SAP Business Add In (BADI)

Introduction

- BAdI – Business Add-ins
- New SAP enhancements using SAP Objects.
- Users of BAdI can customize the logic according to the specific requirements (User-defined) or use the standard logic available.
- Each Business Add-In has
  - At least one BAdI definition
  - A BAdI interface
  - A BAdI class that implements the interface
- For User-defined BAdI,
  - developer creates an interface for the add-in.
  - Enhancement management creates an adapter class that implements the interface
  - Developer creates an instance of the class in the application program and calls the corresponding methods.
- For standard BAdI, interface and class will be predefined by SAP.
- Adapter class performs these tasks

- Control (the class calls all active implementations)
- Filtering (If the Add-in has to be executed under certain conditions, the class ensures that only certain implementations are executed)
- In BAdI, all the enhancement components are grouped together.
- Program Enhancements (interface methods)
- Menu Enhancements (function codes in interface definition)
- Screen Enhancements .

Advantages of BAdI's

- Business Add-Ins no longer assume a two-level infrastructure (SAP and customer solutions), but instead allow for a multi-level system landscape (SAP, country-specific versions, industry solutions, partner, customer, and so on). You can create definitions and implementations of Business Add-Ins at any level of the system landscape.
- SAP guarantees the upward compatibility of all Business Add-In interfaces. Release upgrades do not affect enhancement calls from within the standard software nor do they affect the validity of call interfaces. You do not have to register Business Add-Ins in SSCR.

- The Business Add-In enhancement technique differentiates between enhancements that can only be implemented once and enhancements that can be used actively by any number of customers at the same time
    - Business Add-Ins can be defined according to filter values. This allows you to differentiate between Add-In implementations using the filter criteria as per customer's requirements.

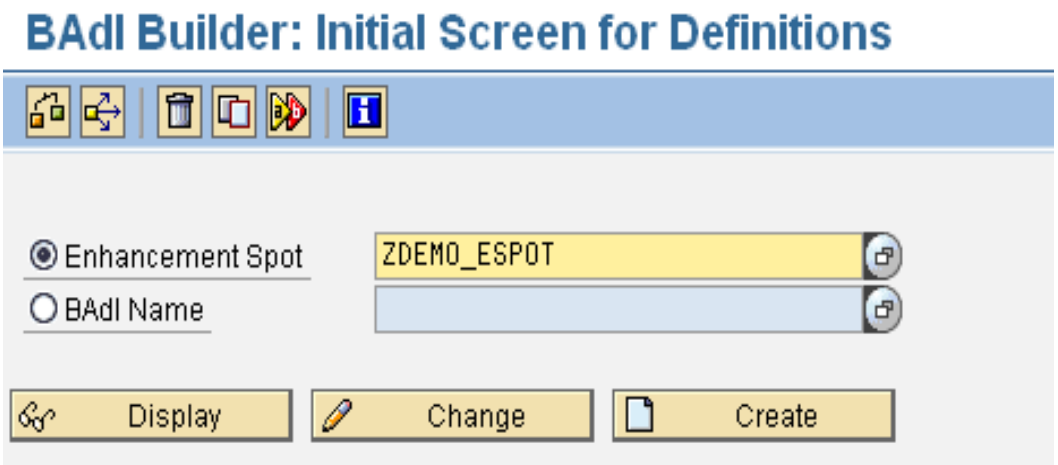
2. Customized BAdI

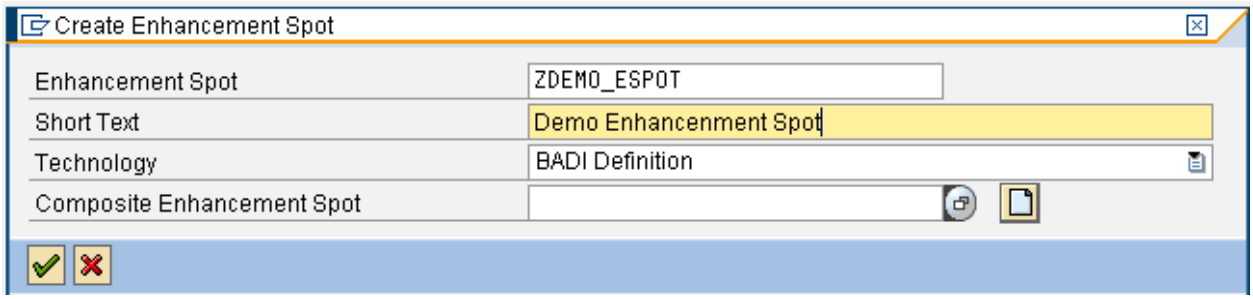
Defining BAdI

- From SAP Easy Access goto Tools -> ABAP Workbench -> Utilities -> Business Ad-Ins -> Definition

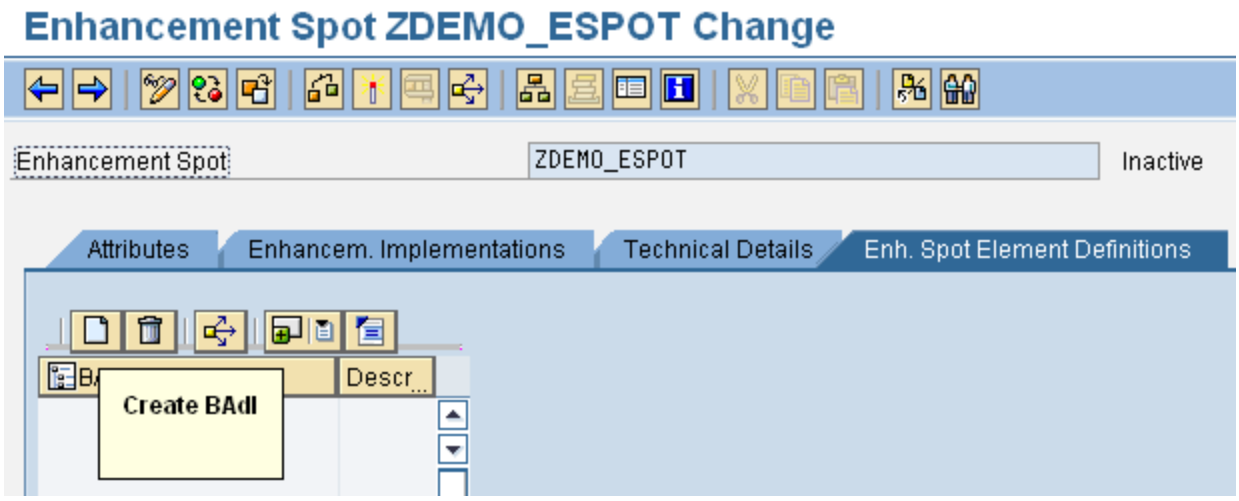
OR

- Go to Tcode **SE18**
- BADI definitions were directly defined in SE18, but from ECC 6.0, SAP has introduced the concept of Enhancement Spots
- An enhancement spot is an object, which can contain one or more BAdI definitions
- In SE18, select the option Enhancement Spot
- Give the Enhancement spot name and click on Create



