Technical Articles

Alexandre Therrien

August 29, 2019 | 15 minute read

Follow

Like

RSS Feed

ABAP / Fiori Elements – Adding BOPF Create Remove Update Delete (CRUD) Actions to CDS Views: Building from Scratch Walkthrough Part 2

6 15 12,617

See Part 1 to see how the data architecture was made.

https://blogs.sap.com/2019/08/09/abap-fiori-elements-organizing-your-data-architecture-using-cds-views-building-from-scratch-walkthrough-ui-and-logic/

Note: I will be using ABAP Development Tools for this tutorial.

Note: If you do not know what is BOPF, you should check Oliver Jaegle's blogs on this website. I will only cover how to make actions from BOPF in this blog, not what it is. This link could be useful, which is the first part of his **BOPF to the future** blogs. Also, this link could be useful to understand how actions are made (part 7 of his **BOPF to the future** blogs), although I had some difficulties understanding the code because it mentioned some objects that I had no idea what they were. I have added my own documentation in this blog for you to understand the parts that I did not.

In the previous tutorial, I made a data architecture using CDS views. In this tutorial, I am going to add actions to the different entities that composed my data architecture.

I used an annotation in the previous tutorial called **@UI.facet.targetElement** in the CDS view **ZCPDB_OFFICE_TP**. In this tutorial, the importance of that annotation is going to be shown.

Front-end JSON Code: Displaying Create and Delete Buttons

I finished the previous tutorial by showing the GUI of the created SAP Fiori Elements project. I will reference that project because one of the file is required to get modified from the frontend. The modification of this file will consequently show all of the **Create** and **Delete** buttons to create/ remove some CDS entity's instances. That file is **manifest.json**. This file is created as soon as the SAP Fiori template project is created on SAP Web IDE. You will have to change the **pages** entry, which is inside of the **sap.ui.generic.app** entry, to the below value:

WARNING: The following JSON code is **character sensitive**. If you misspell any CDS view's name or put a capital letter instead of a lowercase letter, you could get weird errors in your application. For example, you could click on the **Create** button generated from an entity and it could return an error. Also, some of the **Create** or **Delete** buttons could be invisible from entries that do not perfectly correspond.

```
"pages": {
            "ListReport|ZCPDB_OFFICE_TP": {
                "entitySet": "ZCPDB_OFFICE_TP",
                "component": {
                    "name": "sap.suite.ui.generic.template.ListR
                    "list": true,
                    "settings": {
                        "condensedTableLayout": true,
                        "smartVariantManagement": true,
                        "enableTableFilterInPageVariant": true
                    }
                },
                "pages": {
                    "ObjectPage|ZCPDB_OFFICE_TP": {
                        "entitySet": "ZCPDB_OFFICE_TP",
                        "component": {
                            "name": "sap.suite.ui.generic.templa
                        },
                                "pages": {
                                    "ObjectPage|to_Workstation": {
                                        "navigationProperty": "to_Works
                                        "entitySet": "ZCPDB_WORKSTATION_
                                        "component": {
                                            "name": "sap.suite.ui.generi
                                        }
                                    },

                                    "ObjectPage|to_Employee": {
                                        "navigationProperty": "to_Emplo
                                        "entitySet": "ZCPDB_EMPLOYEE_TP'
                                        "component": {
                                            "name": "sap.suite.ui.generi
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
```

This will generate the missing **Create / Delete** buttons that were not appearing by default. Looking back at the previous tutorial, only a **Create** button was missing for the entity **Employee**. With this JSON input, the **Create** button will appear.

So this snippet of JSON code above describes the flow of your application as seen through the GUI. Here are some things you have to keep in mind to understand the JSON code above:

- **ZCPDB_OFFICE_TP** is the name of my root CDS consumption view (Consumption view = CDS for UI).
- The **entitySet** entry is the name of a CDS consumption view.
- The entries **ObjectPage|to_Workstation** and **ObjectPage|to_Employee** are related to some annotation I used in the **ZCPDB_OFFICE_TP**, which is the **@UI.facet.targetElement** annotation. **ObjectPage** is a certain type of model page that Fiori Elements generates. **to_Workstation** and **to_Employee** relate to the **@UI.facet.targetElement** values **_Workstation** and **_Employee** I have set in the CDS view **ZCPDB_OFFICE_TP**. Notice that the **navigationProperty** uses the same **to_Workstation** and **to_Employee** values.
- Why does the entry **ObjectPage|ZCPDB_OFFICE_TP** does not have a **navigationProperty** nor a **targetElement**? Because it is the root CDS view.
- Notice the **component** entry is always added to describe the type of page that we are using to show the entities. I used a **ListReport** page to show the **Office** entities,

and that changes to **ObjectPage** when I select an **Office** entity.
The **ZCPDB_OFFICE_TP** has two entries with different **component**s: one with the prefix **ListReport** and another with the prefix **ObjectPage**. This is because I can see the **Office** entity's information once I have selected it. See my pictures in the conclusion to notice that I see some **Office** information from the **ObjectPage** (all the pictures in the conclusion show the **ObjectPage**)

- Also notice the order of the entries. First, we put the root CDS view **Office** entity **ZCPDB_OFFICE_TP**, the one we see as we open the **ListReport** application. Then, when we click on an **Office** entity, we get information about this **Office**. This is why the entry **ListReport|ZCPDB_OFFICE_TP** contains its own **pages** entry. However, when we click on an **Office** entity, we also see some **Employee**s and **Workstation**s associated to that **Office**. Thus, the **Office** entity contains **Employee**s and **Workstation**s. This is why the entry **ObjectPage|ZCPDB_OFFICE_TP** also contains its own **pages** entry.

