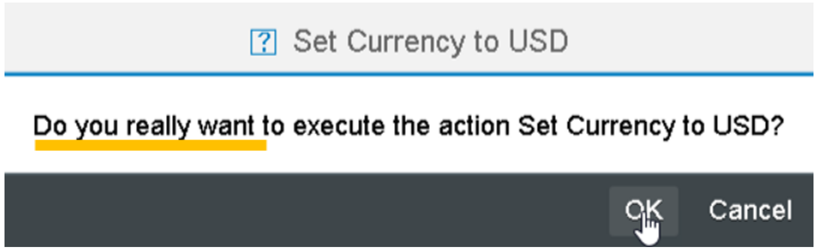
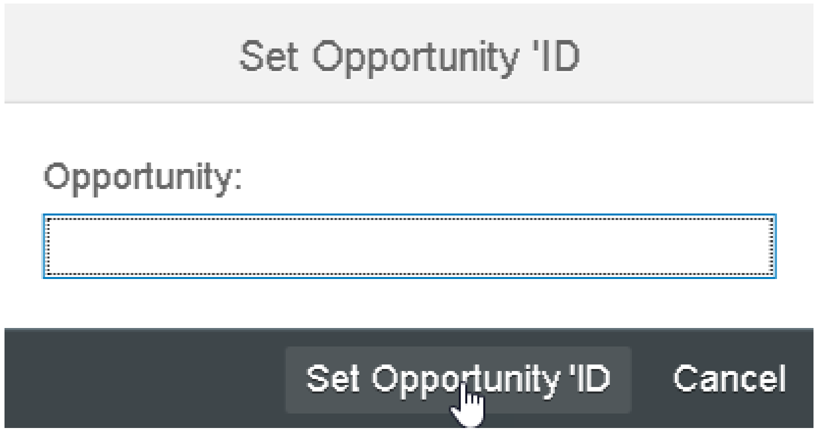
Fiori Elements – A Journey of Building an Action Dialog on a List Report using Annotations

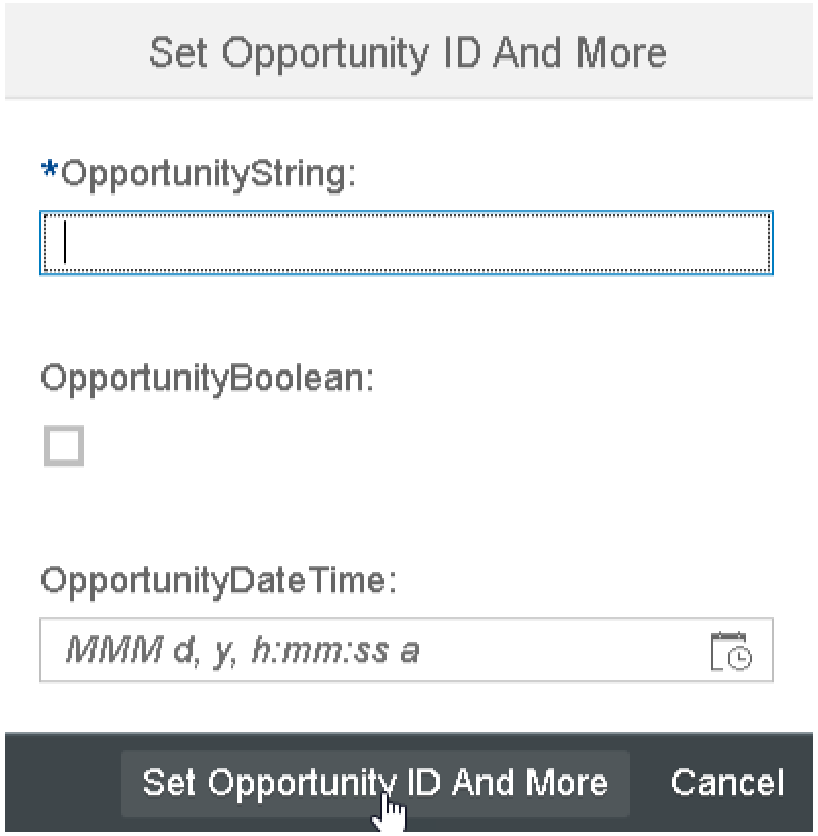
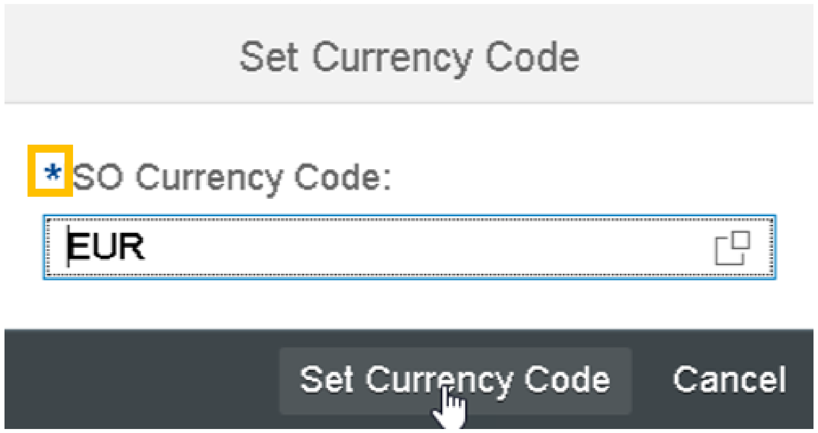
Abstract

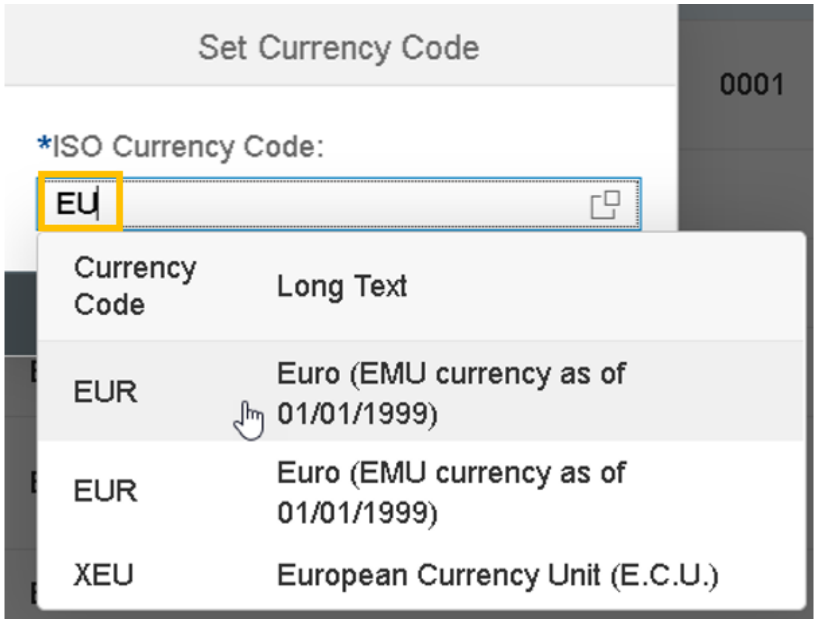
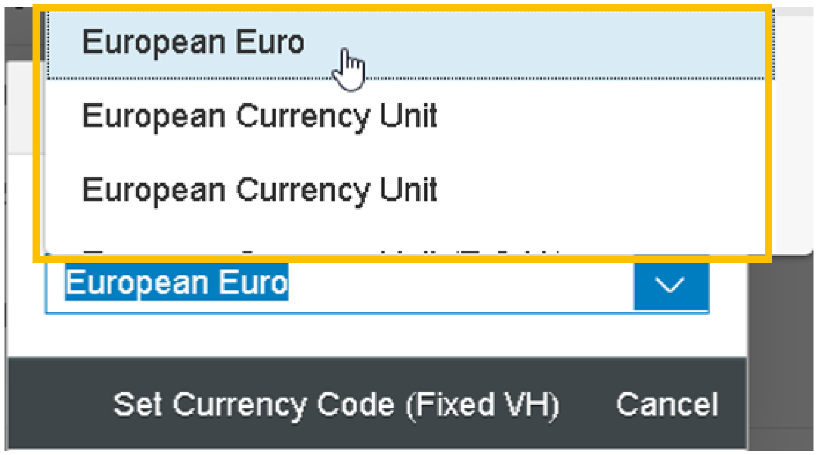
In this blog post you can find information on how to create complex action dialogs in SAP Fiori Elements applications without the use of app extensions (aka breakout). The blog post covers the topics of implementing the dialogs, gives technical details, mentions limitations and includes links to documentation and runnable test apps.

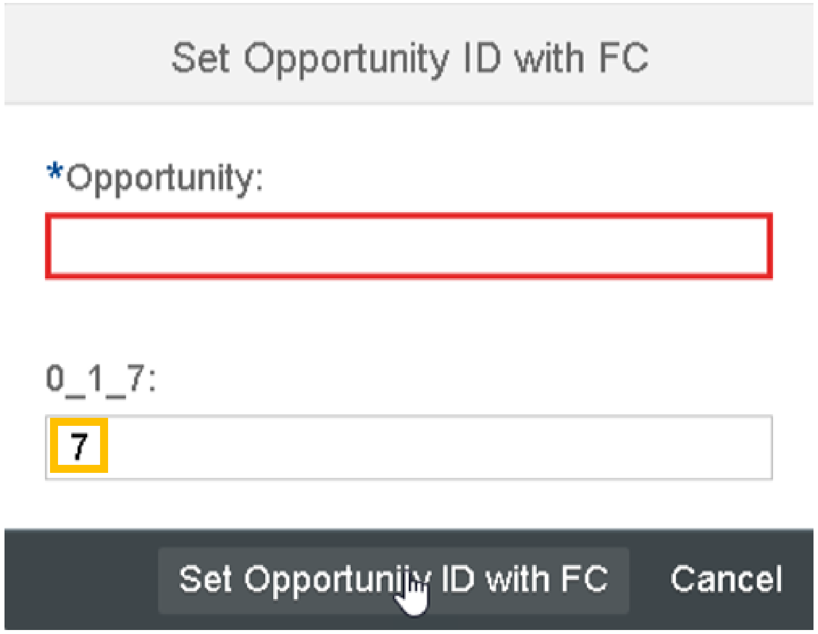
See below an overview with links to what you will learn in this blog post.

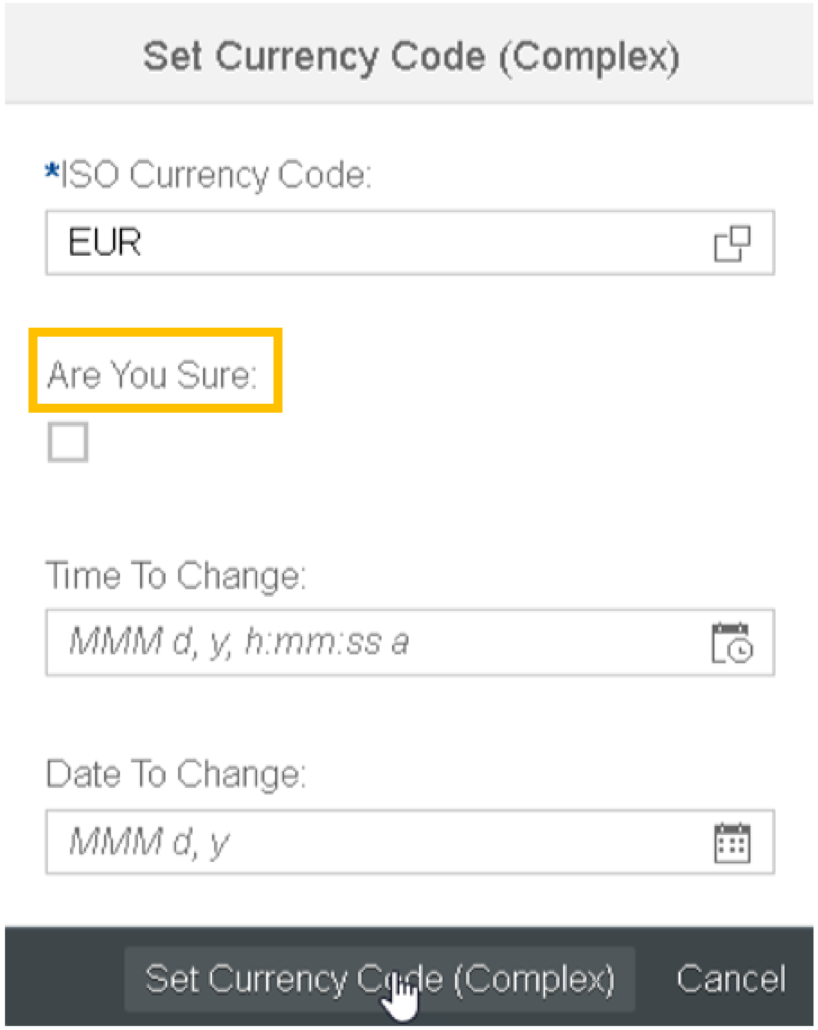
Link to Chapter	Brief Description
#1 Simple action	A simple action can be executed without additional input.

	
#2 Critical action	Critical actions are special actions that require user confirmation.  Set Currency to USD Do you really want to execute the action Set Currency to USD?
#3 Action with dialog	Actions with a dialog are actions that require additional user input, for example, an approval comment or a value that should be overwritten – as below.  Set Opportunity 'ID Opportunity:
#4 Action with more complex dialog	You see below a more complex dialog using different data types.

	 Set Opportunity ID And More *OpportunityString: OpportunityBoolean: OpportunityDateTime: MMM d, y, h:mm:ss a Set Opportunity ID And More Cancel
#5 Action dialog with mandatory and optional parameter	You want to have a dialog with a parameter the user must fill before proceeding.  Set Currency Code *SO Currency Code: EUR Set Currency Code Cancel
#6 Action dialog with value help	Show a value help dialog. Below you see one form of value help – a type ahead value help.

	 Set Currency Code *ISO Currency Code: EU Currency Code / Long Text EUR / Euro (EMU currency as of 01/01/1999) EUR / Euro (EMU currency as of 01/01/1999) XEU / European Currency Unit (E.C.U.)
#7 Action dialog with drop down value list	Show a value help as a drop-down list.  European Euro European Currency Unit European Currency Unit European Euro Set Currency Code (Fixed VH) Cancel
#8 Field Control	Influence the dialog using field controls, e.g. hiding fields instance based. Below you see a mandatory field which is not filled.

	 Set Opportunity ID with FC *Opportunity: 0_1_7: 7
#9 Action dialog changing labels	Learn how to set different labels on the dialog.



Context

Imagine you are a Fiori application developer and your Product Owner wants you to build an app with a table and different clickable actions above it. The actions shall execute application specific business logic. Some of these actions may have parameters a user needs to fill; others might be critical actions. In general the actions are very different. You know how to do that with a SAPUI5 freestyle app, but you have heard of Fiori Elements before and wonder if you can use that framework. Checking the List Report Elements documentation getting a table seems straight forward. If you can realize all those different clickable actions with Fiori Elements is now your question. Your Product Owner adds, preferably to do that without any app extensions (also known as custom or breakout actions), because he heard that increases the maintenance effort for the application.

