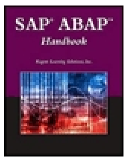


SAP ABAP Handbook

by Kogent Learning Solutions, Inc.
Jones and Bartlett Publishers. (c) 2010. Copying Prohibited.

Chapter 14: Business Add-Ins (BADIs)

Overview

At times, some special functions or programs need to be predefined in a software application to enhance the functionality of the application. For example, there are many Microsoft Excel add-ins (such as converters, sorting tools, and complex table filters) to enhance the basic functionality provided by Excel. Similarly, SAP facilitates in predefining such functions by providing Business Add-Ins (BADIs). A BADI is an enhancement technique that facilitates an SAP programmer, a customer, or a specific industry to enhance or add some additional code to existing standard programs in an SAP system. You can use standard or customized logic to enhance the SAP system. A BADI must first be defined and then implemented to enhance an SAP application. While defining a BADI, an interface is created. The BADI is implemented by this interface, which in turn is implemented by one or more adaptor classes. Finally, you need to create an instance of the adaptor class and method calls in the SAP application as per the requirements to use the BADI.

In this chapter, you learn about the concept of BADIs, Enhancement Framework, enhancement concepts, different enhancement techniques in Enhancement Framework, and BADIs as one of the enhancement techniques in Enhancement Framework. Next, you learn about the structure of BADIs and how to define, implement, and call BADIs. You also learn about the filter-dependent BADIs. Finally, at the end of this chapter, you learn about different features of BADIs, such as function code enhancements (a new name adopted for the menu enhancements), and screen enhancements. Now, let's start by exploring the concept of BADIs.

Concept of BADIs

A BADI is an enhancement technique of SAP that uses ABAP objects to create predefined points in the SAP ERP components. These predefined points are then implemented by the individual industry solutions, country variants, or even by partners and customers to suite their specific requirements. SAP introduced the BADI enhancement technique with Release 4.6A; the technique was re-implemented in Release 7.0.

Prior to BADI, a technique called customer exits was used to add custom functionalities to existing standard business applications in an SAP system. Customer exits were created to add some specific programs, screens, and menus to standard applications. These exits, however, do not contain any functionality of their own, but help add further functionality. In addition, customer exits are based on a two-level structure that incudes an SAP system and a customer solution. In contrast to customer exits, however, BADIs are based on a multi-level structure that includes an SAP system, industry solutions, country-specific versions, partner solutions, and customer solutions.

The BADI technique is different from other enhancement techniques in two respects. First, enhancement techniques can be implemented only once; second, enhancement techniques can be used by multiple customers simultaneously. In addition, you can create filter BADIs, which means BADIs are defined on the basis of filtered data, which is not possible with enhancement techniques. The concept of BADIs has been redefined in SAP Release 7.0 with the following goals:

- Enhancing the standard applications in an SAP system by adding two new elements in the ABAP language, i.e., `GET BADI` and `CALL BADI.`
- Providing more flexibility features, such as contexts and filters, for the enhancement of standard applications in an SAP system.

When a BADI is created, it contains an interface and other additional components, such as function codes for menu enhancements or screen enhancements. A BADI creation allows customers to include their own enhancements in the standard SAP application. The enhancement, interface, and generated classes are located in an appropriate application development namespace.

Now, let's explain the concept of Enhancement Framework and the use of a BADI as one of the enhancement techniques of Enhancement Framework.

Enhancement Framework

Enhancement Framework is an enhancement concept of ABAP Workbench that integrates different elements of an SAP application to modify and enhance the development objects. Prior to Release 7.0 of SAP, Enhancement Framework was used to fulfill two goals: enhancing an SAP application by inserting user developments at predefined points in an SAP system and modifying the delivered development objects. With Release 7.0 of SAP, the aim of Enhancement Framework has changed. Now, Enhancement Framework is used to unify all possible ways to modify or enhance the SAP products; for example, repository objects, which could not be customized in earlier releases of SAP. In addition, you can replace a repository object with an object having the same name, and you can enhance a repository object at a predefined position in an SAP system.

Note Within Enhancement Framework, any explicit reference to the BADI concept prior to Release 7.0 is termed as classic BADIs. References to the BADI concept in Release 7.0 is simply termed as BADI.

It is interesting to note that, unlike modifications, you can introduce enhancements at different development levels, which are as follows:

1. Core development
2. Application development
3. Add-on development
4. Customer development

Figure 14.1 shows the enhancements at the different development levels:

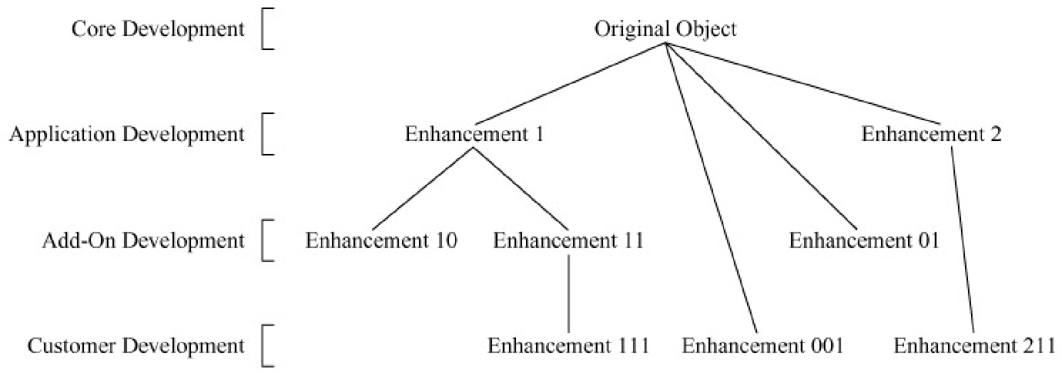


Figure 14.1: Enhancements at the different development levels

Note You can create multiple enhancement implementations or replace an enhancement implementation on different layers. However, enhancements cannot be nested.

Now, let's discuss the following topics in the context of Enhancement Framework:

- Enhancement concept
- Enhancement Builder
- Different enhancement techniques in Enhancement Framework

Overview of Enhancement

The enhancement concept enables you to integrate the following concepts to modify and enhance development objects:

- Enhancement options
- Enhancement spots
- Enhancement implementations

Enhancement Options

Enhancement options are positions in repository objects where enhancements can be made. These options are either defined explicitly (by the developers) or exist implicitly.

Enhancement Spots

Enhancement spots are points in an SAP system where enhancement options are created. An enhancement spot contains information about how an enhancement option is defined in an SAP system. One enhancement spot can be used to manage one or more enhancement options. Conversely, one enhancement option can be managed by one or more enhancement spots.

An enhancement option is created when an enhancement spot is created as a repository object at a point and processed by the relevant tool. This enhancement option can then be called at different points by calling the elements of relevant enhancement spots. The definition and the corresponding call of an element of an enhancement spot together comprise the definition of an enhancement option.

The definition of an element of an enhancement spot must be assigned to one or more simple enhancement spots. Further, enhancement spots are assigned to one or more composite enhancement spots. Simple and composite enhancement spots are repository objects that form a tree-like structure. The branches of this tree-like structure represent composite enhancement spots and leaves represent simple enhancement spots. Note that a simple enhancement spot is always assigned to an enhancement technology, such as ABAP source code enhancement or BADI; whereas composite enhancement spots are a collection or group of simple enhancement spots. Moreover, a composite enhancement spot not only contains one or more simple enhancement spots but also one or more composite enhancement spots of the relevant type. Figure 14.2 shows the grouping of simple and complex enhancement spots:

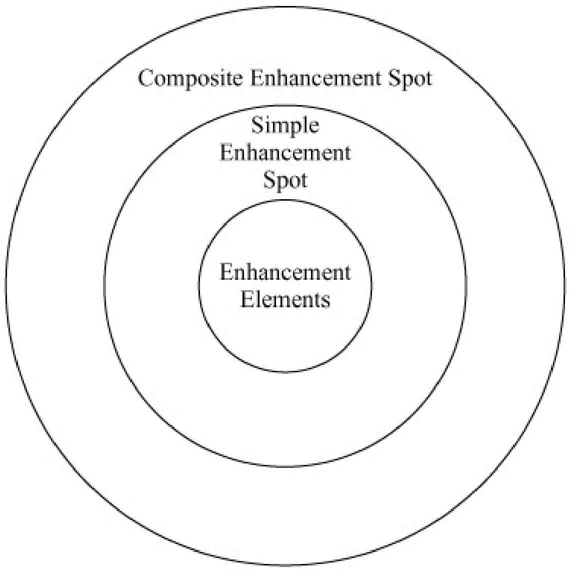


Figure 14.2: Grouping of the simple and complex enhancement spots

Enhancement spots are processed with the Enhancement Builder tool, which is integrated in the ABAP Workbench tool. Enhancement spots in Enhancement Builder indicate that enhancement options are explicit and are managed by developers.

Enhancement Implementations

Enhancement implementations manage enhancement options, both explicitly and implicitly. Similar to differentiated enhancement options and their management, we also differentiate the actual enhancement and its management in the case of enhancement implementations.

An enhancement implementation describes the enhancement of a repository object at one or more enhancement options. Note that an element of an enhancement implementation can belong to one enhancement option. However, elements of several enhancement implementations can be assigned to one enhancement option. The enhancement implementations are also divided into two categories: simple and composite. A composite enhancement implementation contains one or more simple enhancement implementations or may contain one or more composite enhancement implementations of the relevant type. An enhancement implementation is processed by using the Enhancement Builder tool.

You can manage an enhancement by assigning one or more simple enhancement implementations to an enhancement spot. However, a simple enhancement implementation belongs to only one enhancement spot. In addition, a simple enhancement implementation must be assigned to at least one composite enhancement implementation. Figure 14.3 shows the grouping of the simple and composite enhancement implementations:

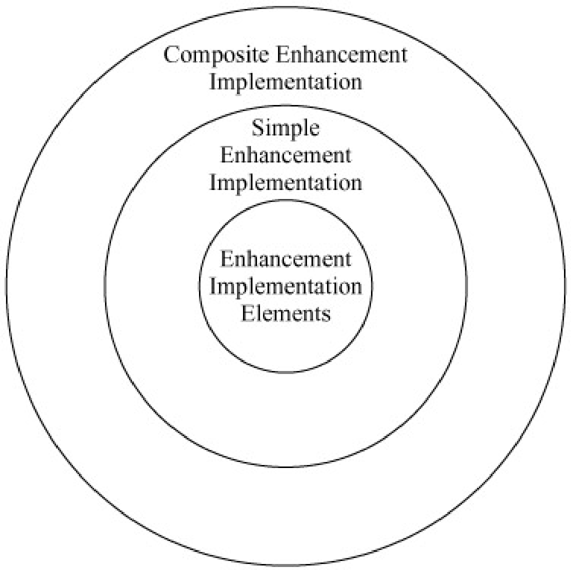

Composite Enhancement Implementation
Simple Enhancement Implementation
Enhancement Implementation Elements

Figure 14.3: Grouping of the simple and complex enhancement implementations

Similar to simple and composite enhancement spots, simple and composite enhancement implementations are repository objects that form a tree-like structure. The branches of this tree-like structure represent composite enhancement implementations, and the leaves represent simple enhancement implementations. To implement the actual enhancement, one or more elements of an enhancement implementation are assigned to a definition of the element of an enhancement spot, which is assigned to an explicit or implicit enhancement option.