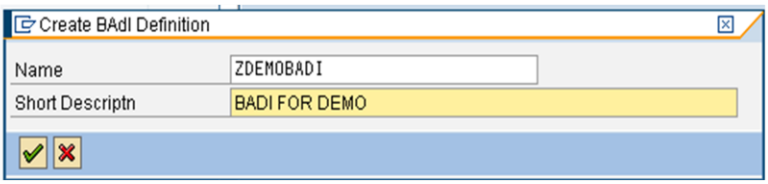
- Provide Short text and Select Technology as BAdI Definition
- 

- Save Enhancement Spot
- Select "Enhancement Spot Definitions" Tab
- On the Left Hand Side Click on the Icon "Create BAdI"

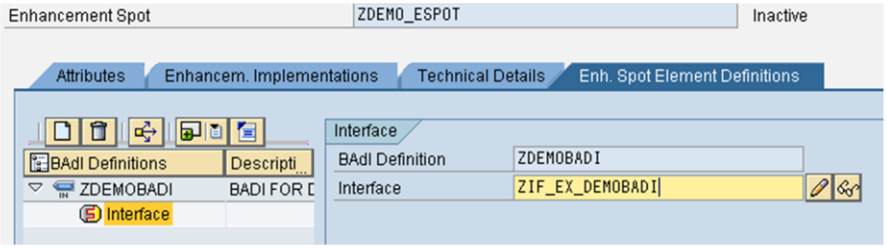


Defining BAdI (Contd.).

- Give the BAdI name and Short Description



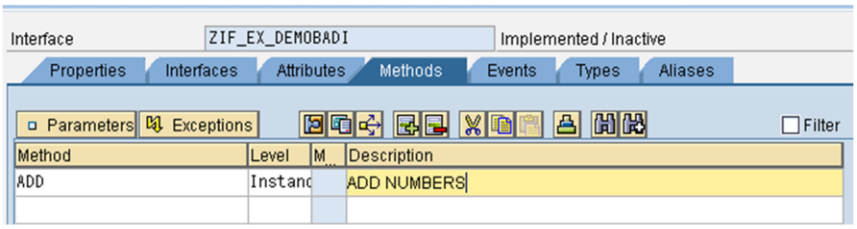
- Expand the BAdI definition in order to give the BAdI Interface name



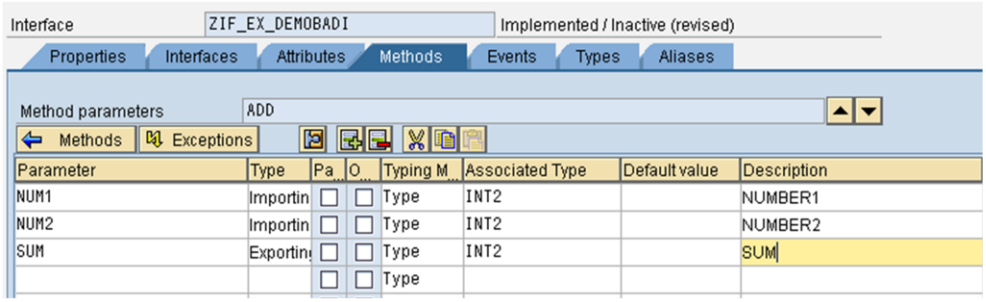
- Give the Interface name and create the Interface

Defining BAdI (Contd.).

- Define the methods in the interface

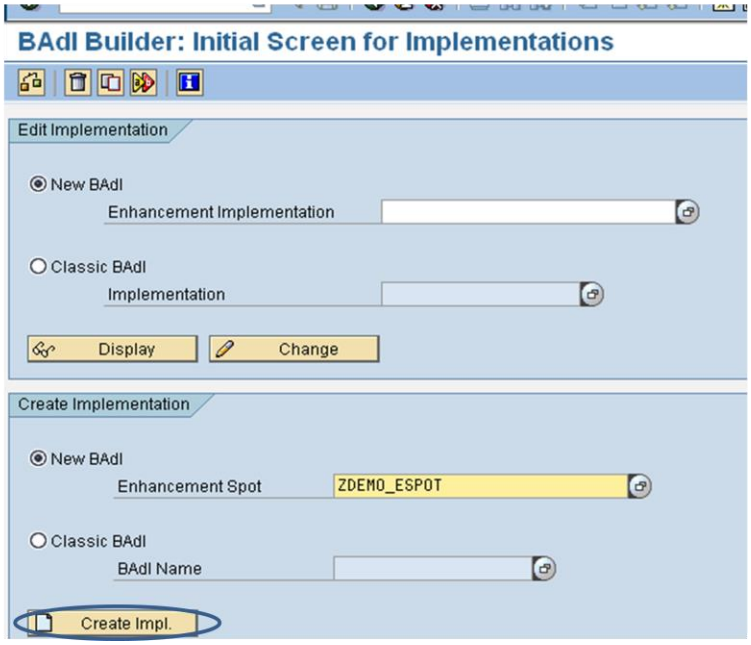


- Define Parameters for your method



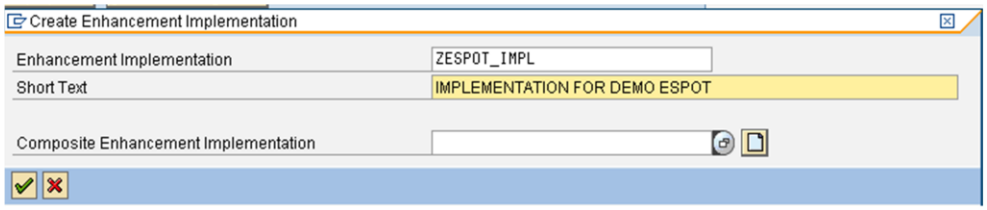
Implementing BAdI

- Go to Tcode **SE19**
- Give the Enhancement Spot name and Click on Create Implementation

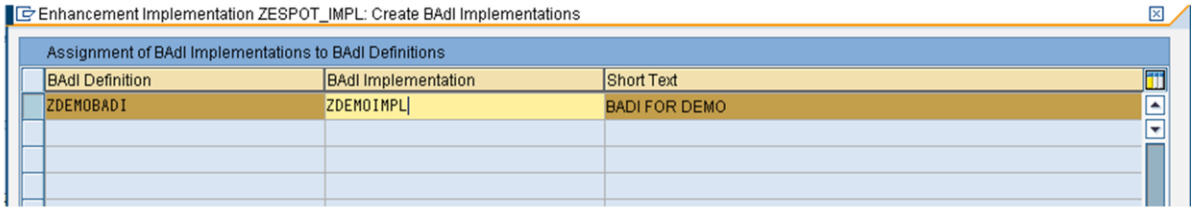


Implementing BAdI (Contd.).

