Document on BADI

After completing this document you will be able to,

- Understand what Business Add In's (BADIs) are
- Define BADIs
- Implement BADIs
- Use BADIs
- Create an implementation for an existing SAP BADI
- Create Multiple Use BADIs
- Create Filter Dependent BADI

Introduction

- Traditional way of doing code modifications - Exits
- 3 Tier Architecture For Presentation Server
- Field Exits
- Screen Exits
- Menu Exits For Application Server
- Programs -> Program exits -> call customer function -> Include At Database Level
- Table -> SE11 -> Go to -> Append Structure
- With BADIs you can make changes only at the Application Layer.
- The traditional way of doing code modifications was via User exits.

As per SAP’s 3 Tier Architecture, at the Presentation layer you had Field Exits, Screen Exits and Menu Exits. At the Application layer you had Program exits; at the Database layer you could append structures to the existing SAP structures.

BADI’s

- Business Add-Ins is a new SAP enhancement technique based on [ABAP Objects]
- They can be inserted into the SAP system based on specific user requirement
- Business Add-Ins should be considered generalized Business Transaction
- Events that can be used to bundle program, menu and screen enhancements into a single add-in.

Each Business Add-In has:

- at least one Business Add-In definition
- a Business Add-In interface
- a Business Add-In class that implements the interface
- In order to enhance a program a Business Add-In must first be defined
- Subsequently two classes are automatically generated:
  - An interface with 'IF_EX_' inserted between the first and second characters of the BAdI name
  - The Application developer creates an interface for this Add-InEnhancement management generates an adapter class for implementing the interface
- This opens a path for custom implementations to be created by partners or customers.
- By instantiating the adapter class in the application program, its corresponding methods can be used

BADI Enhancement options

- In the business Add INS, all the components of an options are grouped together
- Program Enhancements: Functional enhancements of the program are defined in the form of interface methods and can be used with or without Menu Enhancements or Screen Enhancements
- Menu Enhancements: Function codes can be entered for a BADI for the corresponding menu entries in the user interface definition.
- Screen Enhancements: Sub screen areas can be entered for a Business Add-In. Screen enhancements can be implemented for these in the form of sub screen screens.

Defining BADIs

Enter BADI name
Click on CREATE SE18
To define a BADI use Sap Menu -> Tools -> ABAP Workbench -> Utilities -> Business Ad-Ins -> Definition

For Application developers to include BADIs in their program they need to define an interface for their enhancement in the 'Business Add-Ins Initial Maintenance Screen'.

**Defining BADIs - Attributes**
- Enter short text
- BADI class is automatically created
- When a BADI is created, BADI class gets automatically created with the suffix 'CL_EX_' after the first character of the BADI name.
- Package: Specifies the Package in which this BADI resides.
- Multiple uses: With this option you can have multiple implementations for the same BADI

  - Filter-Dependent: Business Add-Ins may be implemented depending on a specific filter value (example, country-specific versions: Distinct implementations of the BADI can be created for each country). A filtertype must be entered when defining your enhancement (a country or industry sector, for example). All methods created in the enhancement's interface have filter value 'FLT_VAL' as their import parameter. The method then selects the active implementation based on the data provided in the filter value

**BADI Definition - Save**
- Enter Package name
- Save the BADI to a Package. (Z_DELTA_STUDENT##)

**Defining BADIs**
- Package name populated
- BADI class created
- A BADI class ZCL_EX_DELTA_7_0_DEMO1 is created automatically.