Figure 14.4 shows an overview of enhancement spots and enhancement implementations:

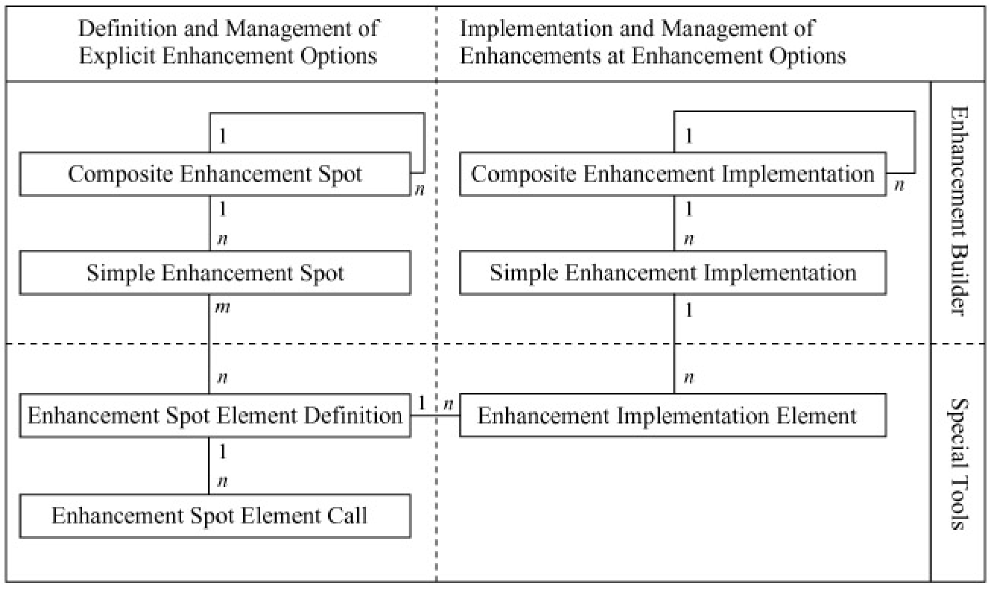

Definition and Management of Explicit Enhancement Options
Implementation and Management of Enhancements at Enhancement Options
Composite Enhancement Spot
Simple Enhancement Spot
Enhancement Spot Element Definition
Enhancement Spot Element Call
Composite Enhancement Implementation
Simple Enhancement Implementation
Enhancement Implementation Element
Enhancement Builder
Special Tools
1
n
m

Figure 14.4: Enhancement spots and enhancement implementations

In Figure 14.4, the left part shows the common terms used for enhancement spots and the types of relationships between them. These enhancement spot terms apply only to explicit enhancement options. Moreover, no enhancement spots are required for implicit enhancement options.

The right part of the figure shows the common terms used for enhancement implementations and the types of relationships between them. However, the enhancement spot terms apply to enhancements of both explicit and implicit enhancement options.

The Enhancement Builder Tool

The Enhancement Builder tool is a collection of individual tools used to create, manage, and maintain the enhancement spots and enhancement implementations. Enhancement Builder is integrated in the ABAP Workbench tool. Moreover, the Enhancement Builder tool contains special tools for navigation and administration.

Enhancement Techniques in Enhancement Framework

The following are some enhancement techniques used in Enhancement Framework:

- **ABAP Source Code Enhancements**—Enhance the ABAP source code by using source code plug-ins. The source code plug-ins do not modify the ABAP source code; rather, they execute the enhancements using implicit and explicit enhancement options. A source code plug-in refers to an enhancement implementation element of a simple enhancement implementation. A source code plug-in is assigned to a single enhancement option. However, an enhancement option can have several source code plug-ins and enhancement implementations. A source code plug-in is created by using the Enhancement Builder tool, which defines the explicit enhancement options and implements the enhancements. The source code plug-in has a unique ID and is displayed in the ABAP Editor. The following syntax shows how to write the enhancement option:

  ```
  ENHANCEMENT id.
  ...
  ENDENHANCEMENT.
  ```

In this syntax, the enhancement is implemented between the `ENHANCEMENT` and `ENDENHANCEMENT` lines.

- **Function Module Enhancements**—Refer to various types of enhancements, which are:
  - **Source code enhancements—** Specify enhancements performed by using the ABAP source code.
  - **Parameter interface enhancements—** Specify enhancements performed by using a parameter interface. This means formal parameters are used as enhancement implementation elements using the Enhancement Builder tool.
- **Global Class Enhancements—**Refer to enhancements made to global classes and interfaces. Global class enhancements are related to the enhancement in the following areas:
  - **Enhancements to the source code of methods, local classes, and others—** Specify enhancements performed by using the ABAP source code.
  - **Enhancements to the components of global classes and global interfaces—** Specify enhancements performed by inserting new attributes, methods, formal parameters to existing methods, and the implementation of an overwritten method, and a pre- or post-method for existing methods. Note that a pre-method is always executed directly before the first statement of the existing method is executed, while a post-method is executed after the execution of the last statement of the existing method, but before the `ENDMETHOD` statement. This condition is valid only if the method ends by using the `ENDMETHOD` statement.
- **BADIs**—Refer to a new enhancement technique to enhance the standard version of an SAP system. The goal behind the development of BADI was to insert some predefined points in the SAP system so that users can insert their applications to enhance the SAP system at the defined points. For example, if you need to predefine a special function in an application to carry out a particular task, SAP allows you to predefine such functions in your application. BADI is not limited to SAP applications; it can be integrated in customer applications as well, where it can be enhanced by other customer applications.

Structure of a BADI

A BADI acts as a definition of elements of an enhancement spot. BADIs are basic requirements for the object plug-ins, which are used to enhance the functions in an ABAP program without modifying them. BADIs enable you to create enhancement options in the form of interfaces, which can be implemented later, either in the same SAP system or different SAP systems.

The definition of a BADI contains a BADI interface, a set of filters, and the settings that affect the runtime behavior later. A BADI interface can form the partial or entire interface of an object plug-in. A BADI implementation consists of a BADI implementation class that implements the BADI interface, and a condition that is imposed on the filters specified in the BADI definition.

The definition of a BADI is usually created in an SAP system. ABAP programs are then created to define the calling points for the definition of a BADI. The definition of a BADI, along with its calling points in ABAP programs, forms the explicit enhancement options in such programs. A BADI implementation is the term used in Enhancement Framework for an enhancement implementation element.

The calling points of a BADI are defined through the ABAP statements, `GET BADI` (to get objects) and `CALL BADI` (to call interface methods). These statements, in the definition of Enhancement Framework, represent the enhancement spot element calls of the explicit enhancement option.

Figure 14.5 shows the object types in the structure of a BADI:

In Figure 14.5, notice that the name of the BADI is `cl_badi_name`. A class with the same name, `cl_badi_name`, is created in the Enhancement Builder tool. This class is the final subclass of the `abscl_badi_base` abstract global class. Note that the BADI class forms the template for the BADI object but cannot be used directly.

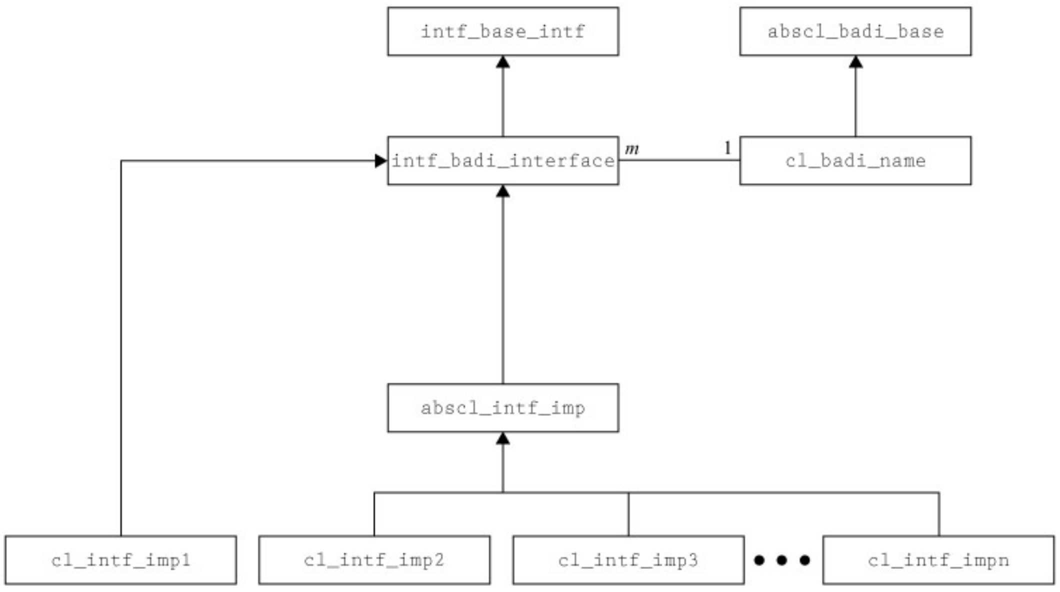


Figure 14.5: Structure of a BADI

The name of the BADI interface, for which the `cl_badi_name` class is defined, is `intf_base_intf` and contains the predefined tag interface, `intf_badi_interface`. The BADI interface must not contain any variable attributes. If the BADI interface is used multiple times in different contexts, it must not contain any methods with exporting or returning parameters.

The BADI implementation classes (`cl_intf_imp1`, `cl_intf_imp2`, `cl_intf_imp13 ...`, `cl_intf_impn`) also implement the BADI `interface intf_base_intf`. There are no limitations with regard to the classes to be implemented, provided they implement the BADI interface only. A BADI implementation class can even implement two different BADI interfaces with the help of an abstract class. In Figure 14.5, the `abscl_intf_imp` abstract class is displayed, which is the partial implementation of the BADI interfaces. In addition, the actual BADI implementation classes are inherited from the `abscl_intf_imp` abstract class.

Note The BADI class is not a BADI implementation class and does not implement the BADI interface, but merely contains references to the actual object plug-ins (objects of BADI implementation classes).

Definition of BADIs

The definition of a BADI contains the name of a BADI, an interface of BADI, and some filter conditions. A filter consists of a filter name, a condition, and a data type (for example, `integer` or `string`). When you create the definition of a BADI, you must use a suitable prefix (for instance `BADI_`) when you specify the name of the BADI. In addition, you also define BADI properties, Instance Generation Mode, and Multiple Use at the runtime of a program. These properties are handled by using the `GET BADI` and `CALL BADI` statements. The Instance Generation Mode property controls the instantiation of an object plug-in when the `GET BADI` statement is executed. You can specify any of the following values for the Instance Generation Mode property:

- **Newly created instantiation—** Creates a new object plug-in every time the `GET BADI` statement is executed.
- **Reused instantiation—** Reuses an object plug-in more than once.
- **Context-dependent instantiation—** Controls the creation of an object plug-in by specifying a context for the `GET BADI` statement. You can reuse a BADI if the `GET BADI` statement needs to be used for a different purpose (other than creating a new object plug-in) in the same context. A context is defined as an instance of a class that implements the `if_badi_context` tag interface.

Note that the first two values are used for context-free BADIs.

The Multiple Use property is used to decide the number of BADI implementations that can be selected. You can select zero, one, or an arbitrary number of implementations when a BADI object is initialized by using the `GET BADI` statement. The `CX_BADI_MULTIPLY_IMPLEMENTED` exception is raised when several implementations are selected in the `GET BADI` statement for a BADI. In addition, the `CX_BADI_NOT_IMPLEMENTED` exception is raised when no implementation is found in a BADI definition.

Note that in the standard version of the SAP system, a BADI is provided for a single use, but it can be used multiple times as well. In the case of multiple use of a BADI, the methods defined in the BADI must not contain any `EXPORTING` or `RETURNING` parameters but can have `CHANGING` parameters. The `EXPORTING` or `RETURNING` parameter cannot be used because a single definition of a BADI cannot return values to multiple implementations of a BADI. However, the `CHANGING` parameter is allowed because changes in the values are made by calling the methods of BADI implementations.

The following are some other properties that can also be assigned to a BADI:

- **An optional fallback BADI implementation class—** Specifies that a BADI implementation with some filter conditions or a standard implementation is not found.
- **Whether or not the BADI is internal—** Specifies that an internal BADI is essentially implemented inside the SAP system and not accessible to external systems.
- **Whether the BADI is a function code or screen enhancement—** Specifies that a BADI defined as a function code enhancement must not have any filters, must not be defined for multiple use, and must not be assigned to any switch.