Creating Custom Actions Using BOPF

Take the case where you would like to modify the gender of one instance of **Employee**. The first thing you are required to do would be to add an action to the entity **Employee** using **BOPF**. To do so, you have to check into the generated **BOPF** Business Object (BO) that gets created when your data architecture has been approved without any errors. See below:

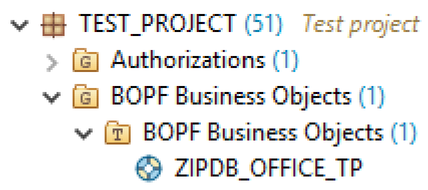
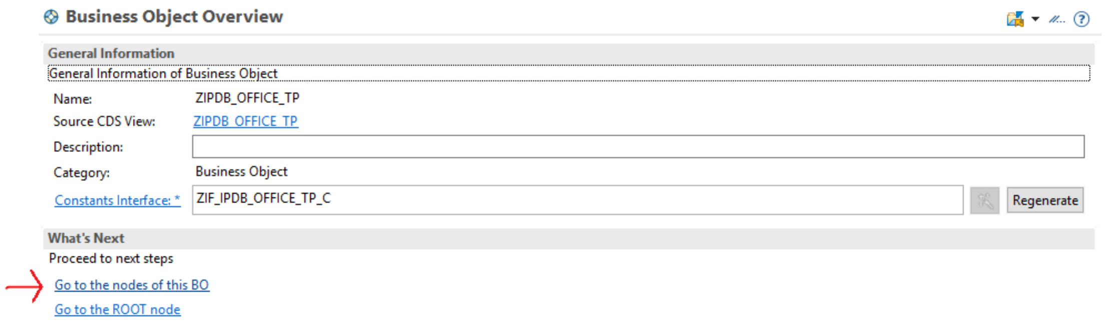

TEST_PROJECT (51) Test project
Authorizations (1)
BOPF Business Objects (1)
BOPF Business Objects (1)
ZIPDB_OFFICE_TP

Figure 1.1, BOPF BO in Projects Explorer in Eclipse
The business object is called **ZIPDB_OFFICE_TP**, just like one of the CDS views.

Double click on that object, and you will see the below screenshot. Press on "**Go to the nodes of this BO**" to get to the **Employee** entity.



Business Object Overview

General Information
General Information of Business Object

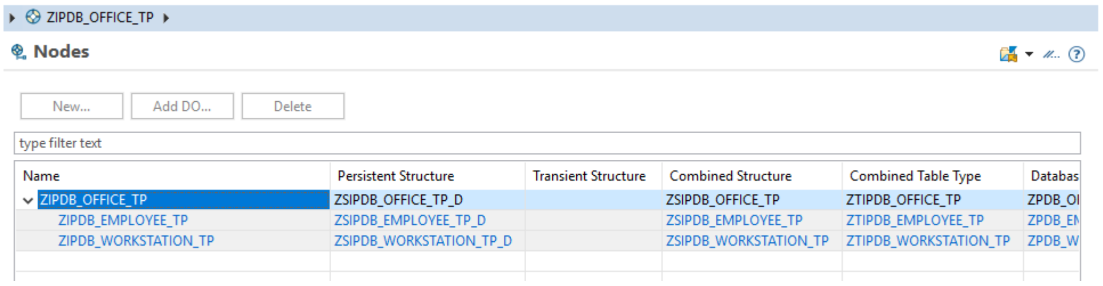
Name:	ZIPDB_OFFICE_TP
Source CDS View:	ZIPDB_OFFICE_TP
Description:	
Category:	Business Object
Constants Interface: *	ZIF_IPDB_OFFICE_TP_C

What's Next
Proceed to next steps
Go to the nodes of this BO
Go to the ROOT node

Figure 1.2, BOPF Main View

This nodes view shows the different nodes of our Business Object (BO). The node **ZIPDB_OFFICE_TP** is the parent node to **ZIPDB_EMPLOYEE_TP** and **ZIPDB_WORKSTATION_TP**. The node **ZIPDB_OFFICE_TP** is also the root node (As you can see, it is the first entry, and that node expands into its children nodes). Remember those as they will be used later in the implementation of the **BOPF** code.

Then press on "**Ctrl+Click**" on the **Name** field of the entity **Employee** called **ZIPDB_EMPLOYEE_TP** (another of the CDS view's name) to get to that object.


ZIPDB_OFFICE_TP

Nodes

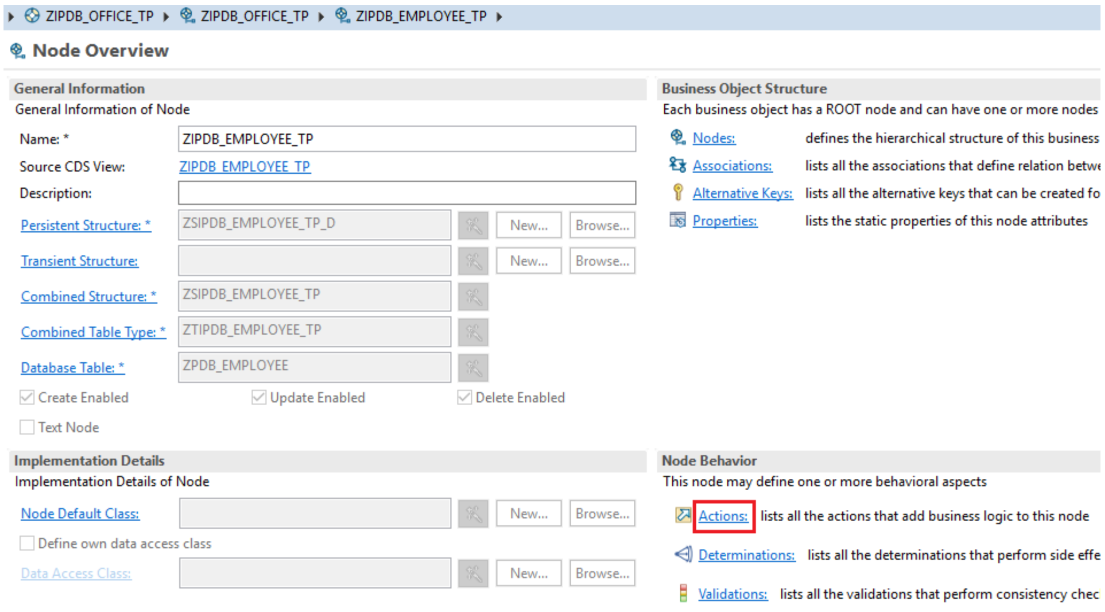
New... Add DO... Delete

type filter text

Name	Persistent Structure	Transient Structure	Combined Structure	Combined Table Type	Databas
ZIPDB_OFFICE_TP	ZSIPDB_OFFICE_TP_D		ZSIPDB_OFFICE_TP	ZTIPDB_OFFICE_TP	ZPDB_OI
ZIPDB_EMPLOYEE_TP	ZSIPDB_EMPLOYEE_TP_D		ZSIPDB_EMPLOYEE_TP	ZTIPDB_EMPLOYEE_TP	ZPDB_EI
ZIPDB_WORKSTATION_TP	ZSIPDB_WORKSTATION_TP_D		ZSIPDB_WORKSTATION_TP	ZTIPDB_WORKSTATION_TP	ZPDB_W

Figure 1.3, BOPF Nodes View

Then, create the action under the **Employee** entity by clicking on the underlined **Actions**.


