Enhancement Framework

Note

This PDF document contains the selected topic and its subtopics (max. 150) in the selected structure. Subtopics from other structures are not included.

Table of content

1 Enhancement Framework

Goal

The Enhancement Framework enables you to add functionality to standard SAP software without actually changing the original repository objects and to organize these enhancements separately from the enhanced objects.

The basic idea of the Enhancement Framework is to make modification-free enhancements of development objects such as programs, function modules, global classes, and Web Dynpro components. It is the new state-of-the-art technology that SAP recommends to enhance and change SAP programs. These technologies and the new kernel-based BAdIs are now integrated in one framework that has the following features:

- Enhancements of existing development objects on different levels - for example, in an industry solution, in the IT department of the customer, and in a customer's company.
- Better upgrade support
- Switching of enhancements with the Switch Framework
- Support for grouping enhancements and appropriate tool support for documentation.

> **Note**
> An enhancement project needs even more planning than a normal development project because it combines the customer code with the code of the underlying application. Enhancements are changes at a very low technical level and should be performed very carefully so that they do not violate the logic of the program they enhance. It is highly recommended to set up a process that defines who is allowed to design and implement enhancements.

Previous Concepts

Up to now, customers could use the following classic technologies to adapt the software when the options for customizing were insufficient:

- Classic enhancement technologies such as customer exits, user exits, ABAP Dictionary appends, ABAP Dictionary includes, and classic BAdIs.
- Modifications Technology - changes to development objects delivered by SAP with or without the Modification Assistant

Scope and Limitations

The new Enhancement Framework is intended to integrate existing enhancement and modification concepts and addresses recent developments such as Web Dynpro. The classic technology for appends and includes cooperates perfectly with the Enhancement Framework but is not yet integrated in it. If you want to be able to enhance all layers of an application, the classic append technology is still necessary.

Allowing and Forbidding Modifications

To be able to enhance repository objects, the software component they belong to should have the property Modifiable or Not modifiable; enhanceable only. For more information about setting this property, see Setting the System Change Option.

Integration

The main tool for performing enhancements is the Enhancement Builder, which is integrated in the ABAP Workbench.

The enhancements can be switched using the Switch Framework. An enhancement takes effect when the package in which the above enhancement components are defined is assigned to a switch of the Switch Framework and this switch is not deactivated.

To have more control over possible enhancements, you can use Enhancement Append Packages. The concept of enhancement append packages is designed for the implementation of enhancements such as source code enhancements and ABAP Dictionary enhancements.

Enhancement implementations are located in a separate append package that is assigned to the original package so that the package treats the content as part of the original package.

See also:

Enhancement Concept

Enhancement Technologies

Enhancement Builder

2 Enhancement Concept

Definition

The Enhancement Framework provides a technology for enhancing source code units without modifying them. The basic mechanism is to offer the so-called enhancement options in development objects. These enhancement options function like "hooks" where you can attach your enhancements. The enhancement options are part of the development objects which can be enhanced. When you assign an enhancement to an enhancement option, at runtime the enhancement is processed when the control flow reaches the option. At runtime the enhancement behaves as if it belongs to the development object it enhances, while the enhancement as a transport object does not belong to the enhanced object.

The Basic Advantages of Enhancements over Modifications

Enhancements result in far less work during an upgrade compared to modifications. Enhancements are transport objects in their own right that can be stored in packages of their own unlike modifications, which are part of the object they modify. Since during an upgrade all SAP objects are replaced by the new version of these objects, after the upgrade all modifications are gone. This results in customers having to adjust all modifications no matter whether the underlying object has changed or not.

In contrast, customer enhancements as transport objects are never overwritten during an upgrade because they are stored in customer packages in the customer namespace. It may happen that the new version of a development object is no longer compatible with an enhancement, if for example an enhanced object is deleted or a data type used in the enhancement does no longer exist. In these cases, the relevant tools of the Enhancement Framework provide the information which enhancements have to be adjusted and offer tool support to do so.

After an upgrade customers still have to adjust enhancement but these are only the enhancements that are no longer compatible with the new version of the object they enhance, but have to touch every single modification in a system. So enhancements are far more robust during upgrade.

Basic Concepts

The basic concepts in the Enhancement Framework are:

- Enhancement options: positions in repository objects where you can make enhancements. Enhancement options can be explicit and implicit.
- Enhancement spots: containers for explicit enhancement options.
- Enhancement implementation elements: these contain the actual enhancement, for example, the source code to be added.
- Enhancement implementations: containers for both enhancement implementation elements of both implicit and explicit enhancement options.

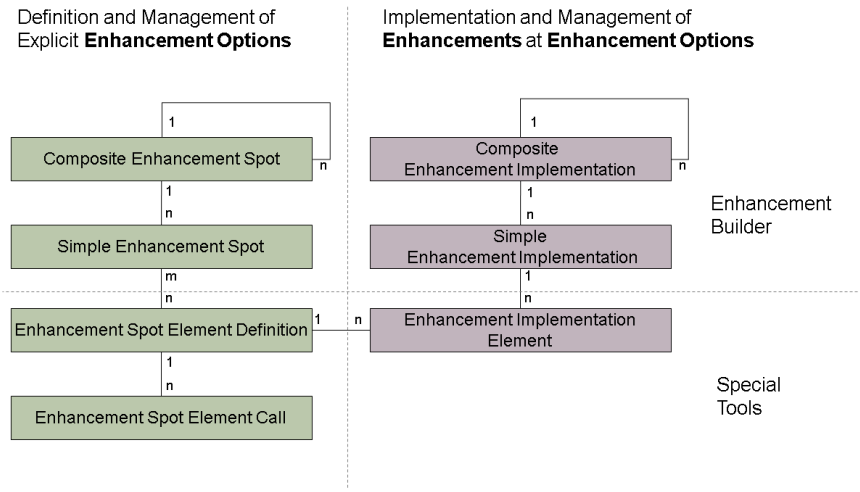
The enhancement options are divided into two classes:

- Implicit enhancement options are provided by the framework and exist without any particular preparation by a developer, they do not have to belong to a container (an enhancement spot).
- Explicit enhancement options have to be inserted explicitly in the source code. They are created in an initial system and must be made known to developers in target systems by means of enhancement spots.

The figure below shows an overview of enhancement spots and enhancement implementations. The left part shows the relevant terminology for enhancement spots and the relationships between them. It only applies to explicit enhancement options. No enhancement spots are required for implicit enhancement options.

The right part shows the relevant terminology for enhancement implementations and the relationships between them. This applies to enhancement implementations of both explicit and implicit enhancement options.

On the implementation side, all implementation elements, regardless of whether they enhance implicit or explicit enhancement options belong to other containers. The containers on the definition side and those on the implementation side are assigned to each other with a particular cardinality.

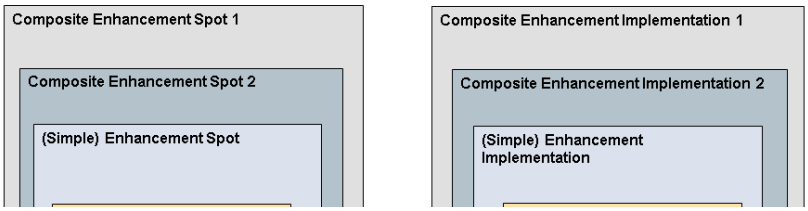


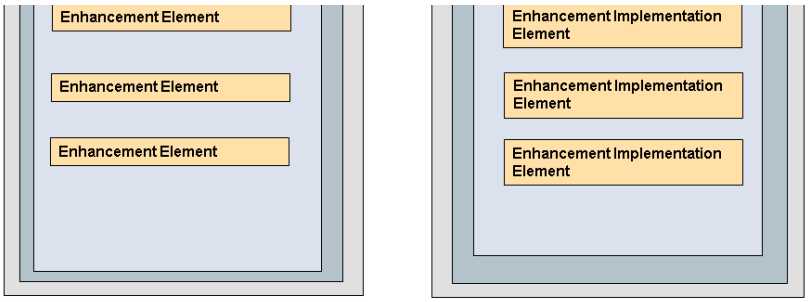
Grouping Enhancements

All explicit enhancement options and all enhancement implementation elements must be part of containers. This is enforced by the tools of the framework. Enhancement options and their implementations must not be grouped together as they belong to different stages of development.

In the Enhancement Framework there are containers for the enhancement options on the definition side and containers for the implementations on the implementation side:

- Simple containers
  These can only contain enhancement options or enhancement implementation elements. These containers cannot be nested. One basic container can only hold elements of one type. For example, a simple container cannot hold BAdIs and enhancement options at the same time.
- Composite containers
  To make room for a nested structure there are composite containers that can hold basic containers and other composite containers. These composite containers can contain basic containers for different enhancement types. Since these composite containers can be nested, you can build a structure that really fulfills the needs of your project. A simple structure looks like this:





When you create an enhancement from scratch, you always have to create the respective containers first. This is similar to creating a method. A method is always part of a class. In the same way, there is no standalone BAdI. Each BAdI is part of an enhancement spot and it is the spot that functions as a transport object. That is why you cannot build a new BAdI and just forget about the framework in the way you might be accustomed to with classic BAdIs. It is also not possible to build the BAdI first and take care of how it fits in the structure of containers later. When building a BAdI, you have to put the BAdI within the relevant structure from the very beginning. But it is only mandatory to have simple containers. Composite enhancements spots and composite enhancement implementations are not enforced by the tools.

> **Note**
> Note that after creating an enhancement spot and a simple enhancement implementation, you have only the containers but you still you have to create an enhancement option or the respective implementation.

Nesting Enhancements

Enhancements can also be nested. An enhancement implementation can also provide (implicit or explicit) enhancements options. As a consequence, an enhancement spot can manage several enhancement options of an enhancement implementation object.

Due to the fact that an enhancement spot combines enhancement options of one single repository object, it is not possible to mix enhancement options of a main object (for example, an ABAP program) and an enhancement implementation in one single enhancement spot object. Similarly, an enhancement implementation can only contain implementation elements for options provided by one repository object, such as a main object or enhancement implementation object. For more information about nesting enhancements, see ABAP Source Code Enhancements.

2.1 Enhancement Options

Definition

Enhancement options are positions in repository objects where enhancements can be made. These options are either explicitly defined (by a developer) or are provided by the framework.

Explicit Enhancement Options

Explicit enhancement options are defined by a developer in a central initial system. Enhancements are made in follow-on systems. Explicit enhancement options can currently be defined by:

- Explicitly flagging source code points or sections in ABAP programs. For an enhancement, these can be enhanced or replaced by source code plug-ins.
- Including Business Add-Ins (BAdIs) in ABAP programs. These programs are then enhanced by object plug-ins with predefined interfaces.

Explicit enhancement options are managed by enhancement spots and enhanced by enhancement implementations.

Implicit Enhancement Options

Implicit enhancement options are provided by the framework. They always exist and do not require enhancement spots. They are also enhanced by enhancement implementations. Implicit enhancement options are:

- Specific options in ABAP programs - such as the end of the program - which can be enhanced by source code plug-ins. There are implicit enhancement points at the beginning and the end of each form, function module, method (of a global or local class) and the end of a report or include and each local structure. As for methods of local classes, you can enhance parameters of all types. There are also implicit options to add something in a defining section of a global class or to add the whole section (for example, a protected section) if it does not exist.
- Parameter interfaces of function modules which can be enhanced with new optional parameters.
- New methods, attributes, and events, which can be added to global classes. Each method of a global class offers pre-, post- and overwrite exits and options to add new optional parameters.
- Enhancement of a Web Dynpro object you have a lot of different implicit enhancement options offered by the framework. There are enhancement options for:
  - UI elements
  - Pre- and post-methods of Web Dynpro methods
  - New methods
  - New navigation paths
  - New actions
  - New attributes
  - New nodes that hold the data for the UI elements.

> **Note**
> You can even add new views to a component as enhancements. Another strong advantage is that you can delete every UI element in the

enhancement mode. This way, the relevant elements are still available as transport objects, but are not compiled at runtime.

2.2 Enhancement Spots

Definition

Enhancement spots are containers for explicit enhancement options and carry information about the positions at which enhancement options were created.

Assigning Enhancement Options to Enhancement Spots

The enhancement spot element definition and the corresponding enhancement spot element calls make up the definition of an explicit enhancement option. For example, when editing an ABAP program with the ABAP Editor, you can define explicit enhancement options in the form of the ENHANCEMENT-POINTstatement, which also represents the element definition and element call.

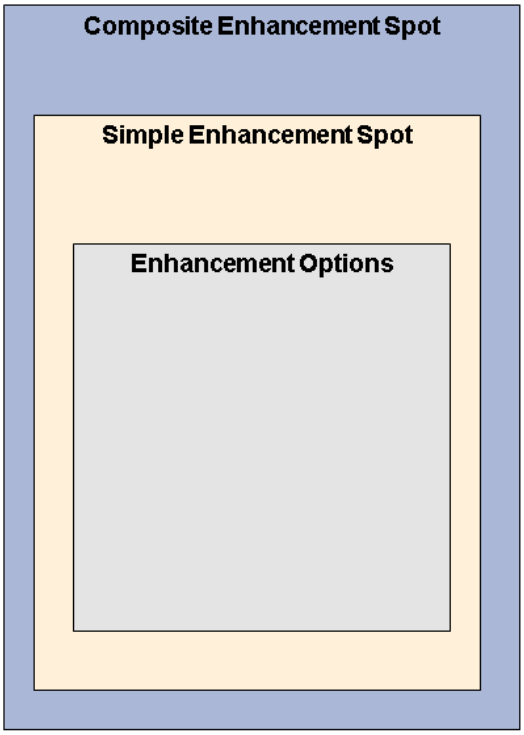
Each enhancement spot element definition must be assigned to at least one enhancement spot.