You are bound to and use the following technical components:

- Your SAP backend system is on version SAP Basis 7.55
- The UI add on shows UI5 Version 1.69.0 – see What's New in SAPUI5 1.69 and Release Notes to find out what was possible with this release
- SAP Web IDE Full-Stack v.190314
- Google Chrome Browser v.72.0.3626.121
- SAP ABAP Development Tools v.3.1.100

Get Familiar with

Your strategy is to start easy with theory, then the simplest form of an action and then little by little you want to test more complicated actions to in the end come up with an overview of what is possible.

What is Fiori Elements?

Fiori Elements is a library for the most common application use cases and offers these as different floorplans. [Fiori Design Guidelines] To get a fast start into the topic, especially to "Understand Fiori Elements technology" you…

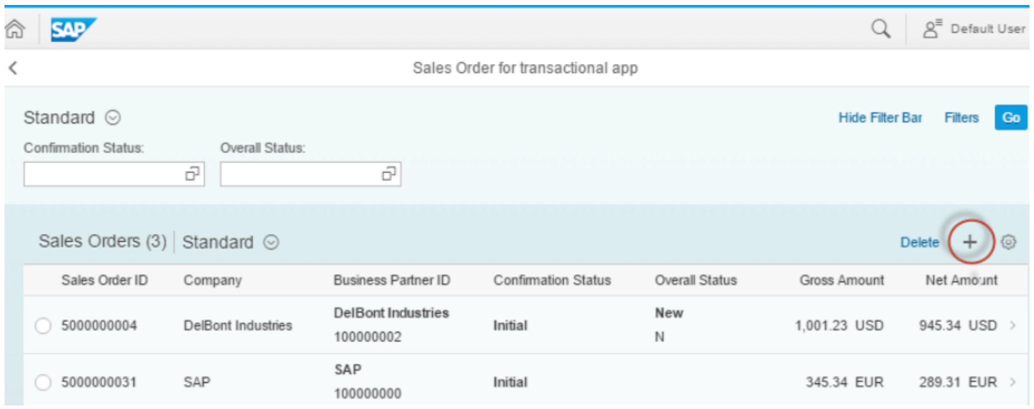
- scanned the official documentation and read "Why use SAP Fiori elements?"
- browsed through the Getting Started with SAP Fiori Elements Video Series
- had a look at the slides of Evolved Web Apps with SAPUI5 – Week 05, especially with slide 5 showing the Fiori Elements technology, which is part of a publicly available hands-on course http://open.sap.com/courses/ui52 (see here an intro video).

Your main takeaways -for the moment- are:

- You read that, for the right use cases, using Fiori Elements is much faster than building freestyle.
- In regards to Fiori Elements technology, you know that an application developer needs to, by any means, expose an "OData Service" and annotate the "OData Service" with "OData Annotations" using "OData Vocabularies". Since you are developing on an ABAP stack, you plan to use the ABAP Development Tools to create a Core Data Service view using CDS annotations, so-called CDS backend annotations. These CDS artifacts will then be automatically mapped and exposed as OData Service & OData annotations – see here and here.

First try's

Having an ABAP backend to expose the OData Service using CDS Services you are following the information on Preparing OData Services. Furthermore you read and implemented the About ABAP Programming Model for SAP Fiori until you reach the point of Running the Resulting SAP Fiori App and your UI looks similar to that:



Sales Order for transactional app

Standard

Confirmation Status:

Overall Status:

Sales Orders (3) | Standard

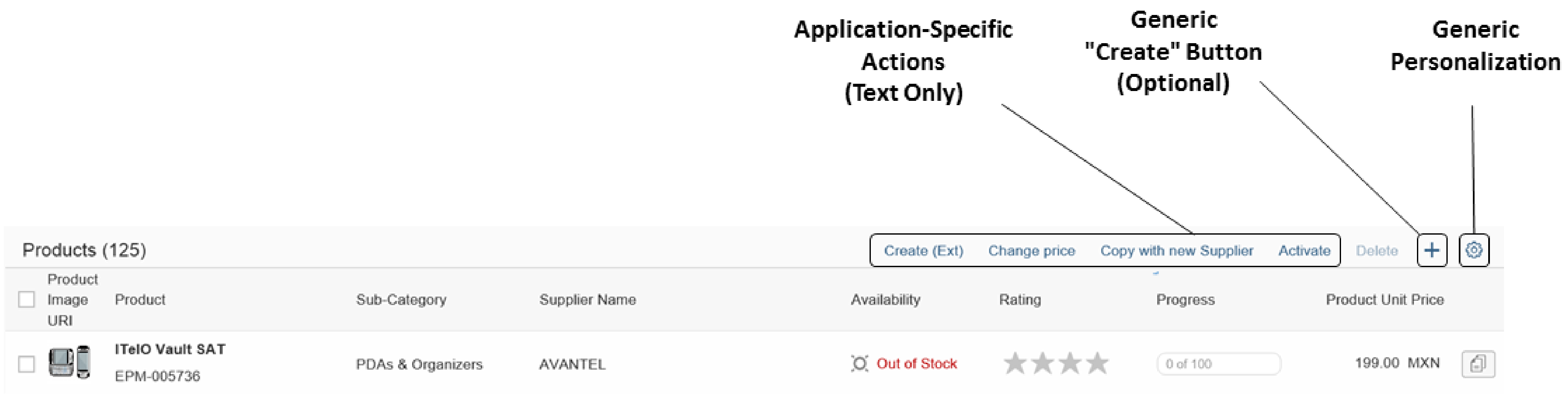
Sales Order ID	Company	Business Partner ID	Confirmation Status	Overall Status	Gross Amount	Net Amount
5000000004	DelBont Industries	DelBont Industries 100000002	Initial	New N	1,001.23 USD	945.34 USD
5000000031	SAP	SAP 100000000	Initial		345.34 EUR	289.31 EUR

On your app there might even be some actions like “Delete” or “+” above the table, which let you interact with the table data.

Action support

The Product Owner mentioned “clickable actions that execute application specific business logic”. In the context of the UI an action is mostly understood as an UI element with which you can interact by clicking. In this blog when talking about actions the following is meant:

- App-Specific Actions



Products (125)

Product Image URI	Product	Sub-Category	Supplier Name	Availability	Rating	Progress	Product Unit Price
	ITelO Vault SAT EPM-005736	PDAs & Organizers	AVANTEL	Out of Stock	★★★★	0 of 100	199.00 MXN

  See in the picture, taken from Actions in the List Report, the distinction to actions that are generic, like pressing on the “Create” button: App specific actions are defined and handled by each application whereas generic actions are the same cross all applications.

- UI actions that execute logic on a business object
  So, you don’t need an UI action that “only” triggers a navigation, which you also found under Enabling Actions in the List Report: “Action triggering external navigation”. But instead your action changes the business object.

- App extension actions
  App extension actions are actions that give full control to the application developer on what happens after clicking the action.
  You searched in the documentation for things coming with app extensions and found Read Before Extending a Generated App: *"After you've created an app extension, its display (for example, control placing, CSS) and system behavior (for example, model and binding usage, busy handling) of the app extension lies within the applications responsibility."*, which seconds the statement of your Product Owner about the maintenance effort.

So, fitting to the requirement of the Product Owner you want, without any application specific client code detour (app extension), trigger an UI action that calls function imports in the backend. The backend executes the function import and returns its result, objects and/or messages.

App-specific actions in OData

An OData service and its capabilities is described in the metadata.xml file. This holds also true for app-specific actions, which are expressed like that – click metadata.xml file and search for *C_STTA_SalesOrder_WD_20Setapprovalstatus*. Just to get a feeling for how such a file and the definition of an action on OData level looks like (remark: it's a mocked *metadata.xml*, not a real one):

```
 <EntityContainer Name="STTA_SALES_ORDER_WD_20_SRV_Entities"…

<FunctionImport Name="C_STTA_SalesOrder_WD_20Setapprovalstatus" ReturnType="STTA_SALES_ORDER_WD_20_SRV.C_STTA_SalesOrder_WD_20Type"En
<Parameter Name="SalesOrder" Type="Edm.String" Mode="In" MaxLength="10"/>
<Parameter Name="DraftUUID" Type="Edm.Guid" Mode="In"/>
<Parameter Name="IsActiveEntity" Type="Edm.Boolean" Mode="In"/>
</FunctionImport>
```

Following you see an example request against such a defined function import: *POST C_STTA_SalesOrder_WD_20Setapprovalstatus?IsActiveEntity=true&DraftUUID=guid'00505691-115b-1ed9-a5cc-09ad63829d29'&…*

A "real" metadata.xml file is generated by the backend and the app-specific actions explicitly can be defined in ADT ABAP Development Tools in the "BOPF Business Objects" – you will see later how that works. When you have specified the BOPF object action the "<FunctionImport Name="C_STTA_SalesOrder_WD_20Setapprovalstatus"…" entry in the metadata.xml is generated for you.

Tests with different kind of actions

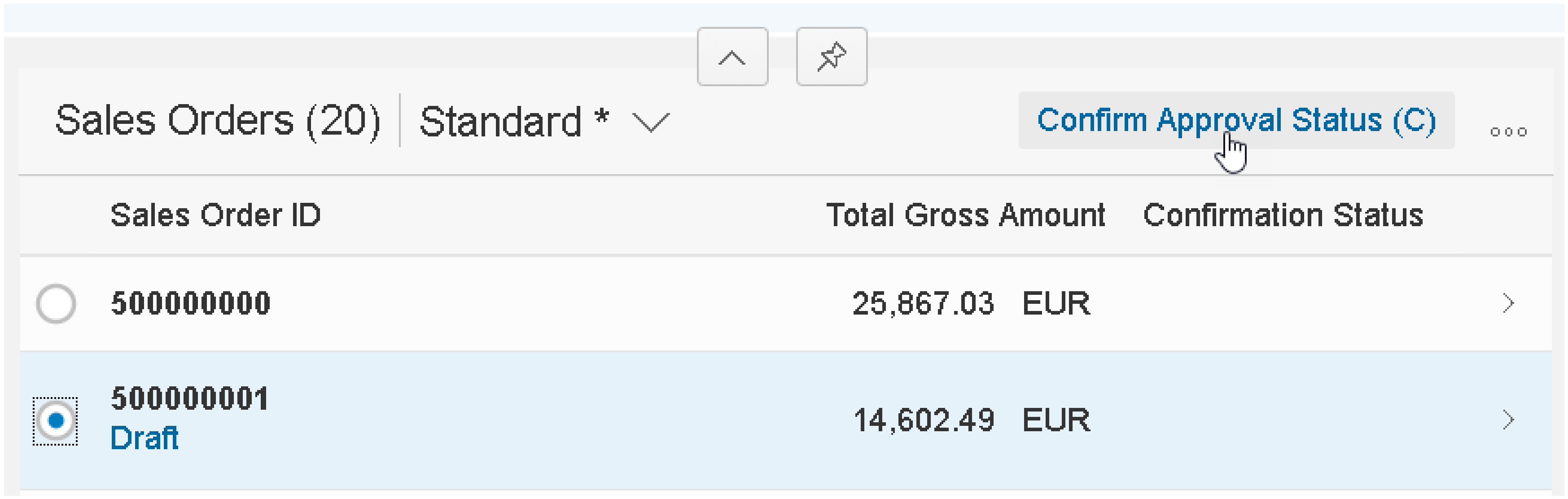
Let’s start with different UI action tests. The titles of the following chapters are case descriptions. For example, depending on whom you ask, the “Simple Action” is understood differently. The chapters start with an explanation of the case.

You can test all these cases in the technical demo app [Start the DEMO app], by clicking on ‘GO’, selecting an item and press the corresponding action.

DISCLAIMER – this technical demo app serves no business use and herewith we only want to focus on the actions above the list, the rest might or might not work. The service behind the technical demo app is a read-only mock service.

1. Simple action

Let’s start with the use-case of creating a “Simple action”. A simple action can be executed without additional input. There is no dialog coming up, asking for input parameters. Also, it executes a simple logic on whatever was selected. An example here would be the action “Confirm Approval Status (C)” in the DEMO app. The action “Confirm Approval Status (C)” changes the value of the field “Confirmation Status” of the selected line item to “C” (confirmed). See below or [Start the DEMO app].



	Sales Order ID	Total Gross Amount	Confirmation Status
○	500000000	25,867.03 EUR	
●	500000001 Draft	14,602.49 EUR	

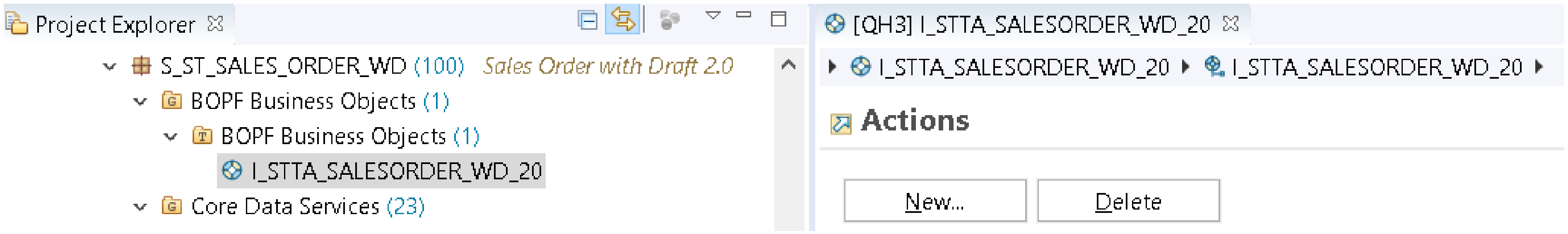
In the picture you can see two line-items. One of them is marked as a “Draft”. See here for an explanation about draft. For this item ‘5000000001’ a draft was created before, by going into the object page and clicking on ‘Edit’. Now you can change that draft object, also with UI actions. To be able to change active objects directly, in the picture above the ‘500000000’ you need to enable so-called quick actions Extending Apps with Quick Actions in your app. With quick actions you are able to change values of line items without going into edit-mode first.