- Give Enhancement Spot Implementation name, short text and continue

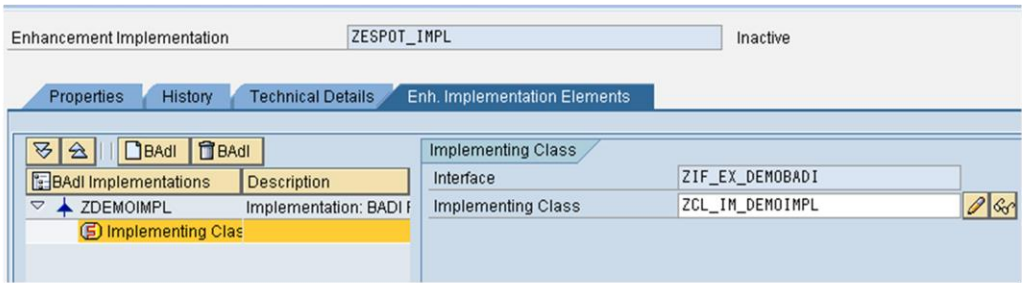


- Give the BADI Implementation Name and continue

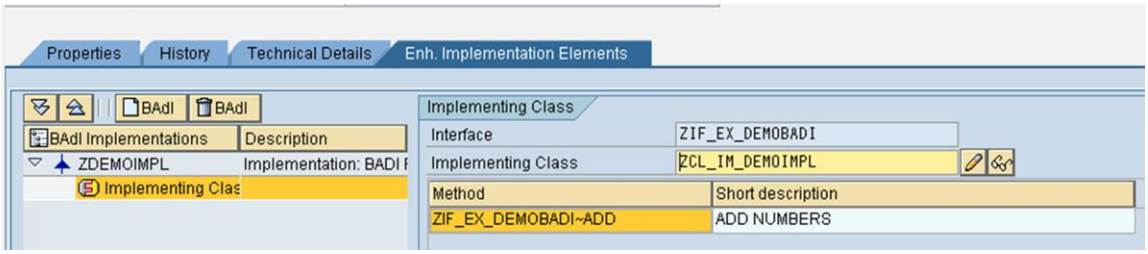


Implementing BAdI (Contd.).

- Double Click on Implementing Class and give the implementing class name and create it.

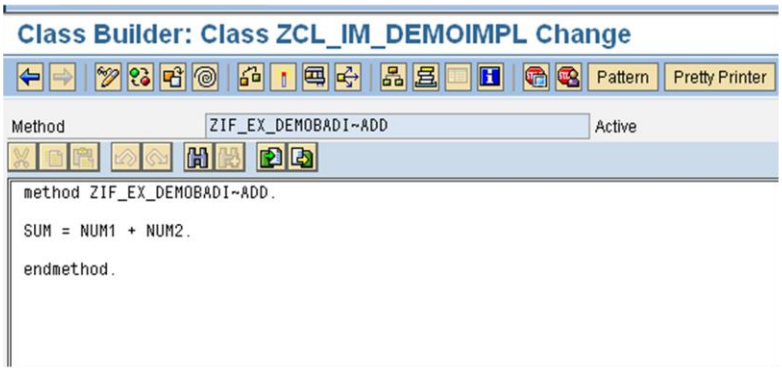


- Once you create the implementing class, the screen looks like this:



Implementing BAdI (Contd.).

- Double click on the method to implement the method of the BAdI



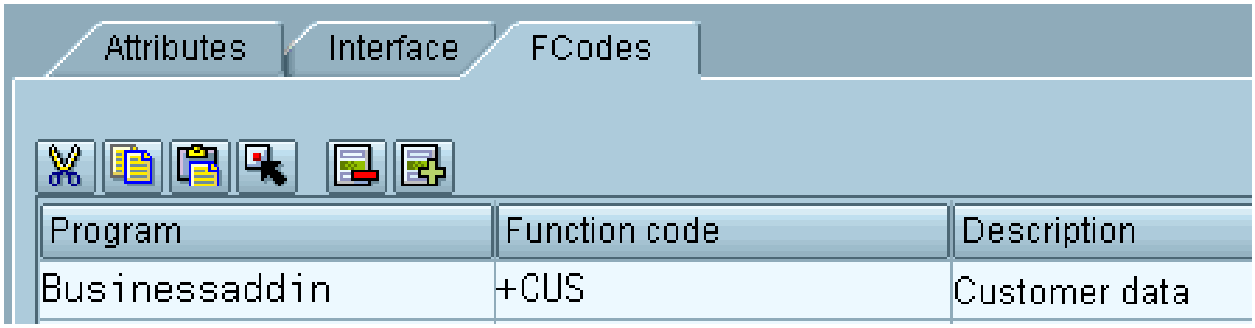
Calling BAdI in programs

- After the definition and independent of the implementation, BAdIs can be called using the ABAP statements **GET BADI** and **CALL BADI**.
- GET BADI
  - Used to generate a new BAdI object
  - GET BADI badiobj FILTER = fltrs
- CALL BADI
  - Used to call the relevant BAdI method

– **CALL BADI badi->meth [EXPORTING p1 = a1 p2 = a2 ...]**
{ {[IMPORTING p1 = a1 p2 = a2 ...]
[CHANGING p1 = a1 p2 = a2 ...]}