> **Note**
> Implicit enhancement options do not need to be assigned to enhancement spots.

Simple and Composite Enhancement Spots

Simple and composite enhancement spots are repository objects that form a tree-like structure, where the leaves and branches represent simple and composite enhancement spots respectively. A simple enhancement spot is always assigned to exactly one enhancement technology (ABAP source code enhancement or BAdI).

Composite enhancement spots are used for the semantic grouping of simple enhancement spots. A composite enhancement spot contains either one or more simple enhancement spots and/or one or more composite enhancement spots of the relevant type as described in the figure below.



You can use composite enhancement spots to combine simple enhancement spots into meaningful units.

2.3 Enhancement Implementations

Enhancement Implementation Elements

An enhancement implementation element contains the actual enhancement - for example, a source code plug-in contains the source code to be added. However, an enhancement implementation element contains no information about at which positions of a repository object the enhancement options were created.

You implement an enhancement option by assigning an enhancement implementation element to it. Assigning an enhancement implementation to an enhancement point or section defines an enhancement implementation element. If you implement a BAdI, first an enhancement implementation (a container) must be assigned to the spot that contains the BAdI, then a BAdI implementation can be created within the container that is assigned to the spot that contains the BAdI

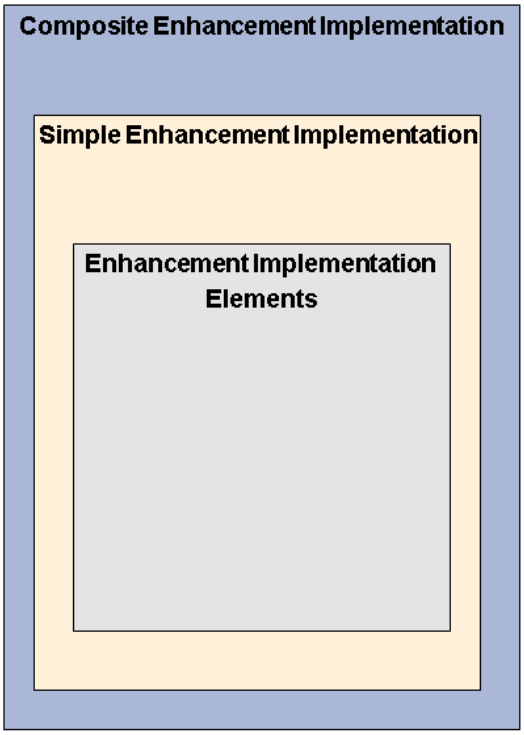
definition.

> **Tip**
> You can assign source code plug-ins, defined between ENHANCEMENT and ENDENHANCEMENT in an ABAP program, to an enhancement option defined using ENHANCEMENT-POINT.

Enhancement Implementations

Simple enhancement implementations are repository objects that serve as containers for enhancement implementation elements. A simple enhancement implementation can contain one or more enhancement implementation elements which are assigned to the enhancement spot element definitions of an enhancement spot.

Composite enhancement implementations are used for the semantic grouping of simple enhancement implementations. A composite enhancement implementation contains either one or more simple enhancement implementations and/or one or more composite enhancement implementations of the relevant type.



Structure of Containers for Enhancement Implementation Elements

Enhancement implementations are processed with the Enhancement Builder, which is integrated in the ABAP Workbench. The hierarchical display of the enhancement implementations in the tool shows the enhancements made in a system.

The state of a Switch Framework switch assigned to the package of an enhancement implementation specifies whether or not an enhancement is executed ina system.

3 Enhancement Technologies

There are different kinds of technologies that provide enhancement options:

- Source code enhancements - enhancement points are positions in the source code where you can attach source code plug-ins which enhance the source code at these positions. While a source code plug-in at an enhancement point is processed in addition to the original code, the code of an enhancement section is substituted by the respective source code plug-in.

  > **Note**
  > Program-bound enhancement implementations in a multiple-use include can only be displayed when the related main programs are generated. The programs, function groups, classes, and so on can be generated within the respective editors. Alternatively, transaction SGEN can be used to perform a mass generation in a system.

- Function group enhancements - you can enhance the interface of a function module by additional parameters using function group enhancements.
- Class enhancements - you can add additional methods, optional parameters, pre- and post-methods to existing methods.
- BAdIs - object-oriented enhancement options. The BAdI defines an interface that can be implemented by classes that are transport objects of their own. The new BAdI is fully integrated into the Enhancement Framework. Within the Enhancement Framework, a BAdI is an enhancement option or an anchor point for an object plug-in.
- Web Dynpro Enhancements - enhancements to individual sections of a Web Dynpro component.

Implicit and Explicit Enhancement Options

Class enhancements, function group enhancements and enhancement options at particular predefined positions (such as the end of a report, a function module, an include or a structure and the beginning and the end of a method) are called implicit enhancement options. They are provided by the framework, only their implementation has to be inserted

If there is a need for additional or different enhancement options than those provided by the framework, you can use explicit enhancement options. They have to be inserted explicitly by the developer. There are two types of explicit enhancement options: BAdIs and explicit enhancement points or sections, where you can insert source code plug-ins.

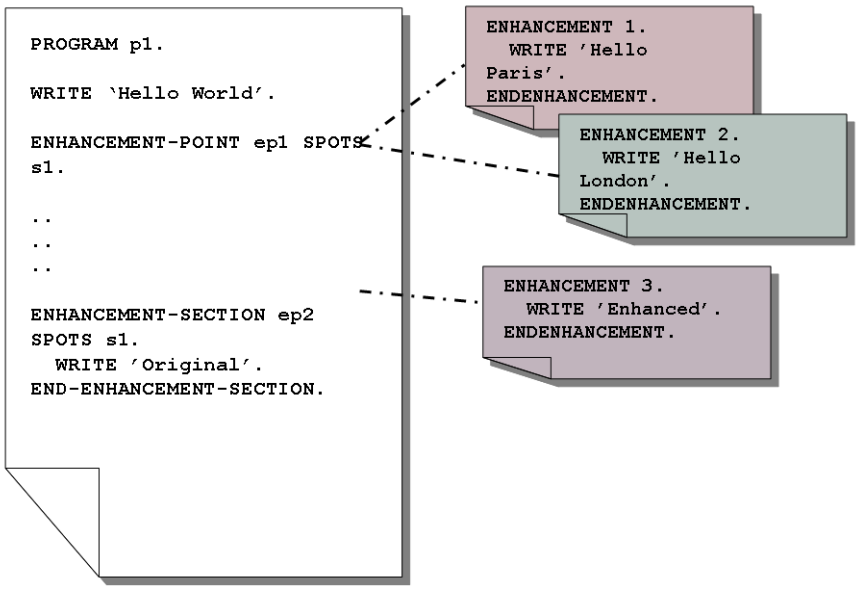
3.1 ABAP Source Code Enhancements

Definition

As a part of the enhancement concept, it is possible to enhance ABAP source code without changing the actual code, simply by adding enhancements.

Source Code Plug-in Technology

Although source code plug-ins are displayed in the same source code as the respective enhancement options, the plug-ins are stored in other include programs managed by the Enhancement Builder:



Types of Source Code Enhancements

The enhancement options in ABAP source code can be explicit and implicit:

- Implicit enhancement options always exist and do not need to be assigned to an enhancement spot. More information: Implicit Enhancement Options in ABAP Source Code.
- Explicit enhancement options are defined by developers and are the places where partners and customers can implement their source-code plug-ins or overlay enhancements. More information: Explicit Enhancement Options in ABAP Source Code.

A source code plug-in defined between **ENHANCEMENT** - **ENDENHANCEMENT** can be enhanced by additional source code plug-ins. This means that the **ENHANCEMENT-POINT** and **ENHANCEMENT-SECTION** statements can be listed in a source code plug-in. In addition, implicit enhancement options are now available before the first and after the last line of a source text plug-in (after **ENHANCEMENT** and before **ENDENHANCEMENT**).

More information about the usage of source code enhancements:

- Implicit Enhancement Options in ABAP Source Codes
- Explicit Enhancement Options in ABAP Source Codes
- Creating Source Code Plug-Ins

3.1.1 Implicit Enhancement Options in ABAP Source Code

Implicit enhancement options always exist and they are not assigned to an enhancement spot.

> **Note**
> The implicit enhancement options can be displayed in the ABAP Editor by following the path: Edit → Enhancement Operations → Show Implicit Enhancement Options, and then enhanced using source code plug-ins.

In ABAP programs, implicit enhancement options are predefined at the following places:

- At the end of an include. There are some restrictions, for example, not at the end of a method include.

- At the end of a `PUBLIC-`, `PROTECTED-`, `PRIVATE-SECTION` of a local class.
- At the end of the implementation part of a class (before the `ENDCLASS`, which belongs to `CLASS … IMPLEMENTATION`).
- At the end of an interface definition (before the `ENDINTERFACE`).
- At the end of a structure definition (before `TYPES END OF`, `DATA END OF`, `CONSTANTS END OF`, and `STATICS END OF`).
- At the beginning and at the end of a procedure (`FORM`, `FUNCTION`, `METHOD`). That is, after commands `FORM`, `FUNCTION`, and `METHOD`, and before statements `ENDFORM`, `ENDFUNCTION`, and `ENDMETHOD`.
- At the end of the `CHANGING-`, `IMPORTING-`, `EXPORTING-` parameter list of a method in a local class. These enhancement options are located in the middle of a statement.
- Before the first and after the last line of a source text plug-in (after ENHANCEMENT and before ENDENHANCEMENT).

3.1.2 Explicit Enhancement Options in ABAP Source Code

Definition

In ABAP programs, you can select either a position or a program section as an explicit enhancement option. Source code plug-ins for an enhancement are either entered at such a position or they replace the selected section.

Activities

To create an explicit enhancement option in the ABAP source code, proceed as follows:

1. In the ABAP Editor, open the program you want to edit.
2. Switch to change mode.
3. In the source code, right click and choose Enhancement Operations → Create Option from the context menu.
   The Create Enhancement Option dialog box appears.
4. Choose between ENHANCEMENT-POINT and ENHANCEMENT-SECTION and specify a name for the type of enhancement option you have selected.
5. Choose whether to include the enhancement option in the source code as a conditional or as an unconditional call. For more information, use F1 to open the ABAP Keyword Help.
6. Create a new enhancement spot where to store your new enhancement option or choose an existing spot.
7. Choose Enter.

> **Note**
> After the program has been saved, the enhancement option is managed by the Enhancement Builder, and can only be deleted using the function Enhancements → Remove Enhancement from the context menu, while your cursor is positioned on the enhancement option.

Declaring Explicit Enhancement Options

A position in an ABAP program marked as an explicit enhancement option looks like that:

```
ENHANCEMENT-POINT <name> SPOTS <spot1> [<spot2>] [STATIC] ...
```

> **Note**
> The **STATIC** addition is intended for the enhancement of data declarations, while the statement **ENHANCEMENT-POINT** without the **STATIC** addition is designed for the enhancement of executable coding.

- Static enhancement statement - for example, additional data declaration
- Dynamic enhancement statement - for example, additional source code

A section in an ABAP program marked as an explicit enhancement option looks like that:

```
ENHANCEMENT-SECTION <name> SPOTS <spot1> [<spot2>] [STATIC]
...
END-ENHANCEMENT-SECTION.
```

- Static enhancement statement - for example, replace an existing data declaration
- Dynamic enhancement statement - for example, replace existing source code.

> **Note**
> In contrast to the statement **ENHANCEMENT-POINT**, the addition **STATIC** of the statement **ENHANCEMENT-SECTION** can only be used with maximum caution for changes of data declarations, because a replacement and no completion is carried out.

Data declarations are always static, even if they are inside a dynamic enhancement option. Form routines, methods, and local classes cannot be part of dynamic enhancement points and sections. You need to place them in static enhancement points or sections.

During program generation, the source code plug-ins that are in the current system for the assigned enhancement implementations and have the switch status stand-by or on, are included at this position or they replace the selected section.

3.1.3 Creating Source Code Plug-Ins

Procedure

1. Open the ABAP source with explicit or implicit enhancement options in the ABAP Editor.
   More information about how to create explicit enhancement options: Explicit Enhancement Options in ABAP Source Code.
2. Switch to enhancement mode by using the Enhance pushbutton from the toolbar.

   > **Note**
   > To exit the enhance mode, choose the Display <-> Changepushbutton.

3. Position your cursor on the explicit or implicit enhancement option and choose Enhancement Implementation →Create Implementationfrom the context menu.
   The Select or Create Enhancement Implementation dialog box appears. You can choose an existing implementation or create a new one.
4. If you choose to create a new enhancement implementation, an empty source code plug-in is automatically created. It has a unique ID and is displayed in the ABAP Editor below the enhancement option as follows:

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

5. Implement the enhancement between the `ENHANCEMENT` and `ENDENHANCEMENT` lines.
   A source code plug-in is assigned to exactly one enhancement option (the container) but an enhancement option can be enhanced through several source code plug-ins.

   > **Note**
   > Although source code plug-ins are displayed in the same source code as the respective enhancement options, they are stored in other include programs managed by the Enhancement Builder.