How to achieve that?
1.1 Steps in the backend (e.g. CDS annotations)

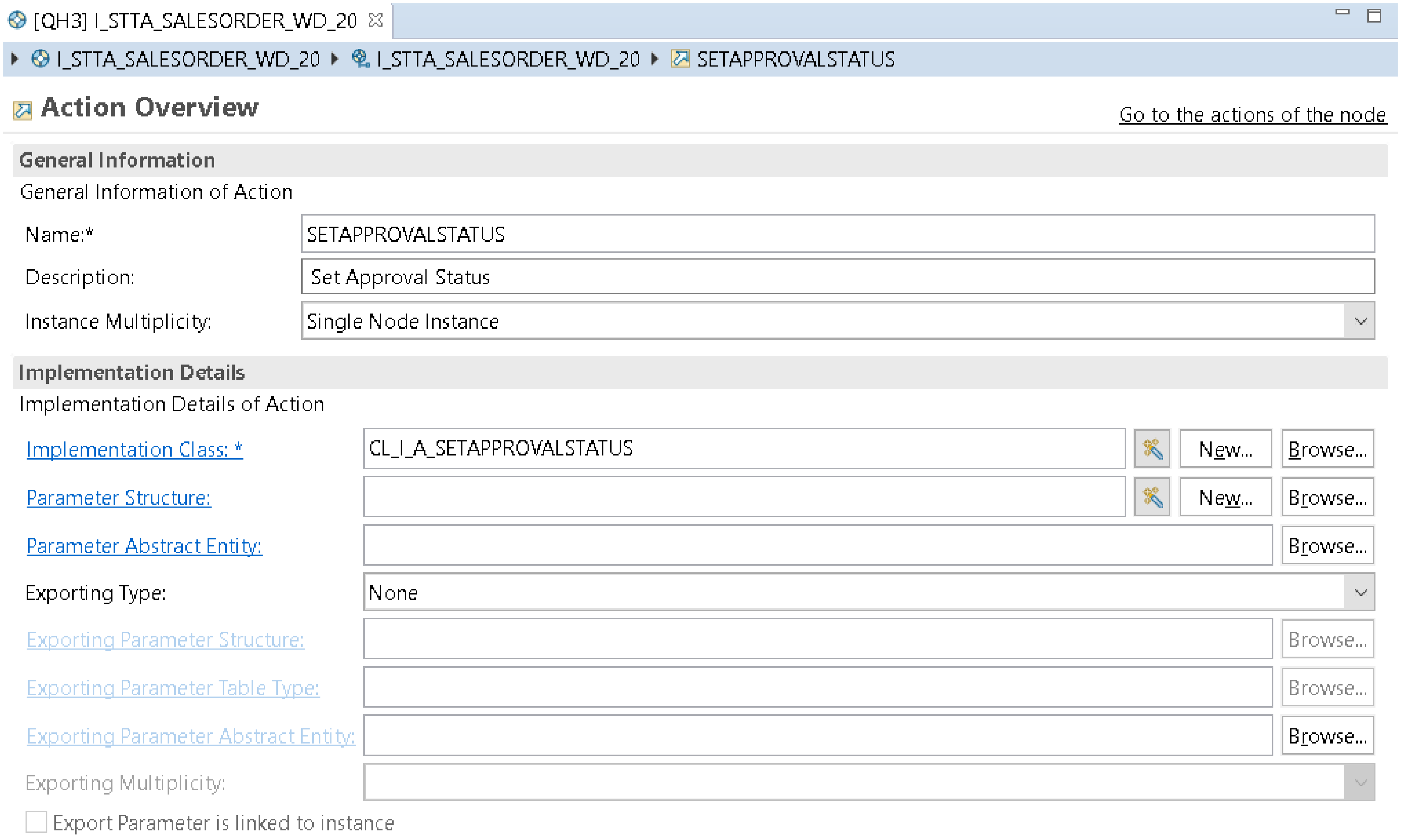
For the first use-case this will be a bit more detailed.

You now want to do the prerequisites in the backend to create a quick action. You can refer to Adding a New BOPF Action for more help.

- in ADT in the “BOPF Business Objects” you create a new BOPF action by clicking on “New…”. This is the backend representation of the quick action.


Project Explorer
S_ST_SALES_ORDER_WD (100) Sales Order with Draft 2.0
BOPF Business Objects (1)
BOPF Business Objects (1)
I_STTA_SALESORDER_WD_20
Core Data Services (23)
[QH3] I_STTA_SALESORDER_WD_20
I_STTA_SALESORDER_WD_20
I_STTA_SALESORDER_WD_20
Actions
New...
Delete

- using these properties



[QH3] I_STTA_SALESORDER_WD_20
I_STTA_SALESORDER_WD_20 ▸ I_STTA_SALESORDER_WD_20 ▸ SETAPPROVALSTATUS
Go to the actions of the node
New...
Browse...

Action Overview

General Information

General Information of Action

Name:*	SETAPPROVALSTATUS
Description:	Set Approval Status
Instance Multiplicity:	Single Node Instance

Implementation Details

Implementation Details of Action

Implementation Class: *	CL_I_A_SETAPPROVALSTATUS
Parameter Structure:	
Parameter Abstract Entity:	
Exporting Type:	None
Exporting Parameter Structure:	
Exporting Parameter Table Type:	
Exporting Parameter Abstract Entity:	
Exporting Multiplicity:	

☐ Export Parameter is linked to instance

- Then you implement the “Implementation Class” like below. Example backend implementation:

```
CLASS CL_I_A_SETAPPROVALSTATUS IMPLEMENTATION.


  METHOD /bobf/if_frw_action~execute.
```

```
    " ---------------------------------------------------------------------------------------------
    " Separate the keys
    " ---------------------------------------------------------------------------------------------
    /bobf/cl_lib_draft_active=>get_instance( is_ctx-bo_key )->separate_keys(
      EXPORTING
        iv_node_key   = is_ctx-node_key
        it_key        = it_key
      IMPORTING
        et_active_key = DATA(lt_active_key)
    ).

    CHECK lt_active_key IS NOT INITIAL.

    DATA: lt_sales_order TYPE itk_stta_salesorder_wd_20_act1.

    /bobf/cl_lib_legacy_key=>get_instance( is_ctx-bo_key )->convert_bopf_to_legacy_keys(
      EXPORTING
        iv_node_key   = is_ctx-node_key
        it_bopf_key   = lt_active_key
      IMPORTING
        et_legacy_key = lt_sales_order
    ).

    LOOP AT lt_sales_order REFERENCE INTO DATA(ls_salesorder).
      UPDATE stta_so_wd SET approval_status = 'C' WHERE node_key = ls_salesorder->activeuuid.
    ENDLOOP.


  ENDMETHOD.
ENDCLASS.
```

The corresponding ODATA representation – fully generated from the backend, after activating the BOPF action – looks like this:

```
<FunctionImport Name="C_STTA_SalesOrder_WD_20Setapprovalstatus"> ReturnType="STTA_SALES_ORDER_WD_20_SRV.C_STTA_SalesOrder_WD_20Type"
   <Parameter Name="SalesOrder" Type="Edm.String" Mode="In" MaxLength="10"/>
   <Parameter Name="DraftUUID" Type="Edm.Guid" Mode="In"/>
   <Parameter Name="IsActiveEntity" Type="Edm.Boolean" Mode="In"/>
</FunctionImport>
```

The UI can now call this action.

You now want to place the action button inside your table toolbar. See Enabling Actions in the List Report ”Application-Specific Actions in Toolbar” where you see that the OData annotation *DataFieldForAction* is needed to do so.

With the following CDS annotation, preferably done in the “Metadata Extensions”, you add two things. First you have included the field approval_status in metadata extension file, by using “position: 10”. Then you need to add an UI annotation to declare an action button with referring to the implemented BOPF action:

```
annotate view C_STTA_SalesOrder_WD_20 with
{
  @UI.lineItem:       [
      { position: 10, importance: #HIGH },
      { type: #FOR_ACTION, dataAction: 'BOPF:SETAPPROVALSTATUS', label: 'Confirm Approval Status (C)'}
  ]
  approval_status;
```

The CDS line item annotation of type ‘#FOR_ACTION’ will lead to the following OData annotation in the service metadata – fully generated from the backend, after activating the BOPF action – looks like this:

```
<Annotations Target="STTA_SALES_ORDER_WD_20_SRV.C_STTA_SalesOrder_WD_20Type">
    <Annotation Term="UI.LineItem">
        <Collection>
```

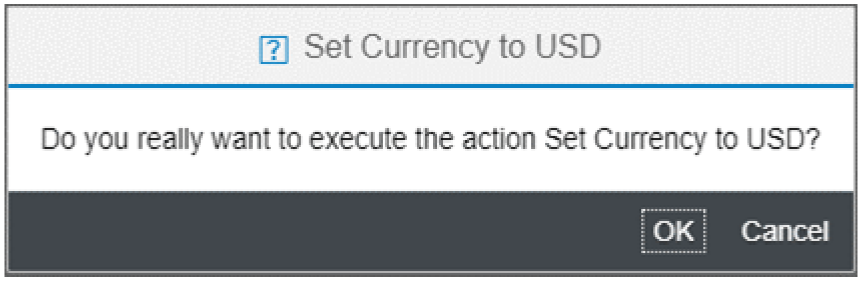
```
            <Record Type="UI.DataFieldForAction">
                <PropertyValue Property="Label" String="Confirm Approval Status (C)"/>
                <PropertyValue Property="Action" String="STTA_SALES_ORDER_WD_20_SRV.STTA_SALES_ORDER_WD_20_SRV_Entities/C_STTA_SalesO
            </Record>
        </Collection>
    </Annotation>
</Annotations>
```

1.2 Steps in UI Project (e.g. local OData annotation file)

There are no changes needed in the UI project.

2. Critical action

Critical actions are special actions that require user confirmation. This can be useful for actions that, for example, have severe consequences. When the user executes such an action the system opens a dialog in which the user must confirm the action. See Adding Confirmation Popovers for Actions for more details. An example for a critical action would be the action "Set Currency to USD" – see below or [Start the DEMO app]:


Set Currency to USD

Do you really want to execute the action Set Currency to USD?

How to achieve that?

2.1 Steps in the in backend (e.g. CDS annotations)

This annotation is not supported by CDS so far, so there are no changes needed in the backend.

2.2 Steps in UI Project (e.g. local OData annotation file)

The OData annotation which is needed for the critical action looks like this – taken from the annotation.xml document:

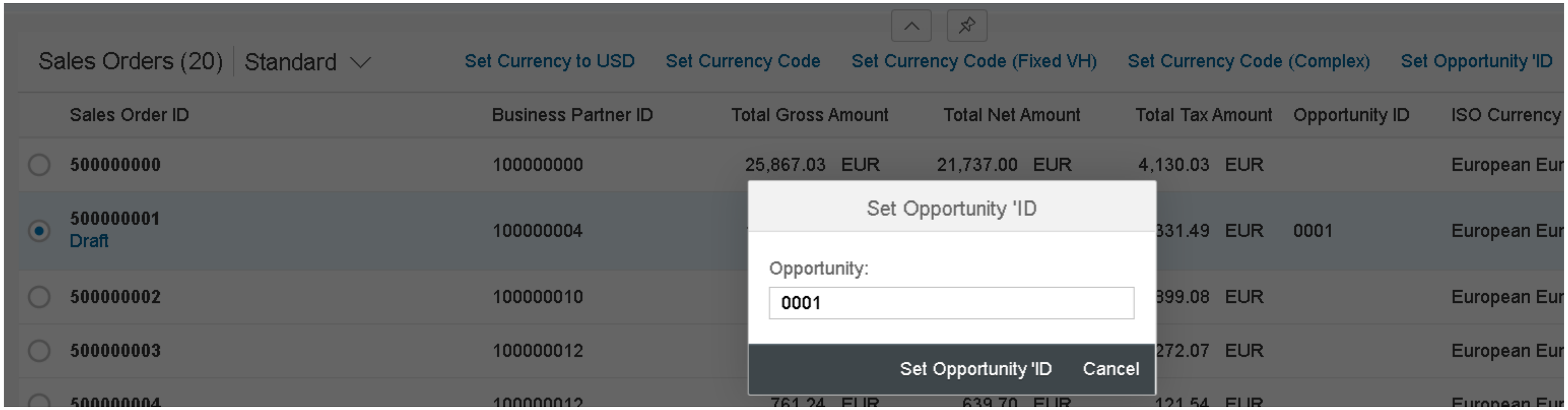
```
<Annotations Target="STTA_SALES_ORDER_WD_20_SRV.STTA_SALES_ORDER_WD_20_SRV_ENTITIES/C_STTA_SalesOrder_WD_20Setcurrencyusd">
  <Annotation Term="COMMON.IsActionCritical" Bool="true"/>
```

```
</Annotations>
```

The target property of this OData annotation points to the function import which is the OData representation of the BOPF action. The target is divided into three parts which contain the service name and the entity container separated with a dot and the function import name separated with a slash. In this example the service name is “STTA_SALES_ORDER_WD_20_SRV”. The entity container is usually named as a combination of service name and the keyword “ENTITIES”. In our example the name of the entity container is therefore “STTA_SALES_ORDER_WD_20_ENTITIES”. The function import name is usually a concatenation of the entity set name and the BOPF action name. In our example the function import name is “C_STTA_SalesOrder_WD_20Setcurrencyusd”. In some cases it may happen that the concatenation of entity set name and BOPF action name is just too long for a function import name. In this case the function import name will be generated differently.

3. Action with dialog

Actions with a dialog are actions that require additional user input, for example, an approval comment. The system opens a dialog with one or more input fields in which the user enters the required data. If applicable, the system can pre-fill data. To achieve an action with dialog you can enrich your action with additional import parameters. ‘Additional’ in this case means: in addition to the key fields which are automatically parameters of the action. An example of an action with a simple dialog can be found here: [Start the DEMO app] and click “Set Opportunity ID”.



Sales Order ID	Business Partner ID	Total Gross Amount	Total Net Amount	Total Tax Amount	Opportunity ID	ISO Currency
500000000	100000000	25,867.03 EUR	21,737.00 EUR	4,130.03 EUR		European Eur
500000001 Draft	100000004			331.49 EUR	0001	European Eur
500000002	100000010			399.08 EUR		European Eur
500000003	100000012			272.07 EUR		European Eur
500000004	100000012	761.24 EUR	639.70 EUR	121.54 EUR		European Eur

Set Opportunity 'ID

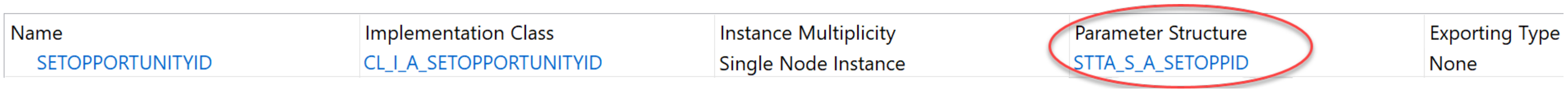
Opportunity: 0001

Set Opportunity 'ID Cancel

How to achieve that?