Now, let's move on to discuss the following topics in the context of the definition of BADIs:

- Defining a BADI
- Displaying, changing, or deleting a BADI definition

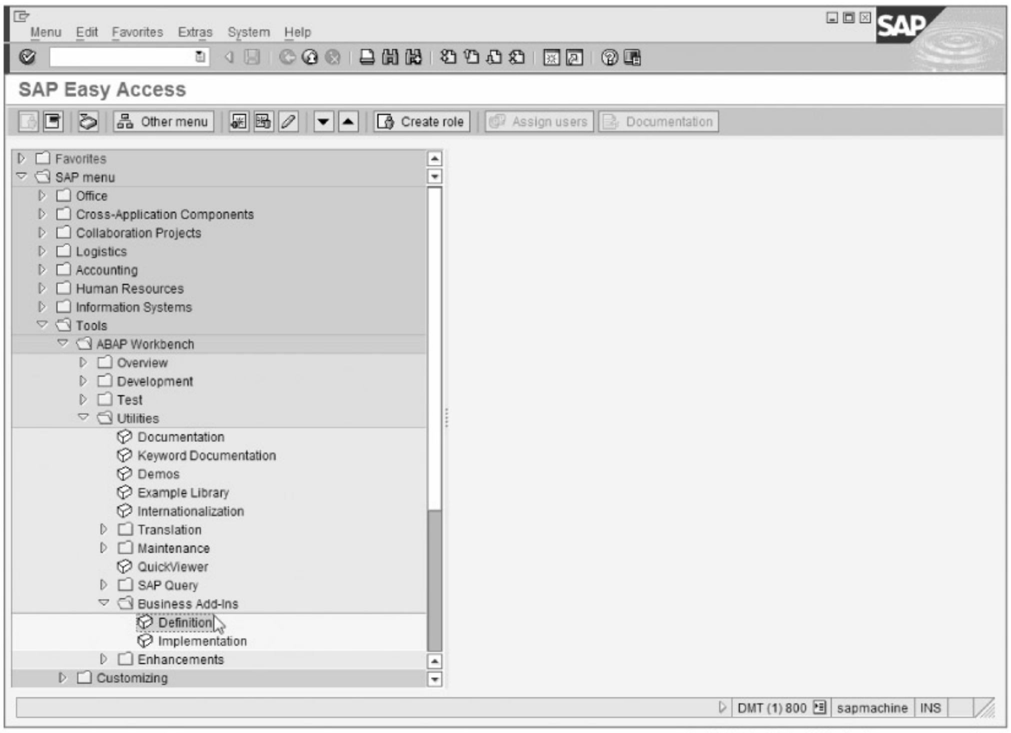
Defining a BADI

Enhancement Builder, one of the tools of ABAP Workbench, is used to create and mange the definition of a BADI. You can access the Enhancement Builder tool by using the `SE18` transaction code. Enhancement Builder manages the definition of classic as well as new BADI. When you have to display the definition of an existing BADI, the Enhancement Builder tool analyzes whether the BADI is a classic BADI or a new BADI and then opens the BADI in the respective tool. In case of a new BADI, Enhancement Builder opens the enhancement spot editor.

Note Creating classic BADIs is no longer supported.

Perform the following steps to create a new BADI definition in the Enhancement Builder tool:

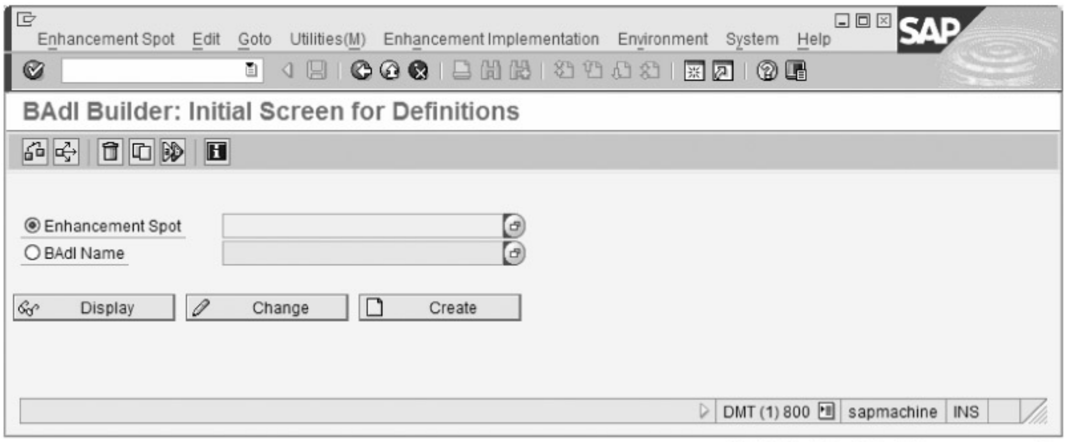
1. In the SAP menu, select `Tools > ABAP Workbench > Utilities > Business Add-Ins > Definition`, as shown in Figure 14.6:



© SAP AG. All rights reserved.

Figure 14.6: Selecting the definition option

The initial screen for definitions opens in the BADI builder, as shown in Figure 14.7:

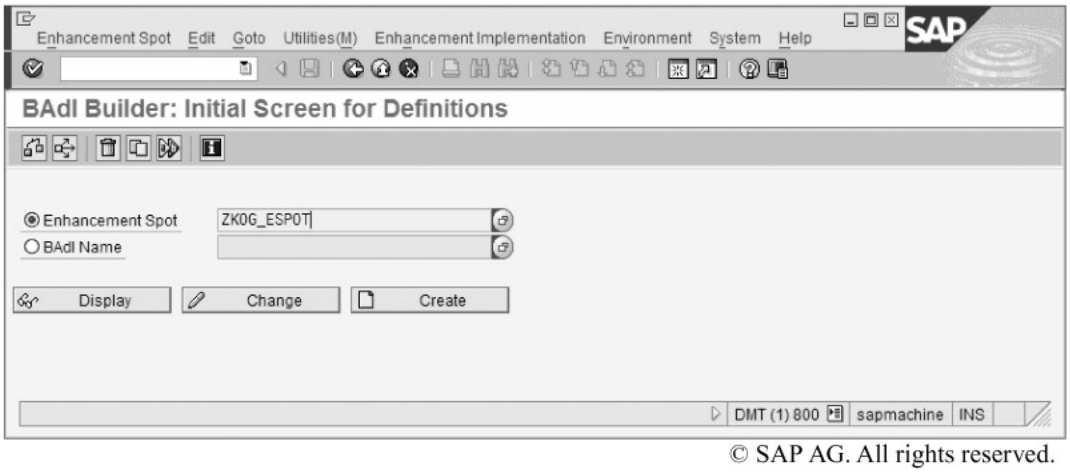

Enhancement Spot Edit Goto Utilities(M) Enhancement Implementation Environment System Help
BAdI Builder: Initial Screen for Definitions
Enhancement Spot
BAdI Name
Display
Change
Create
DMT (1) 800
sapmachine
INS

© SAP AG. All rights reserved.

Figure 14.7: The initial screen for definitions

Note An alternative way to open the initial screen for definitions is by using the `SE18` transaction code.

2. In the initial screen for definitions, enter a name for a new `enhancement spot`. In this case, we have entered the name as ZKOG_ESPOT, as shown in Figure 14.8:
3. Click the `Create` button (see Figure 14.8).



BAdI Builder: Initial Screen for Definitions

Enhancement Spot [✓] ZKOG_ESPOT

BAdI Name

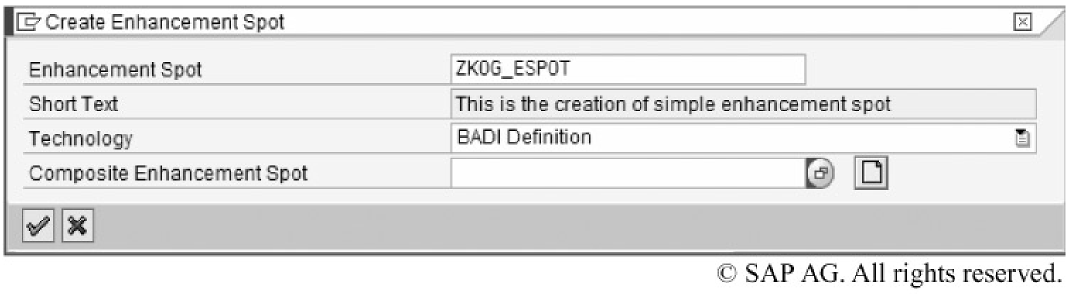
DMT (1) 800 sapmachine INS

© SAP AG. All rights reserved.

Figure 14.8: Specifying an enhancement spot

The `Create Enhancement Spot` dialog box appears, as shown in Figure 14.9:

4. Enter a short description to create a new enhancement spot in the `Short Text` field. Ensure that the `Enhancement Spot` field and the `Technology` field are populated automatically, and then click the `Continue` (✓) icon, as shown in Figure 14.9. The `Create Object Directory Entry` dialog box appears.
5. In the `Create Object Directory Entry` dialog box, enter the package name ZKOG_PCKG in the `Package` field and click the `Save` () icon. The ZKOG_ESPOT enhancement spot is opened in the `Change` mode of Enhancement Builder, as shown in Figure 14.10:
6. Select the `Enh. Spot Element Definitions` tab, as shown in Figure 14.10. The `Enh. Spot Element Definitions` tab page appears, as shown in Figure 14.11:
7. Click the `Create BADI` () icon; see Figure 14.11.

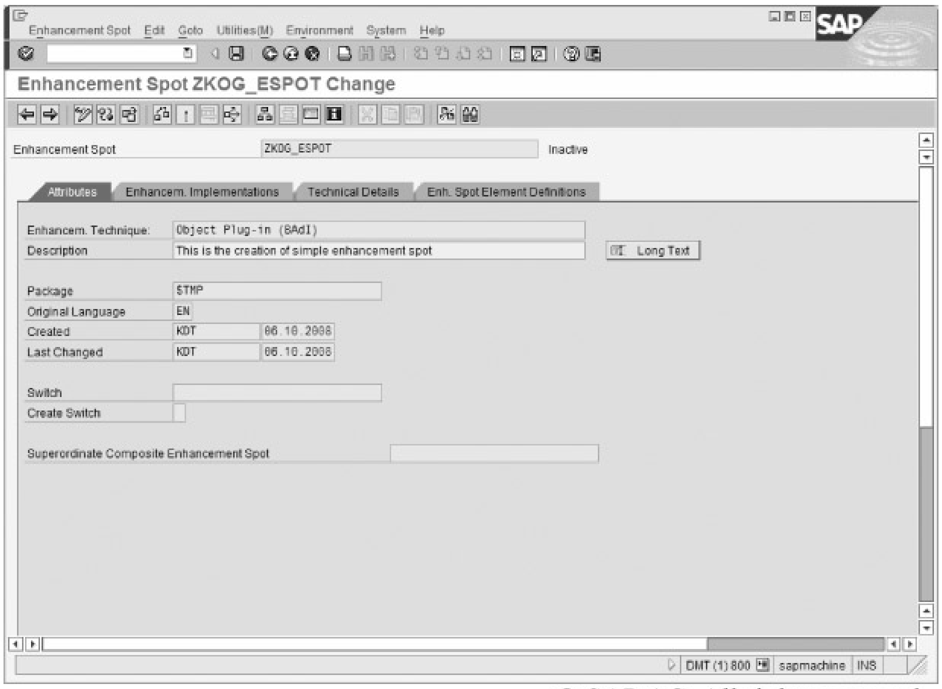


Create Enhancement Spot

Enhancement Spot	ZKOG_ESPOT
Short Text	This is the creation of simple enhancement spot
Technology	BADI Definition
Composite Enhancement Spot	

© SAP AG. All rights reserved.