ZIPDB_OFFICE_TP ▸ ZIPDB_OFFICE_TP ▸ ZIPDB_EMPLOYEE_TP

Node Overview

General Information

General Information of Node

Name: *	ZIPDB_EMPLOYEE_TP	
Source CDS View:	ZIPDB_EMPLOYEE_TP	
Description:		
Persistent Structure: *	ZSIPDB_EMPLOYEE_TP_D	New... Browse...
Transient Structure:		New... Browse...
Combined Structure: *	ZSIPDB_EMPLOYEE_TP	
Combined Table Type: *	ZTIPDB_EMPLOYEE_TP	
Database Table: *	ZPDB_EMPLOYEE	

[✓] Create Enabled [✓] Update Enabled [✓] Delete Enabled

[ ] Text Node

Business Object Structure

Each business object has a ROOT node and can have one or more nodes

Nodes:	defines the hierarchical structure of this business
Associations:	lists all the associations that define relation betw
Alternative Keys:	lists all the alternative keys that can be created fo
Properties:	lists the static properties of this node attributes

Implementation Details

Implementation Details of Node

Node Default Class: New... Browse...

[ ] Define own data access class

Data Access Class: New... Browse...

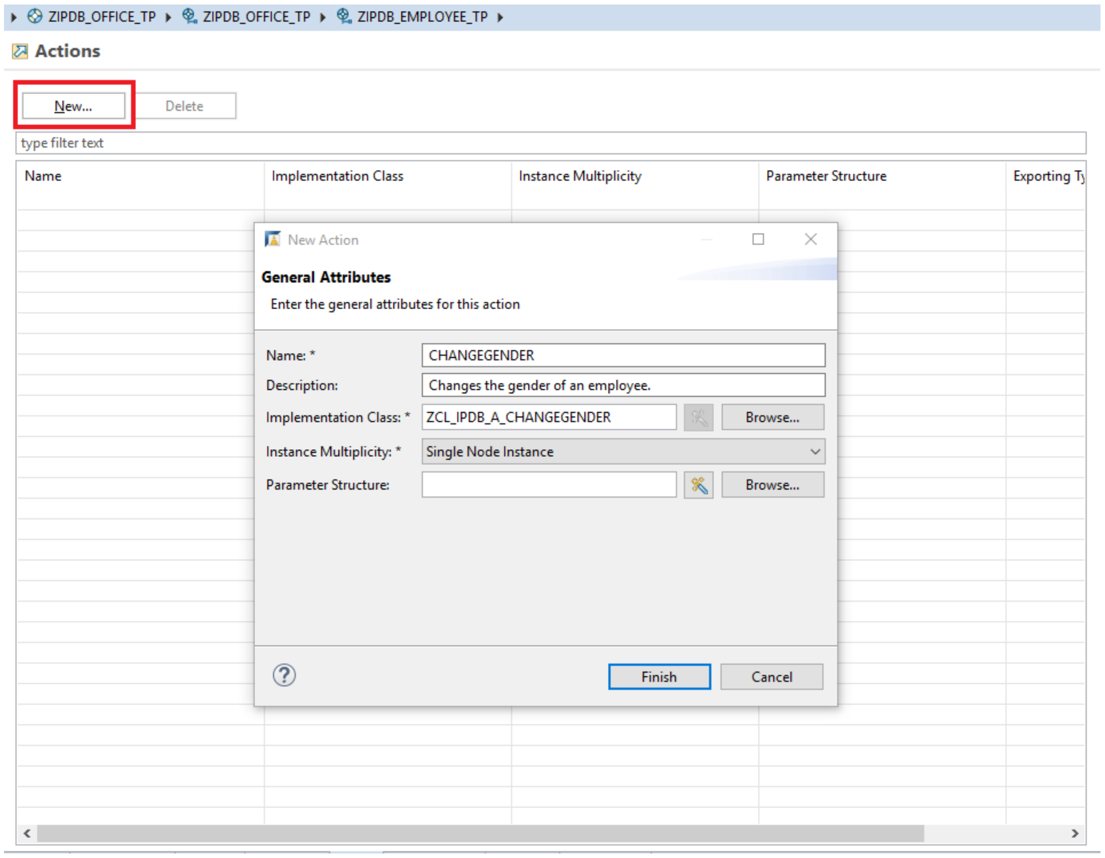
Node Behavior

This node may define one or more behavioral aspects

Actions:	lists all the actions that add business logic to this node
Determinations:	lists all the determinations that perform side effe
Validations:	lists all the validations that perform consistency chec

Figure 1.4, BOPF Employee Node

Click on the **New...** button to create a new action. This will generate a new popup. Fill it with these values for the purpose of this custom action:



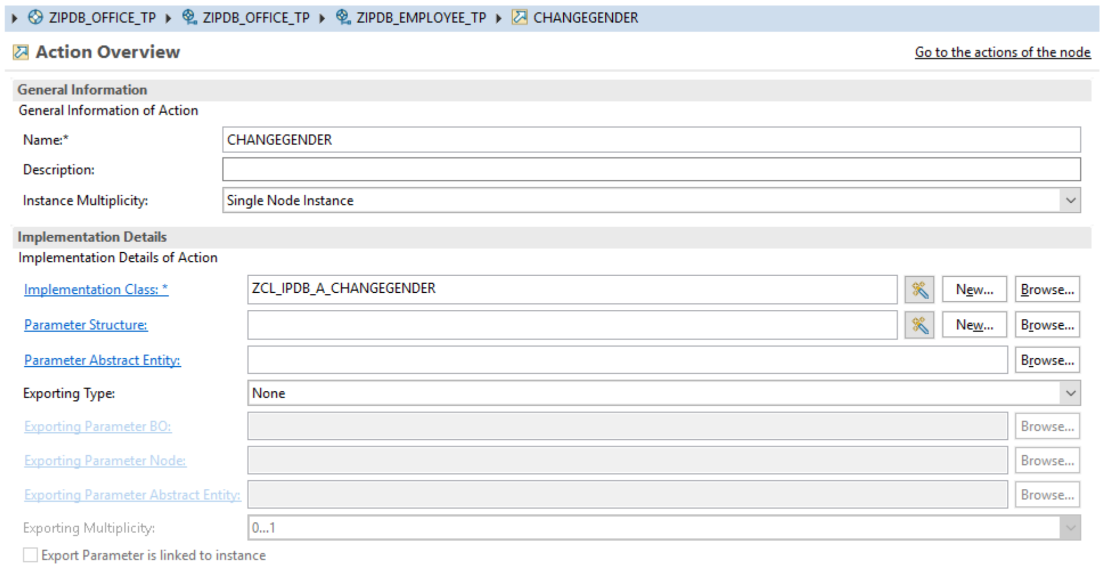
Name: * CHANGEGENDER
Description: Changes the gender of an employee.
Implementation Class: * ZCL_IPDB_A_CHANGEGENDER
Instance Multiplicity: * Single Node Instance
Parameter Structure:

Figure 1.5, BOPF Creating an Action

The **Implementation Class**' name is generated by default while typing the name of the action.

I believe the **Instance Multiplicity** parameter defines how many instances you are taking as input.

Click on **Finish**. Then, your action will appear in the list of actions. If it does not, save and activate the BOPF Business Object, or wave your mouse on the first available line from the top and it should appear (no idea why). Then, go into that action by pressing "**Ctrl+Click**", and you will get something similar to the following screen:


ZIPDB_OFFICE_TP ▸ ZIPDB_OFFICE_TP ▸ ZIPDB_EMPLOYEE_TP ▸ CHANGEGENDER

Action Overview — Go to the actions of the node

General Information
General Information of Action

Name:*	CHANGEGENDER
Description:	
Instance Multiplicity:	Single Node Instance

Implementation Details
Implementation Details of Action

Implementation Class: *	ZCL_IPDB_A_CHANGEGENDER
Parameter Structure:	
Parameter Abstract Entity:	
Exporting Type:	None
Exporting Parameter BO:	
Exporting Parameter Node:	
Exporting Parameter Abstract Entity:	
Exporting Multiplicity:	0...1

☐ Export Parameter is linked to instance

Figure 1.6, BOPF Changing an Action's Parameters

I have already set the **Exporting Type** to **None**. That is because I am returning nothing from the action I am about to create. If you forget to set it to **None**, you will get an error mentioning something about a returned value **ZCPDB_Employee_TPType** from **OData**, because the program is expecting a returned value that you are not giving.

Once you are done, save and activate the BO. Then, create a class with the same name as the implementation class' name. In this case, **ZCL_IPDB_A_CHANGEGENDER**.

Coding

In the beginning, after doing some modifications, the class should look like this:

```
class ZCL_IPDB_A_CHANGEGENDER definition
  public
  inheriting from /BOBF/CL_LIB_A_SUPERCL_SIMPLE
  final
  create public .

public section.
  methods /BOBF/IF_FRW_ACTION~EXECUTE
      redefinition .
protected section.
private section.
ENDCLASS.
```

```
CLASS ZCL_IPDB_A_CHANGEGENDER IMPLEMENTATION.

  METHOD /BOBF/IF_FRW_ACTION~EXECUTE.

  ENDMETHOD.

ENDCLASS.
```

The modifications I made show that this class is inheriting from the BOBF class **/BOBF/CL_LIB_A_SUPERCL_SIMPLE**, and that we are going to modify the method **/BOBF/IF_FRW_ACTION~EXECUTE**, which is the method that an action is going to execute. There is another method name when you create a determination or a validation. You can also know that from looking at Oliver Jaegle's blogs.

Constructing the Method CHANGEGENDER

Updating and Reading BOPF Instances

```
class ZCL_IPDB_A_CHANGEGENDER definition
  public
  inheriting from /BOBF/CL_LIB_A_SUPERCL_SIMPLE
  final
  create public .

public section.
  methods /BOBF/IF_FRW_ACTION~EXECUTE
      redefinition .
protected section.
private section.
ENDCLASS.



CLASS ZCL_IPDB_A_CHANGEGENDER IMPLEMENTATION.

  METHOD /BOBF/IF_FRW_ACTION~EXECUTE.
    "Notice the type of the object: where is this object mentior
    "If you look in bobf, this is the combined table type associ
    data lt_item type ztipdb_employee_tp.

    "You can find the generated interface in your project struct
    "find the right one. You need to know this because the firs
    "zif_ipdb_office_tp_c, which is the interface for the root c
    io_read->retrieve(
    exporting
    iv_node = zif_ipdb_office_tp_c=>sc_node-zipdb_employee_tp
    it_key = it_key

    importing
    et_data = lt_item ).

    "now modifying the item. Remember: you received a table of i
    "on the items that corresponded with your search criteria fr
    "Normally returns one result because we are searching for a
    LOOP AT lt_item REFERENCE INTO DATA(ls_item).

      "Checks if the employee's gender contains the string 'M'.
      IF ls_item->gender CS 'M'.
        ls_item->gender = 'F'.

      "Checks if the employee's gender contains the string 'F'.
      ELSEIF ls_item->gender CS 'F'.
        ls_item->gender = 'M'.
      ENDIF.

      "parameters are pretty easy to understand. iv_node is the
      "the key of the entity, and is_data is the data we want to
      io_modify->update(
        iv_node = zif_ipdb_office_tp_c=>sc_node-zipdb_employee_t
        iv_key = ls_item->key
        is_data = ls_item ).

    ENDLOOP.

  ENDMETHOD.

ENDCLASS.
```

Some of the comments in this code will help you understand what is happening. Also, let me tell you that the method **execute** has different objects it receives as input, such as the **io_read** and **io_modify** objects, which allows you to perform CRUD operations on some data you have. You also received the table **it_key** which gives you the keys for the entity received as input.