3.1 Steps in the in backend (e.g. CDS annotations)

To add additional import parameters to a function import, you must add a parameter structure to the BOPF action:



Name	Implementation Class	Instance Multiplicity	Parameter Structure	Exporting Type
SETOPPORTUNITYID	CL_I_A_SETOPPORTUNITYID	Single Node Instance	STTA_S_A_SETOPPID	None

The structure contains all additional import parameters which are needed for the user input. It can be chosen via value help or be created on the fly. In our example the structure contains 1 parameter “Opportunity”:

```
 @EndUserText.label : 'Parameter for action Set Opportunity ID'
@AbapCatalog.enhancementCategory
define structure stta_s_a_setoppid {
        Opportunity: snwd_op_id;
}
```

The field *Opportunity* is the additional import parameter which will let the system generate the dialog with an input field. Best – the id and type of the parameters should be the same as the corresponding field in the CDS view. Then you get for free:

- A label for the parameter (see chapter #9 Action dialog changing labels)
- The value of the property of the entity type gets propagated to the dialog – the value “0001” on the picture in the beginning of the chapter
- Type validations (if the field would be Edm.Int, UI control checks would not allow to enter a string)

The corresponding ODATA representation – fully generated from the backend, after activating the BOPF action – looks like this:

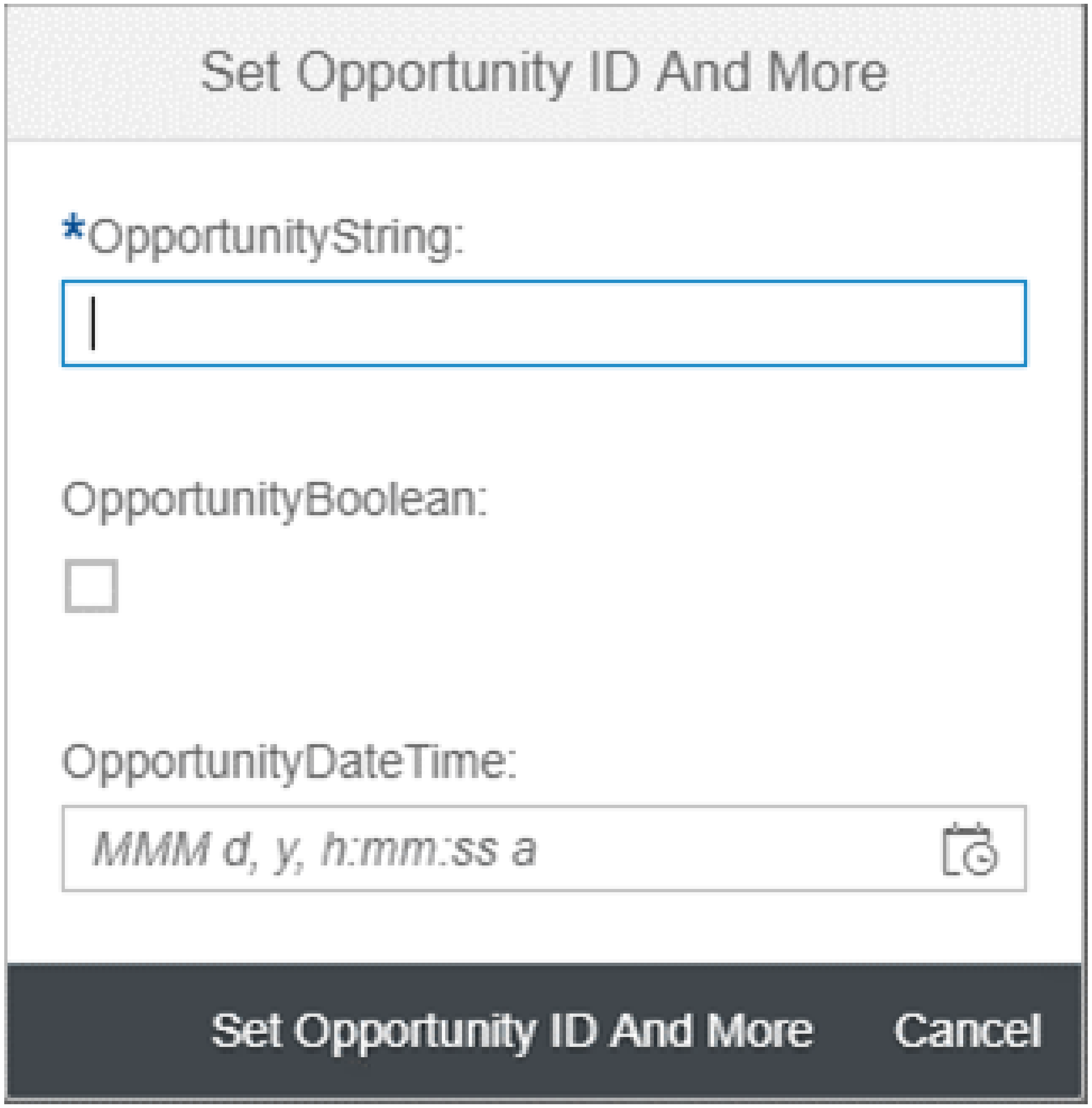
```
 <FunctionImport Name="C_STTA_SalesOrder_WD_20Setopportunityid" ReturnType="STTA_SALES_ORDER_WD_20_SRV.C_STTA_SalesOrder_WD_20Type" Er
   <Parameter Name="SalesOrder" Type="Edm.String" Mode="In" MaxLength="10"/>
   <Parameter Name="DraftUUID" Type="Edm.Guid" Mode="In"/>
   <Parameter Name="IsActiveEntity" Type="Edm.Boolean" Mode="In"/>
   <Parameter Name="Opportunity" Type="Edm.String" Mode="In" MaxLength="35"/>
</FunctionImport>
```

3.2 Steps in UI Project (e.g. local OData annotation file)

No changes required.

4. Action with more complex dialog

The next example 'Set Opportunity ID And More' is an action with a more complex dialog. The dialog requires three different parameters to be filled by the user. The parameters are represented by three different controls. For testing this action [Start the DEMO app].



How to achieve that?
4.1 Steps in the in backend (e.g. CDS annotations)

Which type of control will be rendered depends on the data type of the function import parameter inside the metadata description file. The so called 'Entity Data Model (EDM) types' are generated automatically for each property inside the OData service. With which EDM type a function import parameter gets generated depends on the ABAP data type of the corresponding property inside the import parameter structure which was assigned to the BOPF action. See below how the structure was defined – e.g. the *snwd_op_id* is of the built-in type CHAR 35.

```
@EndUserText.label : 'Parameter for action Set Opportunity ID'
@AbapCatalog.enhancementCategory : #NOT_EXTENSIBLE
define structure stta_s_a_setoppid {
        OpportunityString   : snwd_op_id;
        OpportunityBoolean  : bool;
        OpportunityDateTime : dateformat;
}
```

The corresponding ODATA representation – fully generated from the backend, after activating the BOPF object – looks like this:

```
<Parameter Name="OpportunityString" Type="Edm.String" Mode="In" MaxLength="35" Nullable="false" />
<Parameter Name="OpportunityBoolean" Type="Edm.Boolean" Mode="In" />
<Parameter Name="OpportunityDateTime" Type="Edm.DateTime" Mode="In" />
```

In our example the first parameter *OpportunityString* is represented as a simple input field. The EDM type for this input field is 'EDM.String'.

- The first parameter: The little asterisk next to 'OpportunityString' in the dialog indicates that this parameter is mandatory. How a field is set to mandatory will be covered in the next chapter #5 Action dialog with mandatory and optional parameter.
- The second parameter 'OpportunityBoolean' is represented as a checkbox based on the type 'EDM.Boolean'.
- The third parameter 'OpportunityDateTime' is represented as a control named 'date time picker'. The corresponding EDM type for this control is 'EDM.DateTime':

The date time picker always comes with a value help to select the date time. One peculiarity with the EDM type 'EDM.DateTime' is that in the context of a function import always a date time picker control will be rendered. This is a different behavior compared to a filter or a smart field referring to an entity property. Here it depends on the ABAP type if a date time picker or a date picker will be rendered. For example: If the ABAP type is 'DATS' which is an 8 character type to represent a date (but not a time), in case of an entity property the additional OData annotation sap:display-format="Date" will be generated. For the action dialog this additional OData annotation is not created.

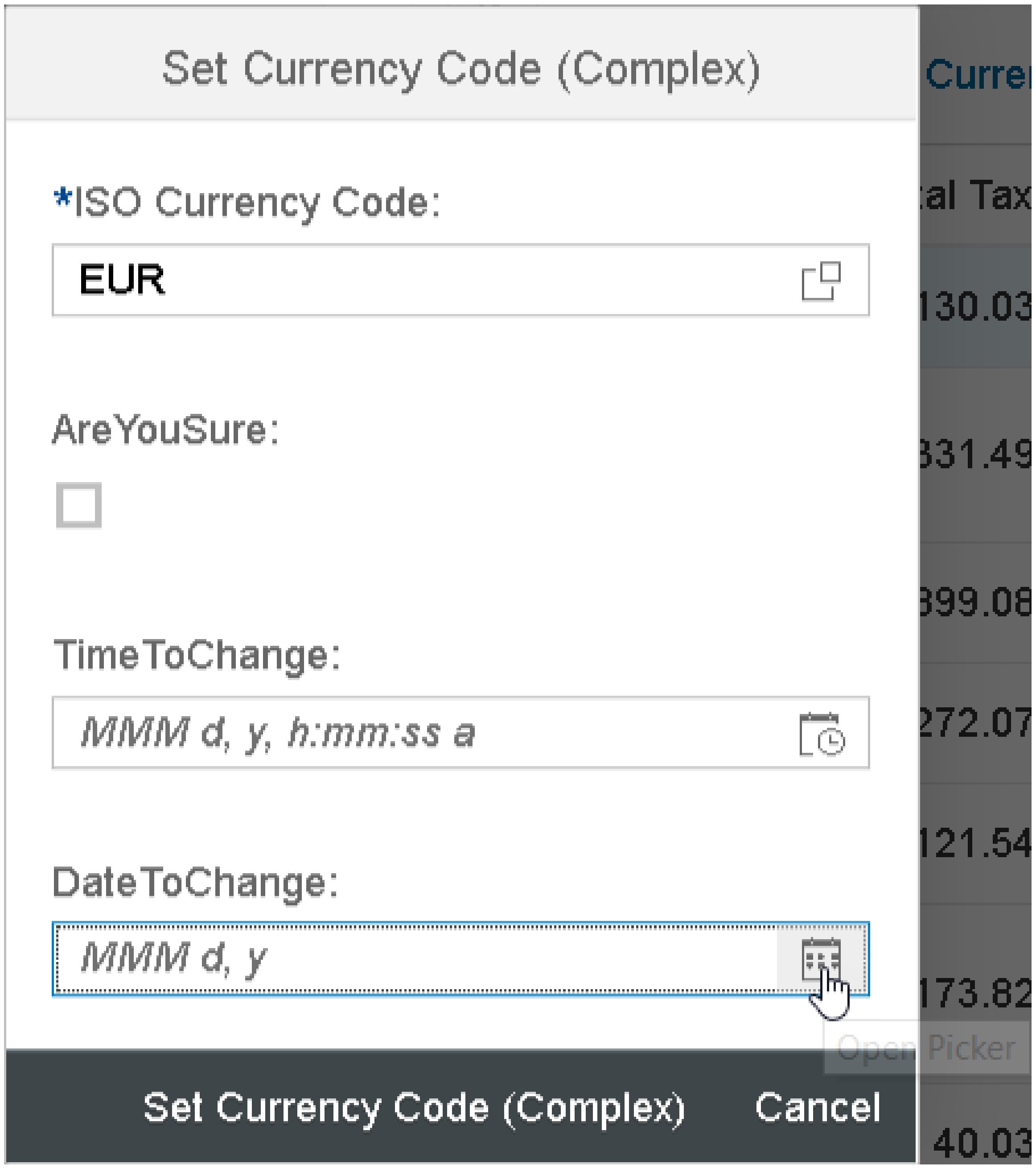
For more information about the EDM types, see here (6. Primitive Data Types).

4.2 Steps in UI Project (e.g. local OData annotation file)

In case you need a Date picker control for your action dialog you need the additional property 'sap:display-format="Date"' in addition to the EDM type 'EDM.DateTime'. This can be added in the local annotation file.

```
<Parameter Name="DateToChange" Type="Edm.DateTime" Mode="In" Nullable="true" sap:display-format="Date" />
```

This annotation will lead to a date picker control instead of a date time picker. See below:


Set Currency Code (Complex)

*ISO Currency Code:

EUR

AreYouSure:

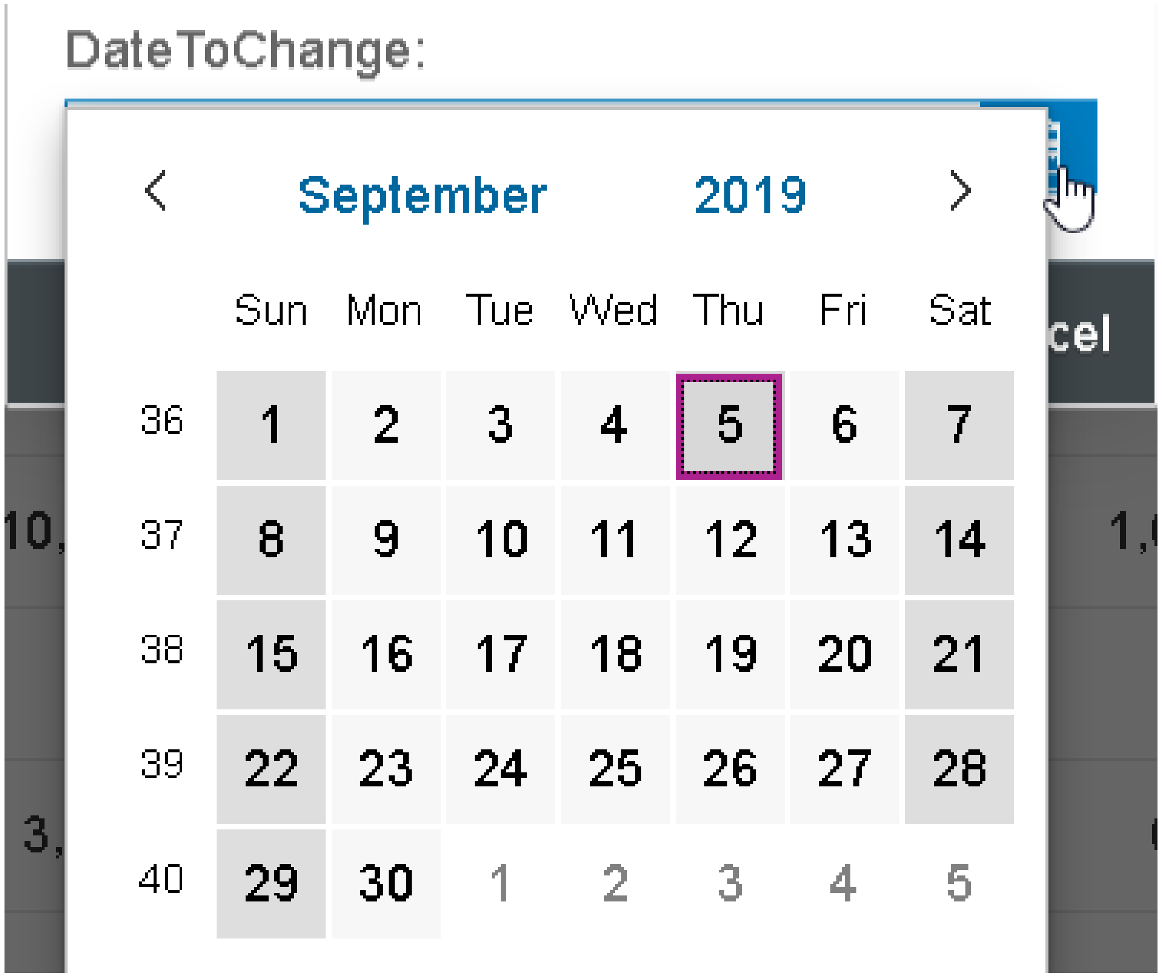
☐

TimeToChange:

MMM d, y, h:mm:ss a

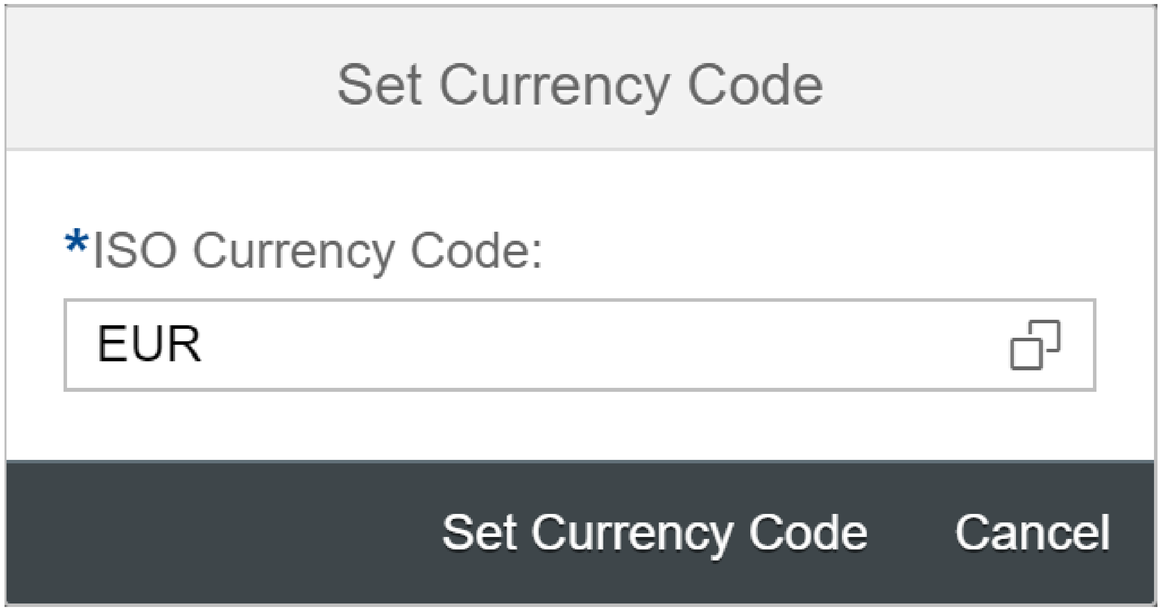
DateToChange:

MMM d, y



5. Action dialog with mandatory and optional parameter

You want to have a dialog with a parameter the user must fill before proceeding. In the last chapter you already could see an example of how a mandatory input field looks like. Another example for a mandatory input field can be seen when executing the action 'Set Currency Code' [Start the DEMO app]



How to achieve that?
5.1 Steps in the in backend (e.g. CDS annotations)
This annotation is not supported by CDS so far.