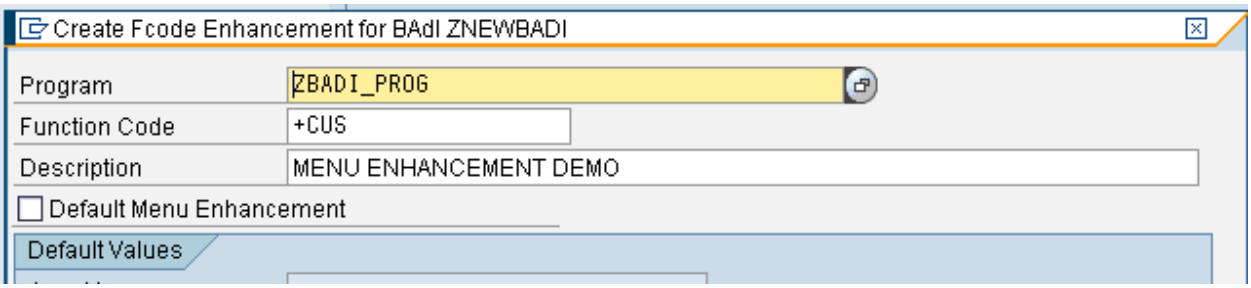
Menu Enhancements

- Menu Enhancements are renamed as Function Code Enhancements
- The Function Code should start with a '**+**'
- Function Code Enhancements can only be used for single-use add-ins
- Should not be filter-dependent
- Have to be created in conjunction with Program Enhancements (Interfaces)



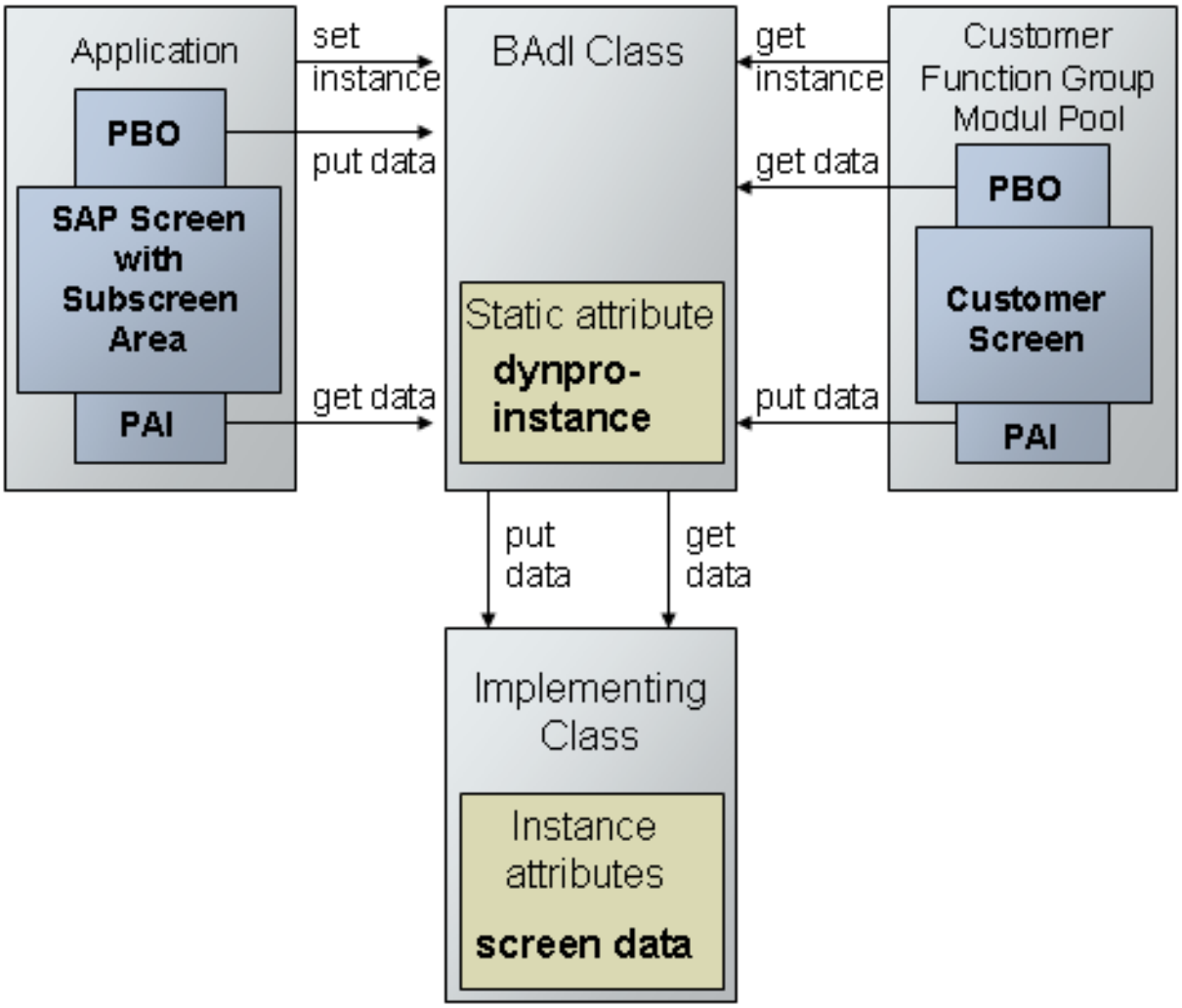
- Choose FCodes Tab

Function Code Enhancements

- Right-click on the BADI definition to Create a Menu Enhancement
- Select 'Add Menu Enhancement'
- Provide the Program Name, Function Code and the description
- 

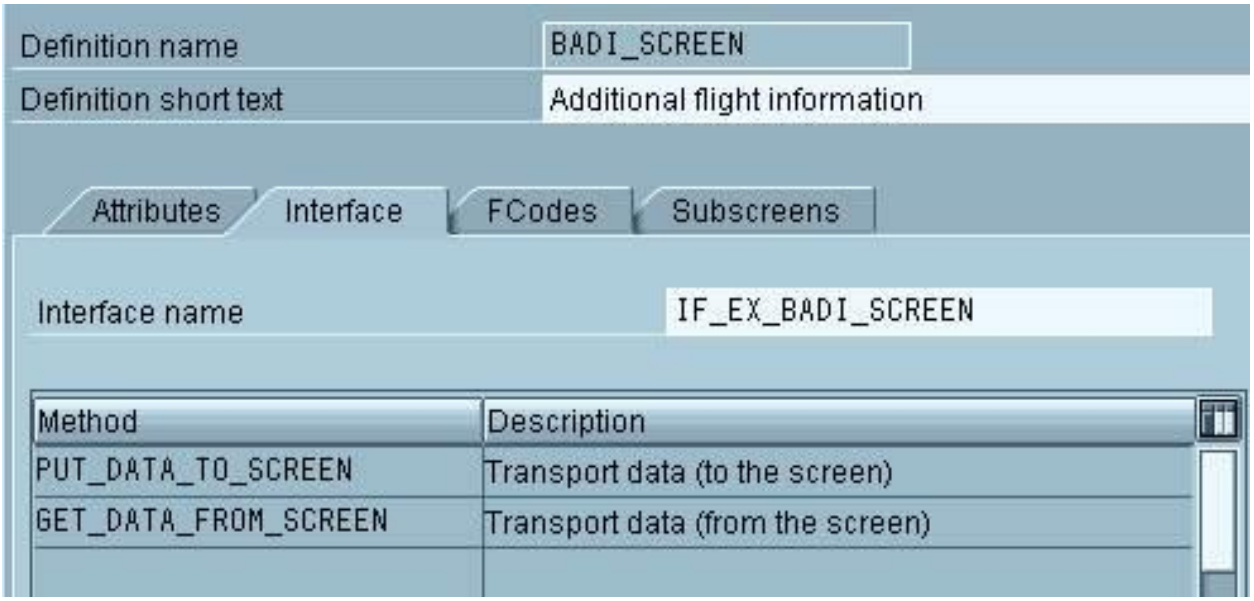
- Create the Implementation for the Function Code Enhancement in SE19
- Provide the Function text, Icon text, etc.
- Develop the processing logic in the interface exit of the BADI