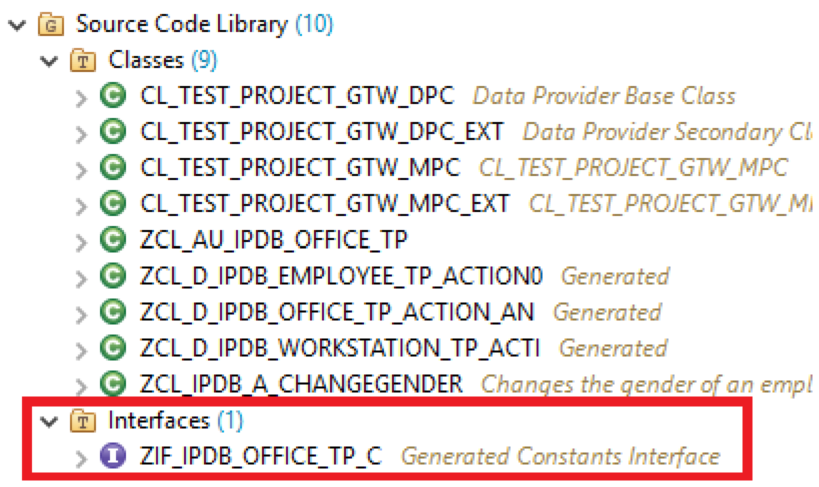
The **sc_node** item is what you must use to access the different entities in your BOPF Business Object instance.

io_read->retrieve does the Read operation. All you need to provide the method is **iv_node**, which is the type of entity concerned (in our case, the **Employee** entity. Remember the nodes that made up our Business Object? **Figure 1.3**), and the key, also known as the unique identifier of the object **it_key**. Then, you get your object as a table outputted from the method as **et_data**.

io_modify->update does the Update operation. The only parameter different from **io_read->retrieve** is the **is_data** parameter, which is your new updated object.

Now something that disturbed me when I looked at BOPF tutorials online was that I never knew what some of the code above meant. What is **zif_ipdb_office_tp_c**? What is **ztipdb_employee_tp**?

You will notice in your project architecture that you have an interface that has been generated (I believe at the same time as your BOPF Business Object was generated, which is when your CDS view data architecture has been approved because it has no errors). See the below image.

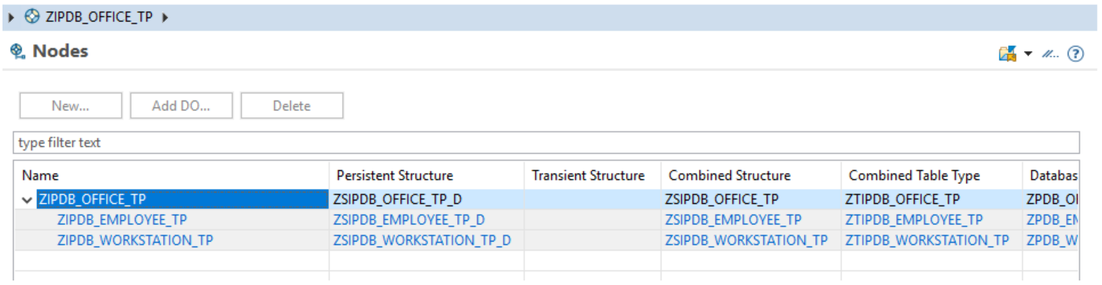


```
Source Code Library (10)
  Classes (9)
    CL_TEST_PROJECT_GTW_DPC  Data Provider Base Class
    CL_TEST_PROJECT_GTW_DPC_EXT  Data Provider Secondary Cl
    CL_TEST_PROJECT_GTW_MPC  CL_TEST_PROJECT_GTW_MPC
    CL_TEST_PROJECT_GTW_MPC_EXT  CL_TEST_PROJECT_GTW_M
    ZCL_AU_IPDB_OFFICE_TP
    ZCL_D_IPDB_EMPLOYEE_TP_ACTION0  Generated
    ZCL_D_IPDB_OFFICE_TP_ACTION_AN  Generated
    ZCL_D_IPDB_WORKSTATION_TP_ACTI  Generated
    ZCL_IPDB_A_CHANGEGENDER  Changes the gender of an empl
  Interfaces (1)
    ZIF_IPDB_OFFICE_TP_C  Generated Constants Interface
```

Figure 1.6, BOPF's Generated Interface from the Project Explorer in Eclipse
So that answers your question on where **zif_ipdb_office_tp_c** comes from.

What about **ztipdb_employee_tp**? Or any other weird objects that have a similar name to the CDS view of **ZIPDB_EMPLOYEE_TP**?

You have to look into the generated BOPF object for that. You can type transaction **bobf** or you can go in Eclipse, go to your BOPF generated object **ZIPDB_OFFICE_TP**, and double click on it. Then click on **Go to the nodes of this BO**. This is the screen you will now see. (Same as **Figure 1.3**) Now look at the **Combined Structure** and **Combined Table Type**. The **Combined Table Type** corresponds to the object **ztipdb_employee_tp**.



Name	Persistent Structure	Transient Structure	Combined Structure	Combined Table Type	Databas
ZIPDB_OFFICE_TP	ZSIPDB_OFFICE_TP_D		ZSIPDB_OFFICE_TP	ZTIPDB_OFFICE_TP	ZPDB_OI
ZIPDB_EMPLOYEE_TP	ZSIPDB_EMPLOYEE_TP_D		ZSIPDB_EMPLOYEE_TP	ZTIPDB_EMPLOYEE_TP	ZPDB_EI
ZIPDB_WORKSTATION_TP	ZSIPDB_WORKSTATION_TP_D		ZSIPDB_WORKSTATION_TP	ZTIPDB_WORKSTATION_TP	ZPDB_W

(Refers back to Figure 1.3, BOPF Nodes View)

Creating Instances with BOPF (Optional)

I wanted to add this because I did struggle a little to find out what were the parameters that this method was requesting.

To create a new employee instance, I used the **io_modify->create** method. I will add this method right after the **io_modify->update** method call, and right before the statement **ENDLOOP**. This is how I did it:

```
io_modify->create(
      iv_source_node_key = zif_ipdb_office_tp_c=>sc_node-zipdb_c
      iv_source_key = ls_item->parent_key
      iv_assoc_key = zif_ipdb_office_tp_c=>sc_association-zipdb_
      iv_node = zif_ipdb_office_tp_c=>sc_node-zipdb_employee_tp
      is_data = ls_item
      ).
```

I will now describe the different parameters of this function.