5.2 Steps in UI Project (e.g. local OData annotation file)
The OData annotation which is needed for the mandatory action parameter looks like this:

```
<Parameter Name="CurrencyCode" Type="Edm.String" Mode="In" MaxLength=”5” Nullable=”false”/>
```

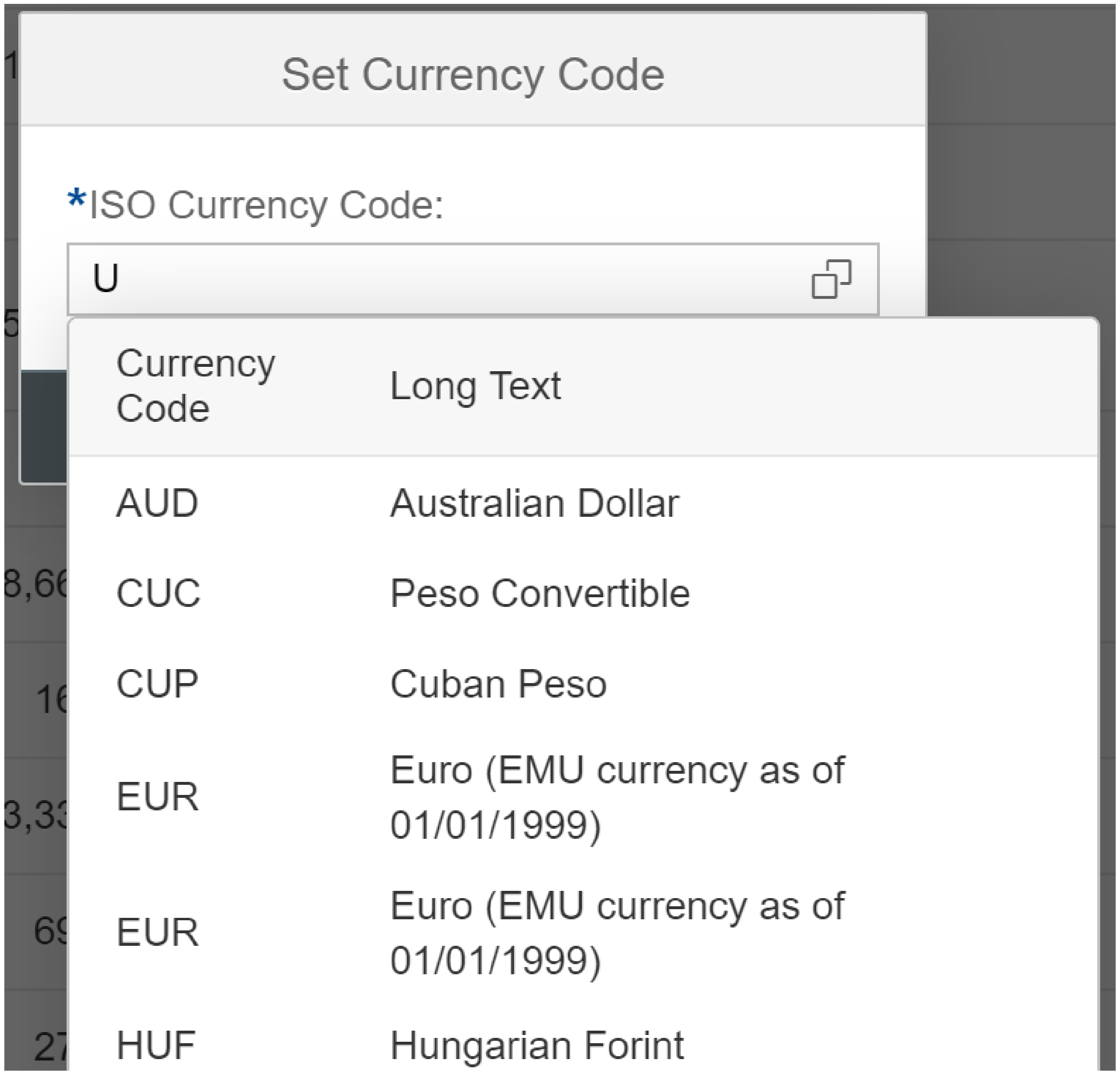
The annotation ‘Nullable=”false”’ will tell the framework explicitly that this parameter is mandatory. To show this to the user the input field will get marked as mandatory (the little asterisk).

With having the annotation ‘Nullable=”true”’, the parameter will be treated as optional.

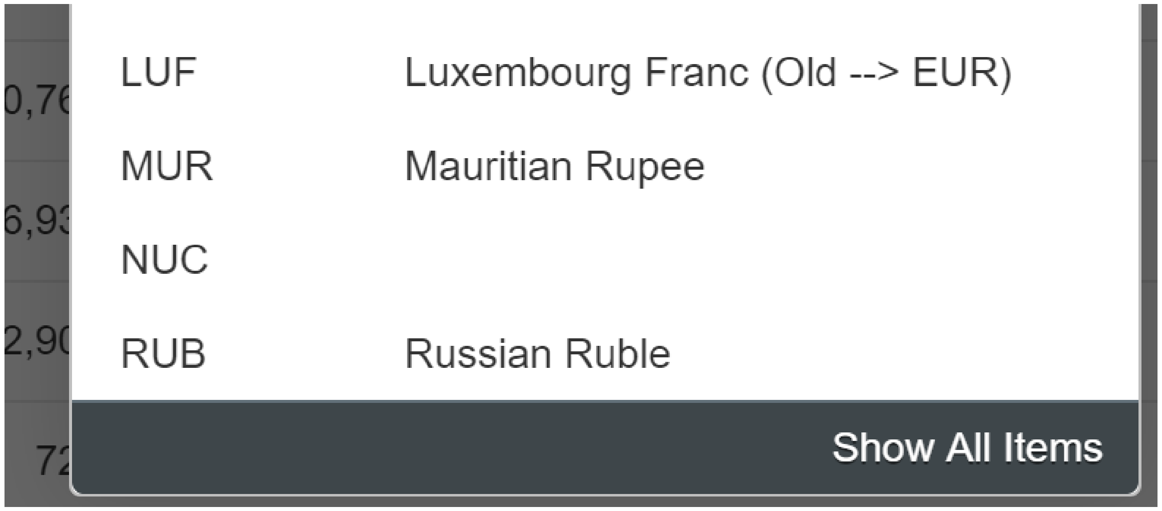
If this annotation Nullable is missing, the parameter will also be treated as mandatory but without the asterisk for the input field. The standard behavior of function imports is, that it treats its parameters as mandatory. Even though, the UI does not mark the fields as mandatory the user will get an error message if he doesn’t pass any value to the input field when executing the action. As this is not very user friendly you should add an additional annotation to the function import parameters to mark them as mandatory or, in case they are optional, change the standard behavior of function imports, so that the parameters are treated as such.

6. Action dialog with value help

Beside the mandatory sign, the input field in the dialog of the action ‘Set Currency Code’ provides a value help. This value help shows a list with possible currency codes for the user to select. It also features a type ahead help which shows possible values in a drop-down list while the user is typing.
[Start the DEMO app]



Currency Code	Long Text
AUD	Australian Dollar
CUC	Peso Convertible
CUP	Cuban Peso
EUR	Euro (EMU currency as of 01/01/1999)
EUR	Euro (EMU currency as of 01/01/1999)
HUF	Hungarian Forint



LUF	Luxembourg Franc (Old --> EUR)
MUR	Mauritian Rupee
NUC	
RUB	Russian Ruble

How to achieve that?
6.1 Steps in the in backend (e.g. CDS annotations)
To get this behavior, an additional value help annotation is needed which has the function import parameter as target.

First, it must be ensured that an entity which provides the value help is part of the OData service. You can have a look on how to model a CDS view to get a value help and how to add it to your service here.

6.2 Steps in UI Project (e.g. local OData annotation file)
If the OData service contains the entity which provides the value help a value help annotation is needed. This annotation must be manually added into the local annotation file. The OData annotation for the value help looks like this:

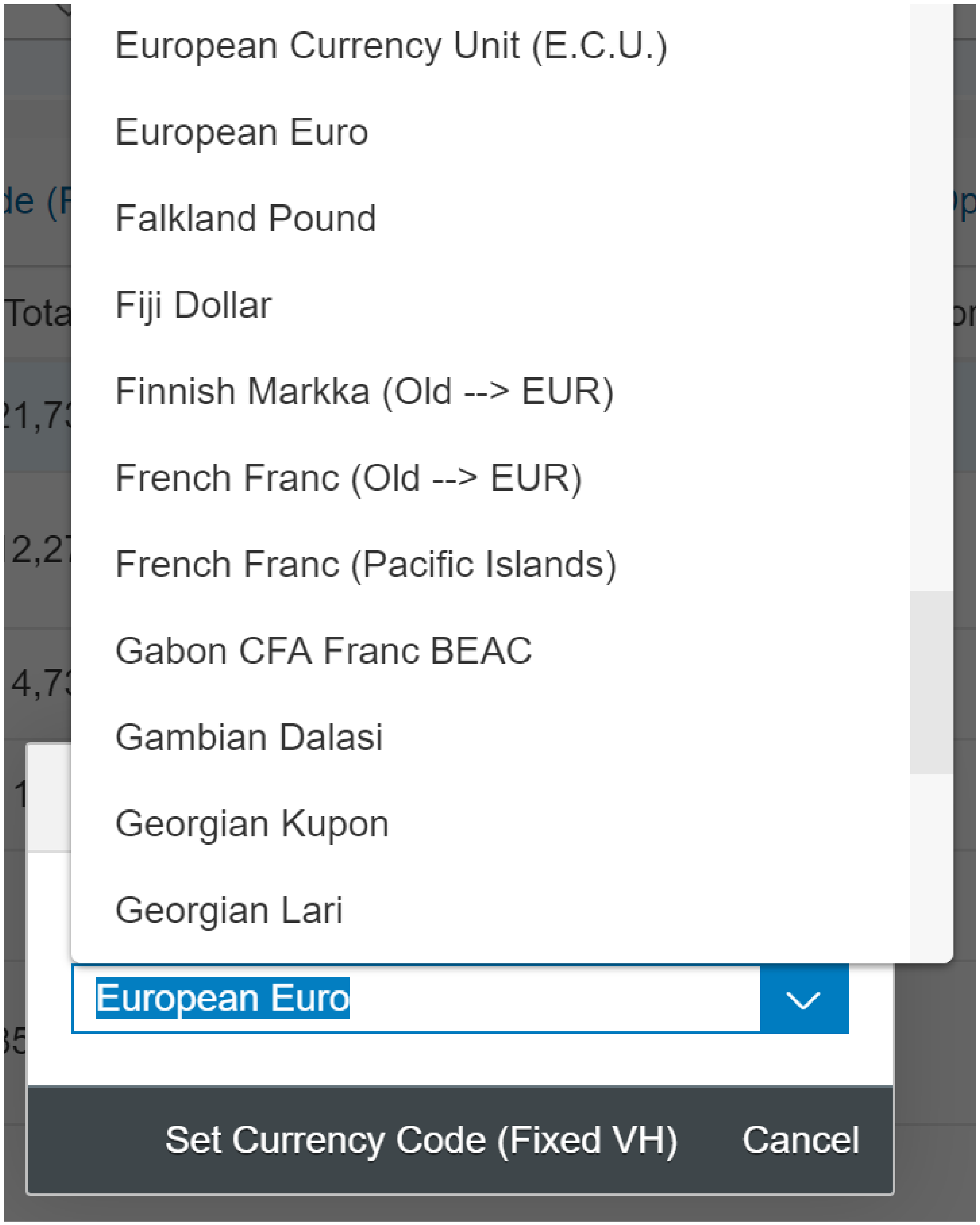
```
<Annotations xmlns="http://docs.oasis-open.org/odata/ns/edm" Target="STTA_SALES_ORDER_WD_20_SRV.STTA_SALES_ORDER_WD_20_SRV_Entities/C
 <Annotation Term="Common.ValueList">
  <Record>
   <PropertyValue Property="Label" String="Value Help for Currency Code" />
   <PropertyValue Property="CollectionPath" String="I_AIVS_Currency_Code" />
   <PropertyValue Property="SearchSupported" Bool="true" />
   <PropertyValue Property="Parameters">
    <Collection>
     <Record Type="Common.ValueListParameterInOut">
      <PropertyValue Property="LocalDataProperty" PropertyPath="CurrencyCode" />
      <PropertyValue Property="ValueListProperty" String="Currency_Code" />
```

```
        </Record>
        <Record Type="Common.ValueListParameterDisplayOnly">
         <PropertyValue Property="ValueListProperty" String="Currency_Code_Text" />
        </Record>
       </Collection>
      </PropertyValue>
     </Record>
    </Annotation>
   </Annotations>
```

The target for this annotation will be the function import parameter *CurrencyCode*. See here open tab “Annotations” and click ValueList to get more insights.