3.2 Function Module Enhancements

The function module enhancements can be separated into:

- Source code enhancements - Enhancements to the source code are carried out by means of ABAP source code enhancements.
- Parameter interface enhancements - You can enhance the parameter interface of a function module with new, optional formal parameters as enhancement implementation elements. For more information: Enhancing Parameter Interfaces.

3.2.1 Enhancing Parameter Interfaces

Use

In the context of the enhancement concept, the parameter interface of a function module provides the necessary enhancement options. You can enhance the parameter interface with new optional formal parameters, which function as enhancement implementation elements. Enhancing by adding new exceptions is not possible.

Procedure

To implement an enhancement of the interface, use the Function Builder to call the Enhancement Builder:

1. In the Function Builder, choose the tab of the parameter type for which you want to insert a formal parameter.
2. Switch to enhancement mode by choosing Function Module → Enhance.
   A dialog box appears.
3. Select an existing enhancement implementation or create a new one.
   This takes you to the enhancement mode of the Function Builder in which the original components of the function module are displayed and cannot be changed.
4. Insert the new formal parameter.
5. Save and activate the function module.
   The new parameter can now be used at implicit and explicit enhancement options in enhancements to the source code of the function module. This can be done using ABAP source code enhancements.

3.3 Enhancements to Classes and Interfaces

Definition

The enhancements to global classes and interfaces can be separated into:

- Enhancements to the source code of methods, local classes, and so on - these enhancements are carried out by means of ABAP Source Code Enhancements.

> **Note**
> No source code enhancements are allowed within interface sections and class sections. The only exception are implicit enhancement options at the end of types.

- Enhancements to components of classes and interfaces

Enhancements to Components of Classes and Interfaces

You can enhance the components of a global class or global interface by:

- Inserting new attributes
- Inserting new optional formal parameters to existing methods (but no new exceptions)
- Adding the implementation of a `pre` and/or `post` method for an existing method of a class. A `pre-method` is called directly after the call of the existing method before the first statement. A `post-method` is called after the last statement of the existing method before ENDMETHOD (only if the method is exited using ENDMETHOD).
  - The chain `pre-method` → method`xyz` →`post-method` can be interrupted at runtime (parts are not executed) under the following circumstances:
  - If an exception is raised within the pre-method, the method `xyz` and the post-method are not executed.
  - If an exception is thrown in the method `xyz`, the post-method is not executed.

Starting with SAP NetWeaver 7.0 Enhancement Package 1 and SAP NetWeaver 7.1 Enhancement Package 1, statements like `CHECK`, `EXIT` and `RETURN` used within the method `xyz` do not stop the post-method from being executed. For more information, see SAP Note 1083387.

- Adding the implementation of an overwrite-method for an existing method of a class. The creation/deletion is similar to the `pre` or `post` methods. However, an `overwrite` method is executed instead of the original method. When an `overwrite` method is created, it is not allowed to have `pre` or `post` methods for the same original method.
- Inserting new methods.

More information:

Enhancing the Components of Classes or Interfaces.

3.3.1 Enhancing the Components of Global Classes or Interfaces

Use

Use this procedure to implement enhancements (`pre`, `post`, and `overwrite` methods) of global classes or interfaces. In the context of the enhancement concept, a global class or global interface provides implicit enhancement options.

Procedure

To implement an enhancement of global classes or interfaces, use the Class Builder to call the Enhancement Builder.

1. In the form-based mode of the Class Builder, choose Class → Enhance or Interface → Enhance for an existing class or an interface.
   A dialog box appears.
2. Select an existing enhancement implementation or create a new one
   This takes you to the enhancement mode of the Class Builder in which the original components of the class or interface are displayed and cannot be changed.
3. Create your enhancements following the steps below:
   1. Create the new attributes in the same way as when you create a class or interface.
   2. Insert new, optional formal parameters in the same way as when you create a method.
   3. Insert or delete pre- and/or post-methods:
      1. Select the desired method
      2. Choose Edit → Enhancement Operations → Enhancement Operations and then choose one of the menu entries:
      3. Insert Pre-Method, Insert Post-Methodor Insert Overwrite-Method
      4. Delete Pre-Method, Delete Post-Method or Delete Overwrite-Method.

      > **Note**
      > An overwrite-method cannot exist simultaneously with pre- or post-methods for the same method.

   4. Insert new methods in the same way as when you create a class or interface.
   5. Insert new interfaces.
   6. Insert new events.
   7. Insert new types
4. Save and activate the class.
   You can further use the new components for implicit and explicit enhancement options in enhancements to the source code of the class. This can be done via ABAP Source Code Enhancements.

Additional Information

When `pre`/`post`/`overwrite` methods are created, a local class named `lcl_<enhancement_name>` is generated for the enhancement `<enhancement_name>`. This local class is attached to the end of the local class implementation section in the original class (as an implementation of the predefined enhancement option).

You can access the components of the original class within `lcl_<enhancement_name>` by the object reference `CORE_OBJECT`.

The local class `lcl_<enhancement_name>` can implement three interfaces:

- `IPR_<enhancement_name>` for `pre` methods.

- `IPO_<enhancement_name>` for `post` methods.
- `IOW_<enhancement_name>` for `overwrite` methods.

The parameters of `pre`/`post`/`overwrite` methods are similar to the parameters of the original method but the following restrictions exist:

- The `pre` method does not have `export` parameters.
- The `post` method does not have `export` parameters; the `export` parameters become `changing` parameters.
- The `returning` parameter of a functional method becomes a `changing` parameter
- `overwrite` methods have the same signature as the original method.
- The parameter definitions of `pre`/`post`/`overwrite` methods cannot be changed.

3.4 Business Add-Ins (BAdIs)

Definition

A BAdI is an object-oriented enhancement option, a hook for an object plug-in and thus the most sophisticated enhancement type. The main characteristic of a BAdI is that it provides a mechanism to change the functionality of a well-defined business function (e.g. a BAPI) without making changes to the delivered source code. Future upgrades of the original business function can be applied without losing the customer-specific enhancements or the need to merge the changes.

When defining a BAdI, you determine its interface - the methods offered by the BAdI. BAdI implementations are classes that implement the BAdI interface. Typically, these implementations are created in another development layer; for example, as objects owned by the customer that is in the customer name space, while the BAdI definition and the call of the BAdI methods belong to the SAP namespace. Calling a BAdI method resembles a dynamic method call with a specified interface. Which methods are actually called is determined by the active BAdI implementation. In contrast to a dynamic method call, the relevant BAdI implementations are selected at compile time.

The BAdI technology is not limited to SAP applications. BAdI calls can be integrated in customer applications, which in turn can then be enhanced by other customer applications.

Reimplementation of the BAdI Technology

The new BAdI concept takes advantage of SAP's extensive experience in offering customers different ways to enhance the standard SAP system. The new BAdIs add some major improvements to the classic BAdIs such as better performance. The new BAdIs are integrated in the kernel and are switchable.

The reimplementation of BAdIs has two main goals:

- By integrating the BAdIs in the ABAP programming language through the new language elements `GET BADI` and `CALL BADI`, their performance is considerably enhanced.
- The new BAdIs provide considerably more flexibility in the conversion of predefined enhancement options through new properties such as contexts and more filtering options than classic BAdIs.

The new ABAP language elements and their additions ensure that these additional options can be easily used in ABAP programs.

> **Note**
> Within the Enhancement Framework, a new BAdI is always meant when the term BAdI is used. If there is an explicit reference to the previous BAdI concept, such BAdIs are referred to as classic BAdIs.

Examples

The following examples explain some of the most common problems that BAdIs solve:

- Standard software - the customer wants specific changes but modifications cause problems.
  The solution is a BAdI which is a type of enhancement with a well-defined interface; BAdIs are more robust to changes than source code plug-ins.
- Different country and industry solutions need their own specific solution of a specified problem.
  The solution is to create a BAdI definition in the core and the different countries can add their own implementations.
- You want to add and later choose parts of code dynamically.
  The solution is to create an internal BAdI and add the different parts in different implementations. The BAdI and its implementations may belong to different software layers.
- As an SAP developer, you have a program in the SAP standard and want to give customers the chance to add some specific code of their own.
  The solution is to include BAdI calls in the standard software. The customer can add the implementations of these objects later.
- You have a program which needs specific implementations for different countries.
  The solution is to put the country-specific parts in BAdI implementations. These implementations can be overwritten later for the respective countries without modifying the original program that contains the BAdI calls.
- You want to program a registry.
  It is possible to use the BAdI infrastructure to build a high-performance registry. In that case, the filter is used as selection criteria for the registry.

More information:

BAdIs - Documentation

BAdIs - Migration of Classic BAdIs

BAdIs - More Information

3.4.1 Step-By-Step Examples with BAdIs

This section presents several step-by-step examples that aim to help you easily navigate and work with BAdIs in the Enhancement Builder:

3.4.1.1 Building Your First BAdI

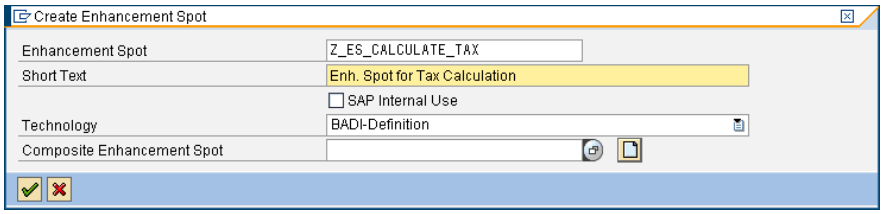
Use Case

A developer needs to calculate the VAT of different ledger entries. The entries have to be passed to a method that calculates and returns the VAT. Since the developer does not know what the VAT in a specific country is, he or she defines a BAdI with the method `get_vat`. This method must return the VAT for a particular value that is passed to the method `get_vat`.

Creating an Enhancement Spot

The first thing you need when creating a BAdI is a container for the BAdI. For that purpose, you need to create a (simple) enhancement spot. This is the container in which you develop your BAdI.

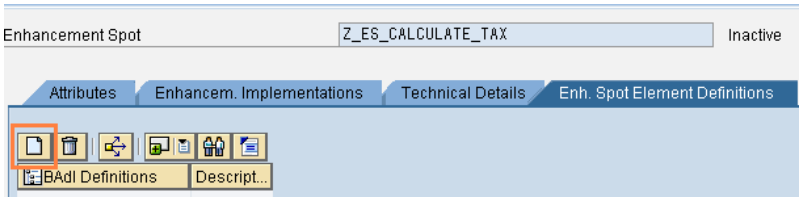
1. In the Object Navigator (transaction SE80), navigate to the package, in which you want to create the enhancement spot.
2. In the context menu of the package, choose Create → Enhancement → Enhancement Spot.
   The following dialog appears:



3. Enter a name and a short text description for the enhancement spot.

Creating a BAdI

1. To create a BAdI within the new enhancement spot, choose the Create BAdI pushbutton on the left, as shown in the figure below:

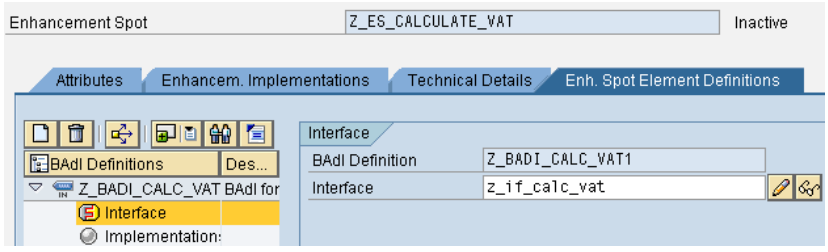


2. In the dialog window that appears, enter the BAdI name `z_badi_calc_vat` and a short description.
   You now have a BAdI in your enhancement spot.
3. Deselect the Multiple Use option because for the current calculation you need a single-use BAdI:

The BAdI Interface

Up to now, you still do not have a BAdI. You need an interface where you can define the methods which determine what you can do with your BAdI.

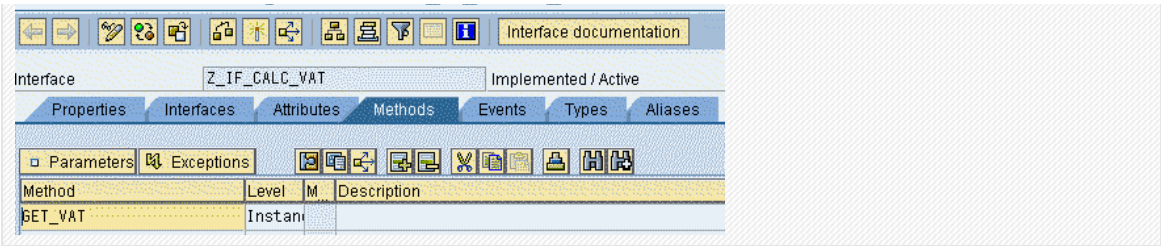
1. Choose the arrow in front of the BAdI name.
2. Double-click on the Interface icon.
   Enter a name for the interface. You can choose an existing name or create a new one.



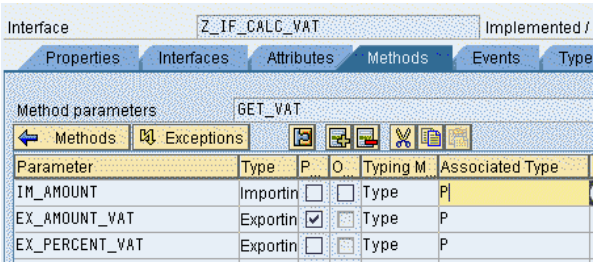
3. Choose the Change pushbutton.
   This leads you to the Class Builder where you can create the methods you need for your BAdI. You simply need to type in the name of the method `get_vat` and enter the parameters you need.