Figure 14.9: The create enhancement spot dialog box


Enhancement Spot Edit Goto Utilities(M) Environment System Help

Enhancement Spot ZKOG_ESPOT Change

Enhancement Spot	ZKOG_ESPOT	Inactive

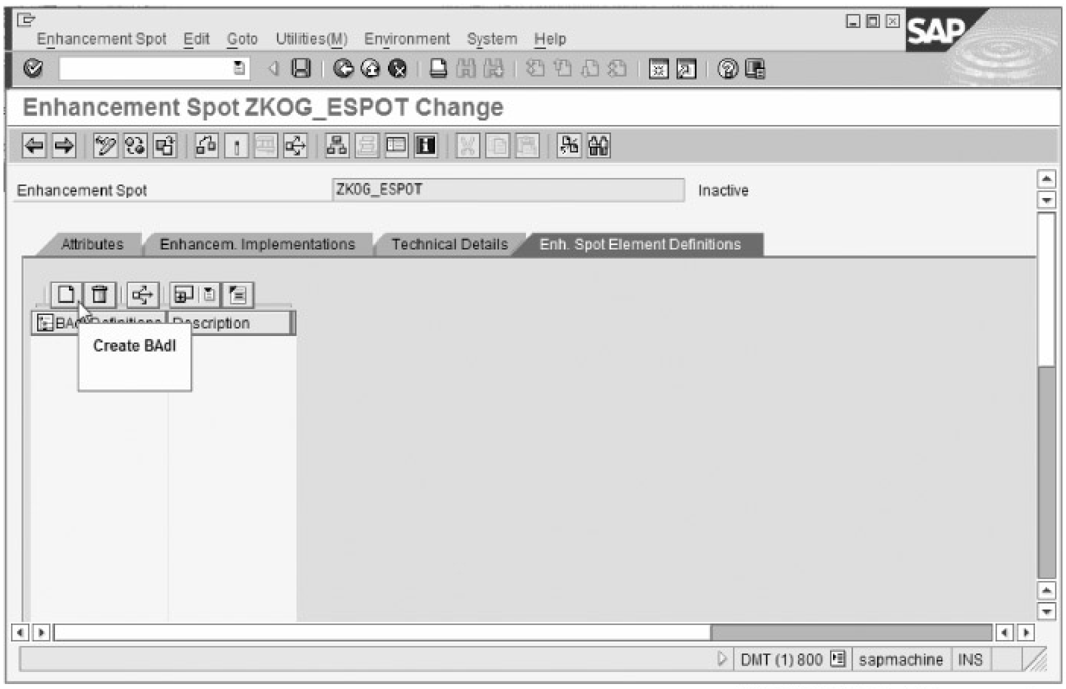
Attributes | Enhancem. Implementations | Technical Details | Enh. Spot Element Definitions

Enhancem. Technique:	Object Plug-in (BAdI)	
Description	This is the creation of simple enhancement spot	Long Text
Package	$TMP	
Original Language	EN	
Created	KDT	06.10.2008
Last Changed	KDT	06.10.2008
Switch		
Create Switch		
Superordinate Composite Enhancement Spot		

DMT (1) 800 sapmachine INS

© SAP AG. All rights reserved.

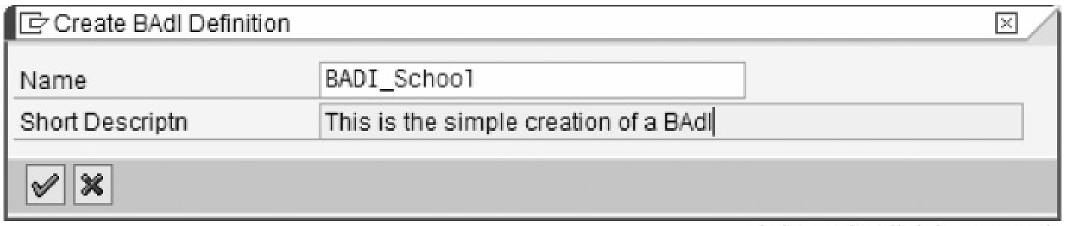
Figure 14.10: The ZKOG_ESPOT enhancement spot


Enhancement Spot Edit Goto Utilities(M) Environment System Help
Enhancement Spot ZKOG_ESPOT Change
Enhancement Spot
ZKOG_ESPOT
Inactive
Attributes
Enhancem. Implementations
Technical Details
Enh. Spot Element Definitions
BAdI Definitions
Description
Create BAdI
DMT (1) 800
sapmachine
INS

© SAP AG. All rights reserved.

Figure 14.11: The enh. spot element definitions tab page

The `Create BADI Definition` dialog box opens, as shown in Figure 14.12:



Name	BADI_School
Short Descriptn	This is the simple creation of a BAdI

© SAP AG. All rights reserved.

Figure 14.12: The create BADI definition dialog box

8. In the `Create BADI Definition` dialog box, enter a name and a description for the BADI definition in the `Name` and `Short Description` fields, respectively. Click the `Continue` (✓) icon, as shown in Figure 14.12.

The definition of a BADI is created in the Enhancement Builder tool, as shown in Figure 14.13. Perform the following on the right-hand side of the page:

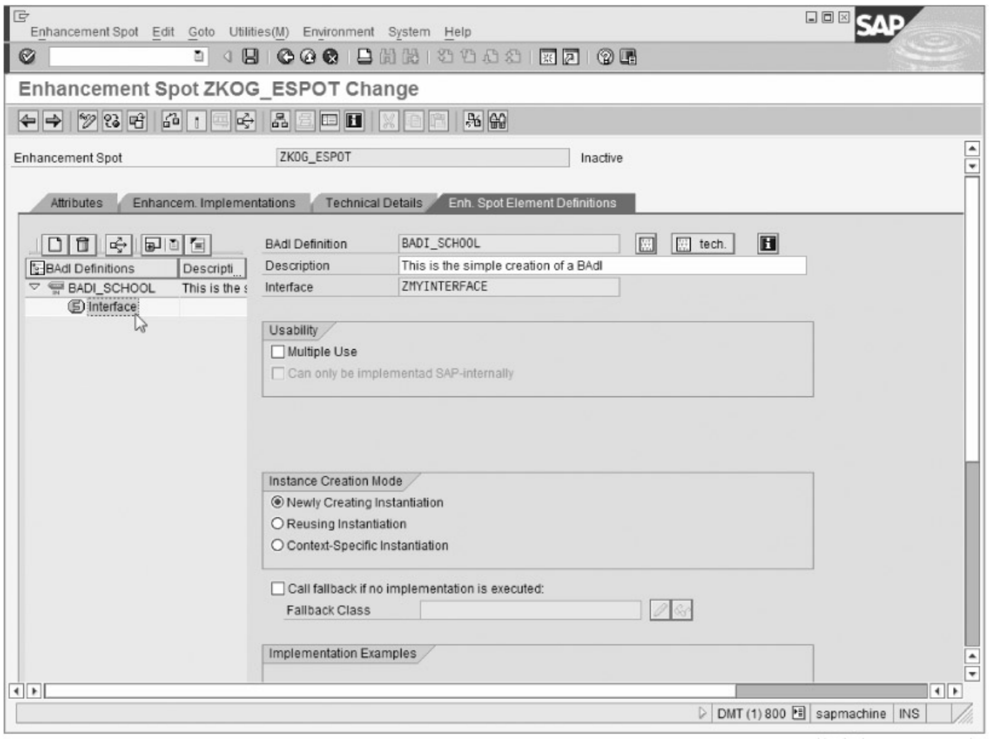
a. Enter the usability by selecting the Multiple Use check box (optional)

b. Enter the instance creation mode

c. Enter the attribute for internal SAP BADIs (only SAP internal use)

d. Enter a Fallback Class (optional)

9. Double-click the `Interface` node, as shown in Figure 14.13:



Enhancement Spot ZKOG_ESPOT Change

Enhancement Spot ZKOG_ESPOT Inactive

BAdI Definitions	Descripti...
BADI_SCHOOL	This is the s
Interface	

BAdI Definition	BADI_SCHOOL
Description	This is the simple creation of a BAdI
Interface	ZMYINTERFACE

Usability
- ☐ Multiple Use
- ☐ Can only be implementad SAP-internally

Instance Creation Mode
- ● Newly Creating Instantiation
- ○ Reusing Instantiation
- ○ Context-Specific Instantiation

☐ Call fallback if no implementation is executed:
Fallback Class

Implementation Examples

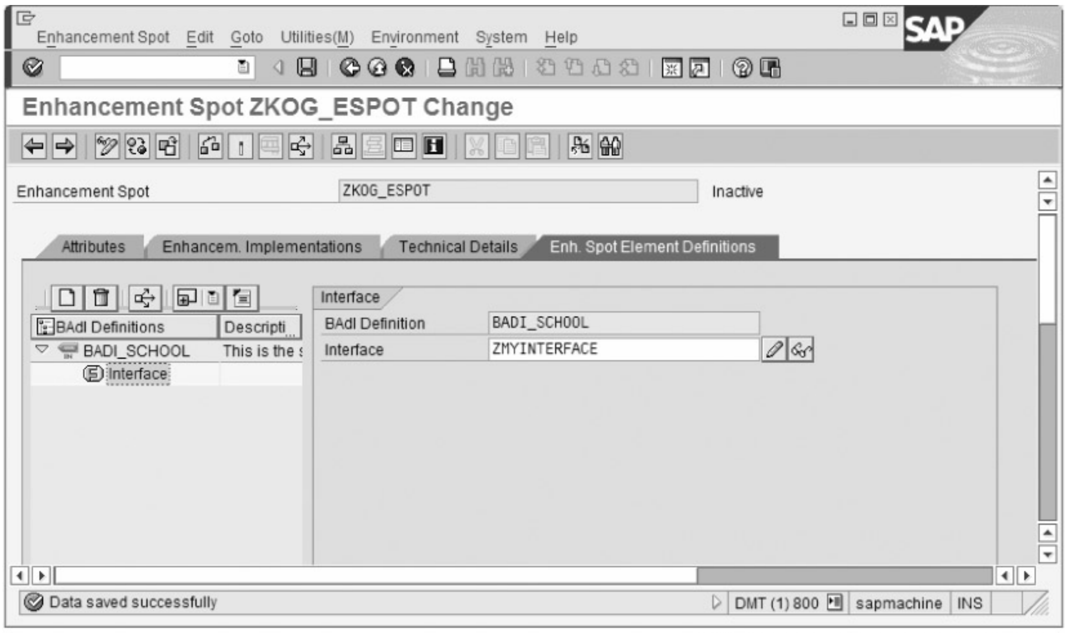
DMT (1) 800 sapmachine INS

© SAP AG. All rights reserved.