- **iv_source_node_key**: this is your root node. Look back at the information above **Figure 1.3**.
- **iv_source_key**: this is the parent's node key. This is the key of the **Office** object, which is the root node.
- **iv_assoc_key**: this is the key representing the association between **Office** and **Employee**. **sc_association** is a generated instance variable from the interface. You do not decide of its name. **zipdb_office_tp** is our root node, and **_employee** is the **targetElement** I have set in **ZCPDB_OFFICE_TP** for **Employee.**
- **iv_node**: this is the node that will have this new item. (our new item is of type **Employee**) **sc_node** is a generated instance variable from the interface. You do not decide of its name.
- **is_data**: the data we want this node to contain.

If you do not recall what is a node, go back to **Figure 1.3**. There are 3 nodes in this BO.

Removing Instances with BOPF (Optional)

I added this after the **ENDLOOP** statement. It will remove the currently selected element.

```
io_modify->delete(
      iv_node = zif_ipdb_office_tp_c=>sc_node-zipdb_employee_tp
      it_key = it_key
      ).
```

The parameters are straight forward now that we have looked at some other CRUD operations, but here is a description of them:

- **iv_node**: this is the node that has the item you want removed (we want to remove an **Employee** instance).
- **it_key**: this is the key of the object you want removed. Remember **it_key** is given as a parameter to this function, **/BOBF/IF_FRW_ACTION~EXECUTE.**

If you do not know what is a node, go back to **Figure 1.3**. There are 3 nodes in this BO.

Final Method Implementation if you Followed the Optional Parts

This is what the code looks like after having followed the optional parts:

```
class ZCL_IPDB_A_CHANGEGENDER definition
  public
  inheriting from /BOBF/CL_LIB_A_SUPERCL_SIMPLE
  final
  create public .

public section.
  methods /BOBF/IF_FRW_ACTION~EXECUTE
      redefinition .
protected section.
private section.
ENDCLASS.



CLASS ZCL_IPDB_A_CHANGEGENDER IMPLEMENTATION.

  METHOD /BOBF/IF_FRW_ACTION~EXECUTE.
    "Notice the type of the object: where is this object mentio
    "If you look in bobf, this is the combined table type assoc
    data lt_item type ztipdb_employee_tp.
```

```
"You can find the generated interface in your project struc
"find the right one. You need to know this because the firs
"zif_ipdb_office_tp_c, which is the interface for the root 
io_read->retrieve(
exporting
iv_node = zif_ipdb_office_tp_c=>sc_node-zipdb_employee_tp
it_key = it_key

importing
et_data = lt_item ).

"now modifying the item. Remember: you received a table of 
"on the items that corresponded with your search criteria f
"Normally returns one result because we are searching for a
LOOP AT lt_item REFERENCE INTO DATA(ls_item).

  "Checks if the employee's gender contains the string 'M'.
  IF ls_item->gender CS 'M'.
    ls_item->gender = 'F'.

  "Checks if the employee's gender contains the string 'F'.
  ELSEIF ls_item->gender CS 'F'.
    ls_item->gender = 'M'.
  ENDIF.

  "parameters are pretty easy to understand. iv_node is the 
  "the key of the entity, and is_data is the data we want t
  io_modify->update(
    iv_node = zif_ipdb_office_tp_c=>sc_node-zipdb_employee_
    iv_key = ls_item->key
    is_data = ls_item ).

  io_modify->create(
  iv_source_node_key = zif_ipdb_office_tp_c=>sc_node-zipdb_
  iv_source_key = ls_item->parent_key
  iv_assoc_key = zif_ipdb_office_tp_c=>sc_association-zipdb
  iv_node = zif_ipdb_office_tp_c=>sc_node-zipdb_employee_tp
  is_data = ls_item
  ).

ENDLOOP.

io_modify->delete(
  iv_node = zif_ipdb_office_tp_c=>sc_node-zipdb_employee_tp
  it_key = it_key
  ).
  ENDMETHOD.

ENDCLASS.
```

CDS View Modifications

After the method is done, you have to add it to the right consumption view where this method will be used. In my case, that would be the consumption view of the **Employee** entity, called **ZCPDB_EMPLOYEE_TP**.

I will thus add the following lines to the consumption view **ZCPDB_EMPLOYEE_TP** (look for the ones that have the comment **//this line below must be added** above them. The rest of the lines are provided for context):

```
...
@Search.searchable: true
@VDM.viewType: #CONSUMPTION
define view ZCPDB_EMPLOYEE_TP as select from ZIPDB_EMPLOYEE_TP
  association [1..1] to ZCPDB_OFFICE_TP as _Office on $projecti
{

  @UI: {
       facet: [
       {
       label: 'Employee',
         id: 'employeeId',
         position: 10,
         type: #COLLECTION
       },
       {
         parentId: 'employeeId',
         type: #FIELDGROUP_REFERENCE,
         targetQualifier: 'employeeIdFG'
       }],
//this line below must be added.
       identification: [{ position: 10, importance: #HIGH, type

  }

  @UI.lineItem.position: 10
//this line below must be added.
```

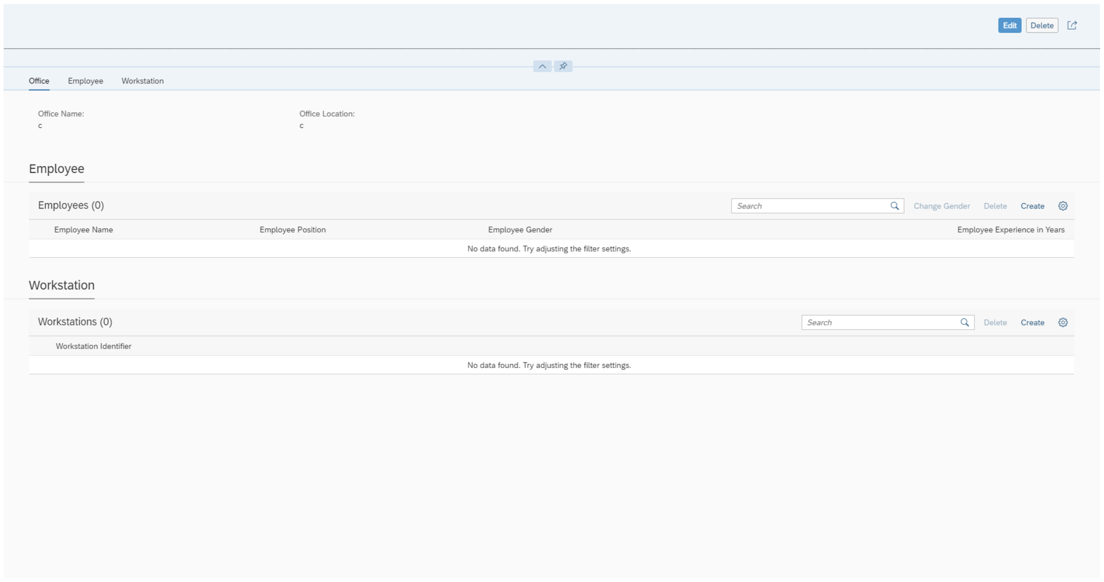
```
  @UI.lineItem: [ {type: #FOR_ACTION, dataAction: 'BOPF:CHANGEG
  @UI.hidden: true
  key ZIPDB_EMPLOYEE_TP.employeeuuid,
...
```