7. Action dialog with drop down value list

The action ‘Set Currency Code (Fixed VH)’ which can be found in the demo app [Start the DEMO app] is an example of an action dialog with a drop-down value help. If there is not enough space as on the picture below, the list opens to the top:

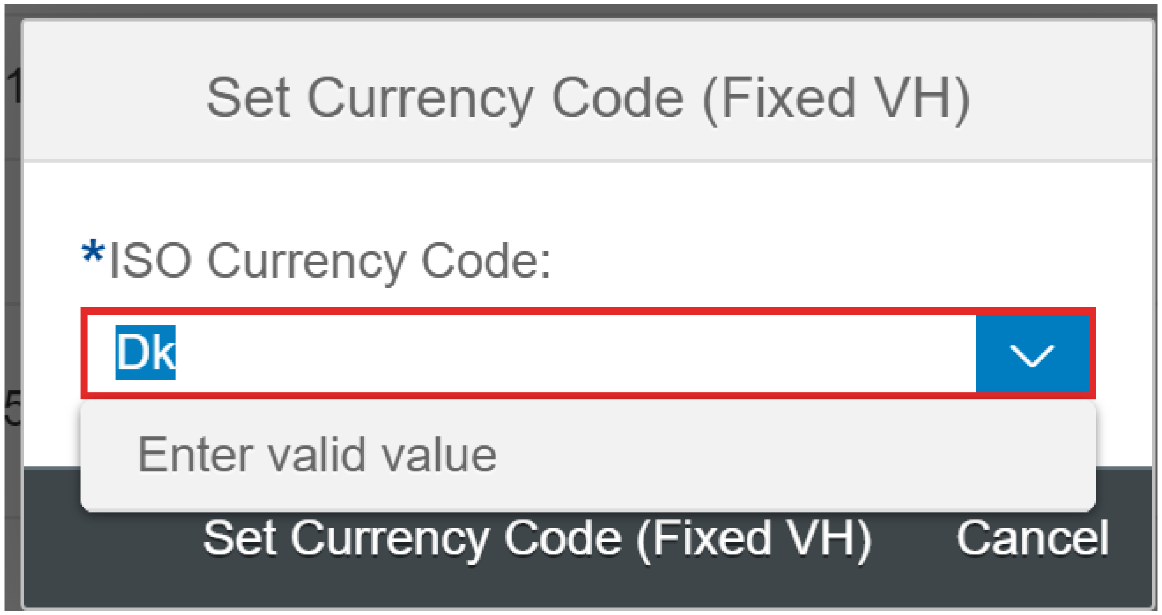


European Currency Unit (E.C.U.)
European Euro
Falkland Pound
Fiji Dollar
Finnish Markka (Old --> EUR)
French Franc (Old --> EUR)
French Franc (Pacific Islands)
Gabon CFA Franc BEAC
Gambian Dalasi
Georgian Kupon
Georgian Lari

European Euro

Set Currency Code (Fixed VH)   Cancel

This is a value help which provides fixed values to the user. It is still possible for the user to type inside the input field, but there is a built-in validation. If the user types in a value which is not part of the drop-down menu the execution of the action won't be possible, and the user gets an error message:



How to achieve that?
7.1 Steps in the in backend (e.g. CDS annotations)
To get this behavior, an annotation in addition to the "normal" value help annotation is needed. This annotation is not supported by CDS so far.

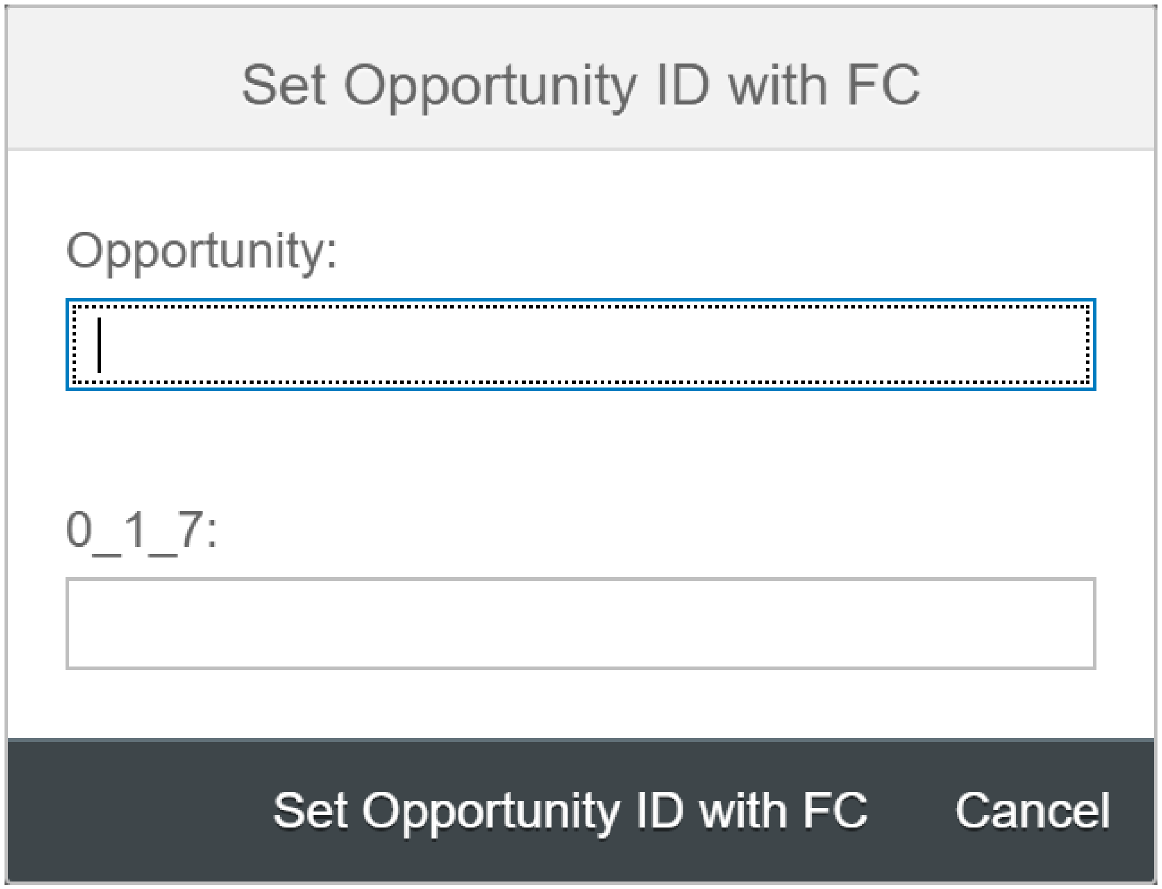
7.2 Steps in UI Project (e.g. local OData annotation file)
This annotation must be added manually into the local annotation file. The OData annotation for the fixed value help looks like this:

```
 <Annotations xmlns="http://docs.oasis-open.org/odata/ns/edm" Target="STTA_SALES_ORDER_WD_20_SRV.STTA_SALES_ORDER_WD_20_SRV_Entities/C
  <Annotation Term="Common.ValueListWithFixedValues" Bool="true" />
</Annotations>
```

The target of this annotation is again the function import parameter. The annotation term which is used is 'Common.ValueListWithFIxedValues" which must be set to true. See here, open tab "Annotations" and click ValueListWithFIxedValues to get more insights.

8. Field Control

Field controls are needed to manipulate the behavior of the fields dynamically. E.g., if a sales order is already shipped, it can't be cancelled anymore. So, based on the data of the selected object, some fields are displayed, other are editable, actions are allowed and so on and so forth. The action 'Set Opportunity ID with FC' is an example of an action dialog with field control. For testing this action you can [Start the DEMO app].



In this example the first input field 'Opportunity' is the 'real' function import parameter. The second input field '0_1_7' represents the field control property. This would usually be hidden and not editable but instead would be calculated dynamically. For demo purpose it is now an editable input field. The user can type in the different values (0, 1 or 7) to see the result on the first input field. The possible values are (can also be seen in here -> "Annotations" -> "FieldControl"):

- 0: 'Hidden': property should not be visible on user interface
- 1: 'Read-Only': property cannot be changed
- 3: 'Optional': property may contain a null value

- 7: ‘Mandatory’: property must not contain a null value

For more information about field control properties please see here.

How to achieve that?
Usually field control properties are part of the metadata but not part of the entity properties. These field control properties encode the behavior of an input field or filter which is bound against an entity property. In order to achieve this there is an attribute for the entity properties called ‘sap:field-control’. This attribute points to the field control property which contains the value which encodes the behavior of the input field.

8.1 Steps in the backend (e.g. CDS annotations)

Usually you get a field control property as soon as you annotate a field in your CDS view in the following way:

```
@ObjectModel.readOnly: ‘EXTERNAL_CALCULATION’
Status_fc
```

or:

```
@ObjectModel.mandatory: ‘EXTERNAL_CALCULATION’
Status_fc
```

Any of these annotations will result in a field control property ‘sap:field-control’ in the metadata. The ‘sap:field-control’ is calculated by the determination ACTION_AND_FIELD_CONTROL of the BOPF object. Then another field shown on the UI, can refer to the field control field in order to define the UI behavior of the field.

CDS Annotations inside the parameter structure definition of the function import are not possible. To still make use of the field control feature for function import parameters it must be set up manually by applying a chain of actions:

First a new field which acts as field control property for the function import parameter needs to be added to the main CDS view. This field should be of type ‘sadl_gw_dynamic_field_property’ and should be annotated as virtual element. The annotations which are needed for this look like this:

```
@ObjectModel.readOnly: true
@ObjectModel.virtualElement: true
@ObjectModel.virtualElementCalculatedBy: ‘ABAP:CL_MY_ABAP_CLASS’
Xyz_fc
```

The first annotation just marks the field as read only as a virtual element must always be read only. The second annotation marks the field as a virtual element whose value will be calculated at runtime. The last annotation sets the ABAP class (in this case the class is named ‘MY_ABAP_CLASS’) which is used for the calculation. This class will now act as a determination for our field control property. For more information about virtual elements and how to use them please refer to this link.

In our demo app the field with its annotations looks like this:

```
SalesOrder.op_id,
      @ObjectModel: {
        readOnly: true,
        virtualElement: true,
        virtualElementCalculatedBy: 'ABAP:CL_STTA_CALCULATE_OP_ID_FC'
      }
      cast( 0 as sadl_gw_dynamic_field_property preserving type )             as opid_fc,
```

After that we also need a function import parameter in the corresponding action which is named the same as the field in the CDS view (in our case ‘opid_fc’). This is important as the binding between those two fields only works if they are named equally:

```
@EndUserText.label : 'Parameter for action Set Opportunity ID'
@AbapCatalog.enhancementCategory : #NOT_EXTENSIBLE
define structure stta_s_a_setoppid {
  op_id   : snwd_op_id;
  opid_fc : sadl_gw_dynamic_field_property;

}
```

8.2 Steps in UI project (e.g. local OData annotation file)

In addition to the set up in the backend, a local OData annotation is needed to link the new field control property to the function import parameter. This annotation must be set manually, since not possible in CDS, as follows:

```
 <Annotations xmlns="http://docs.oasis-open.org/odata/ns/edm" Target="STTA_SALES_ORDER_WD_20_SRV.STTA_SALES_ORDER_WD_20_SRV_Entities/C
  <Annotation Term="com.sap.vocabularies.Common.v1.FieldControl" Path="oppid_fc” />
</Annotations>
```

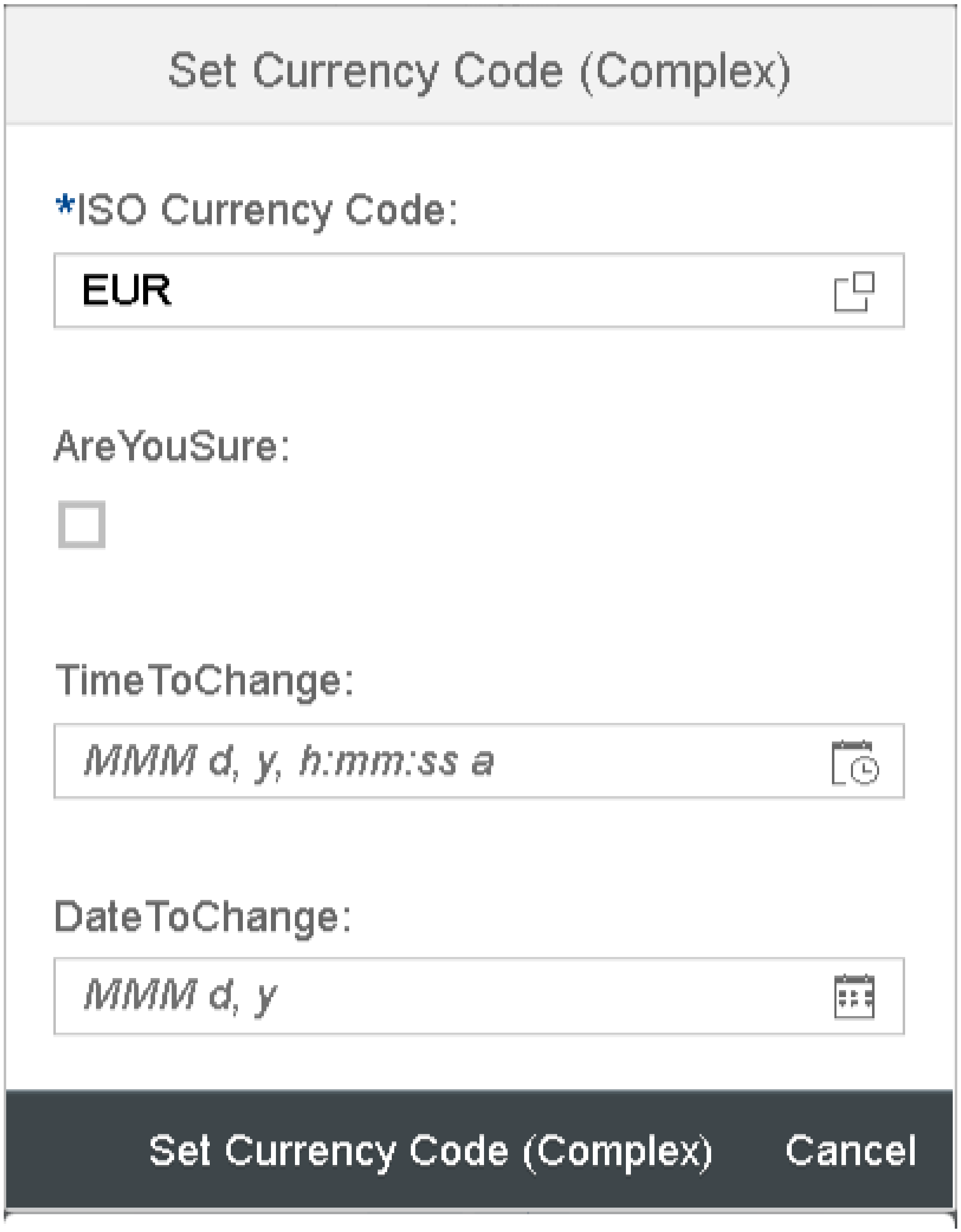
The target of this annotation is the function import parameter. So, in this case it is the function import parameter 'Opportunity' of the function import 'Set Opportunity Id Field Control'. The annotation term shows that this is a field control annotation. The path property of the annotation refers to the virtual element which was created earlier which is used as a field control property.