Figure 14.13: A BADI is created in the enhancement builder tool

The tab page, storing the information about the interface of the BADI definition, appears, as shown in Figure 14.14:

10. Enter the name of an interface in the `Interface` text field. In this case, we have specified the interface name as ZMYINTERFACE (see Figure 14.14). Click the `Change` (✎) icon, so that you can define methods to this interface, in the `Class Builder` tool. Figure 14.15 shows the screen of the `Class Builder` tool to define methods for the ZMYINTERFACE interface:


Enhancement Spot Edit Goto Utilities(M) Environment System Help
Attributes
Enhancem. Implementations
Technical Details
Enh. Spot Element Definitions
DMT (1) 800
sapmachine
INS

Enhancement Spot ZKOG_ESPOT Change

Enhancement Spot ZKOG_ESPOT Inactive

BAdI Definitions	Descripti
BADI_SCHOOL	This is the s
Interface	

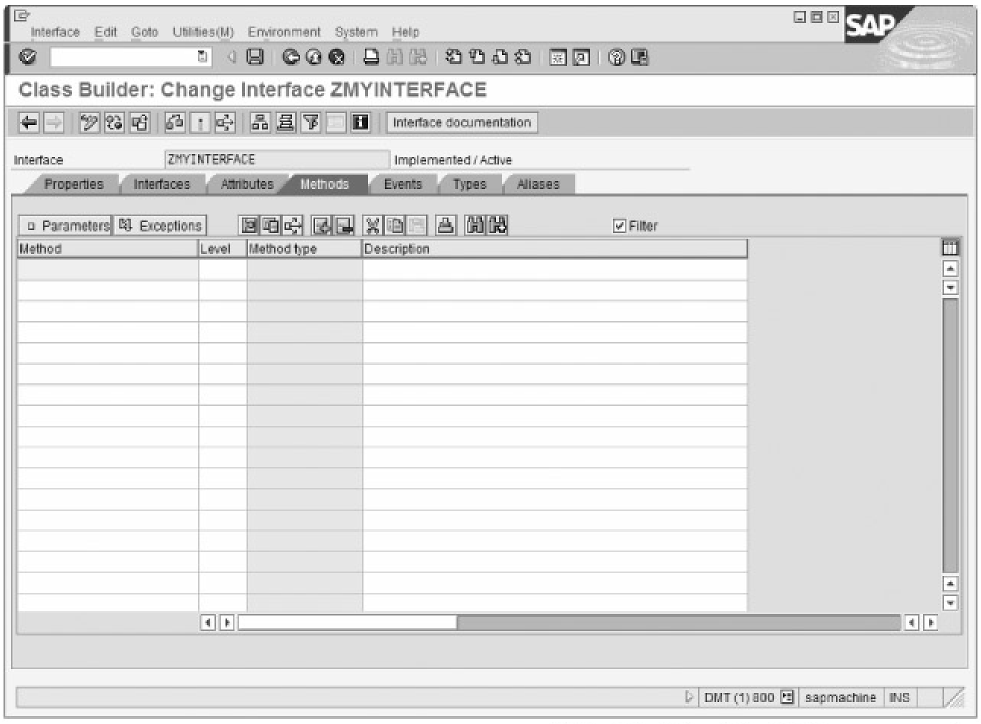
Interface

BAdI Definition	BADI_SCHOOL
Interface	ZMYINTERFACE

Data saved successfully

© SAP AG. All rights reserved.

Figure 14.14: Information about the interface of the BADI definition



Class Builder: Change Interface ZMYINTERFACE

Interface ZMYINTERFACE Implemented / Active

Method	Level	Method type	Description

© SAP AG. All rights reserved.

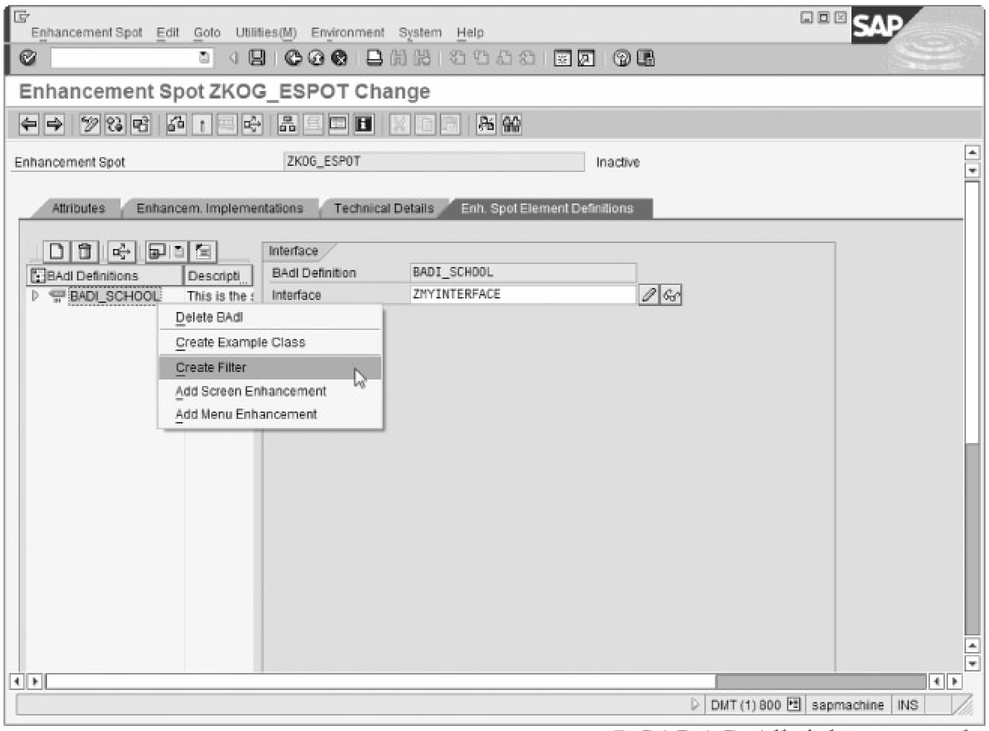
Figure 14.15: The screen of the class builder tool

You can assign a method to the interface and define a parameter with the attributes in the `Class Builder` tool. Use the `Save` and `Activate` icons to activate the interface and navigate back to the BADI definition. The method created for the interface appears in the BADI screen.

Note that when the methods are created in the BADI interface, the corresponding executing class (the Adapter class) is also generated, which further implements the interface methods.

Note Steps 11 to 14, which follow, are optional. Users can follow these steps according to their requirement of the BADI definition.

11. You can create a filter by selecting the `Create Filter` option from the context menu, which is displayed when the BADI definition is selected in the `Navigation` menu of the created BADI definitions, as shown in Figure 14.16:



Enhancement Spot ZKOG_ESPOT Inactive

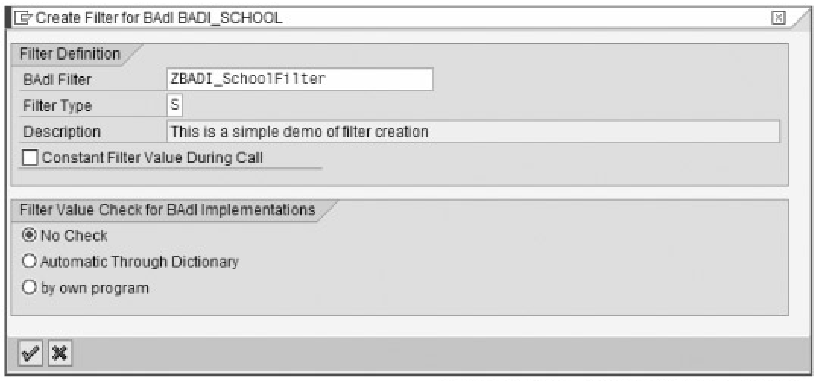
BADI_SCHOOL This is the

BAdI Definition BADI_SCHOOL

Interface ZMYINTERFACE

Figure 14.16: Selecting the create filter option

The previous action displays the `Create Filter for BADI` dialog box, as shown in Figure 14.17:


Create Filter for BAdI BADI_SCHOOL
Filter Definition
Filter Value Check for BAdI Implementations

BAdI Filter ZBADI_SchoolFilter

Filter Type S

Description This is a simple demo of filter creation

☐ Constant Filter Value During Call

[✓ No Check]

Automatic Through Dictionary

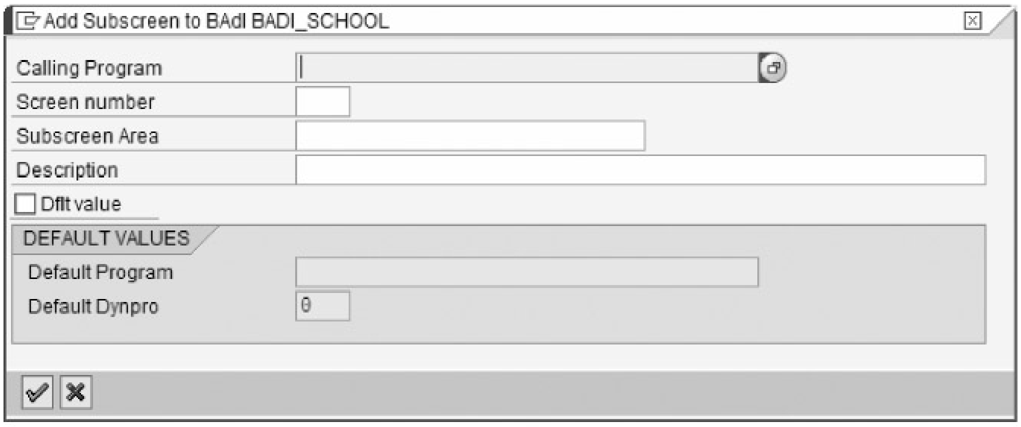
by own program

Figure 14.17: The create filter for BADI dialog box

In the `Create Filter for BADI` dialog box, enter a filter name, filter type, and description. In this case, the filter name is ZBADI_SchoolFilter, the filter type is S, and the description is This is a simple demo of filter creation (see Figure 14.17).

If you select the `Constant Filter Value During Call` check box, you can specify only a constant value for the respective filter when using the `GET BADI` statement. Click the `Continue` (✓) icon (see Figure 14.17).

12. Select the `Add Screen Enhancement` option (see Figure 14.16) to create the BADI as a screen enhancement. The `Add Subscreen to BADI` dialog box, as shown in Figure 14.18:



Add Subscreen to BAdI BADI_SCHOOL

Calling Program	
Screen number	
Subscreen Area	
Description	
☐ Dflt value	

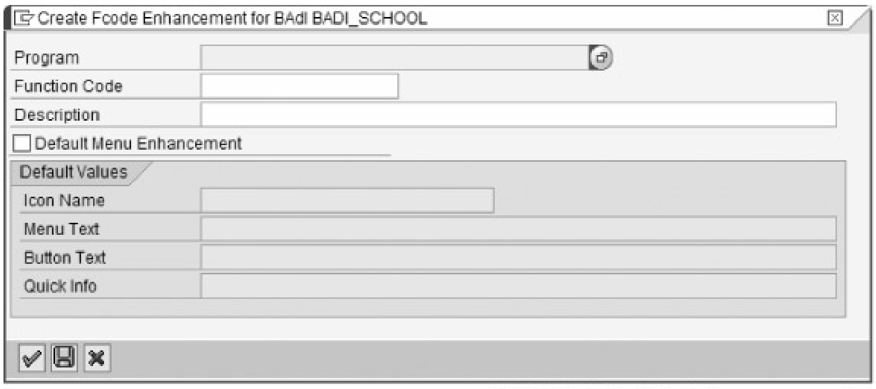
DEFAULT VALUES

Default Program	
Default Dynpro	0

Figure 14.18: The add subscreen to BADI dialog box

Enter the values for the `Calling Program`, `Screen number`, `Subscreen Area`, and `Description` fields. If you select the `Dflt value` check box, you have to specify a screen of a program that is used if no active implementation is found at runtime. Click the `Continue` (✓) icon. Note that when the BADI is created as a screen enhancement, it must not be of the multiple use type.

13. Select the `Add Menu Enhancement` option (see Figure 14.16) to create the BADI as a function code enhancement. The `Create Fcode Enhancement for BADI` dialog box, as shown in Figure 14.19:



Create Fcode Enhancement for BAdI BADI_SCHOOL

Program	
Function Code	
Description	
☐ Default Menu Enhancement	

Default Values

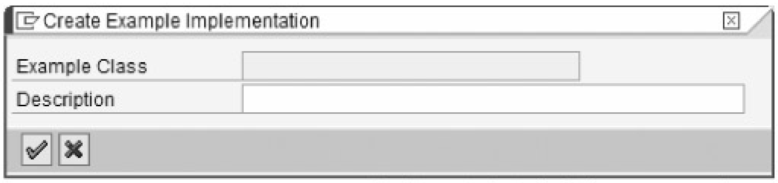
Icon Name	
Menu Text	
Button Text	
Quick Info	

Figure 14.19: The create fcode enhancement for BADI dialog box

Enter the values for the `Program`, `Function Code`, and `Description` fields. If you select the `Default Menu Enhancement` check box, you have to specify an icon, a menu text, a button text, and quick info, which are used when no active implementation is found at runtime.

Click the `Continue` (✓) icon. Note that when the BADI is created as a function code enhancement, it must not have any filters and must not be of the multiple use type.