Screen Enhancements



Defining Screen Enhancements

- Create the following methods.

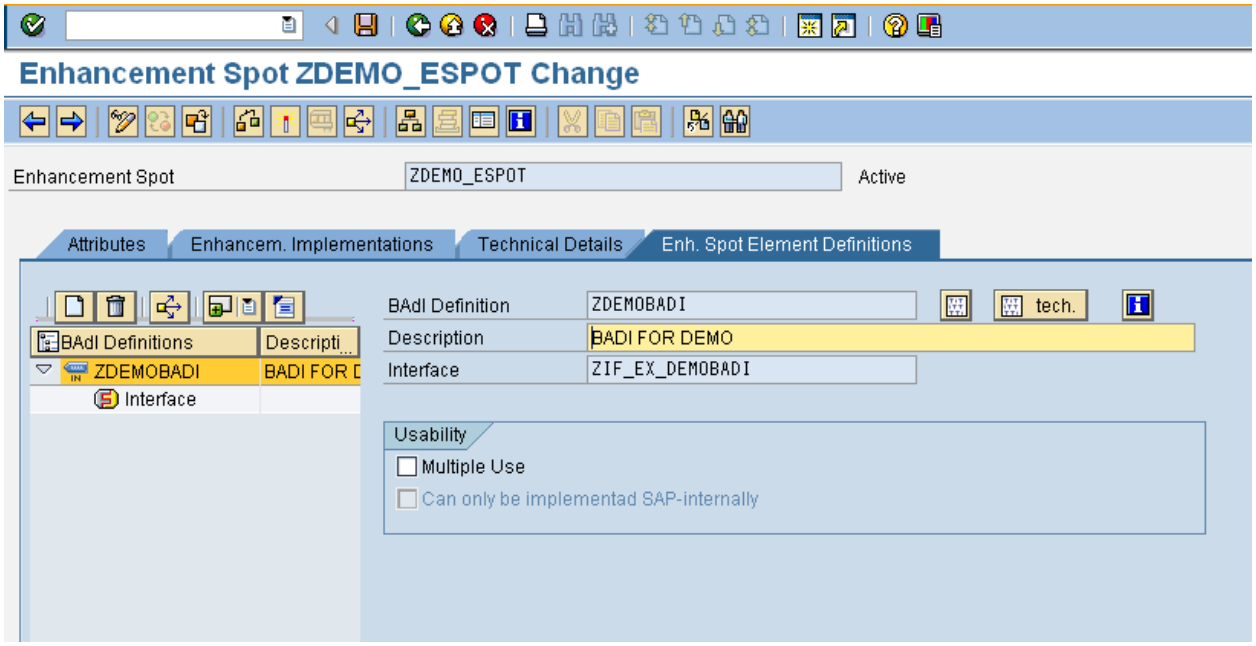


3. Multiple Use BAdI's

Multiple Use BAdI's

- Multiple implementations are possible for the same BAdI
- There is no sequence control for multiple implementation since at the time of definition, it doesn't know which implementation will be active.
- All active implementations will be triggered by the application program.
- To display the list of all implementations of a BAdI definition, go to Implementation -> Display in **SE18**.

Multiple Use BAdI's (Contd.).



- When defining a Multiple use BAdI, the interface methods of the BAdI should not contain **export or returning parameters**

4. Filter dependent BAdI

Filters

- BAdI's are implemented based on some filter values
- Filter type must be specified while defining the enhancement.
- It can be a single filter value or a set of values.
- All methods in the interface will have the filter value FLT_VAL as their importing parameter.
- The method then selects the active implementation based on the data provided in the filter value.

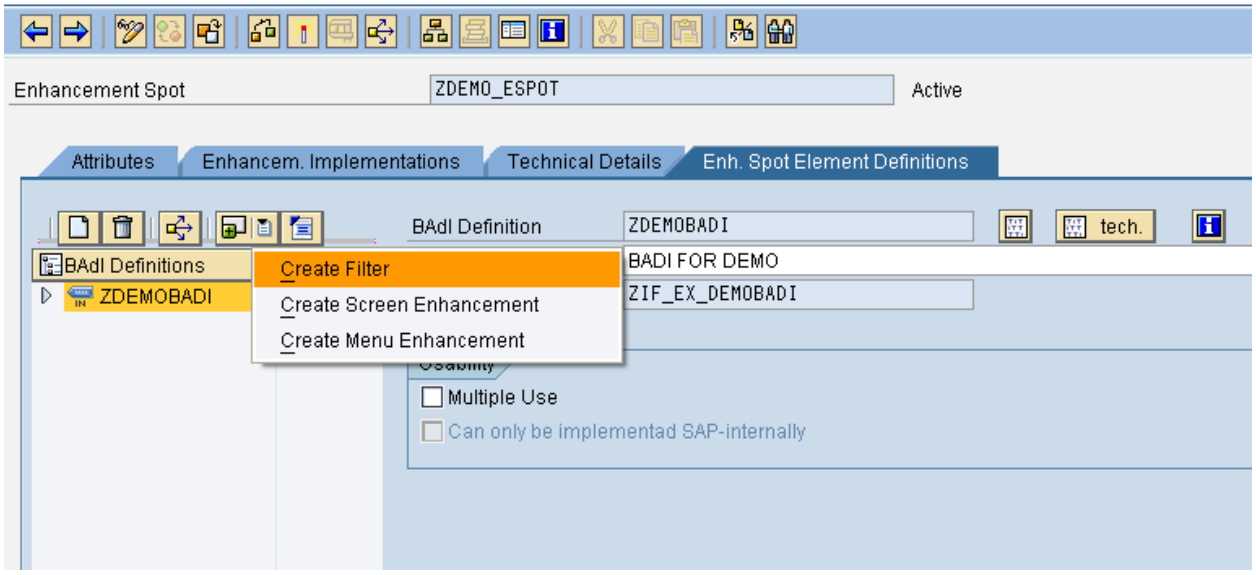
- A filter type can be a data element or a structure. A data element must fulfill the following criteria:
- The data element's domain may contain a maximum of 30 characters and must be of type *Character*.
- The data element must
  - Either have a search help with a search help parameter of the same type as the data element and this parameter must serve as both the Import and export parameter