> **Note**
> A BAdI interface has to implement the interface `if_badi_interface`. But if you create a BAdI interface in the way shown above, this marker interface is already integrated in the BAdI interface.

Class Builder: Change Interface Z_IF_CALC_VAT



4. Determine the parameters of the method:



5. Save and activate the interface and the spot.

By completing the steps above, you have created an enhancement spot and a BAdI with an interface. The interface has one method so far.

However, just building a BAdI is not enough as it does not do anything. You need a BAdI instance and this instance must be called somewhere in the code. Since the BAdI only defines an interface, you need a class that implements this interface.

> **Note**
> A BAdI definition is the place to insert an object plug-in that does something at runtime. You still need an object that is plugged in to get something done.

Writing the Source Code

Now you need to write some ABAP code to use the BAdI. You need a variable that can refer to the BAdI and some variables that are given as actual parameters to the BAdI method.

Next, you need to create a handle for that BAdI and call the BAdI method `get_vat`. The respective commands are `GET BADI` and `CALL BADI`.

```
DATA: handle TYPE REF TO z_badi_calc_vat,
sum TYPE p,
vat TYPE p,
percent TYPE p.
sum = 50.
GET BADI handle.
CALL BADI handle->get_vat
EXPORTING im_amount = sum
IMPORTING ex_amount_vat = vat
ex_percent_vat = percent.
WRITE: 'percentage:', percent, 'VAT:', vat.
```

A Fallback Class

If you run the program at this stage, it dumps. This is because it is mandatory to have exactly one active implementation for a single-use BAdI. You can handle this error by catching the respective exception `cx_badi_not_implemented`.

The better solution is to use a fallback class. This class is used if there is no active BAdI implementation.

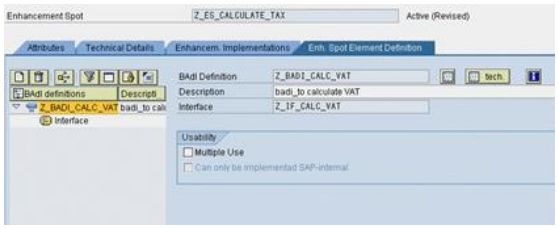
The `GET BADI` command returns a handle to an instance of the fallback class and the respective `CALL BADI` calls the methods of the fallback class instance. As soon as there is an active BAdI implementation, the fallback class is no longer used at runtime.

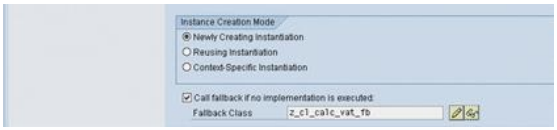
Using a fallback class serves two functions at once:

- The program runs with a single-use BAdI without throwing an exception.
- It is guaranteed that the fallback class is not used any more as soon as a BAdI implementation is supplied. The fallback class is selected conditionally. This is important because in most cases the BAdI provider does not know the details of some process. This is why a BAdI is used.

To add a fallback class, proceed as follows:

1. Select the Call fallback class if no implementation is executed option and enter a name for the class:





2. Choose Change to go back to the Class Builder.
   The respective method of the BAdI interface is already automatically defined. You only have to implement it:
   ```
   DATA: percent TYPE p VALUE 20.
   ex_amount_vat = im_amount * percent / 100.
   ex_percent_vat = percent.
   ```
3. Save and activate the class.
4. Navigate back to the enhancement spot and activate it.
5. Run the program.
   The result should be:
   percentage: 20 VAT: 10.

Next step is: How to Implement a BAdI and Use a Filter

3.4.1.2 How to Implement a BAdI

In the previous example (Building Your First BAdI) you defined a BAdI, provided a fallback class, instantiated the BAdI and called a BAdI method. You also provided an enhancement spot to serve as a container for the BAdI.

As you already know, the fallback class is chosen in case no BAdI implementation is available. As you have not created a BAdI implementation so far, the method implementation of the fallback class is used in your example code.

In this section you will learn how to create a BAdI implementation. As soon as there is a suitable BAdI implementation, the methods of the fallback class are not used any longer.

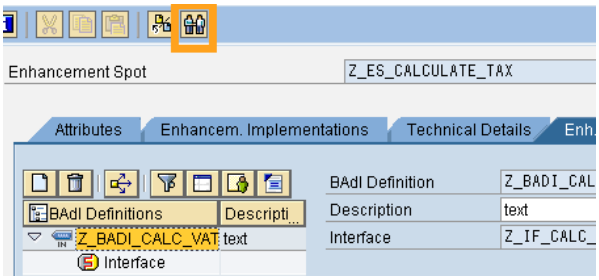
The entries you have so far are:

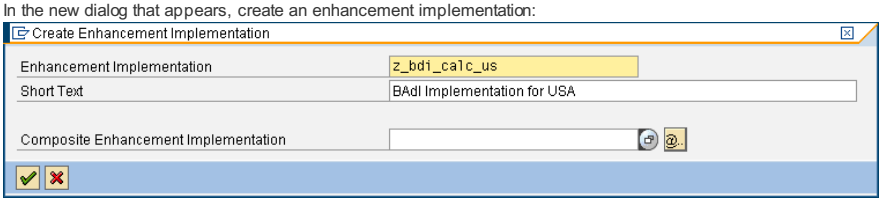
- Enhancement spot `z_es_calc_tax`, in which the BAdI `z_badi_calc_vat` lives.
- Interface `z_if_calc_vat` with one method `get_vat()`.
  The BAdI interface defines an important part of the BAdI identity. It defines the BAdI methods that can be used.

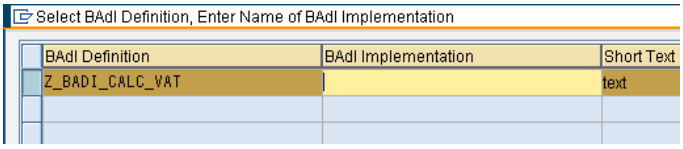
Create a Container for the Implementation

The relevant container type for BAdI implementations is called an enhancement implementation. A simple enhancement implementation can keep many different BAdI implementations, but with one restriction: a simple enhancement implementation is uniquely assigned to an enhancement spot. That is, a (simple) enhancement implementation can keep only BAdI implementations of BAdIs that belong to the spot the simple enhancement implementation is assigned to. Therefore, a (simple) enhancement implementation cannot keep BAdI implementations that implement BAdIs belonging to different spots.

To create a container that is uniquely assigned to the enhancement spot to which your BAdI belongs, proceed as follows:

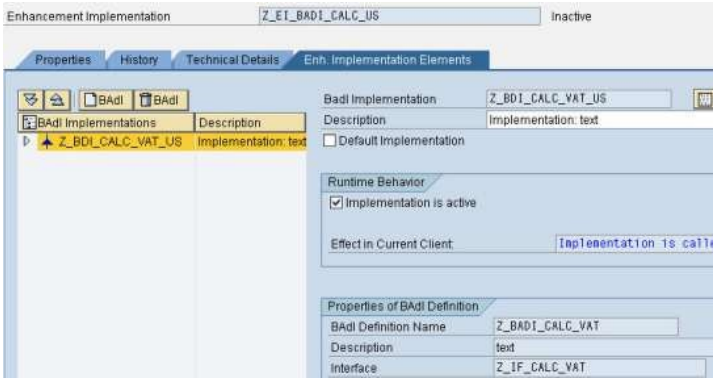
1. In the Object Navigator (transaction SE80), open the enhancement spot you have already created and choose the Create Enhancement Implementation pushbutton.
   

2. In the new dialog that appears, create an enhancement implementation:
   

3. The following new window opens:
   

   What are you asked to enter here?
   You have created so far a container for BAdI implementations- an enhancement implementation. This container is uniquely assigned to your enhancement spot. Once this connection is established, you can create a BAdI implementation for the BAdI within the enhancement spot. Since you have defined only one BAdI within the enhancement spot, you have no choice. If you had more BAdIs in your enhancement spot, this will be the point where you would select which BAdI you want to implement.
4. Enter `z_bdi_calc_vat_us` as the name of the BAdI implementation, confirm and save the enhancement spot in the next figure, which shows a (simple)

enhancement implementation that contains the BAdI implementation `z_bdi_calc_vat_us`:



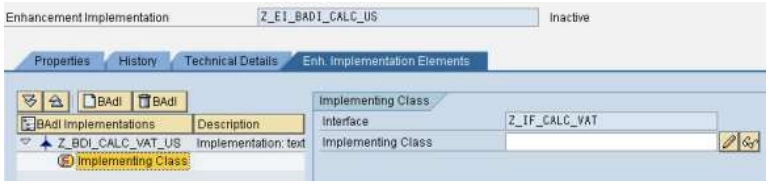
The appearance of a (simple) enhancement implementation in the tool is pretty much like the one of an enhancement spot. Under the Enh. Implementation Elements tab there is a tree with the BAdI implementation(s) contained on the right-hand side. On the left, you see the properties of the marked BAdI implementation.

> **Note**
> Select the property Implementation is active in the Runtime Behavior pane in the figure above. If you do this, the text below changes to Implementation is called. This is intended to help you understand what the selection you have just made is for. Further below, there is a list that shows the properties of the BAdI definition your BAdI implementation is assigned to.

The Implementing Class

Now that you have a BAdI implementation, you need an implementing class.

1. Choose the triangle in front of the name of the BAdI implementation in the tree.
2. In the Implementing Class field, enter `z_cl_calc_vat_us` and choose the Change pushbutton.



The Class Builder opens. The relevant interface methods are already defined there. In your case it is only the method `get_vat()`.

3. Implement the method using the following source snippet:

```
DATA: percent type p value 4 .
ex_amount_vat = im_amount * percent / 100 .
ex_percent_vat = percent .
```

4. Save and activate the class.
5. Return to the program and enter the following code:

```
DATA: handle TYPE REF TO z_badi_calc_vat,
sum TYPE p,
vat TYPE p,
percent TYPE p.
sum = 50.
GET BADI handle.
CALL BADI handle->get_vat
EXPORTING im_amount = sum
IMPORTING ex_amount_vat = vat
ex_percent_vat = percent.
WRITE: 'percentage:', percent, 'VAT:', vat.
```

6. Run the program.
   The result is a percentage of four percent. This is because the fallback class is not selected when there is an active BAdI implementation.

Creating a Second BAdI Implementation

1. Create another BAdI implementation, this time with the VAT rate of Great Britain.
   To make your example more like real world programming, you create another enhancement implementation, that is, another container. This is because implementing the taxes for different countries most probably belongs to different projects and because the structure of the (simple) enhancement implementations must mirror the project structure.
2. Navigate to your enhancement spot and use the same button as you have done above. Name the BAdI implementation `z_bdi_calc_vat_gb`, and the implementing class `Z_CL_CALC_VAT_GB`. The implementation of the method `get_vat` is the same as for USAexcept for the VAT rate, which you assume to be 16.5%.
3. After saving and activating the enhancement implementation and the class, return to the program and run it again.
   This time you get a short dump with the exception `cx_badi_multiply_implemented`.

> **Note**
> You have defined a single-use BAdI by deselecting the Multiple Use option. When instantiating a single-use BAdI, you have to make sure that only one implementation comes into effect at runtime.

What you need now is a way to select among different BAdI implementations. That can be easily done with filters. For more information, see How to Use Filters.

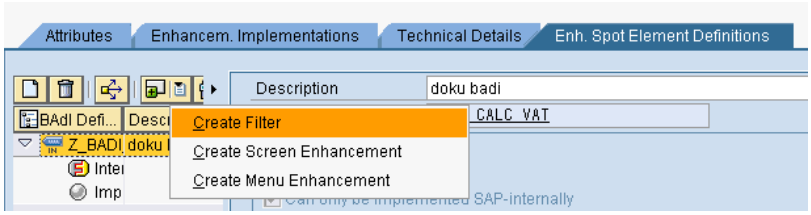
3.4.1.3 How to Use Filters

Recalling from the previous step (How to Implement a BAdI), you need a way to select between different BAdI implementations. This is where you need the BAdI filter.

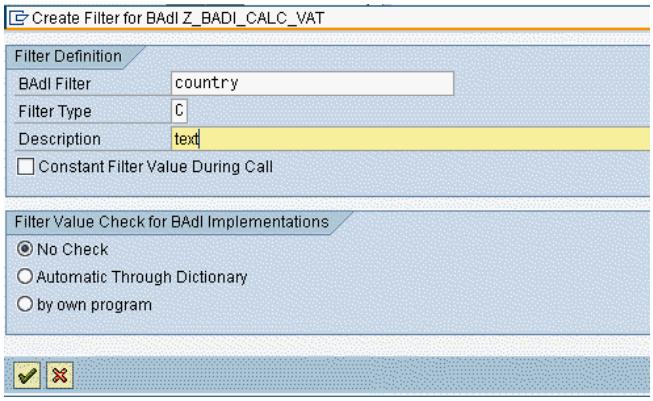
What you need is to change the BAdI definition. You can define one or many filters for a BAdI. For the purpose of this example, you create only one filter. Therefore, in this step you define the filter in the BAdI definition, determine filter values for the respective BAdI implementation, and use the filter in the instantiation of the BAdI handle.