Now the last thing to do is to hide the field control parameter in the function import. For this we need another annotation which can be added via local annotations:

```
 <Annotations xmlns="http://docs.oasis-open.org/odata/ns/edm" Target="STTA_SALES_ORDER_WD_20_SRV.STTA_SALES_ORDER_WD_20_SRV_Entities/C
  <Annotation Term="UI.Hidden"/>
</Annotations>
```

9. Action dialog changing labels

In the dialogs below you sometimes see a label value combination:



Set Currency Code (Complex)

*ISO Currency Code:

EUR

AreYouSure:

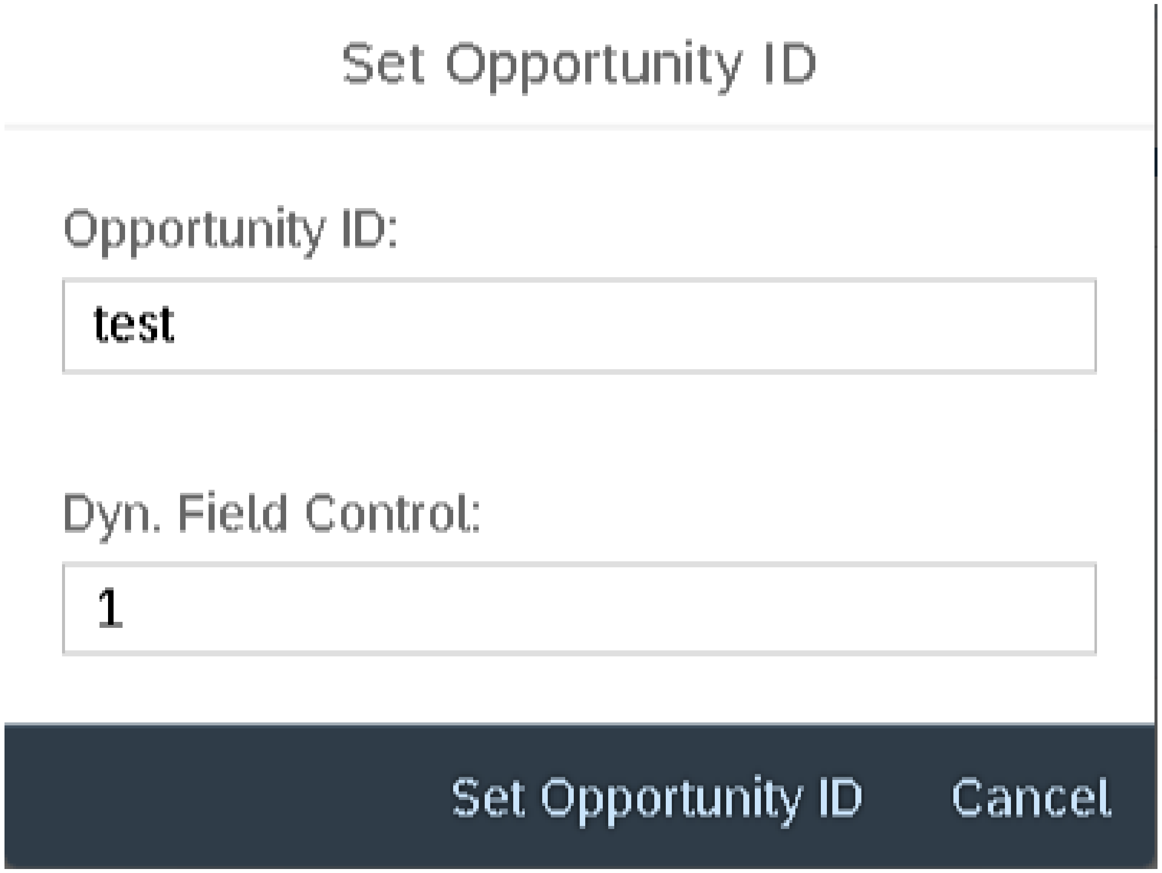
☐

TimeToChange:

MMM d, y, h:mm:ss a

DateToChange:

MMM d, y



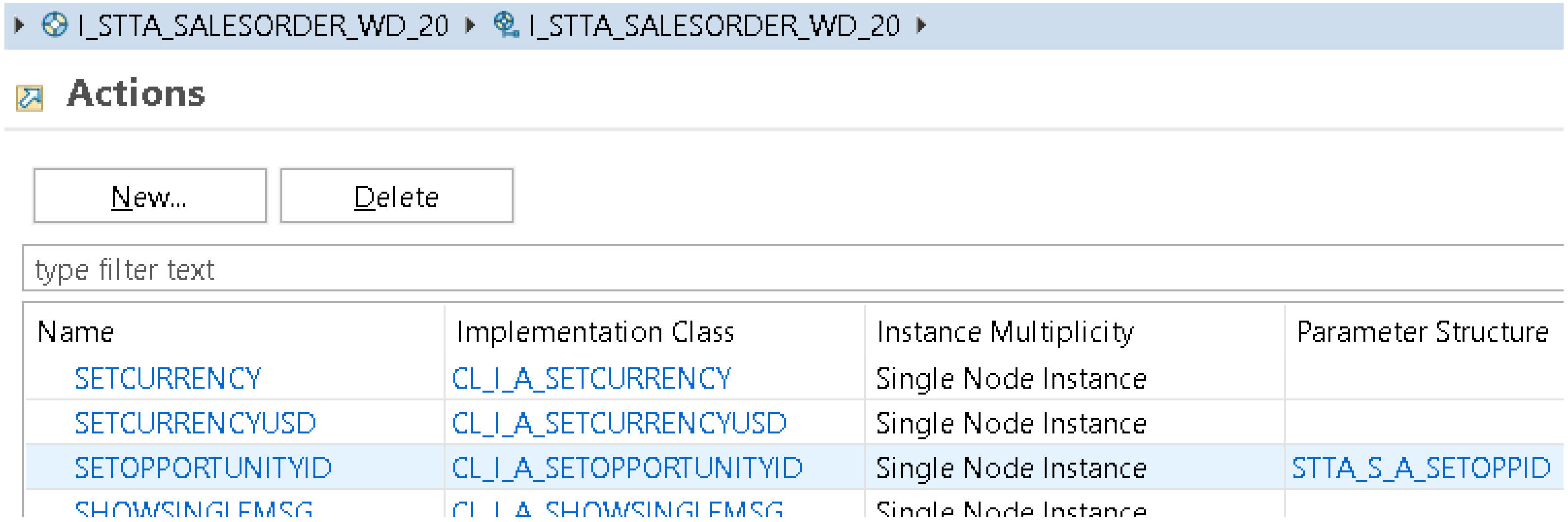
Opportunity ID: test

Dyn. Field Control: 1

The label “ISO Currency Code” or “Opportunity ID” is drawn from the underlying entity type. This is based on property name equality. If there is no equal name, the technical name of the property is shown – as it is the case for “AreYouSure”, “TimeToChange” and “DateToChange”. You will see later how to change that.

How to achieve that?
Knowing the property name equality behavior, you just need to use the same property name for the function import parameter as for the entity type property. Then the label, the type of the value and possible value help will be used in the dialog.

9.1 Steps in the backend (e.g. CDS annotations)
Below we have a BOPF object with an action with a filled parameter structure.


I_STTA_SALESORDER_WD_20 ▸ I_STTA_SALESORDER_WD_20 ▸

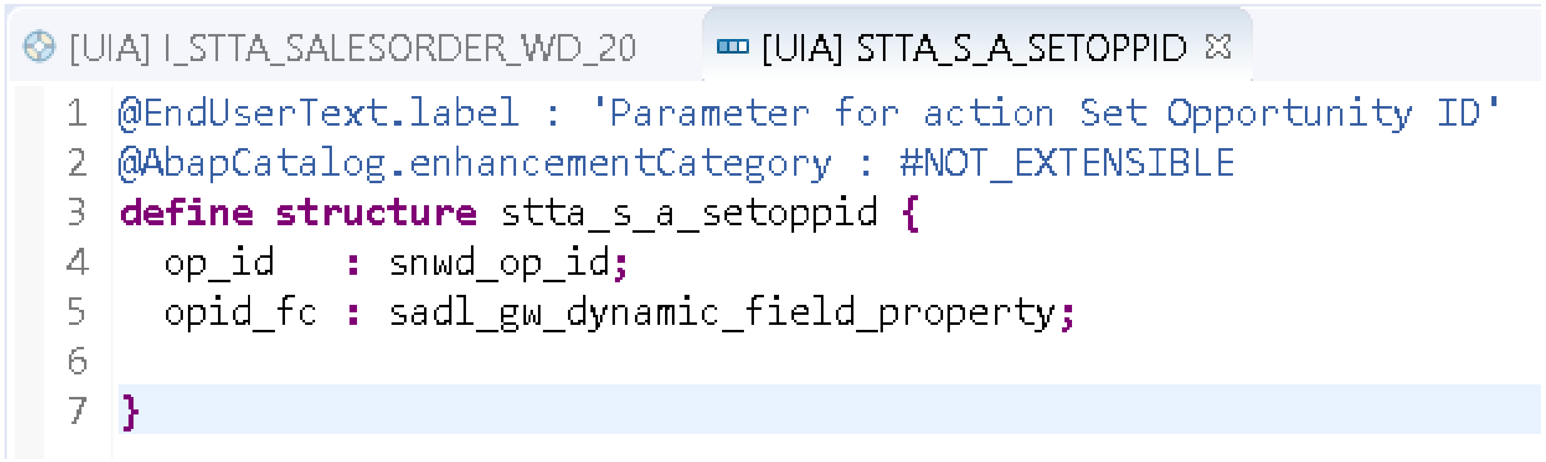
Actions

New... Delete

type filter text

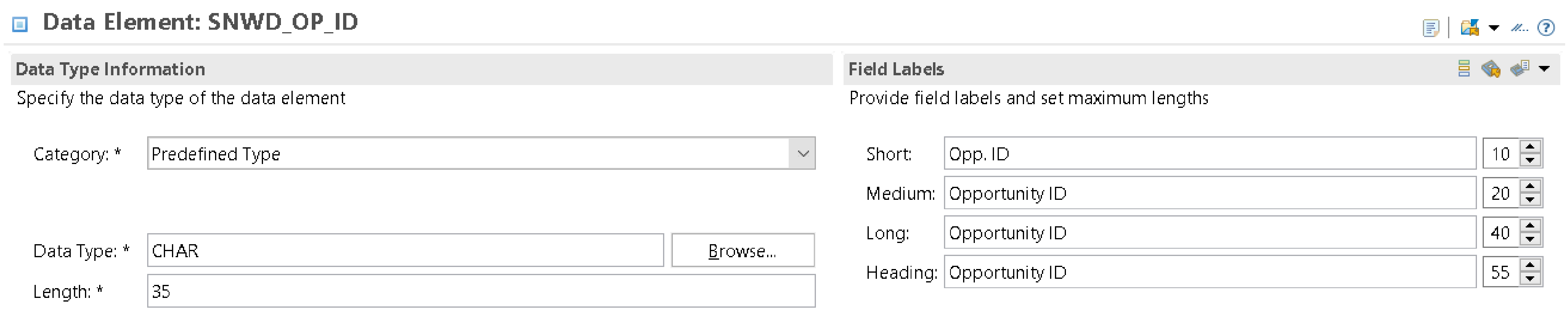
Name	Implementation Class	Instance Multiplicity	Parameter Structure
SETCURRENCY	CL_I_A_SETCURRENCY	Single Node Instance	
SETCURRENCYUSD	CL_I_A_SETCURRENCYUSD	Single Node Instance	
SETOPPORTUNITYID	CL_I_A_SETOPPORTUNITYID	Single Node Instance	STTA_S_A_SETOPPID
SHOWSINGLEMSG	CL_I_A_SHOWSINGLEMSG	Single Node Instance	

The “Parameter Structure” “STTA_S_A_SETOPPID” contains 2 parameters. One is named “op_id”. With the same name and within the main entity type, there is also a property named “op_id” which is again of type “SNWD_OP_ID”. Therefore the action dialog label shows “Opportunity ID”, taken from the “Medium” text.



```
@EndUserText.label : 'Parameter for action Set Opportunity ID'
@AbapCatalog.enhancementCategory : #NOT_EXTENSIBLE
define structure stta_s_a_setoppid {
  op_id    : snwd_op_id;
  opid_fc  : sadl_gw_dynamic_field_property;

}
```


Data Element: SNWD_OP_ID

Data Type Information

Specify the data type of the data element

Category: * Predefined Type

Data Type: * CHAR

Length: * 35

Field Labels

Provide field labels and set maximum lengths

Short:	Opp. ID	10
Medium:	Opportunity ID	20
Long:	Opportunity ID	40
Heading:	Opportunity ID	55

The corresponding ODATA representation – fully generated from the backend – looks like this:

```
<FunctionImport Name="C_STTA_SalesOrder_WD_20Setopportunityid" ReturnType="STTA_SALES_ORDER_WD_20_SRV.DummyFunctionImportResult" m:Ht
   <Parameter Name="node_key" Type="Edm.Guid" Mode="In"/>
   <Parameter Name="IsActiveEntity" Type="Edm.Boolean" Mode="In"/>
   <Parameter Name="op_id" Type="Edm.String" Mode="In" MaxLength="35"/>
   <Parameter Name="opid_fc" Type="Edm.Byte" Mode="In"/>
</FunctionImport>
```

It contains the 2 explicitly defined parameters and also the key fields of the entity. Also generated is the property below. There you see where the label in the end is taken from.