Notice that the two lines to be added have the parameter **type**, which describes the type of event (we created an action, so **#FOR_ACTION**), the parameter **dataAction**, which selects the action we want with this entity (it must start with **BOPF:** followed by the name of the action, which is **CHANGEGENDER** – Refer to **Figure 1.6**) and then the parameter **label**, which is the text that will be inside the button doing the action.

Conclusion

After all those steps, you will end up with this view of your Office entity. This is an Office instance I created with name 'c' and location 'c' with no Employee or Workstation instance.

Edit: You might have noticed there is no mention of OData. This is because there is nothing you need to to with OData: your OData project will create automatically a function import for the **Change Gender** action.



Office Name: c
Office Location: c

Employee

Employees (0)

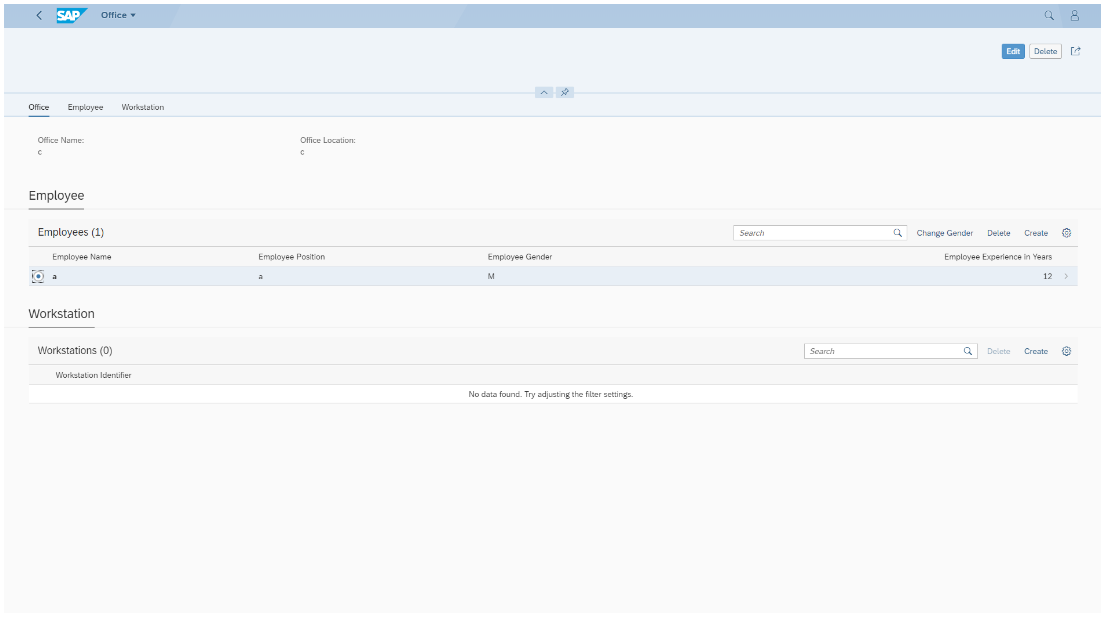
Employee Name	Employee Position	Employee Gender	Employee Experience in Years
No data found. Try adjusting the filter settings.			

Workstation

Workstations (0)

Workstation Identifier
No data found. Try adjusting the filter settings.

Once you add an Employee instance, and you select it by clicking on the circle that will be to the left of the entity, the button "Change Gender" that we created will enable itself.



Office Name: c

Office Location: c

Employee

Employees (1)

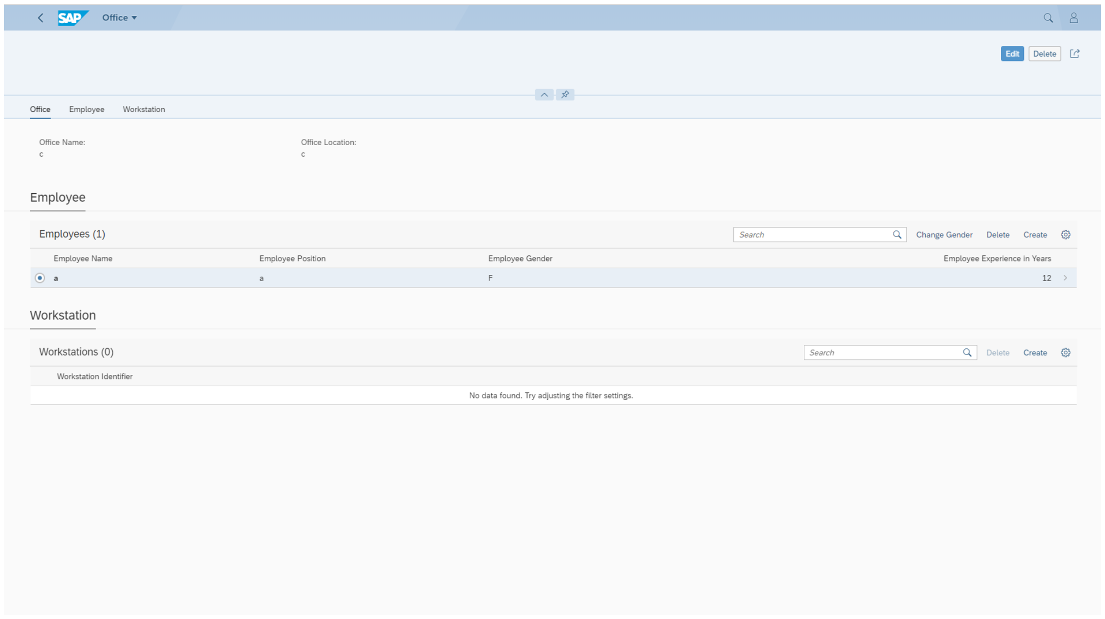
Employee Name	Employee Position	Employee Gender	Employee Experience in Years
a	a	M	12

Workstation

Workstations (0)

Workstation Identifier
No data found. Try adjusting the filter settings.