> **Note**
> When modifying your example you do not take into account the differentiation between the BAdI provider and the implementer. In real life it is the BAdI provider who defines a BAdI with a filter or adds a filter to an existing BAdI. It is also part of this role to use the filter condition to select the respective BAdI implementation in the ABAP code. It is the implementer who determines the filter value(s) or an interval for one or many BAdI implementations.

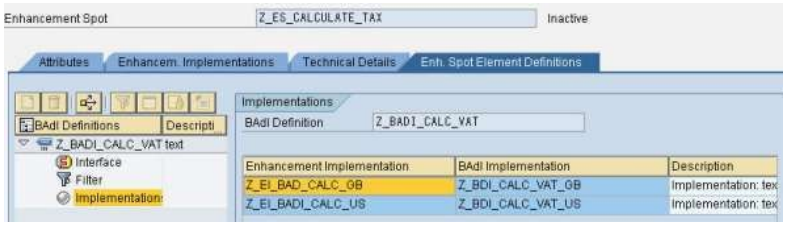
1. Navigate to your enhancement spot and open the Enh. Spot Element Definition tab that shows a list of all the enhancement elements of the spot.
2. Switch to a change mode, select your BAdI in the list, and in the small toolbar above choose Create BAdI Subobject icon and then Filter.



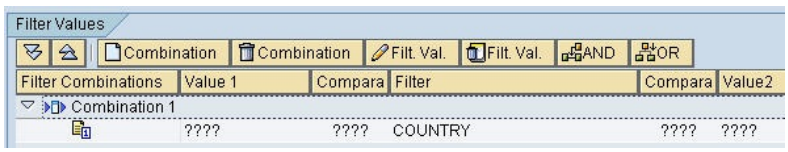
3. The following dialog appears, where you fill in the corresponding fields:



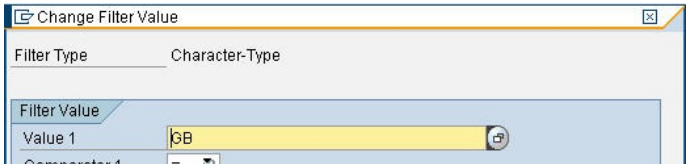
4. Confirm the entries.
   The filter is now visible as a property below the BAdI.
5. Activate the enhancement spot, double-click on the implementation and navigate to the respective BAdI implementation by double-clicking the respective row in the table of BAdI implementations.

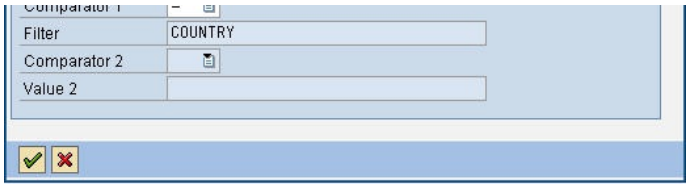


6. Switch to change mode, double-click the triangle in front of the BAdI implementation in the tree and then double-click on the filter icon below.
7. On the Filter Values screen, choose the Create Filter Combination pushbutton.
8. Select Country as the filter and confirm
9. Double-click on the row below Combination 1.



A new dialog opens:





1. Activate the (simple) enhancement implementation and navigate back to the spot.
   The other implementation, that is the one for USA, also needs the respective filter value. So you go to the respective (simple) enhancement implementation and change the BAdI implementation in the same way as you just did above. The respective filter value for this country is 'US'.
2. Return to your program and adapt it to the modified BAdI. Running the syntax check shows you that you need to have a filter parameter in the `GET BADI` command if the BAdI you try to instantiate is defined with a filter. The required addition to the `GET BADI` command is `FILTERS`. It is after this keyword that you insert the name of the BAdI filter and a value the filter is compared to. You also have to take care that an appropriate value can be passed to the `GET BADI` routine:

```
REPORT  Z_DEMO_ENH.
parameters: ctry(2) type c.
DATA: handle TYPE REF TO z_badi_calc_vat,
sum TYPE p,
vat TYPE p,
percent TYPE p.
sum = 50.
GET BADI handle FILTERS Country = ctry.
CALL BADI handle->get_vat
EXPORTING im_amount = sum
IMPORTING ex_amount_vat = vat
ex_percent_vat = percent.
WRITE: 'percentage:', percent, 'VAT:' ,vat.
```

   If you pass `GB` to the parameter `ctry`, you get a VAT rate of 17.5 percent. If the value of the parameter `ctry` is `US`, the VAT rate is 4 percent. When the parameter `ctry` has any other value, you still get a calculated value for the data field percent. This is because you have defined a fallback class for your BAdI. The fallback class is not only selected if no BAdI implementation is available, but also if none of the existing BAdI implementations meets the filter conditions in the `GET BADI` command. Accordingly, you get a VAT rate of 20 percent, which is the VAT rate you have implemented in the method `get_vat` in the fallback class.

3.4.1.4 Function Code and Screen Enhancements

Use

The main use of BAdIs is to enhance ABAP programs using object plug-ins. For compatibility reasons and to be able to use the classic means for designing a user interface in an ABAP-based SAP system (GUI status and screens), menu enhancements and screen enhancements were adopted almost unchanged from the classic BAdIs into the BAdIs of the new enhancement concept.

Function Code Enhancements

Menu enhancements have been adopted under the new name "Function Code Enhancements" but the classic concept has been retained. The use of the ABAP statements **`GET BADI`** und **`CALL BADI`** is not necessary. The runtime environment inserts the implementation of a function code enhancement automatically during program regeneration.

Screen Enhancements

In the case of screen enhancements, the class concept has been adopted with the following exceptions:

- The previous call to the method `CL_EXITHANDLER=>GET_PROG_AND_DYNP_FOR_SUBSCR`, has been replaced by the call `CL_ENH_BADI_RUNTIME_FUNCTIONS=>GET_PROG_AND_DYNP_FOR_SUBSCR` with the same interface.
- The methods `PUT_DATA_TO_SCREEN and GET_DATA_FROM_SCREEN` are no longer be generated. You can create your own BAdI methods for data transport and call them using **`CALL BADI`**.
- You no longer need to call the `CL_EXITHANDLER=>SET_INSTANCE_FOR_SUBSCREENS` and `CL_EXITHANDLER=>GET_INSTANCE_FOR_SUBSCREENS` methods. These methods are now unnecessary as they only place the BAdI reference in a temporary storage.
  `SET_INSTANCE_FOR_SUBSCREENS` is no longer necessary.
  `GET_INSTANCE_FOR_SUBSCREENS` can, if necessary, be replaced by **`GET BADI`**.

3.4.2 Definition of BAdIs

Name

When you define a BAdI, you must specify a name, a BAdI interface as the interface for the enhancement option, and the required filters.

The name of a BAdI is in the same namespace as data types from the ABAP Dictionary, global classes, or interfaces. It is recommended that you use suitable prefixes, such as "BADI_".

Filter

A filter consists of a filter name and a data type (integer, string, and so on).

Properties

In addition, you also define BAdI properties that are relevant at the runtime of a program with the appropriate statements **GET BADI** and **CALL BADI**. For more information, see Instance Generation Mode and Multiple Use.

Other properties that can be assigned to a BAdI include:

- An optional fallback BAdI implementation class. This option is used if no BAdI implementation with suitable filter conditions and no standard implementation is found.
- Whether the BAdI is internal or not.

  > **Caution**
  > An internal BAdI must only be implemented by SAP and is not visible outside of SAP.

- Whether the BAdI is a function code or screen enhancement.
  A BAdI that is defined as a function code enhancement must not have any filters, must not be defined for multiple use, or assigned to any switch. It may contain methods that are independent of the actual function code enhancement.
  A BAdI that is defined as a screen enhancement must be defined in the instance generation mode for the reused instantiation and must not be defined for multiple use. It may contain BAdI methods that can be used to fill or evaluate the screen fields of the respective subscreens.

3.4.2.1 Creating a BAdI

Procedure

1. Start the Object Navigator (SE80).
2. Open an enhancement spot. For more information, see Creating, Editing, and Deleting Enhancement Spots.
3. Select the Enh. Spot Element Definitions tab page.
4. Choose Create BAdI.
   A dialog box appears.
5. Enter a name and a short text for the BAdI.

   > **Note**
   > BAdIs are in the same namespace as global data types from the ABAP Dictionary, global classes, or interfaces. For BAdIs, we recommend using the prefix "BADI_" (or "ZBADI_", and so on, in the customer namespace).

   The new BAdI appears as a node in the tree display of the tab page.
6. On the right-hand side of the page, do the following:
   1. Enter the attribute for multiple use.
   2. Enter the instance creation mode.
   3. Enter the attribute for internal SAP BAdIs (only SAP internal use).
   4. Optional: Enter a fallback class.
   5. In the tree, expand the BAdI and choose the Interface node. Enter the name of an existing BAdI.
7. Optional: Choose the function Create Filterto create a filter.
   Here you can:
   1. Enter the filter name, filter type, and description.
   2. Optional: If you choose Constant Filter Value During Call, you may only specify a constant value at the respective filter when using **GET BADI**. This is provided for future performance improvements of the statement.
   3. Optional: Enter either a data element or a domain with fixed values, or a search help. Alternatively, enter a check or input help class, and a length (together with decimal places). In this way, the filter values specified at **GET BADI** can be checked during the BAdI implementation.
8. Optional: Choose the function Create Screen Enhancement in order to create the BAdI as a screen enhancement.
   The BAdI must not be of a multiple use type.
   1. Enter Calling Program, Screen Number, Subscreen Area, and Description.
   2. Optional: Select Default Value to specify a screen of a program that is used if no active implementation is found at runtime.
9. Optional: Choose Create Function Code Enhancement to create the BAdI as a function code enhancement.
   The BAdI must not have any filters and must not be of the type for multiple use.
   1. Enter Program, Function Code, and Description.
   2. Optional: Select Default Value to specify an icon, a menu text, a pushbutton text, and a quick info, all of which are used when no active implementation is found at runtime.
10. Optional: Create an example implementation.
    1. Select the BAdI, and choose Create Example Class from the context menu.
    2. Enter the name of a BAdI implementation class and a description.

3.4.2.2 Instantiation of BAdIs

This property controls the instantiation of the object plug-ins during execution of the statement GET BADI. The following are possible specifications:

- Newly created instantiation
- Reused instantiation
- Context-dependent instantiation

The first two specifications define context-free BAdIs. In the case of the newly created instantiation, new object plug-ins are created at each execution of the statement **`GET BADI`**. In the case of the reused instantiation, an object plug-in that was used once in the current internal mode is reused - if it is required more than once.

In the case of context-dependent instantiation, you need to specify a context for **`GET BADI`**. This context controls the instantiation. Only one object plug-in is created for each context and implementing class. Each time there is another `GET BADI` with the same context, it is reused. A context is an instance of a class that implements the tag interface `if_badi_context`. The specification takes place in the form of a reference to such an instance.

> **Note**
> If object plug-ins are to get reusable data, only specifications 1 and 2 are allowed.

Example

The figures below show the situation for context-free BAdIs, after two statements following each other:

```
GET BADI bd1.
```

```
GET BADI bd2.
```

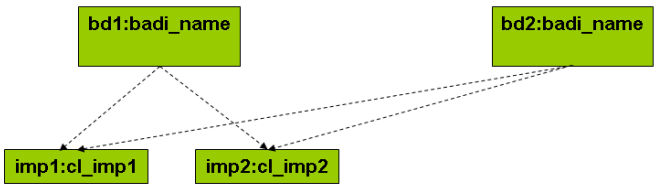
The upper rectangles represent the BAdI objects. The broken-line arrows show which object plug-ins are referenced by the BAdI objects, whereby the case of two suitable BAdI implementation classes `cl_impl` and `cl_imp2` is demonstrated.

Figure 1 shows the newly created instantiation. Each time a BAdI object is created, new object plug-ins are created - first `imp1` and `imp2`, then `imp3` and `imp4`.

Figure 1

Figure 2 shows the reused instantiation. For each BAdI implementation class, only one instance is created. All BAdI objects reference the same instance, irrespective of where (in internal mode) and how the BAdI object is called.

Figure 2



Contexts for BAdIs

The instance generation mode of a BAdI can be context-dependent. If you have context-dependent BAdIs, you must specify an object reference variable to a BAdI context object (an object whose class implements the tag interface `IF_BADI_CONTEXT`) in the **`GET BADI`** statement. As a result, within an internal session for the same BAdI context object, the same object plug-ins are always used. Such object plug-ins are singletons as far as their BAdI implementation class and a BAdI context object are concerned. Using a context-dependent BAdI is linked to a program is illustrated in the example below.

Example

By using the predefined global class `cl_badi_report_context` you can easily couple BAdI implementations to an (executable) program (or more generally, to a compilation unit). This class implements the interface `if_badi_context`, has a public attribute `repid`, and a public static method `get_instance`, which creates an object and sets the attribute.

By calling the method `cl_badi_report_context=>get_instance`, you can create unique instances as far as the attribute, which can serve as BAdI context object is concerned.

Suppose the BAdI `badi_ctx_badi` is context-dependent and has no filters. The code fragment below shows how to use the `cl_badi_report_context`:

```
DATA: bd  TYPE REF TO badi_ctx_badi,
ctx TYPE REF TO cl_badi_report_context.

ctx = cl_badi_report=>get_instance( repid = sy-repid ).
GET BADI bd CONTEXT ctx.
```

The reference variable `ctx` references an instance that is unique for the current program name. In this context a BAdI object is requested. If the same code fragment is executed in other programs in the same internal session, different instances of the BAdI implementation classes are always associated with the BAdI object.