14. Select the `Create Example Class` option (see Figure 14.16) to create an example implementation. Figure 14.20 shows the `Create Example Implementation` dialog box:



Create Example Implementation

Example Class	
Description	

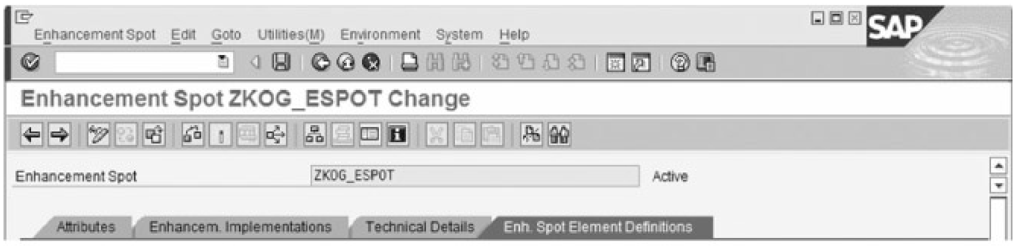
Figure 14.20: The create example implementation dialog box

Enter the name of a BADI implementation class and a description. Click the `Continue` (✓) icon.

After performing any one or more of the optional steps from 11 to 14, you must save and activate the BADI definition by using the `Save` (), `Check` (), and `Activate` () icons, in the same order.

Figure 14.21 shows the activation of the ZKOG_ESPOT enhancement spot:

Note that the activation of the ZKOG_ESPOT enhancement spot also activates the definition of the BADI_School BADI.


Enhancement Spot Edit Goto Utilities(M) Environment System Help
Enhancement Spot ZKOG_ESPOT Change
Attributes
Enhancem. Implementations
Technical Details
Enh. Spot Element Definitions

Enhancement Spot ZKOG_ESPOT Active

© SAP AG. All rights reserved.

Figure 14.21: Activation of the ZKOG_ESPOT enhancement spot

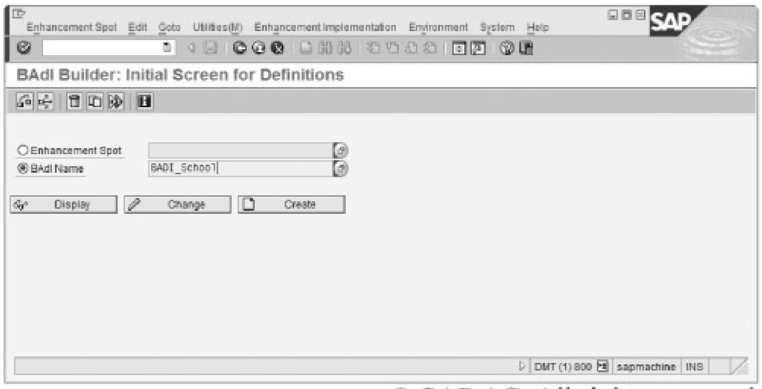
Displaying, Changing, or Deleting a BADI Definition

Perform the following steps to display, change, or delete the definition of a BADI:

1. Open the initial screen for a BADI by using the SE18 transaction code or by selecting `Tools > ABAP Workbench > Utilities > Business Add-Ins > Definition` in the SAP menu.

2. Enter the name of an existing BADI definition that needs to be displayed or changed. In this case, the BADI definition name entered is BADI_School, as shown in Figure 14.22:

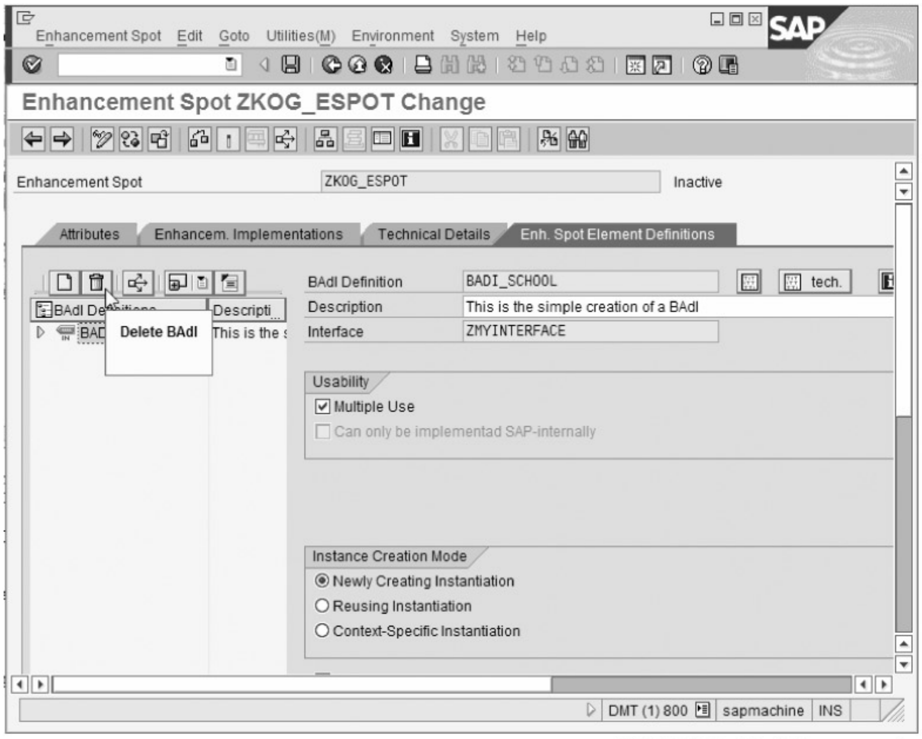
3. Now, click the `Display` or `Change` button to bring the selected BADI definition in the display or change mode, respectively. Moreover, in the `change` mode of the enhancement spot, click the `Delete` () icon to delete the selected BADI definition, as shown in Figure 14.23:


Enhancement Spot Edit Goto Utilities(M) Enhancement Implementation Environment System Help
BAdI Builder: Initial Screen for Definitions
Display
Change
Create
DMT (1) 800
sapmachine
INS

○ Enhancement Spot
◉ BAdI Name BADI_School

© SAP AG. All rights reserved.

Figure 14.22: The initial screen for definitions


Enhancement Spot Edit Goto Utilities(M) Environment System Help
Attributes
Enhancem. Implementations
Technical Details
Enh. Spot Element Definitions
BAdI Definitions
Descripti...
Delete BAdI
tech.

Enhancement Spot ZKOG_ESPOT Change

Enhancement Spot	ZKOG_ESPOT	Inactive

BAdI Definition	BADI_SCHOOL
Description	This is the simple creation of a BAdI
Interface	ZMYINTERFACE

Usability
[✓] Multiple Use
[ ] Can only be implementad SAP-internally

Instance Creation Mode
[● Newly Creating Instantiation]
○ Reusing Instantiation
○ Context-Specific Instantiation

DMT (1) 800 sapmachine INS

© SAP AG. All rights reserved.

Figure 14.23: Deleting a BADI definition in the change mode of the enhancement spot

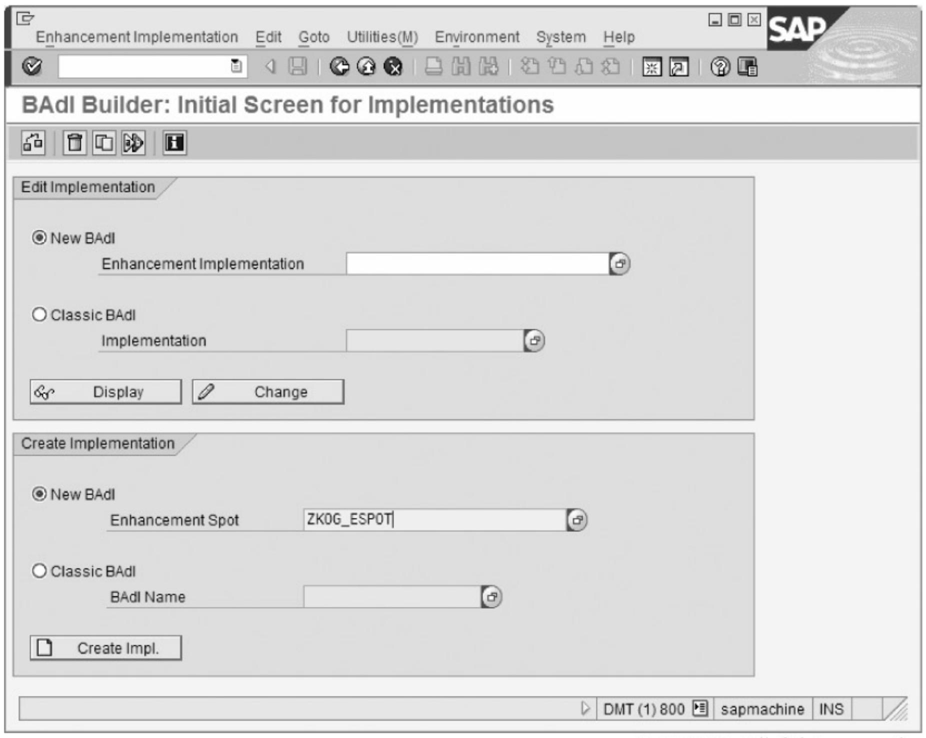
Note Click the Save icon to save any changes in the enhancement spot.

Implementation of BADIs

The implementation of a BADI refers to the implementation of the interface and the filters defined in the BADI definition. A BADI implementation consists of an implementation class and a filter condition by which the BADI implementation is selected using the `GET BADI` statement. The BADI implementation is identified by a unique name in the enhancement concept. Perform the following steps to implement a BADI definition:

1. Open the initial screen for BADI implementation by using the `SE19` transaction code or by selecting `Tools > ABAP Workbench > Utilities > Business Add-Ins > Implementation` from the SAP menu. Figure 14.24 shows the initial screen for BADI implementation:
2. In the `Create Implementation` group box, enter the name of an existing enhancement spot. In this case, the enhancement spot entered is ZKOG_ESPOT, as shown in Figure 14.24.
3. Click the `Create Impl.` button (see Figure 14.24). The `Create Enhancement Implementation` dialog box appears, as shown in Figure 14.25:
4. In the `Create Enhancement Implementation` dialog box, enter a name and short description for the enhancement implementation. In Figure 14.25, `Enhancement Implementation` is ZImp_KOG_ESPOT, and `Short Text` is Demo of BADI Implementation. Now, click the `Continue` icon. The `Create Object Directory Entry` dialog box appears.
5. In the `Create Object Directory Entry` dialog box, enter the package name ZKOG_PCKG in the `Package` field and click the `Save` icon. The `Enhancement Implementation` dialog box, respective to the enhancement spot implementation, is displayed, as shown in Figure 14.26:
6. In the `Enhancement Implementation` dialog box, enter a name for BADI Implementation, such as ZIMP_BADI_SCHOOL, as shown in Figure 14.26.
7. Click the `Continue` icon to display the screen for Enhancement Implementation, as shown in Figure 14.27:
8. In the `Navigation` menu for the BADI implementation, double-click the `Implementing Class` option and provide

a name for the `Implementing Class` field, as shown in Figure 14.28:


Enhancement Implementation Edit Goto Utilities(M) Environment System Help
Display
Change
Create Impl.

BAdI Builder: Initial Screen for Implementations

Edit Implementation

(●) New BAdI
Enhancement Implementation:

( ) Classic BAdI
Implementation:

Create Implementation

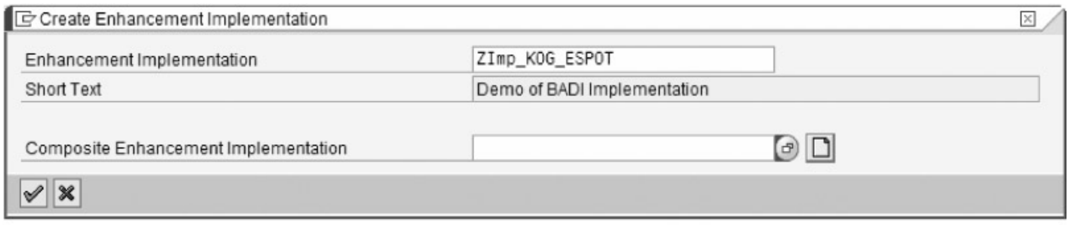
(●) New BAdI
Enhancement Spot: ZKOG_ESPOT

( ) Classic BAdI
BAdI Name:

DMT (1) 800 | sapmachine | INS

© SAP AG. All rights reserved.