OR

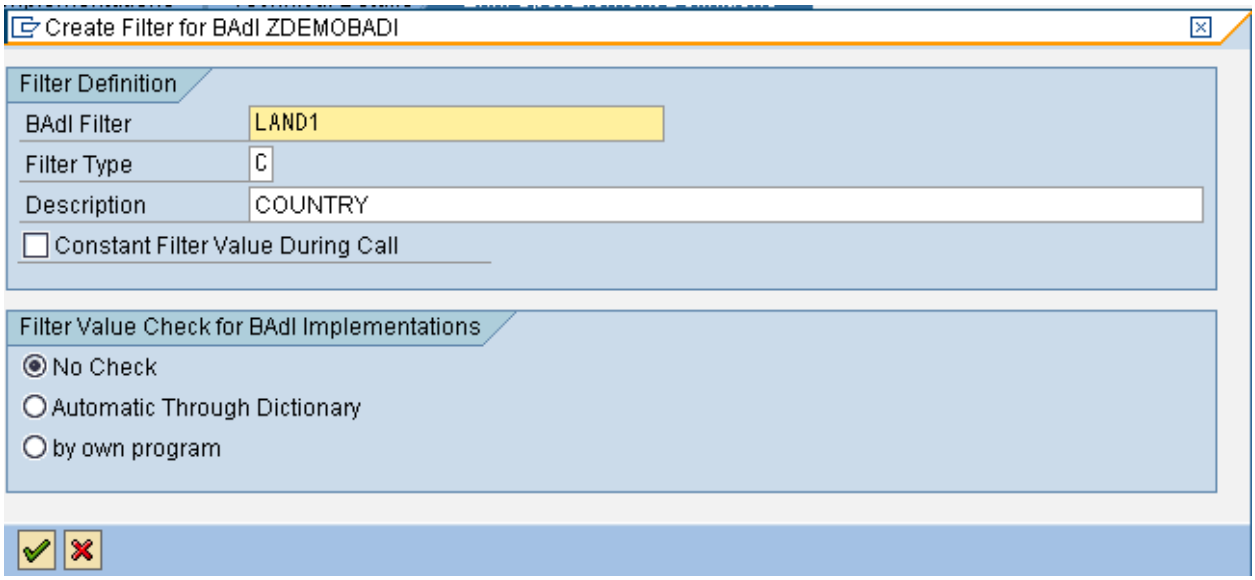
  - Element's domain must have fixed domain values or a value table containing a column with the same type as the data element.
- Now create an interface with a method. Be aware that for each method you create in the interface of a filter-dependent enhancement, the appropriate filter value must be defined as the import parameter so that the application program can provide the filter value to the enhancement method. The method then selects the active implementation for that value.

Creating a Filter

- Select the option Create Filter



- Specify the required parameters – BAdI Filter Field, Filter Type and Description



- When creating the BAdI implementation, the filter values for which the implementation should be processed should be defined

5. Finding Standard BAdI's

Finding Standard BAdI

- There are multiple ways of searching for BAdI:
  - Finding BAdI Using CL_EXITHANDLER=>GET_INSTANCE
  - Finding BAdI Using SQL Trace (TCODE-ST05).
  - Finding BAdI Using Repository Information System (TCODE- SE84).
  - Finding BADI Using CL_EXITHANDLER=>GET_INSTANCE
  - Go to the Transaction, for which we want to find the BADI, in our case we will take the example of Transaction VD02.
  - Get the Program Name of Corresponding Transaction.

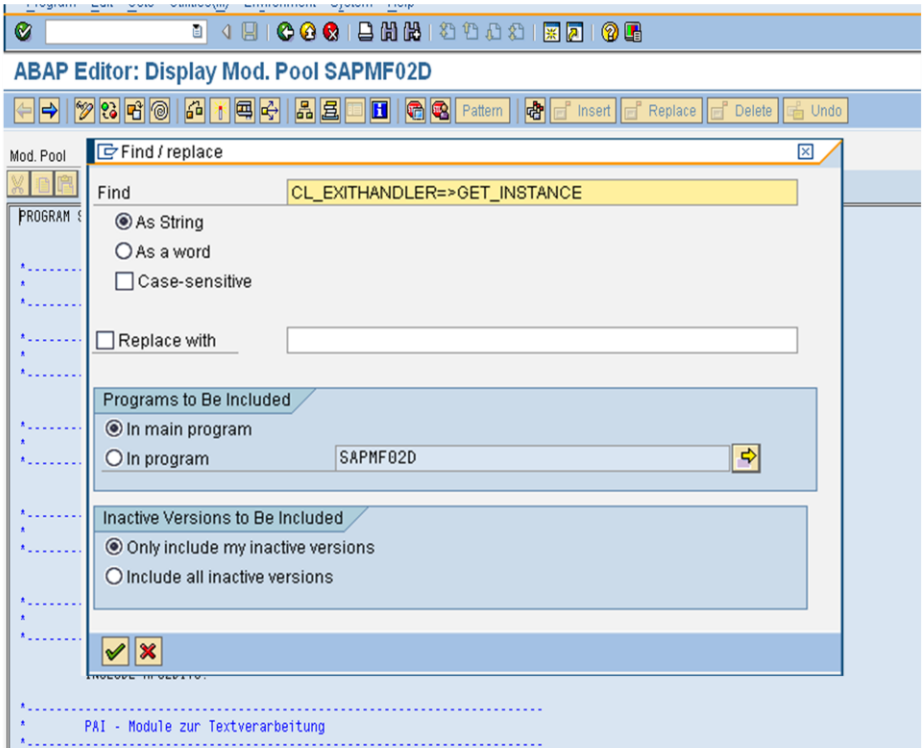
(Click on System->Status. Double Click on Program Name).

  - Once inside the program search for 'CL_EXITHANDLER=>GET_INSTANCE'.

Make sure the radio button "In main program" is checked

Finding Standard BAdI (Contd.).