**Defining BADIs - Interface**
- BADI Interface is created automatically
- To edit double click on the interface
- An interface is created automatically with the suffix 'IF_EX_' after the first character of the BADI name
- When you double click the interface 'ZIF_EX_DELTA_7_0_DEMO1 you will be taken into the class builder where you can define the methods, parameters and exceptions for that interface

**Defining BADIs - Interface methods**
- Method name
- Method type
- Specify the methods in the Class Builder for the interface
- Instance methods can access all of the attributes of a class, and can trigger all of the events of the class
- Static methods can only access static attributes and trigger static events
- Click on the Parameters button to specify any Import/Export parameters for the method

**Defining BADIs - Interface methods - parameters**
- Import and Export parameters
- The Import/Export parameters for the method are specified in the 'Parameters' section
- The type specifies whether the parameter is an Importing, Exporting, Changing or Returning parameter.
- Check the Optional field if it is not a mandatory parameter
- The type, default value and description of the parameter can be specified

**Defining BADIs - Exceptions**
- Exceptions
- You can specify any exceptions in the Exceptions section.
- For class based exceptions, check the checkbox 'Exception Classes'.

**BADI Implementation**
- Double click on the method to modify method code
- During implementation creation, a class for implementing the enhancement's interface is also created
- Navigate to the Class Builder by double-clicking the method.

**Implementing BADIs: Method implementation**
- Modify the method code within the METHOD/ENDMETHOD statements and activate
- Insert the source code between the method and end method statements and activated.
- Exception handling for the method can be handled in this section.

**Using a BADI**
- When a BADI is created, the enhancement manager automatically generates classes that implement the interface
- In order to be able to call static methods, you must declare the corresponding class in ABAP Objects using the 'class ... definition load' statement
- In the example having created a BADI for division, you can program the call for the BADI in the following manner:
  - Create an Interface Reference Variable in the program 'L_BADI_INSTANCE'
  - Perform the statement 'call method cl_exithandler ...' using the BADI and the interface reference variable.
  - Call method DIVIDE_NUMBERS using the interface reference variable L_BADI_INSTANCE.

**Create an Implementation for an SAP provided BADI**
- Find the corresponding BADI for the SAP transaction
- Create a custom Implementation for the BADI
- From the methods available in the BADI and select the appropriate method
- Enter custom code in the method and activate
- Execute SAP transaction and test if method with custom code is being invoked

**Finding a BADI - Transaction VD02**
Double click on the program name

**Three BADIs used in this program**
- Double click required
- BADI to see the source code
- Double click on the method to enter the source code.
- Interface reference
- Variable for the BADI
- Finding a BADI - Interface reference to the BADI
- Interface reference
- Variable defined
  - If the BADI name is not specified in the CL_EXITHANDLER=>GET_INSTANCE method call, you can find out the BADI name by removing the prefix IF_EX_ from the interface name.
- Using an SAP BADI - Create a Custom Implementation
  - Enter the name of the BADI
  - SE19
  - Enter the implementation name
  - Click on CREATE
- To create a custom implementation for an SAP defined BADI do the following:
- Go to transaction SE19
- Enter a name for the BADI
- Hit the 'Create' button
- The BADI definition selection popup will be displayed
- Enter the name of the standard SAP BADI that you want to implement

**BADI Implementation - Attributes**
- Enter description
- Enter the description for the custom implementation
- Click on the Activate button to activate the implementation

**BADI Implementation - Interface methods**
- Click on the interface tab to view the available methods
- Double click on method
- SAVE_DATA to modify it
- In the interfaces tab you will find a list of methods included in the BAdI Interface
- Double clicking on any of the methods will take you into the Class Builder where you can enter custom code for the corresponding method

**BADI Implementation - Class builder**
- Enter custom code
- In our example we have chosen the SAVE_DATA method to enter our code changes
- Double clicking on the SAVE_DATA method takes us into the class builder
- You can enter your code changes between the methods, end method statements
- Enter the custom code that will be executed when customer data is saved in transaction VD02.

**BADIs - Execute the Change customer transaction**
- Click on SAVE button
- VD02
- Go into the change customer transaction VD02 and change the name of the customer
- Click on the SAVE button.
- This should invoke the BADI - CUSTOMER_ADD_DATA and subsequently execute the custom implementation.

**BADI Implementation - Invoked by VD02**
- Custom code for method invoked via BADI CUSTOMER_ADD_DATA
- A break point is hard-coded into the method SAVE_DATA.
- Execute transaction VD02 and change the customer address.
- When you hit the SAVE button, the code entered in the SAVE_DATA method is executed.