Figure 14.24: The initial screen for BADI implementation

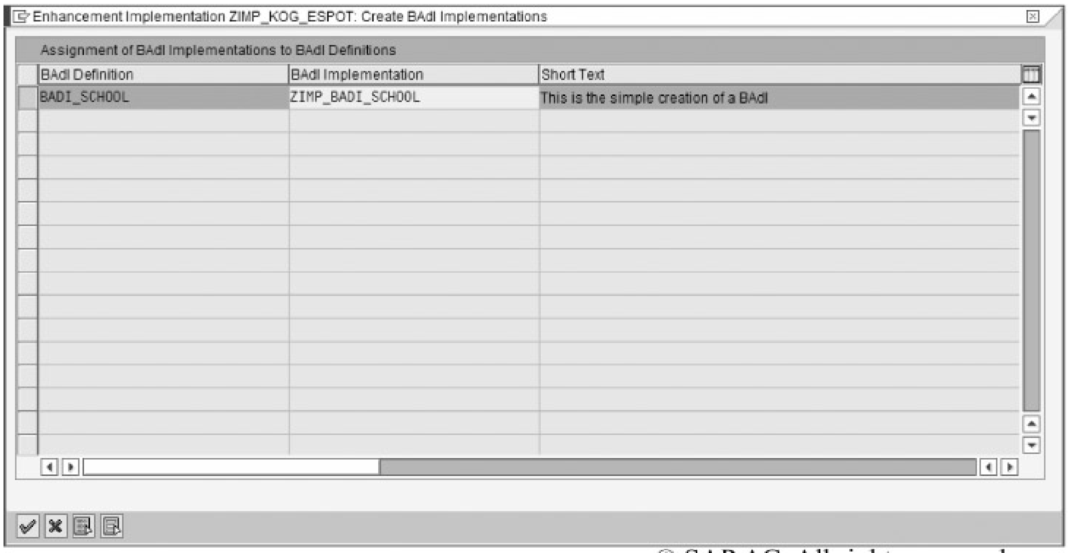


Create Enhancement Implementation

Enhancement Implementation	ZImp_KOG_ESPOT
Short Text	Demo of BADI Implementation
Composite Enhancement Implementation	

© SAP AG. All rights reserved.

Figure 14.25: The create enhancement implementation dialog box



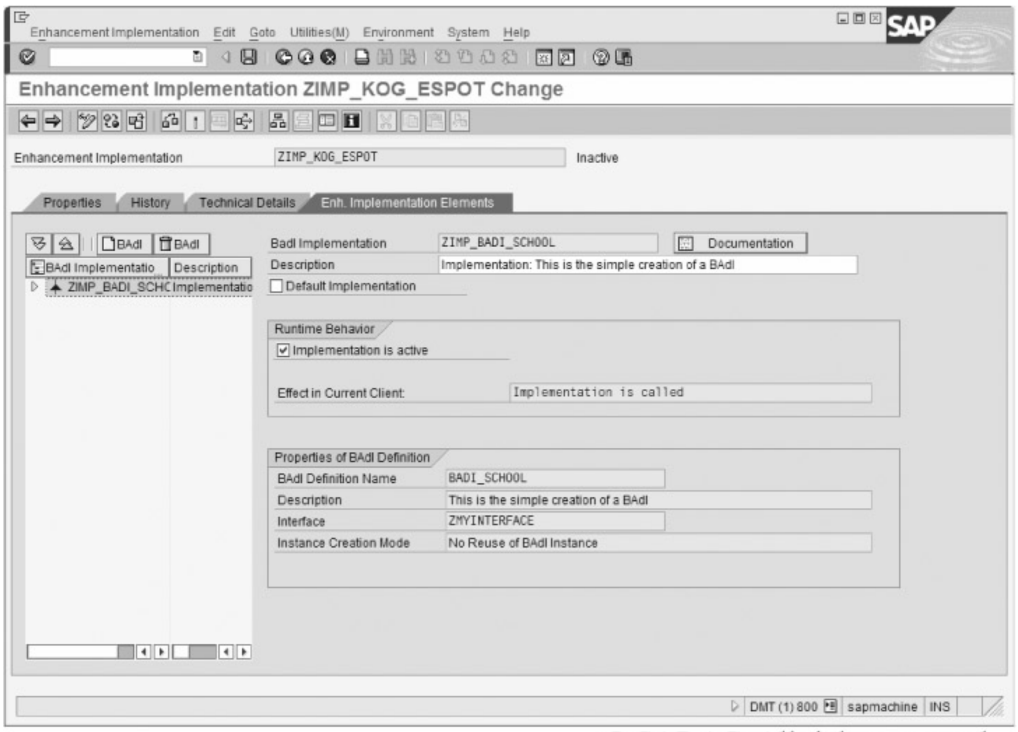
Enhancement Implementation ZIMP_KOG_ESPOT: Create BAdI Implementations

Assignment of BAdI Implementations to BAdI Definitions

BAdI Definition	BAdI Implementation	Short Text
BADI_SCHOOL	ZIMP_BADI_SCHOOL	This is the simple creation of a BAdI

© SAP AG. All rights reserved.

Figure 14.26: The enhancement implementation dialog box


Enhancement Implementation Edit Goto Utilities(M) Environment System Help
Properties
History
Technical Details
Enh. Implementation Elements
BAdI Implementatio
Description
Documentation

Enhancement Implementation ZIMP_KOG_ESPOT Change

Enhancement Implementation: ZIMP_KOG_ESPOT Inactive

ZIMP_BADI_SCHC Implementatio

BadI Implementation: ZIMP_BADI_SCHOOL
Description: Implementation: This is the simple creation of a BAdI
☐ Default Implementation

Runtime Behavior
☑ Implementation is active
Effect in Current Client: Implementation is called

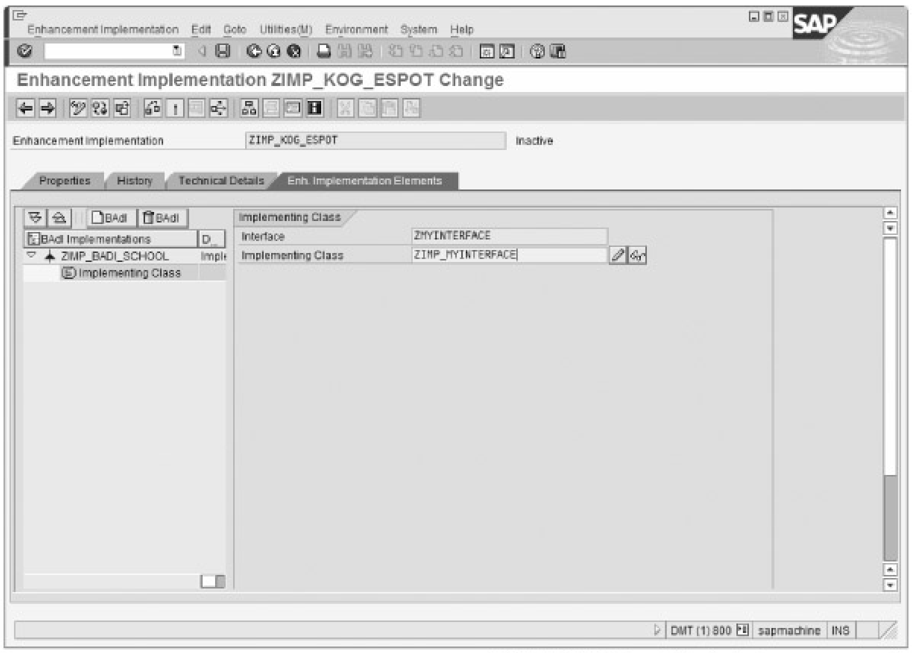
Properties of BAdI Definition

BAdI Definition Name	BADI_SCHOOL
Description	This is the simple creation of a BAdI
Interface	ZMYINTERFACE
Instance Creation Mode	No Reuse of BAdI Instance

DMT (1) 800 | sapmachine | INS

© SAP AG. All rights reserved.

Figure 14.27: The screen for enhancement implementation


Enhancement Implementation Edit Goto Utilities(M) Environment System Help
Properties
History
Technical Details
Enh. Implementation Elements
BAdI Implementations
D..
Impl

Enhancement Implementation ZIMP_KOG_ESPOT Change

Enhancement Implementation: ZIMP_KOG_ESPOT Inactive

ZIMP_BADI_SCHOOL
Implementing Class

Implementing Class
Interface: ZMYINTERFACE
Implementing Class: ZIMP_MYINTERFACE

DMT (1) 800 | sapmachine | INS

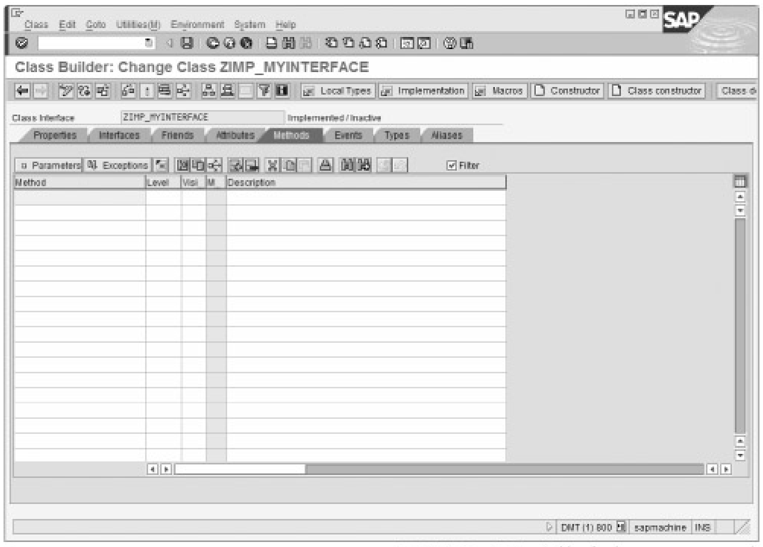
© SAP AG. All rights reserved.

Figure 14.28: Implementing a class

In Figure 14.28, the name for `Implementing Class` is ZIMP_MYINTERFACE. Click the `Change` icon to write the methods in the Class Builder tool for the ZIMP_MYINTERFACE class.

> **Note** The `Display` icon is used to display the methods of the implementing class.

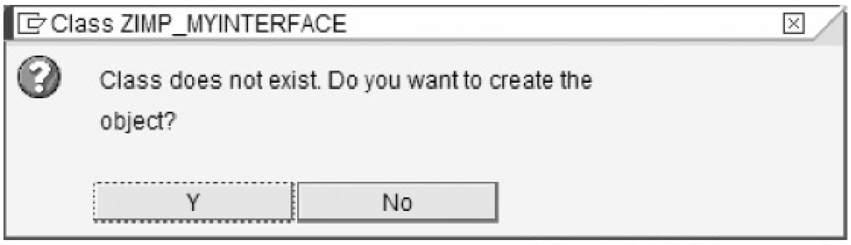
The `Class Builder` tool opens the implementing class in the change mode to define the methods in it. Figure 14.29 shows the `Class Builder` tool for implementing the class:


Class Builder: Change Class ZIMP_MYINTERFACE
Class Interface ZIMP_MYINTERFACE Implemented / Inactive
Properties Interfaces Friends Attributes Methods Events Types Aliases
Method Level Visi M Description

© SAP AG. All rights reserved.