Then, if all goes well, you should see that the gender of the employee changed after you clicked on the button ‘Change Gender’.



Office Name: c

Office Location: c

Employee

Employees (1)

Employee Name	Employee Position	Employee Gender	Employee Experience in Years
a	a	F	12

Workstation

Workstations (0)

Workstation Identifier
No data found. Try adjusting the filter settings.

In conclusion, this tutorial covered how to add all the **Create/Delete** buttons of the different entities as well as adding custom actions with **BOPF**.

Hope this tutorial was helpful to you. Happy coding!

-Alexandre Therrien

Alert Moderator

Assigned tags

ABAP Development

ABAP CDS view

fiori elements

Similar Blog Posts

Creating a draft enabled Sales Order Fiori App using the new ABAP Programming Model - Part 1: Overview

By Geert-Jan Klaps Mar 11, 2019

Getting Started with ABAP Core Data Services (CDS)

By Carine Tchoutouo Djomo Feb 01, 2016

Getting Started with the ABAP Programming Model for SAP Fiori

By Carine Tchoutouo Djomo Apr 04, 2016

Related Questions

ABAP CDS View: OData service CRUD enabled?

By Bert Deterd Jan 22, 2016

Object Model Annotations in CDS Views without corresponding BOPF Objects

By Prateek Sonthalia Jan 11, 2018

ABAP programming model for Fiori with legacy apis

By Mattias Johansson Dec 21, 2018

Join the Conversation

SAP TechEd

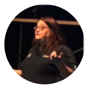
Tune in for tech talk. Stay for inspiration. Upskill your future.

Coffee Corner

Join the new Coffee Corner Discussion Group.

6 Comments

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



Michelle Crapo

August 30, 2019 at 11:33 am

Very nice. Step by step is always appreciated. The screen shots are a nice touch too. I hate to even guess how long this took you to put together.

Excellent!

Michelle

Like 2 | Share

Alexandre Therrien | Blog Post Author

August 30, 2019 at 1:16 pm

Thank you! I really appreciate it!

Alex

Like 2 | Share

Mahesh Kumar Palavalli

August 30, 2019 at 4:13 pm

Very informative series Alexandre Therrien

Hope to see more ABAP Programming model concepts being covered in the future blogs 🙂

Thanks,

Mahesh

Like 2 | Share

Nabheet Madan
September 4, 2019 at 1:01 am

Thanks for the nice blog Alex. One small correction

"Create Remove Update Delete (CRUD) Actions to CDS Views" should be "Create Read Update Delete (CRUD) Actions to CDS Views"

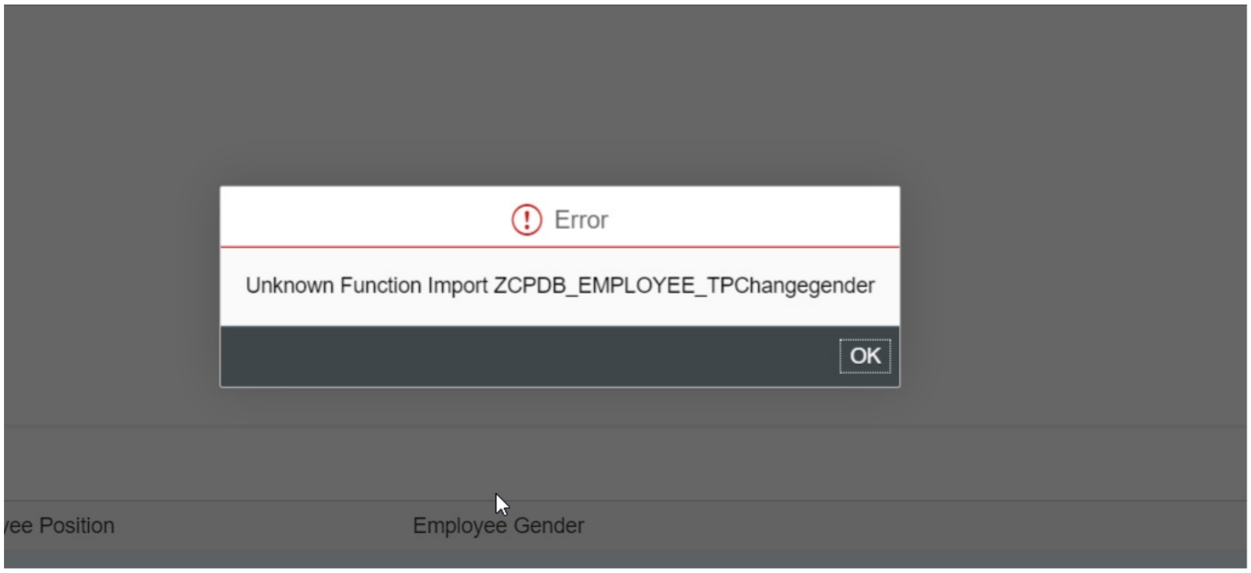
Like 1 | Share

vikas gupta
January 6, 2020 at 2:48 pm

Thanks Alex for this blog.

I followed your blog and succesfully created the FIORI App but i got the error while creating the action button.

Mahesh Kumar Palavalli Please look into this if you can help too.


Error

Unknown Function Import ZCPDB_EMPLOYEE_TPChangegender

Like 0 | Share

Giuseppina Montenegro
April 20, 2020 at 3:20 pm

Thank’s a lot for the walkthrough, it is very detailed!

I followed the step-by-step tutorial but my ‘Change Gender’ action in my Fiori app is disabled.
If I test the action from the trx BOBX it works!

I would be very grateful if you could help me.

Like 0 | Share