The class `cl_badi_report_context` allows you to bind object plug-ins to compilation units. The methods of these object plug-ins can be called anywhere in the internal session by using the combinations of **`GET BADI`** and **`CALL BADI`** shown above. The state is kept in this connection, which means that the same instances are always called for the same compilation units.

3.4.2.3 The Multiple Use Property

Definition

This property defines whether, during the initialization of a BAdI object using **`GET BAD,`** just one implementation needs to be selected through the specified filter values, or whether an arbitrary number of implementations (or no implementation at all) can be selected.

If more than one implementation is selected at **`GET BADI`** for a BAdI provided for single use, the exception `cx_badi_multiply_implemented` is triggered. If no implementation is found, the exception `cx_badi_not_implemented` is triggered.

Use

You can define whether a BAdI is to be provided for single or multiple use. In the standard version, a BAdI is provided for single use, but a multiple use can also be selected.

In the case of a multiple use, there is a general restriction - in addition to the general restriction regarding variable attributes - that the BAdI methods must not have any `EXPORTING` or `RETURNING` parameters. The reason for this is that if you have a call with **`CALL BADI`**, the methods of all the object plug-ins referenced by the BAdI object are called and that there is no definition regarding from which of the implementations a returned value will actually come. `CHANGING` parameters, on the other hand, are allowed since these are changed by all the calling methods, one after the other, so that a method can also access the parameter changed in a previous method.

3.4.3 BAdI Use Cases

Here you can find different use cases for BAdIs separated into the following groups:

Section	Description
Single-Use BAdI	Single-use BAdIs for requesting something and getting something back
Multiple-Use BAdI	Multiple-use BAdIs for implementing the possibility to do something additional
Registry Pattern	Different registry patterns

3.4.3.1 Single-Use BAdI

Single Customer Enhancement

> **Tip**
> It is necessary calculate the VAT for several ledge entries. Since the developer at SAP who defines the enhancement option does not know the VAT values for the specific country, he or she creates a BAdI.

The BAdI has no filters and is called once. No state has to be kept, so you can use static methods. Such a BAdI is defined in the core or in an industry solution and is implemented by the customer.

Since this is single-use BAdI (it requires exactly one implementation), a default class must be provided by SAP. Otherwise, the `GET BADI` statement will raise an exception if it is executed before the customer has inserted the respective implementation.

Multiple Customer Enhancements

Tip

You have a calculation rule that depends on the country, for example a VAT rate which is different for different countries. It has default implementations for Europe, Asiaand America and a default class for the rest of the world.

This case is almost the same as the previous use case but with one important difference: there are different implementations for different countries.

Whenever you add an implementation for a specific country, it is selected with the respective filter value. The default implementation is selected if the country is in Europe, Asia, or America. Otherwise the default class is used.

Use Existing Coding As Default

If you have a piece of procedural coding which cannot be easily re-written in object-oriented style, you can use this code in combination with a BAdI. For that purpose, you need to use a single-use BAdI.

If there is no implementation, you get the exception `CX_BADI_NOT_IMPLEMENTED`. You can put the code you want to reuse in the respective `CATCH` clause. This code is executed if the BAdI has no active implementation. You code serves as a default implementation in that case.

If there are many implementations when you need exactly one, this is a more severe mistake and the handling of this exception must return you to a point in the call hierarchy where the next step of the current process is a step that can do without the result of the calculation.

Tip

Use a `TRY-ENDTRY` construct and put the procedural code you want to reuse in the `CATCH` construct:

```
TRY
GET BADI
CALL BADI
CATCH cx_badi_multiply_implemented.
*Go somewhere up in the call hierarchy
CATCH CX_BADI_NOT_IMPLEMENTED.
*Your code to be reused
ENDTRY
```

3.4.3.2 Multiple-Use BAdI

Multiple-Use BAdI

Multiple-use BAdIs allow multiple implementations to exist in parallel and it is also possible to have no implementation at all. With multiple implementations in place there cannot be a single return value; therefore the methods of the interface of a multiple-use BAdI must not have "returning" or "exporting" parameters. This type of BAdI is not used to compute results but enables one or more activities to be executed at a certain point in time. Multiple-use BAdIs can be used to implement an event mechanism that allows multiple implementers to specify their individual reaction to some event.

Tip

`badi_additional_export`

Export the book entry to additional destination(s) at the end of a book entry.

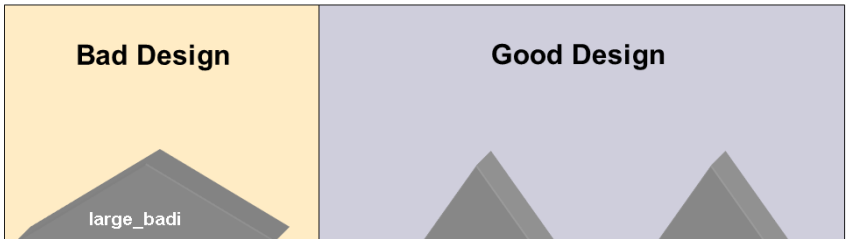
```
*Some_Code
*End of program
CALL BADI badi_additional->export
```

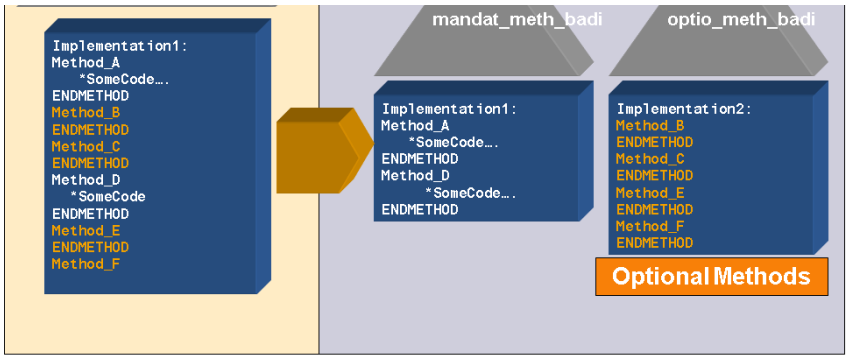
The result of this example is:

- Implementation 1: Export to XML file
- Implementation 2: Export to sequential file
- Implementation 3: Export to external file

BAdI With Optional Functionality

In the example figure below, the methods `method_a` and `method_b` have to be put into one BAdI. All the optional methods are put in another BAdI:





Never put mandatory and optional methods for a given solution or customer in one BAdI.

> **Note**
> A large number of empty implementations indicates that your BAdI is too large.

- If there are many empty methods that are often not implemented, divide the BAdI.
- A design which leads to empty implementations of methods is a poor design: you will have unnecessary work with empty methods if there are any changes to the BAdI.
- Instead of leaving optional methods empty, define them in different BAdIs.
- Adding new smaller BAdIs instead of modifying large BAdIs to reduce the effort for implementation during import or upgrade.

Encapsulating Data for Two BAdIs Within One Class

Sometimes you have two BAdIs that have to work on the same encapsulated data. An example can be a BAdI that manages some data and writes it to the database and a BAdI that reads this data.

If you have a class that implements the interfaces of both BAdIs, this class is selected as the implementation for both selections. If you pass the same context object in both cases, you get the same implementation because the class mentioned implements both interfaces. Because of the same context object you get also the same instance.

> **Tip**

```
GET BADI bd1 CONTEXT me
CALL METHOD bd1->meth
```

and

```
GET BADI bd2 CONTEXT me
CALL METHOD bd2->meth
```

If the BAdIs are instantiated from within the same context-object (which implements the interface `if_badi_context`), you work with the same implementation and the same instance. This implementation implements the interfaces of both BAdIs.

Encapsulating Data for Two BAdIs Within Several Compilation Units

If you want to encapsulate data for two BAdIs that are called in different compilation units, you need an additional context object, which is accessible from both compilation units.

> **Tip**

```
GET BADI bd1 CONTEXT mycontext  "(some context given by a global factory class)
CALL METHOD bd1->meth
```

and

```
GET BADI bd2 CONTEXT mycontext  "(some context given by a global factory class)
CALL METHOD bd2->meth
```

In this case you need a separate context object. Both positions where the `GET BADI` occurs must be able to reach the same factory class. Both BAdI calls will yield calls to the same instance of class `X`. In addition, it is possible to share data between BAdIs.

This technique allows small and simple BAdI definitions.

Inheriting from a Default Class

When you define a default class also as an example class, the one who implements the BAdI can easily inherit from this class. This is a very comfortable way of redefining just the methods that must be different from the example or the default class in your implementation.

3.4.3.3 Registry Pattern

This use case shows how you enhance objects that mirror database table lines, where the database table stands for business objects such as an order, a customer, or a material.

When you call the BAdI from inside the respective class, you get one BAdI instance for each object. In that way you can easily enhance your objects with new methods or attributes. You just have to make some additional provisions if you want to access private or protected attributes or methods of your object with the BAdI. In this case you have to pass the object reference to the BAdI and declare the friend relation to the BAdI interface.

Tip

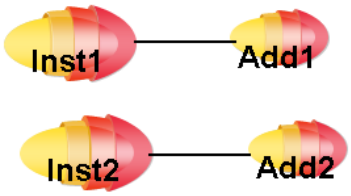
There is a class for a database table. For each line of the respective database table there is an instance of the class `cl_mat`. `cl_mat` implements `if_badi_context`.

The respective BAdI is designed in such a way that it enhances the respective object. Methods of BAdIs enhance the functionality of the class `cl_mat`.

If you need access from the BAdI instance to the original object, you have to pass the reference of the original object to the BAdI instance.

If you need access to private or protected attributes or methods, declare the friend relation in the interface.

```
GET BADI my_badi CONTEXT me
CALL BADI my_badi->meth
```



Each instance of the `cl_mat` gets an addition of its own, which enhances the class.

3.4.4 Migrating Classic BAdIs

Note

Due to that fact that calling new BAdIs is significantly faster, we recommend that you define only new BAdIs and migrate all classic BAdIs and their calls to new BAdIs. Make sure that you migrate only your own BAdIs. Customers should never migrate the BAdIs that are provided by SAP.

A completely automated migration of all existing classic BAdIs is impossible because of the existing differences between classic and new BAdIs. For that reason, we recommend that you use the procedure for semi-automatic migration.

See also:

Differences Between Classic and New BAdIs.

Migrating BAdIs

3.4.4.1 Differences Between Classic and New BAdIs

Classic and new BAdIs differ in a number of features important for migration:

- Differences in the BAdI objects
  1. In the case of classic BAdIs, you create a BAdI object by calling a factory method and reference it by means of a reference variable of the type of the BAdI interface.
  2. In the case of new Badis, you create a BAdI object with the ABAP statement **`GET BADI`** as a handle for the calls of BAdI methods and you reference it with a reference variable of the type of the BAdI. A BAdI object is an instance of an internal BAdI class which is invisible to the outside world.
- Passing comparison values to the filter
  1. In the case of classic BAdIs, the filter values are stored in a structure and passed with the call of the BAdI methods.
  2. In the case of new BAdIs, the comparison values for the filters used to search for implementations are passed when the BAdI object is created with the **`GET BADI`** statement.

3.4.4.2 Migrating BAdIs

Automatic (Partial) Migration

1. Call the classical BAdI Builder (transaction SE18).
2. Enter the name of the BAdI you want to migrate.
3. Choose Utilities → Migrate Classic BAdI.
4. Create for an enhancement spot with the same name as the classic BAdI.
   It is also possible to use another name or to choose an existing enhancement spot.
5. In the enhancement spot, create one new BAdI which has the same name as the classic BAdI.
6. The new BAdI should be defined with a context-dependent instance generation mode.
   The properties of the classic BAdI (such as multiple use, texts) and the screen and menu enhancements (such as function code enhancements) are copied one to one.
   The tag interface `IF_BADI_INTERFACE` is included in the BAdI interface of the classic BAdI.
7. If you define a filter for the classic BAdI, copy the components of the related filter structure to the new BAdI as separate filters. Use the name of the component is used as a name of the new filter; its type is always `C`.
8. Copy all existing implementations of the classical BAdI to new BAdI implementations. Create the new BAdI methods within the existing BAdI implementation classes.
9. Analogous to the migration of the filter definition, the structure of the filter condition of a classic BAdI implementation should also be dissembled into individual filter conditions.
10. Transport the change into a follow-up system:
    1. Call transaction SPAU if there is a BAdI implementation for the migrated BAdI.
       In this case, a message with identifier 6 will appear in the input log of the transport request.
    2. Choose the traffic light in front of the BAdI implementation.
    3. Enter an enhancement implementation.

Complete Migration

After finishing the (partial) migration, you may to proceed with the optional full migration to profit from all of the advantages of the new BAdIs.

> **Caution**
> To perform this procedure you need expert knowledge of the application. The effort may vary from several minutes up to several days per BAdI

1. Delete the classic BAdI. (Answer the question: Shall the migrated BAdI be deleted, too? with No.
2. Change the new BAdI according to your needs.
   In most cases you do not need the context, so change the instantiation mode accordingly.
3. Find all calls of the classic BAdI by GET_INSTANCE and reprogram the BAdI call using the new commands GET BADI and CALL BADI
4. To improve the performance you can change the context settings for BAdIs called more than once in one program.

Additional Information

BAdIs can only be migrated in their original systems. The migration of BAdI implementations is triggered in all follow-up systems by the transport of a migrated BAdI.