- **Finding BAdI Using CL_EXITHANDLER=>GET_INSTANCE**



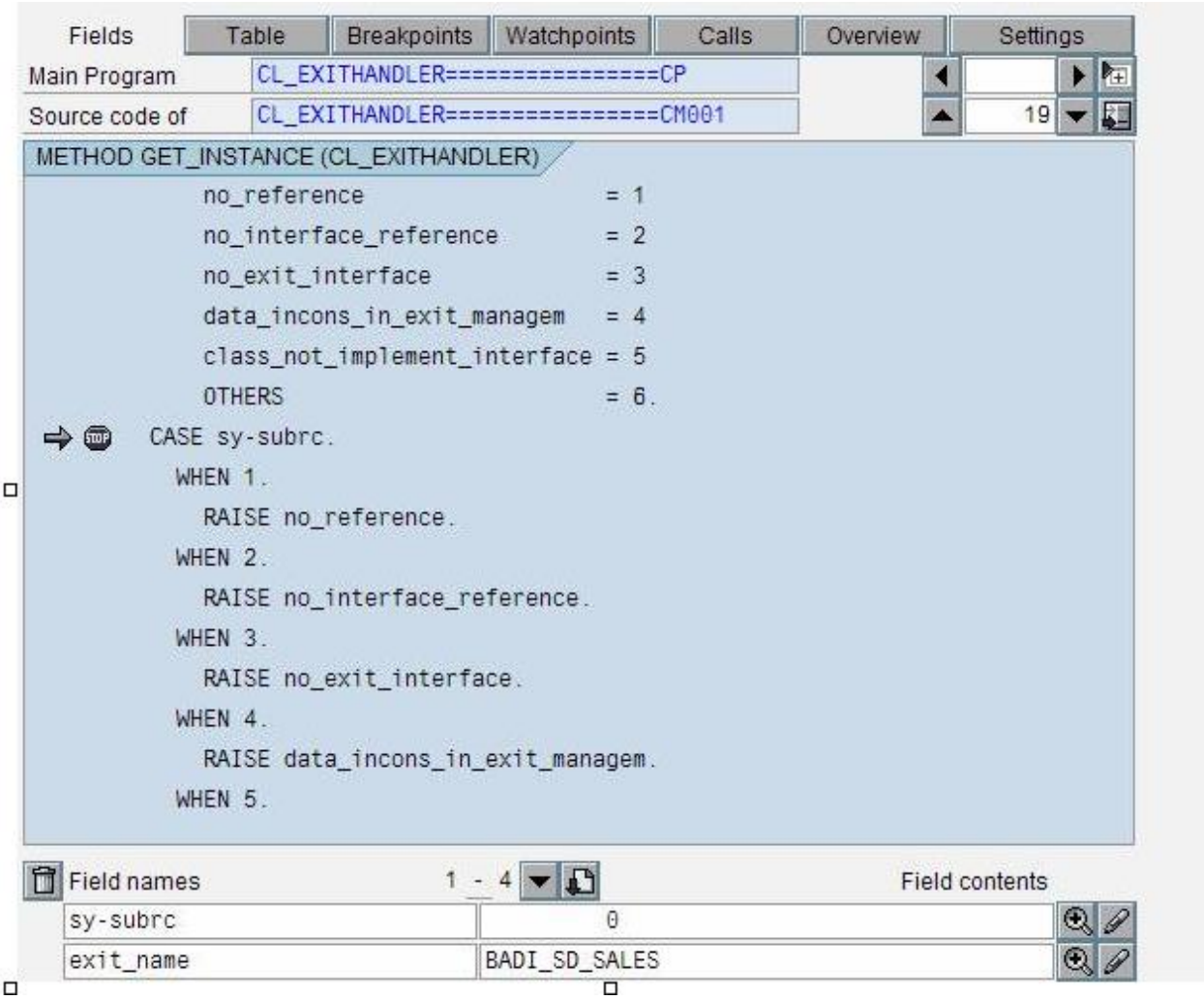
Finding Standard BAdI (Contd.).

- A list of all the programs with call to the BAdI's will be listed.
- The export parameter 'EXIT_NAME' for the method GET_INSTANCE of class CL_EXITHANDLER will have the user exit assigned to it.
- The changing parameter 'INSTANCE' will have the interface assigned to it.

Program	Found locs/short description
MF02D_ADD_ON_FG0	17 CALL METHOD CL_EXITHANDLER=>GET_INSTANCE exporting " \TP 563352 exit_name = 'CUSTOMER_ADD_DATA' " \TP 563352 null_instance_accepted = 'X' " \TP 563352 CHANGING INSTANCE = G_ADDITIONAL_DATA. 76 CALL METHOD CL_EXITHANDLER=>GET_INSTANCE exporting " \TP 563352 exit_name = 'CUSTOMER_ADD_DATA_CS' " \TP 563352 null_instance_accepted = 'X' " \TP 563352 CHANGING INSTANCE = G_ADDITIONAL_DATA_CS. 97 CALL METHOD CL_EXITHANDLER=>GET_INSTANCE exporting " \TP 563352 exit_name = 'CUSTOMER_ADDRSCR_CHG' " \TP 563352 null_instance_accepted = 'X' " \TP 563352 CHANGING INSTANCE = G_ADDRESS_SCREEN_CHG.

- Once you have identified the BAdI definition, create an implementation for the BAdI definition.
- Double click on the method to develop the implementation source code
  - <u>A simple way to FIND ALL BAdI exits used in a particular transaction</u>
  - GO to the transaction SE24 – (Transaction for classes) .
  - Enter the class name **CL_EXITHANDLER**.
  - Double-click the method **GET_INSTANCE**.