Figure 14.29: The class builder tool for implementing the class

Note that when the BADI implementation is created for the first time, the creation of the class implementation and the enhancement spot implementation is confirmed by the respective dialog boxes. For example, the creation of an implementing class is confirmed in Figure 14.30:


Class ZIMP_MYINTERFACE

Class does not exist. Do you want to create the object?

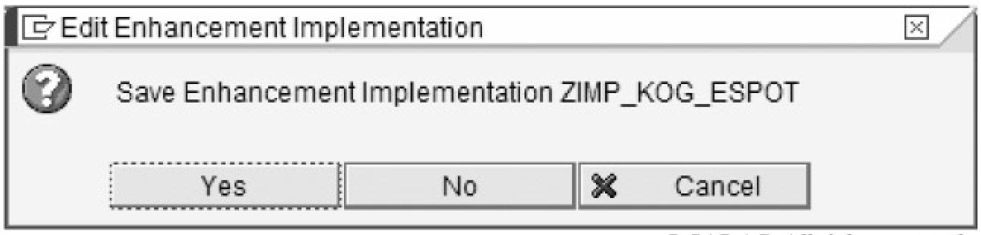
© SAP AG. All rights reserved.

Figure 14.30: Confirming the creation of the implementing class

Click the Y button (see Figure 14.30) to create the implementing class.

Figure 14.31 shows the `Edit Enhancement Implementation` dialog box to save the changes in the enhancement implementation:

Click the `Yes` button (Figure 14.31) to save the changes in the enhancement implementation.


Edit Enhancement Implementation

Save Enhancement Implementation ZIMP_KOG_ESPOT

© SAP AG. All rights reserved.

Figure 14.31: Saving the enhancement implementation

Calling BADIs

In the classic BADIs, when we create a BADI definition, the enhancement management generates an adaptor class that implements the interface created in the class definition. An instance of this adaptor class is created by a factory method. The instance is used to call the methods defined in the adaptor class.

The following ABAP statements are used to call the newly created BADIs:

- `GET BADI`—Selects the BADI implementations based on the filter condition specified in the statement. The BADI implementation classes assigned to the selected implementations are instantiated and passed to a simultaneously created BADI object that serves as a handle for the implementation.
- `CALL BADI`—Uses the reference of a BADI object that passes the call of BADI methods to the BADI handler using object plug-ins.

Differences between Classic and New BADIs

Classic and new BADIs differ on various features, some of which are described in Table 14.1:

Table 14.1: Differences between classic and new BADIs

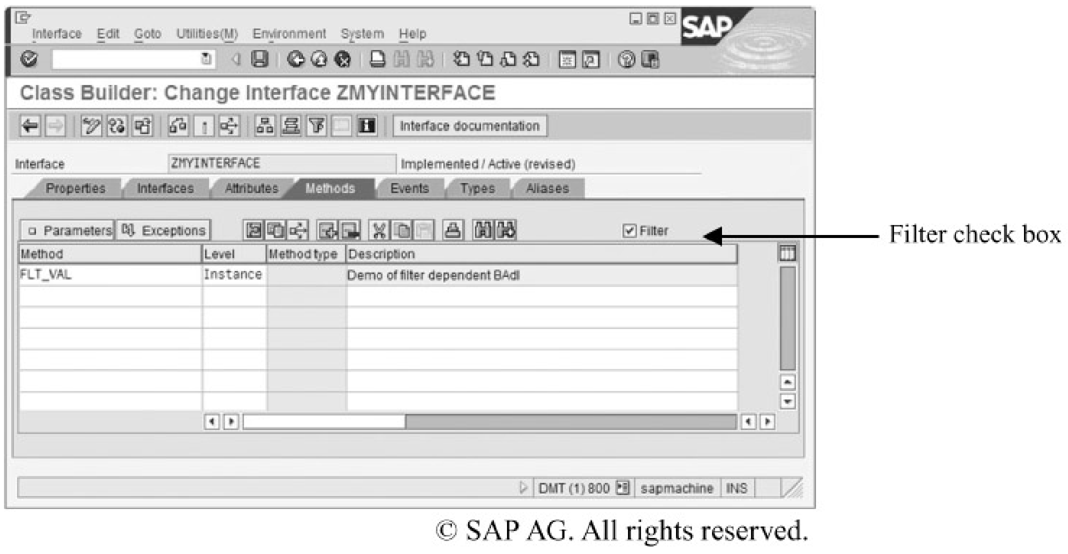
Feature	Classic BADI	New BADI
Object creation	A factory method is used to call a classic BADI object. In addition, a reference variable of the type of BADI interface is used to refer to the BADI object.	The `GET BADI` statement is used to create a new BADI object. In addition, a reference variable of the type of BADI class is used to refer to the BADI object.
Filter values	The filter values are stored in a structure and passed when the BADI methods are called.	The comparison values for the filters are used to search for implementations that are passed when the BADI object is created with the `GET BADI` statement.
BADI calling	A classic BADI can be called only once, and the call positions are registered centrally.	Multiple calls are possible, and the call positions are not registered centrally.

Filter-Dependent BADIs

BADIs may be implemented on the basis of a filter value. For example, if a country-specific enhancement is available in the standard version of SAP, the partner countries can implement this enhancement individually by using filters to create and activate their respective implementations. To implement a BADI on the basis of a filter value, a filter type is entered while defining an enhancement. All the methods created in the enhancement's interface have some filter value as their importing parameter. An application program then provides the filter value for the enhancement method. This method finally selects the active implementation for that value.

To define a filter-dependent BADI perform the following steps:

First, create the definition of the BADI and select the `Filter` checkbox, as shown in Figure 14.32:



Class Builder: Change Interface ZMYINTERFACE

Interface ZMYINTERFACE Implemented / Active (revised)

☑ Filter

Method	Level	Method type	Description
FLT_VAL	Instance		Demo of filter dependent BAdI

DMT (1) 800 sapmachine INS

Figure 14.32: Creating a filter-dependent BADI

Next, specify the data element or structure as a filter type. A data element can be predefined or user-defined. You can use different elements as filter types, such as a country, a material, an object type, and an internal parameter. These data elements also are valid while using structures.

Now create an interface with methods. Note that for each method, the filter value must be defined as the importing parameter so that the application program can provide the filter value to the enhancement method. The enhancement method then selects the active implementation for the specified value. The filter value is always declared by using the `flt_val` parameter, and is predefined in the list of parameters.

Now, when you create an instance of the generated class in the application program and call the corresponding method at the appropriate time, the filter value is passed to the method as an export parameter.

When a filter-dependent BADI is called by using only one filter value, you can use the `SXC_EXIT_CHECK_ACTIVE` function module to check whether an active implementation for this filter value exists.

Note If the filter type is specified as Extendable, you can create implementations for filter values that do not exist any more.

Function Code Enhancements

Menu enhancement has been renamed function code enhancement in Release 7.0 of the SAP R/3 system.

In this release, you do not need to use the `GET BADI` and `CALL BADI` statements of ABAP; the runtime environment of SAP system automatically inserts the implementation of a function code enhancement. Function code enhancements are used for BADIs that are single-use add-ins and not filter-dependent. Nowadays, function code enhancements can be created by using a program interface.

Function code enhancements are provided to customers along with function codes. Note that at the development time, a developer creates a BADI definition that includes a specific function, which is assigned to a menu option in the appropriate menu list by using the Menu Painter tool. In addition, the developer ensures that when such a menu option is searched by the customer, the corresponding BADI method is invoked. On the customer side, function code enhancements are used by creating a new implementation, which includes the text selection for a menu option and the method to apply the implementation. This method depends on the action performed by the menu option and is implemented differently for different menu options. To create a function code enhancement, perform the following steps:

Create a BADI definition and define its interface.

Now, select the `Add Menu Enhancement` option from the context menu, which is displayed when the BADI definition is selected in the `Navigation` menu of the created BADI definitions, as shown in Figure 14.33:

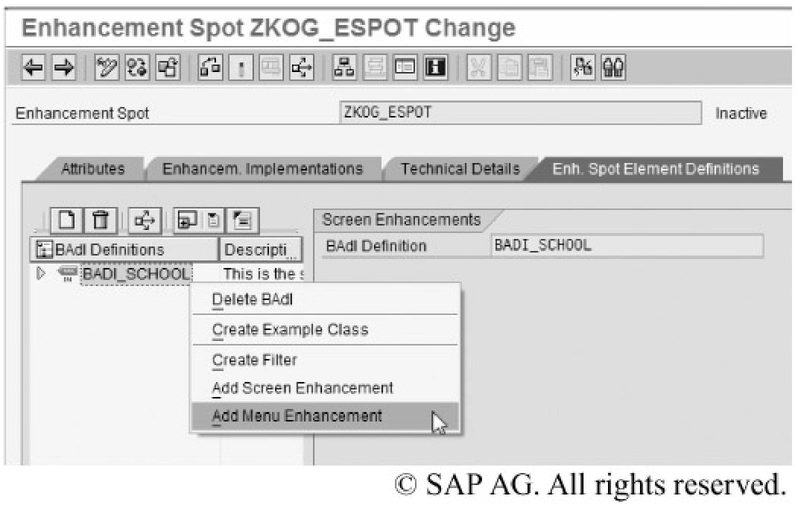
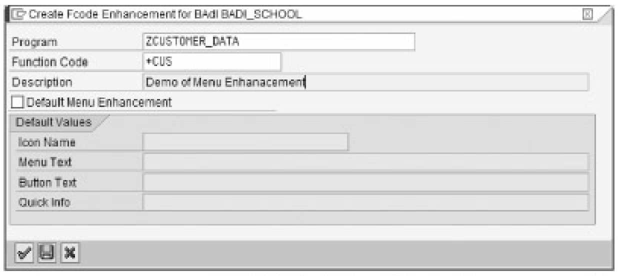


Figure 14.33: Selecting the add menu enhancement option

The `Create Fcode Enhancement for BADI` dialog box appears, as shown in Figure 14.34:


Create Fcode Enhancement for BAdI BADI_SCHOOL

Program	ZCUSTOMER_DATA
Function Code	+CUS
Description	Demo of Menu Enhanacement

☐ Default Menu Enhancement

Default Values

Icon Name	
Menu Text	
Button Text	
Quick Info	

Figure 14.34: The create fcode enhancement for BADI dialog box

Enter the values in the `Program`, `Function Code`, and `Description` fields for the function code enhancements. The following code snippet shows the code to a function code enhancement:

```
(…)
case fcode.
when 'SAP'.
(…)
when '+CUS'
call method …
```

Note Function code enhancements become visible only after the implementation has been activated and the application that calls the BADI has been executed.

Screen Enhancements

Besides programs and function code enhancements, you can create screen enhancements for BADIs. However, screen enhancements are not supported for BADIs designed for multiple use. In the case of screen enhancements, the concept of a class has been adopted with the following exceptions:

- The call to the `CL_EXITHANDLER=>GET_PROG_AND_DYNP_FOR_SUBSCR` method has been replaced by the call to the `CL_ENH_BADI_RUNTIME_FUNCTIONS=>GET_PROG_AND_DYNP_FOR_SUBSCR` method with the same interface.
- The `PUT_DATA_TO_SCREEN` and `GET_DATA_FROM_SCREEN` methods cannot be generated. Users can create their own BADI methods for the data transport and call these BADI methods by using the `CALL BADI` statement.
- Users do not need to call the `CL_EXITHANDLER=>SET_INSTANCE_FOR_SUBSCREENS` and `CL_EXITHANDLER=>GET_INSTANCE_FOR_SUBSCREENS` methods because these methods are now unnecessary because they only place the BADI reference in a temporary storage.

Summary

In this chapter, you have learned about the BADIs, which are one of the enhancement techniques of Enhancement Framework. This chapter also describes Enhancement Framework, the various techniques supported by it, and the structure of BADIs. You have also learned about the definition, the implementation, and the use of the BADIs. In addition, you have learned about the differences between the classic and the new BADIs, and about filter-dependent BADIs. At the end of this chapter, the features of BADIs are explained by describing function code enhancements (previously called menu enhancements) and screen enhancements.