The call of `CL_EXIT_HANDER=>GET_INSTANCE` and the subsequent calls of the BAdI methods are not migrated automatically. The present ABAP proxy class of the classic BAdI is kept in order to guarantee this semi-automatic migration, but it is regenerated. After that, it implements the tag interface `IF_BADI_CONTEXT` for BAdI context objects and uses the new statements **`GET BADI`** and **`CALL BADI`** in its proxy methods to call the migrated BAdI instead of determining the implementations itself.

Screen enhancements with classic BAdIs are called using the `CL_EXITHANDLER=>GET_PROG_AND_DYNP_FOR_SUBSCR` method. During the migration, the method and call are kept. In the method, however, after the migration the method `CL_ENH_BADI_RUNTIME_FUNCTIONS=>GET_PROG_AND_DYNP_FOR_SUBSCR` is called for the new BAdIs.

The method `SET_INSTANCE_FOR_SUBSCREEN` from the old BAdIs is no longer necessary and `GET_INSTANCE_FOR_SUBSCREEN` can be replaced by a `GET BADI` command due to the fact that BAdIs with screen extensions always have reuse as the instantiation mode

Menu enhancements or function code enhancements are evaluated for classic and new BAdIs during the program generation. The generation considers classic and new BAdIs alike.

> **Caution**
> After the migration, you are not allowed to change the resulting new BAdI as long as the original classic BAdI still exists. If the classic BAdI is not deleted immediately after the migration, but is changed again, then the resulting new BAdI is adapted. This is the reason why BAdIs must be migrated only in their original system.
>
> We recommend that you make all required manual changes (calls) as soon as possible and then delete the classic BAdI.

Influence on Upgrades

BAdI implementations are usually created in follow-up systems of the system in which a BAdI is defined. The migration of a BAdI in its original system triggers the migration of the implementation in all follow-up systems as follows:

- For all related BAdI implementations, an upgrade flag is set.
- In the tool for the enhancement comparison, these BAdI implementations are displayed with a green traffic light.
- By selecting the comparison function, you can start the migration of the individual implementation manually.

Example

BAdI Interface After the Migration

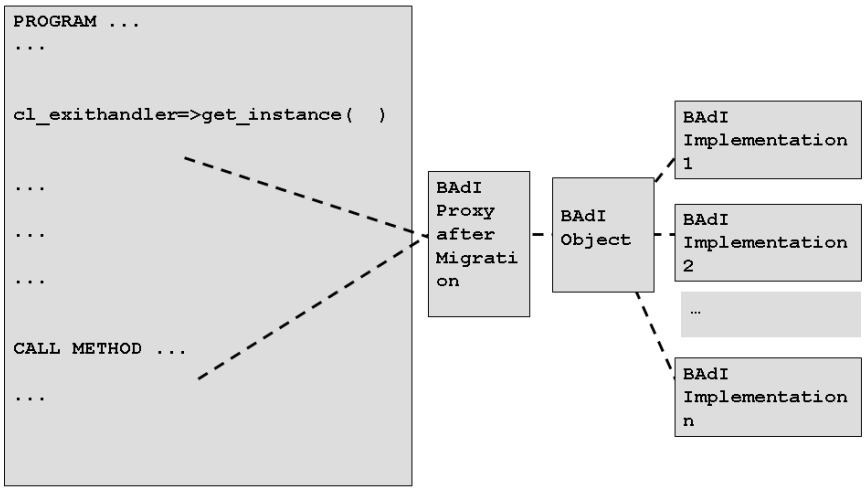
```
INTERFACE if_ex_badi_migtest PUBLIC.

INTERFACES if_badi_interface.

METHODS m  IMPORTING  value(flt_val) TYPE flt_migtest  CHANGING  test TYPE char1.
```

```
ENDINTERFACE.
```

ABAP proxy class after the migration

```
CLASS cl_ex_badi_migtest DEFINITION PUBLIC  FINAL  CREATE PUBLIC.
PUBLIC SECTION.  INTERFACES: if_ex_badi_migtest,   if_badi_context .
ENDCLASS.
CLASS cl_ex_badi_migtest IMPLEMENTATION.
METHOD if_ex_badi_migtest~m.
DATA l_badi TYPE REF TO badi_migtest.
GET BADI l_badi  filter  f1 = flt_val-f1  f2 = flt_val-f2  CONTEXT me.
CALL BADI l_badi->m  EXPORTING  flt_val = flt_val.  changing  test = test.
ENDMETHOD.
ENDCLASS.
```

This results in the following call hierarchy:



1.

3.4.5 Additional Information

This section provides additional information about the usage of BAdIs. The following topics are covered:

- Documentation
- FAQs

3.4.5.1 Documentation

You can create documentation about BAdIs at the following places:

Definition/Implementation Side	Type of Documentation

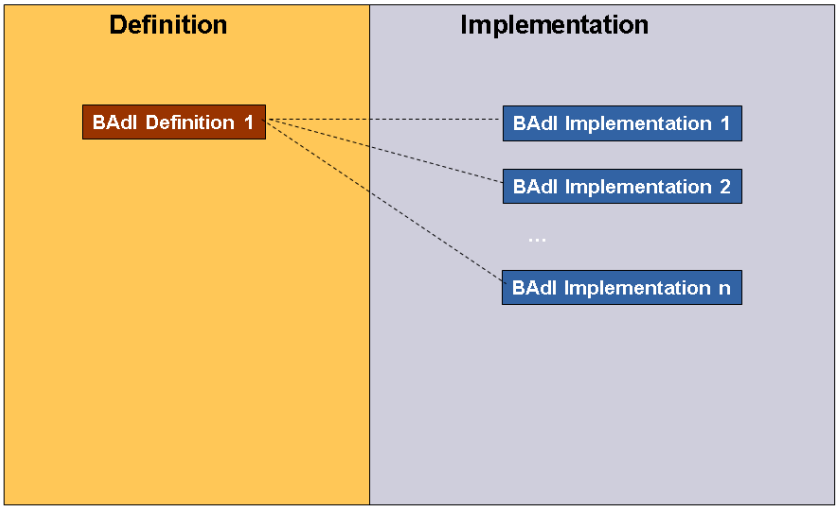
Definition Side	• Documentation of the relevant enhancement spot
	• Documentation of the BAdI itself:
	  ○ Externally visible documentation - this documentation can be stored, for example, in the IMG. The documentation has to describe the role and purpose of the BAdI. The methods of the BAdI must not be described here, but in the BAdI interface instead.
	  ○ Technical documentation - this documentation describes the BAdI from a technical point of view. It has to contain information that supports BAdI developers with any further developments or error handling.
	• Documentation of the BAdI interface and its components
	• Documentation of the default and example implementation classes

Implementation Side	• Documentation of the enhancement implementation • Documentation of the BAdI implementation • Documentation of the BAdI implementation class Note

	As far as the content and structure of the documentation is concerned, you must adhere to the existing standards and guidelines.

3.4.5.2 BAdIs Embedded in the Enhancement Framework

The following figure shows the basic idea behind the implementation of a BAdI. You have one BAdI definition and many implementations that may implement the same BAdI:



The definition of BAdIs is managed by enhancement spots. A simple enhancement spot for BAdIs can contain several BAdI definitions as enhancement spot element definitions. BAdIs that would otherwise be spread out in the system can be grouped together semantically.

This structuring is mirrored on the implementation side through the enhancement implementations. A BAdI can be implemented by several BAdI implementations that are managed by enhancement implementations. A simple enhancement implementation for object plug-ins can contain several BAdI implementations of an enhancement spot as enhancement implementation elements and can implement a BAdI multiple times. Therefore, several enhancement implementations can be assigned to one enhancement spot.

A BAdI is always assigned to the same package, like the enhancement spot, to which it belongs.

The structuring features of BAdIs in enhancement spots can be of a technical or a subject nature.

If, in the definition of several BAdIs, you have the provision that they are to be used in the same context (this is a feature known only by the person who defines the BAdIs), they must be used by the same simple enhancement spot.

From a subject point of view, all BAdIs must then be grouped together by affiliation to functional units - for example, payroll - in combined enhancement spots.

3.4.5.3 FAQs

When do I use a context-dependent BAdI? When do I select instance reuse?

It makes sense to bind a BAdI to a context only if the same BAdI is called several times and has instance methods. The context guarantees that an already instantiated implementation class is reused provided that the same context is transferred. If the implementation uses local data, the data is kept.

- If a BAdI interface contains only static methods and is used more than once in an internal session, for performance reasons, you have to use a non context-dependent BAdI with instance reuse.
- If a BAdI is used only once in an internal session, again for performance reasons, you should always use a non context-dependent BAdI without instance reuse.
- In all other cases, use a non context-dependent BAdI without instance reuse.

When do I use the fallback class in the BAdI definition and when do I create a default BAdI implementation?

A default implementation can have filter conditions and you can have many different default implementations with different filter conditions. If at compile time no non-default implementation for a given filter value is provided, the system selects the suitable default implementation. If there is no suitable default implementation, the system selects the fallback class.

> **Tip**
> A BAdI defines the filter language. You create one default implementation for German and one for English, and for all other languages you create a default fallback class.

Can I change a BAdI after the delivery and what are the consequences?

The new BAdIs which are part of the Enhancement Framework have upgrade support. If a BAdI definition is imported into a system in which there are implementations of this BAdI, the framework checks if the imported BAdI definition and the BAdI implementations are still compatible. If incompatibilities are detected, there is an adjustment tool that provides information about which implementations must be adjusted and supports the user in adjusting the relevant BAdI implementation.

Before a BAdI implementation is adjusted, the incompatibility between the BAdI implementation and the definition can cause syntax errors (for example, due to changes to a method interface). This is critical especially for central BAdIs. An error in a BAdI in the user management can, for example, make logons impossible.

In general, it makes sense to perform only compatible changes, such as:

- Adding new optional parameters.
- Changing a single-use BAdI to a multiple-use BAdI (without changing parameters).
- Changing an SAP-internal BAdI to a non-SAP-internal BAdI.
- Adding a screen or menu enhancement with the related default values.
- Adding new methods as long as an empty implementation works.

Can I deliver a normal class from which the BAdI implementation classes can inherit? Can I implement several BAdIs in the same BAdI implementation class?

Yes, for the BAdIs of the enhancement concept, BAdI implementation classes can be treated like normal classes. They can inherit from other classes and implement several interfaces. It is also possible to inherit from any BAdI implementation class.

Is there a naming convention recommended by SAP for BAdIs in the Enhancement Framework?

We recommend that you use the `BADI_` prefix for all BAdIs. In this way you avoid namespace conflicts with other global data types, classes, or interfaces. BAdIs must belong their own namespace so that in ABAP you can use `REF TO` to create a unique BAdI reference variable.

4 Working with Enhancements

This section describes how to work with enhancements.

- Creating and Editing Enhancements
  The tool for creating and editing enhancement spots and enhancement implementations is called Enhancement Builder and is fully integrated into the ABAP Workbench, you cannot use a transaction code to call it directly.
- Looking Up Enhancements
  The Enhancement Information System (also known as Enhancement Browser) helps you to obtain a list of all enhancements available in your ABAP-based SAP system. You can search for enhancements, enhancement upgrades, (composite) enhancement implementations, and (composite) enhancement spots.
- Adjusting Enhanced Objects
  Whenever an enhanced repository object has been changed in a previous system and its enhancement implementation has been affected or may be affected by a semantic change, use one of the available tools to adjust your enhancements.

4.1 Enhancement Builder

The Enhancement Builder is a tool for creating and managing enhancements which is fully integrated into the ABAP Workbench. You cannot use a transaction code to call it directly.

In the Enhancement Builder you can edit enhancement spots and enhancement implementations.

More information:

Creating, Editing, and Deleting Enhancement Spots

Creating, Editing, and Deleting Enhancement Implementations.

Managing Enhancements

4.1.1 Creating, Editing, and Deleting Enhancement Spots

Procedure

1. Start the Object Navigator (SE80)

If you want to	Then
Create a composite enhancement spot	1. Select the package in which you want to create the enhancement spot. 2. Open the context menu and choose Create → Enhancement → Composite Enhancement Spot. A dialog appears. 3. Specify a name and a short description. Optionally, you can assign the new spot to an already existing composite enhancement spot. 4. Choose Activate to activate the new spot.
Create a simple enhancement spot	1. Select the package in which you want to create the simple enhancement spot. 2. In the context menu of the package, choose Create → Enhancement → Enhancement Spot. A dialog appears. 3. Specify a name and a short description. Optionally, you can assign the new spot to an already existing composite enhancement spot. 4. Select the desired technology from the Technology list.

	Currently it is only possible to create enhancement spots of type "BAdI". Source code plug-ins can only be created inside the editor, because they contain points and sections. 5. Choose Activate to activate the new simple enhancement spot.
Edit an enhancement spot	1. Select the package and expand it. 2. Expand the Enhancements node. 3. Expand the Enhancement Spot node. 4. Select the desired enhancement spot. 5. Choose Change from the context menu. Choose Display if you only want to display the enhancement spot.
Delete an enhancement spot	1. Select the package and expand it. 2. Expand the Enhancements node. 3. Expand the Enhancement Spot node. 4. Select the desired enhancement spot. 5. Choose Delete from the context menu.

4.1.2 Creating, Editing, and Deleting Enhancement Implementations

Use

- If you have explicit enhancement options, you have to create the enhancement implementations for the related enhancement spots.
- If you have implicit enhancement options, you create the enhancement implementations directly.

Procedure

1. Start the Object Navigator (SE80).
2. Open the package in which an enhancement implementation is defined or is to be created.