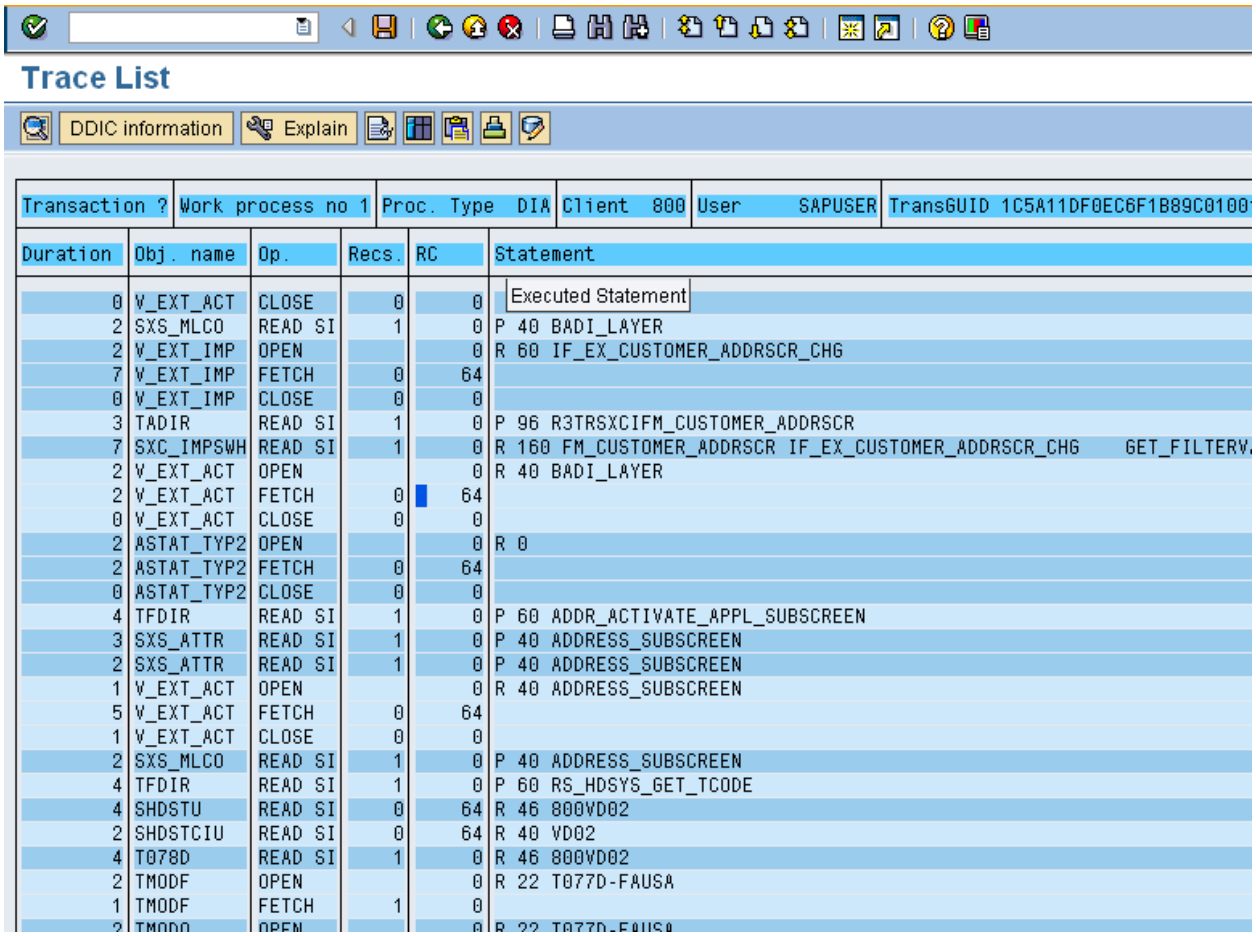
- Set a break-point at command line CASE sy-subrc (line 25).
- Execute the transaction that you want to analyse. It will stop at the break-point you have just set on class **CL_EXITHANDLER** whenever it finds any BADI method/Exit call.
- On the debug screen, type the field name *'E*xit_name'. This field contains the BAdI method/Exit name which is being called in the program at that time.
- Press **F8** to see the next calls.
    - Eg: When I select transaction VA01 , field 'Exit_name' has the BADI name 'BADI_SD_SALES'.



ST05

- ***Finding BAdI Using SQL Trace (TCODE:-ST05)***
- Here, we will find BAdIs for “Change Customer (Sales)” (Transaction Code: VD02).
- Start transaction ST05 (Performance Analysis).
- Set flag field "Buffer trace"

- Remark: We need to trace also the buffer calls, because BAdI database tables are buffered. (Especially view V_EXT_IMP and V_EXT_ACT)
- Push the button "Activate Trace".



Duration	Obj. name	Op.	Recs.	RC	Statement
0	V_EXT_ACT	CLOSE	0	0	
2	SXS_MLCO	READ SI	1	0	P 40 BADI_LAYER
2	V_EXT_IMP	OPEN		0	R 60 IF_EX_CUSTOMER_ADDRSCR_CHG
7	V_EXT_IMP	FETCH	0	64	
0	V_EXT_IMP	CLOSE	0	0	
3	TADIR	READ SI	1	0	P 96 R3TRSXCIFM_CUSTOMER_ADDRSCR
7	SXC_IMPSWH	READ SI	1	0	R 160 FM_CUSTOMER_ADDRSCR IF_EX_CUSTOMER_ADDRSCR_CHG GET_FILTERV
2	V_EXT_ACT	OPEN		0	R 40 BADI_LAYER
2	V_EXT_ACT	FETCH	0	64	
0	V_EXT_ACT	CLOSE	0	0	
2	ASTAT_TYP2	OPEN		0	R 0
2	ASTAT_TYP2	FETCH	0	64	
0	ASTAT_TYP2	CLOSE	0	0	
4	TFDIR	READ SI	1	0	P 60 ADDR_ACTIVATE_APPL_SUBSCREEN
3	SXS_ATTR	READ SI	1	0	P 40 ADDRESS_SUBSCREEN
2	SXS_ATTR	READ SI	1	0	P 40 ADDRESS_SUBSCREEN
1	V_EXT_ACT	OPEN		0	R 40 ADDRESS_SUBSCREEN
5	V_EXT_ACT	FETCH	0	64	
1	V_EXT_ACT	CLOSE	0	0	
2	SXS_MLCO	READ SI	1	0	P 40 ADDRESS_SUBSCREEN
4	TFDIR	READ SI	1	0	P 60 RS_HDSYS_GET_TCODE
4	SHDSTU	READ SI	0	64	R 46 800VD02
2	SHDSTCIU	READ SI	0	64	R 40 VD02
4	T078D	READ SI	1	0	R 46 800VD02
2	TMODF	OPEN		0	R 22 T077D-FAUSA
1	TMODF	FETCH	1	0	
2	TMODO	OPEN		0	R 22 T077D-FAUSA

Using Repository Information System

- ***Finding BAdI Using Repository Information System (TCODE- SE84)***
- Here, we shall find BAdI's for Transaction VD02

- Go to “Maintain Transaction” (TCODE- SE93).
- Enter the Transaction VD02 for which you want to find BAdI.
- Click on the Display push buttons.
- Get the Package Name. (Package VS in this case)
- Go to TCode: SE84->Enhancements->Business Add-inns->Definition
- Enter the Package Name and Execute.