**Multiple Use BADIs**
- Single Use Business Add-Ins

Do not select the multiple use checkbox from the Administration tab when defining a Single Use BADI

BADI has a single Implementation

Similar to procedures

Return values can be expected
Can use importing, changing and returning parameters N/A

A single implementation will be triggered by the application program

- Multiple Use Business Add-INS

Select the Multiple use checkbox from the Administration tab when creating the BADI definition

BADI has multiple implementations

Similar to event calls

Cannot expect return values

Can use importing and changing parameters

For multiple implementations there is no sequence control because at the time of BADI definition it does not know which implementation will be active

All active implementations will be triggered by the application program using the BADI

**Multi Use BADIs - CUSTOMER_ADD_DATA Definition**

- Multiple use buttons checked
- SAP defined BADI - CUSTOMER_ADD_DATA
- CUSTOMER_ADD_DATA is an SAP defined BADI that is invoked by the transaction VD02
- Multi Use BADIs - Custom Implementation for BADI - CUSTOMER_ADD_DATA
- Create a second implementation for BADI CUSTOMER_ADD_DATA
- Specify BADI name
- Click on CREATE
- Create a second implementation for BADI CUSTOMER_ADD_DATA
- Multi Use BADIs - Interface method SAVE_DATA
- Double click on method
- SAVE_DATA
- Double click on method SAVE_DATA TO enter the class builder where you can enter custom code for this method.

**Multi Use BADIs - Modify method SAVE_DATA**

- Enter custom code for method SAVE_DATA
- Enter the custom code for the method SAVE_DATA in the class builder and activate.
- Activate the BADI Implementation before exiting.
- Multi Use BADIs - View implementations for BADI
- SE18
- Displays list of
- Implementations
- Select to display list of
- Implementations for BADI
- CUSTOMER_ADD_DATA
- To view all implementation for BADI CUSTOMER_ADD_DATA go to the BADI definition in se18. Click on Implementation->Display.
- A list of all the Implementations is displayed.
- You can CREATE CHANGE, DISPLAY and DELETE Implementations from here.
Both BADIs will be triggered when the customer data is modified in transaction VD02 and saved.

Filter Dependent BADIs

Business Add-Ins can be implemented based on a specific filter value

If you want to call the implementation of a Business Add-In depending not only on one filter value but on various values, you can enter the name of a structure into the Filter type field. The structure can consist of several data elements that fulfill the above conditions for data elements.

Filter Dependent BADIs - BADI Definition

SE18

Click checkbox for Filter Dependent BADIs

Enter Filter type

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 1). 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 2). The element's domain must have fixed domain values or a value table containing a column with the same type as the data element.

Filter Dependent BADIs - Interface - Method Definition

Double click on the Interface to add parameters and methods

Enter text

When you double click on the interface it takes you to the class builder where you can add methods, parameters, and exceptions to the interface.

Filter Dependent BADIs - Interface - Method Definition

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.

Filter Dependent BADIs - Interface - Parameter Definition

FLT_VAL parameter preset for method

Create parameter to export tax rate

The filter value is declared using parameter flt_val and is preset in the list of parameters.

Filter Dependent BADIs - Implementation

Enter name of BADI

Select CREATE button

Filter Dependent BADIs - BADI Implementation

SE19

Select a value from the search help or enter a valid value

Click on button to create an entry for filter value

Create a BADI

Filter Dependent BADIs - BADI Implementation

Double click on the Method to enter source code

Select the Interface tab

Filter Dependent BADIs - Method code

Enter code for the
- Method to calculate the tax rate based on the company code in Parameter FLT_VAL
- Save and activate
- The parameter FLT_VAL will have the company code value passed to the implementation.
- Move the tax rate to the export parameter Z_TAX_RATE.
- Save and Activate the method
- Filter Dependent BAdIs - Source code to Invoke BADI
- Call BADI from within the program
- Selection screen parameter for Company code
- Call method get_bukrs passing the company code
- The Tax rate is returned
- A second implementation can be created wherein a different set of company codes can be specified.