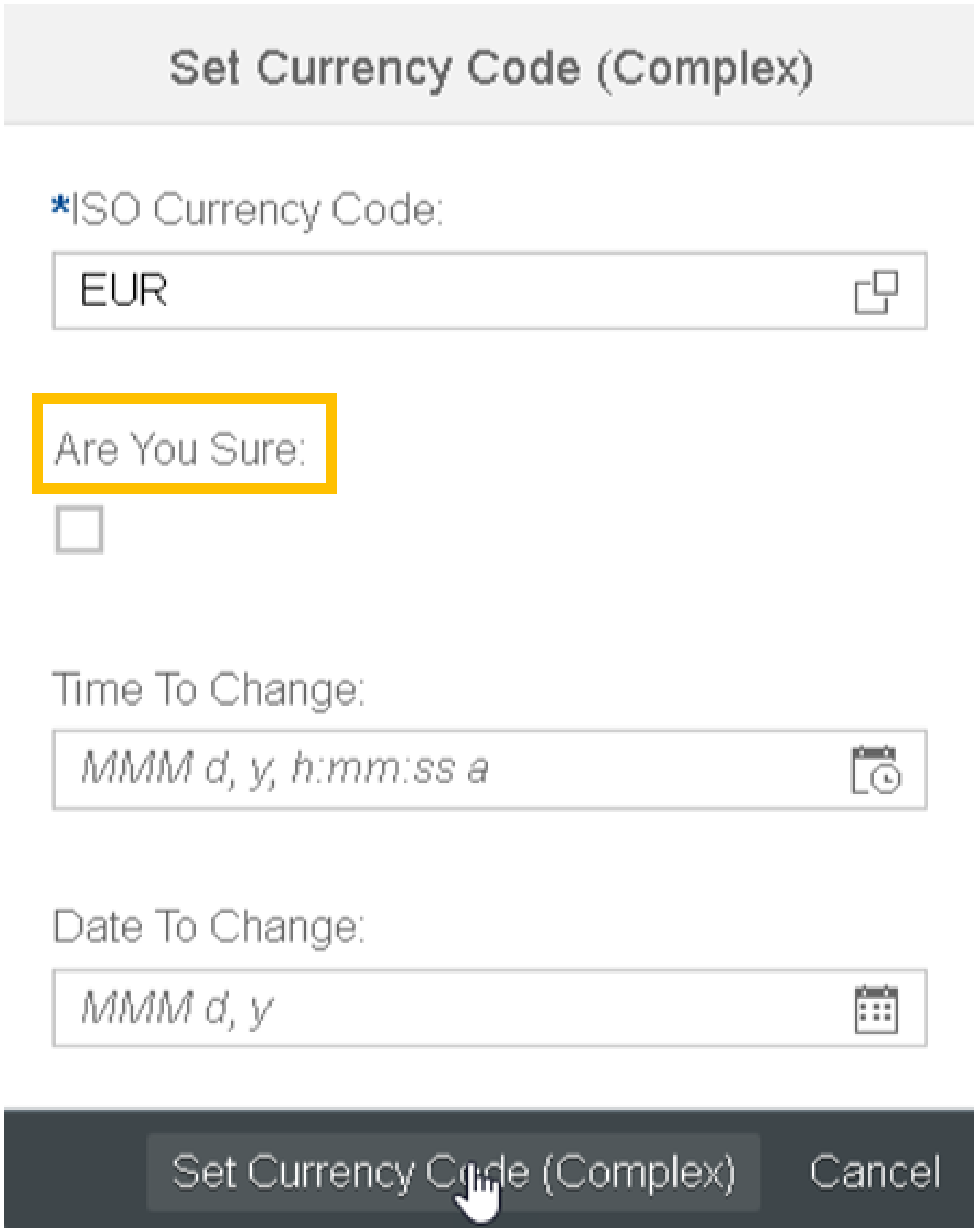
```
<Property Name="op_id" Type="Edm.String" MaxLength="35" sap:display-format="UpperCase" sap:label="Opportunity ID" />
```

9.2 Steps in UI Project (e.g. local OData annotation file)

Since we don't have a corresponding property in the entity type for all fields, like it is with "AreYouSure", we need a local annotation as below – important here is to use translatable i18n keys in the label string:

```
<Annotations Target="STTA_SALES_ORDER_WD_20_SRV.STTA_SALES_ORDER_WD_20_SRV_Entities/C_STTA_SalesOrder_WD_20Setcurrencycodecomplex/Are
    <Annotation Term="com.sap.vocabularies.Common.v1.Label" String="{@i18n>AreYouSure}"/>
</Annotations>
```

Then the UI will show the labels as on the following picture:


Set Currency Code (Complex)

*ISO Currency Code:

EUR

Are You Sure:

☐

Time To Change:

MMM d, y, h:mm:ss a

Date To Change:

MMM d, y

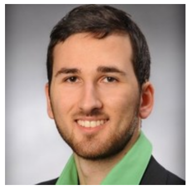
Known Limitations

This chapter will list the known limitations as of UI5 version 1.69 in regards of generating action dialogs. In order to achieve the following points the developer needs to create a breakout and develop his own dialog:

- Multi-line actions: Currently it is not possible to create an action with parameters which is executable for multiple selected records. Even if there are parameters specified the dialog will not be opened.
- More complex dialogs: Currently the framework is only able to generate a dialog with the former described capabilities of user input. If the developer wants to create a more complex dialog (e.g. a dialog with additional read only fields or different controls like charts or sliders) the developer needs to build the dialog by himself.

Authors

This is a common blog post of the following authors mentioned, along with their roles at the time when published.

	Peter Jirsak – Developer in S/4 HANA Produce Manufacturing
	Martin Steinhauser – Product Owner for SAP Fiori Elements List Report

We hope that you as reader and possible Fiori application developer have enjoyed our blog post and are now (better) in the position to decide whether you can realize your action with dialog within the Fiori Elements possibilies using annotations or you need an app extension.

Best regards Peter and Martin

Alert Moderator

Assigned tags

SAP Fiori | SAPUI5 | SAP Fiori Elements | SAP Fiori for SAP ERP | SAP Fiori for SAP S/4HANA |

Related Blog Posts

Fiori Elements – How to Develop a List Report – Using Local Annotations
By **Jocelyn Dart** , Nov 27, 2016

Fiori Elements – How to develop a List Report - Basic Approach
By **Jocelyn Dart** , Nov 16, 2016

Getting Started with SAP Fiori elements Video Series: Creating an Analytical List Page Application now available
By **Conrad Bernal** , Sep 19, 2019

Related Questions

Fixed value List with CDS annotations
By **Former Member** , Jul 13, 2017

Variant management not working on Fiori Elements
By **Former Member** , Nov 27, 2018

How to use ObjectIdentifier in SmartTable Fiori Elements List Report?
By **Katrin Petzold** , Sep 17, 2018

24 Comments

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

Mahesh Kumar Palavalli

October 21, 2019 at 2:36 am

This is one of the best blog on Fiori elements so far!!!

Thanks for sharing!!

Like(4)

---

abhijeet kankani

October 22, 2019 at 3:36 pm

Excellent

Like(2)

Jocelyn Dart

October 24, 2019 at 7:31 am

Thanks Peter! I've added this to the wiki https://wiki.scn.sap.com/wiki/display/Fiori/Fiori+elements

Like(1)

Syambabu Allu

October 24, 2019 at 3:27 pm

Absolutely,Best blog on SAP Fiori Elements

Like(1)

Ben Patterson

October 24, 2019 at 11:21 pm

Hi Peter and Martin,

thank you so much for such a well explained and comprehensive explanation of this topic. This will be a valuable resource.

I have a question regarding section 1.1 where the action is added to the BOPF object, and inherently applies to the remaining approaches. The object is in the SAP namespace, is this done as a modification? And if so what would be the proposed approach to avoid modification as a customer?

Best regards, Ben.

Like(1)

Martin Steinhauser

October 28, 2019 at 8:59 am

Hi Ben,

Thanks for the question. The blog post has been written in the situation in which Peter and me were “*your Product Owner wants you to build an app*” from scratch (1. Build own app).

In general and simplified there can be 2 scenarios:

1. Build own app — In this scenario, and if you are bound to the same technical components, you can use the blog post as guideline 1:1. Of course, if you are a partner or customer building an own app then you would not use the SAP namespace.

Side comment: The mentioned “*app extensions (also known as custom or breakout actions)*” in the blog post have to be seen in the context of Fiori Elements as library with a defined (and limited) feature scope. “*App extensions: Are made by* [Fiori Elements] *developers during the creation of an SAP Fiori elements-based app*“, if the developer needs something beyond the feature scope of Fiori Elements. See here.

2. Extending shipped apps — you want to change an app of someone else. Shipped by SAP… . This refers to "*Adaptation extensions: Let customers and partners introduce their own functionality to an existing app*". See here. In that scenario you can use the blog post as information source to know the artifacts that you need to create somehow. E.g. you might need to do stuff in the backend, (maybe) create an own function import, adding fields to a parameter structure, changing the annotations, … and you might need to do something in the frontend as well.

For extending shipped apps I just collected the following for you:
– "Extensibility" wiki overview page about Extensibility
– "Extensibility" help.sap.com documentation in context of "SAP Fiori Apps for SAP NetWeaver"
– "Extending a SAP Fiori Elements application using Adaptation Projects" video
– "SAPUI5 Flexibility: Adapting UIs Made Easy", especially "*I enable my apps for key user adaptation or adapt my SAP Fiori elements apps*"
– "Extending Delivered Apps Using Adaptation Extensions"

Best regards Martin

Like(0)

Ben Patterson

October 30, 2019 at 12:47 am

Hello Martin,

thanks for your reply and even though the blog is limited to the scope you have stated it is still a very helpful resource for the 'Build own app' scenarios.

Unfortunately none of the links provided help when you need to extend the **actions** a shipped app, based on **CDS BOPF.** So far I have not found anything that provides a straight forward approach to this. The custom fields and logic scenarios are quite limited and cannot be used for many shipped applications. My small hope was that you may have been privy to some insight in this area.

Thanks again for your input.

Kind regards,

Ben.

Like(0)

Jan Nyfeler

November 14, 2019 at 2:23 pm

I do not see how to set the Nullable="false" Annotation in the annotation modeller.

I have BOPF actions with a Parameter structure, so I do not see how I could alter the generated metadata....

Edit: I did it via Model Provider Class

Like(1)

Mohit Bansal

November 26, 2019 at 7:11 am

Good write up Peter !! Very informative.

Like(0)

Sergey Demianchenko

December 6, 2019 at 8:50 am

Very informative. Untill this blog I thought that Fiori programming model is elegant, but useless for compex transactional applications.

Still many quastions remains:

- Is it possible to set up static (context-independent) action with (local, MPC_EXT, CDS?) annotations? "Create special" for instance? In test I defined static BOBF action. But framework didn't care. I have to select item in the list to be able to press the button.
- Is it possible to influence the size of pop-up screen for customer actions? Define facets?
- Is it possible to set up default values for action parameters?

- What is the best way to make BOPF aware through which consumption view/fiori application it accessed (to alter behavior)?
- I didn't quite get how you managed to annotate locally sap:display-format="Date"? Actually, I failed with this annotation even with MPC_EXT class. Code lo_annotation->add(iv_key = 'display-format' iv_value = 'Date') didn't work specifically for action parameters in my test system. Bug or fearture?

Regards,

Sergey

Like(0)

Suresh Babu Muthusamy

June 10, 2020 at 8:27 pm

The below code you mentioned above is related to MetaData.XML but you mentioned to add it in annotation and I confused with that. How did you manage to do this? I have put a DATS field but it is not working and still I get the Date/Time Picker.

```
<Parameter Name="DateToChange" Type="Edm.DateTime" Mode="In" Nullable="true" sap:display-format="Date" />
```

Regards,

Suresh.

Like(1)

Suresh Babu Muthusamy

June 10, 2020 at 9:47 pm

```
<FunctionImport Name="ZC_DASTPCreate_time_slots" ReturnType="ZC_DASTP_CDS.ZC_DASTPType" EntitySet="ZC_DASTP" m:HttpMethod="POST"
sap:action-for="ZC_DASTP_CDS.ZC_DASTPType" sap:applicable-path="Create_time_slots_ac"><Parameter Name="db_key" Type="Edm.Guid" Mode="In'
```

I think he wrongly mentioned the metadata.xml as annotation. I added the same in metadata.xml in the WebIDE project and it worked for me.

I added this in the Function import in metadata.xml. And this worked good.

sap:display-format="Date" sap:label="Start Date"

Like(1)

Suresh Babu Muthusamy

June 10, 2020 at 8:29 pm

The below code you mentioned above is related to MetaData.XML but you mentioned to add it in annotation and I confused with this. Where should I add this in the annotation.xml and can you please let us know. I see that there is also one more member who has my same doubt. How did you manage to do this? I have put a DATS field but it is not working and still I get the Date/Time Picker instead of Date control.

```
<Parameter Name="DateToChange" Type="Edm.DateTime" Mode="In" Nullable="true" sap:display-format="Date" />
```

Regards,

Suresh.

Like(0)

Attila Berencsi

January 29, 2021 at 8:55 am

Hello Suresh,

metadata is delivered by the backend. You can include your CDS in SEGW as reference, and adjust metadata delivered by SADL in the Model Provider Class.

If your SEGW service is not CDS based, but manual implementation, you can tick/untick any time the nullable property.

Doing local changes in WebIDE or SAP Business Application works until next refresh of metadata from backend.

BR,Attila

Like(0)

Louis-Arnaud BOUQUIN

July 20, 2020 at 2:24 pm

Hello,

Is there a way to call the function from the extension API (method InvokeActions) with the generated dialog for parameters ?

Thanks

Like(0)

Virendra Pathak

August 11, 2020 at 5:45 pm

Hi Peter, I am stuck at an important juncture;

I am using a CDS generated OData and created a Fiori Elements List report application. I have added an BOPF action on the object page and the corresponding Action in BOPF (have followed the steps as well in the blogs) below is the error being faced.

1. When i use the instance multiplicity of the BOPF action as "Single node instance" the Fiori element app on click of that button throws an error stating " "This action cannot be performed for the selected object"
2. When i change the instance multiplicity to "Static Action" then the control flow to the class but the it_keys are not passed in the method interface. The BO keys of the existing selected row from the list are not being sent.

Any help will be appreciated, as i am not sure what else is being missed here.

Like(0)

CHIH-JUNG HO

September 16, 2020 at 9:02 am

Hi,

I have the same problem. Did you find a solution?

Like(0)

Andrew Fordham

August 14, 2020 at 9:35 am

Hi Peter,

Fantastic blog!

I have a question about the formatting of parameter dates, which you discuss in 4.2. You specify that the 'sap:display-format="Date" should be added to the local annotation file, but the code snippet that follows looks like something from the service-generated metadata file rather than the annotation.xml file.

Where in the Fiori Elements app can I enter the 'sap:display-format="Date" to change my parameter to a date picker?

Many thanks,

Andrew

Like(0)

Sunil H

August 25, 2020 at 5:32 pm

Yes, Peter. The field is in metadata file rather than annotation file.

Like(0)

Vaibhav Goel

October 14, 2020 at 6:44 pm

Hi Sunil,

When using BOPF and oData.publish: true annotations.

How exactly we add that ‘sap:display-format=”Date” in the metadata.

Can you please help.

Regards,

Vaibhav

Like(0)

---

ramjee korada

January 23, 2021 at 4:34 pm

Hi Peter,

very nice blog!

Is it possible to prepopulate values for some fields in the dialog dynamically using CDS annotations in ABAP RAP?

Regards,

Ramjee.

Like(1)

Attila Berencsi

January 29, 2021 at 10:02 am

Hi Ramjee,

there is no such, at the moment at least.

I think the way the field control is imlemented in point 8, we can pass values for identical property names in the dialog for action input parameters from the entity instance for which the action is defined. What I do not understand at the moment, that how the field opid_fc is selected by the OData request by Fiori elements in point 8 in the mentioned example, due this should be also hidden on the list report (to not disturb users), it is normally not contained then in the SELECT list. Maybe I miss sg here conceptually.

I am using an abstract entity for action import parameters in RAP, and will try the trick mentioned in point 8. If works great, if not I have some workaround ideas, like setting the property of the entity manually in the model for the entity instance for which the action is invoked. There does not matter it is selected by the template or not, but it needs kind a lot workaround coding for the action (maybe better then to implement custom dialog)

BR,Attila

Like(0)

Attila Berencsi

January 29, 2021 at 9:37 am

Hello,

great blog!

Just an update to point 7.

CDS Annotation

@ObjectModel.resultSet.sizeCategory: #XS

can be used to render a ComboBox(dropdown) in Fiori Elements/Smartfield. It will result in the required annotation file. This can be aplied in Metadata extensions or custom CDS views.

Of course it was always possible to reference the CDS in SEGW and alter SADL generated metadata programmatically in the model provider class, but such operation is bit more complicated, than simply using CDS annotations.

BR,Attila

Like(0)

Axel Mohnen

February 19, 2021 at 2:11 pm

Hi Peter,

thanks lot for the excellent blog.

I'm trying to build a action dialog with an text area (e.g. multiLineText: true).

My parameter data element is "STRINGVAL". Unfortunately, I only see a simple input field in the dialog.

Which annotation or vocabulary is needed therefore?

Thanks,

Kind regards

Axel

Like(0)