If you want to	Then
Create an enhancement implementation	• If you have an explicit enhancement option, select the enhancement spot in the Object Navigator and in the context menu choose Implement. • For an implicit enhancement option, open the Enhancement Builder in the relevant tool (ABAP Editor, Function Builder, Class Builder) for executing an enhancement and continue with the following steps: 1. Enter a name for the (simple) enhancement implementation. 2. Enter a short text for the (simple) enhancement implementation. 3. Select a composite enhancement implementation, or create a new one. 4. Choose Creation of Enhancement (Enter). The subsequent process depends on the enhancement technology of the enhancement spot.
Edit an enhancement implementation	1. Select the package and expand it. 2. Expand the Enhancements node. 3. Expand the Enhancement Implementations node. 4. Select the desired enhancement implementation. 5. Select Change from the context menu. If you only want to display the enhancement implementation, choose Display.
Delete an enhancement implementation	1. Select the package and expand it. 2. Expand the Enhancements node. 3. Expand the Enhancement Implementations node. 4. Select the desired enhancement implementation. 5. In the context menu, choose Delete.

4.2 Enhancement Information System

Definition

The Enhancement Information System provides an overview of all enhancement possibilities and enhancements of the Enhancement Framework for an ABAP-based SAP system. There are several tree displays which render the hierarchy of the figure in the Enhancement Concept:

- An enhancement spot element definition is displayed below a simple enhancement spot, which in turn is displayed below a composite enhancement spot.
- An enhancement implementation element definition is displayed below a simple enhancement implementation, which in turn is displayed below a composite enhancement implementation.
- Composite enhancement spots or implementations can be displayed below further composite enhancement spots or implementations.
- Simple or composite enhancement spots or implementations can be at the uppermost level.

The tree displays provide an overview of the enhancements in a system. If there is no corresponding node in the implementation tree for a node in the definition tree, this means that an enhancement is possible but not yet implemented in the current system.

The Enhancement Information System shows the entire enhancement logic of the Enhancement Framework in an SAP system and enables users to find enhancements quickly and navigate to their environment.

As in the Object Navigator, you can go straight to the relevant tools from the nodes in the tree display or start direct processing using the context menu (for example, Copy, Rename, or Delete).

Like the Repository Browser or Repository Information System, the Enhancement Information System is a tree display (browser) in the navigation area of the Object Navigator (transaction SE80). You can call it either by choosing Enhancement Information System in the Object Navigator, or by using transaction SPAU_ENH.

> **Note**
> If the Enhancement Information System option does not appear on the initial screen of the Object Navigator, choose Utilities → Settings → Workbench (General) → Enhancement Info Systemoption to add it to the list.

In addition to the usual selection field for object categories and input field for names, the following options are available in the Enhancement Information System:

- Rebuild Tree - rebuilds the display for the current selection.
- Adjustment Preparation - explicitly starts the program ENH_SAPRUPGM. This function converts the flags from table ENHOBJ to flags of table ENHHEADER, and determines the status of the enhancements to be adjusted. These enhancements are then displayed under the category Enhancements Upgrade.

See also:

Display Options

4.2.1 Display Options

The Enhancement Information System offers the following display possibilities:

- Enhancements (Upgrade View)
- Enhancements (Overall View)
- Enhancement objects by type: Composite Enhancement Spots, Simple Enhancement Spots, Composite Enhancement Implementations and Simple Enhancement Implementations.

Enhancements (Overall View)

To see an overview of all the enhancements in a system, choose Enhancements in the selection list and enter an asterisk (*) in the input field.

The tree structure shows the hierarchy of all enhancement in the current SAP system. Initially, only the nodes of the uppermost level - that is, composite enhancement spots - are displayed. By expanding the nodes, you can display all the lower levels down to the enhancement implementations.

> **Note**
> The overall view of all enhancement objects in a system is required for generating the tree of Objects to Be Adjusted. The tree is generally created automatically after an upgrade. If it does not exist, it must be created manually for the above selection by choosing Rebuild Tree.

Enhancements (Upgrade View)

When you are upgrading, importing support packages, or importing notes with the Note Assistant, new enhancement objects can be imported into the system. As a result, you may have to adjust objects already in the system. For more information, see Adjustment Tools.

The Enhancement Information System enables you to display the entire set of objects to be adjusted. To do so, choose Enhancements (Upgrade View) in the selection list.

Unlike all other display options, the tree of objects to be adjusted is completely regenerated every time it is called. The system reads an existing tree of all enhancement objects (Overall View) and creates the tree of objects to be adjusted from it.

If no tree of objects to be adjusted is displayed, there are two possible reasons for this:

- There are no objects to be adjusted
- The tree of the Overall View has not yet been generated

To create the tree of the overall view, you can choose Rebuild Tree in the overall view display.

In the Enhancement Information System, an object is regarded as to be adjusted if the following conditions are met:

- In the database table ENHOBJ, the UPGRADE flag must be set for the relevant enhancement object.
- The program ENH_SAPRUPGM must be started to delete this flag and set it in the dependent follow-on tables.

Only if both conditions are met can you select and display objects to be adjusted. The prerequisites are generally fulfilled automatically after an import. The Enhancement Information System features the Adjustment Preparation function for calling the program ENH_SAPRUPGM.

Enhancement Objects by Type

You can choose a type in the selection field such as Simple Enhancement Implementation or Composite Enhancement Spot - and enter a corresponding name (generic also). The selected objects are then displayed according to their position in the overall hierarchy.

You have the following options displaying the objects by type:

- Composite Enhancement Spots
- Simple Enhancement Spots
- Composite Enhancement Implementations
- Simple Enhancement Implementations

All of the above display options use the same hierarchy and depending on the number of objects existing in the system, the trees they generate may be deeper or shallower.

Irrespective of what type of enhancement object you enter in the search field, you will get definition view and implementation view (if there is an implementation).

4.3 Adjusting Enhanced Objects

Use

If you have to deal with repository objects that have undergone an enhancement in a previous system, or in a subsequent system and their enhancements have to be adjusted, you need the adjustment tools of the Enhancement Builder.

The import log notifies you if there are imported objects which have an enhancement.

The adjustment tools are directly integrated into the Enhancement Editor. If an enhancement implementation is set to adjustment mode, an additional Adjustment tab is displayed in the Enhancement Editor.

All objects which are currently in adjustment mode can be listed in the upgrade view of transaction SPAU_ENH, or in the Enhancement Infosystem in the Object Navigator (transaction SE80).

More information:

- Objects Requiring Adjustment
- Displaying Set of Objects to be Adjusted
- Performing Adjustments

4.3.1 Displaying the Object Set to be Adjusted

Use

This procedure describes how you can use the Enhancement Information System to display the entire set of objects to be adjusted.

Procedure

1. Start the Enhancement Information System via the Object Navigator (transaction SE80) or transaction SPAU_ENH.
2. Choose Enhancements (Upgrade View) to display a list of the enhancements to be adjusted.
3. Select one of the enhancements for processing.

See also:

Objects to be Adjusted in the Display Options section.

4.3.2 Objects Requiring Adjustment

After an upgrade or import, the adjustment tools record all changed repository objects with enhancements that may have been affected. The set of objects to be adjusted is usually smaller since not all changes in an enhanced object have a real influence on the enhancement.

The following enhanceable repository objects are registered by the adjustment tools:

- ABAP source code with enhancements
- Function modules with enhancements
- Classes and interfaces with enhancements
- BAdIs
- Web Dynpro Components.

4.3.2.1 Cases When ABAP Source Code Needs Adjustment

The following changes make the ABAP source code eligible for adjustment:

- An object (class, function group, program) with enhancement options was deleted.
- A procedure (method, function module, subroutine) with enhancement options was deleted.
- The definition of an enhancement option with `ENHANCEMENT-POINT` or `ENHANCEMENT-SECTION` was deleted.
- The coding of an explicit enhancement option defined between `ENHANCEMENT-SECTION` and `END-ENHANCEMENT-SECTION` has changed.
- In a processing block that contains an enhancement option, the coding before the enhancement option has changed.
- A function module with enhancement options has been moved to another function group.
- A processing block with enhancement options was moved within its program to another include program.
- A conflict has occurred between source code plug-ins for enhancements defined with `ENHANCEMENT-SECTION`.
- An program-bound enhancement option (referring to the main program) has turned into an include-bound enhancement point (referring to an include with multiple usages), or vice versa.
- A static enhancement option has become a dynamic enhancement option, or vice versa.

More information:

Adjusting Source Code Plug-Ins

4.3.2.2 Cases When Function Modules Need Adjustment

Function modules can be enhanced by means of enhancements to the parameter interface at implicit enhancement options.

The following changes to function modules make them eligible for adjustment:

- A new importing parameter was declared that has the same name as an importing parameter introduced by an enhancement.
- An enhanced function module was deleted.
- An enhanced function module was moved to another function group.
- An enhanced function module is not RFC-compatible.

More information:

Adjusting Classes, Interfaces, Web Dynpro Components and Function Groups

4.3.2.3 Cases When Classes and Interfaces Need Adjustment

Classes and interfaces can be enhanced with implicit enhancement options by means of enhancements to the components of classes or interfaces. For more information, see Enhancing the Components of Classes or Interfaces.

After you apply one of the following changes, the changed classes or interfaces are included in the set of objects to be adjusted:

- There are name clashes with the imported class or interface or another imported enhancement implementation in:
  - An attribute name
  - A method name
  - A name of a parameter of a method
  - An event name
  - A name of an event in a method
  - A name of a type
- An enhanced class/interface is deleted.
- An enhanced method or event is deleted.

More information:

Adjusting Classes, Interfaces, Web Dynpro Components and Function Groups

4.3.2.4 Cases When BadIs Need Adjustment

After one of the following changes, the changed BAdIs or BAdI interfaces are included in the set of objects to be adjusted:

- The enhancement spot of the BAdI has been deleted.
- The BAdI has been deleted.
- The implementing class does not implement the BAdI interface.
- The implementing class does not implement all methods of the BAdI interface.
- A BAdI filter has been deleted.
- The type of the BAdI filter has been changed.
- The value check of a BAdI filter has been changed.
- The filter usage of a BAdI has been limited.
- A BAdI may only be implemented by SAP.
- An implementing class is syntactically incorrect.

More information:

Adjusting BAdI Implementations.

4.3.3 Performing Adjustments

Use

The adjustment tools are integrated into the Enhancement Builder. You can select one of them if you have to process an enhancement implementation that needs adjustment.

Procedure

1. Start the Enhancement Information System from the Object Navigator (transaction SE80) or from transaction SPAU_ENH.
2. Choose Enhancements (Upgrade View) to display a list of the enhancements to be adjusted.
3. Select one of the enhancements for processing.
4. Select the additional tab page Adjustment.
5. The conflict list displays the adjustment status.

> **Note**
> What a conflict means, why it has occurred, and how it can be solved is described in the long text for the conflict, which is accessible form the Long Text pushbutton in the toolbar of the conflict list.

6. Perform the kind of adjustment suited for the enhancement technology:
   1. Adjusting Source Code Plug-Ins
   2. Adjusting Classes, Interfaces, Web Dynpros and Function Groups
   3. Adjusting BAdI Implementations
7. Set the enhancement implementation to state Adjusted by choosing the Adjust Enhancement Implementation pushbutton.

> **Note**
> All conflicts that can be resolved automatically (signified by a green traffic light) are adjusted when you choose the Adjust Enhancement Implementation pushbutton in the toolbar of the conflict list. If all conflicts are resolved, the pushbutton is deactivated.

8. Activate the enhancement implementation.
   The edited enhancement is removed from the set of objects to be adjusted after the adjustment is complete.

4.3.3.1 Adjusting BAdI Implementations

1. Start the Enhancement Information System from the Object Navigator (transaction SE80) or transaction SPAU_ENH.
2. Choose Enhancements (Upgrade View) to display a list of the enhancements to be adjusted.
3. Select the enhancement implementation which contains your BAdI implementation.
4. Choose the Adjustment tab page.
5. Go through the listed conflicts by double-clicking each entry.
6. For each entry, follow the screen instructions, decide how you would like to solve the conflict, and adjust the conflict
7. Set the enhancement implementation to adjusted by using the Adjust Enhancement Implementation pushbutton.
8. All conflicts that can be solved automatically (green traffic light) are adjusted.
9. Save and activate the enhancement implementation.

See also:

Migrating BAdIs

4.3.3.2 Adjusting Classes, Interfaces, Web Dynpros and Function Groups

1. Start the Enhancement Information System from the Object Navigator (transaction SE80) or transaction SPAU_ENH.
2. Choose Enhancements (Upgrade View) to display a list of the enhancements to be adjusted.
3. Select the enhancement implementation which contains the enhancement of your class, interface, Web Dynpro component, or function group.
4. Select the Adjustment tab page.
5. Go through the conflicts listed in the ALV by double-clicking each entry.
6. Remove all name clashes by renaming or deleting the Enhancement Implementation Element.
7. Mark the enhancement implementation as adjusted by choosing the Adjust Enhancement Implementation pushbutton.
   All conflicts that can be solved automatically (green traffic light) will be adjusted automatically.
8. Save and activate the enhancement implementation.

4.3.3.3 Adjusting Source Code Plug-Ins

1. Start the Enhancement Information System from the Object Navigator (transaction SE80) or transaction SPAU_ENH.
2. Choose Enhancements (Upgrade View) to display a list of the enhancements to be adjusted.
3. Enter the enhancement implementation which contains the enhancement of your source code plug-in.
4. Select the Adjustment tab page.
5. Go through the listed conflicts by selecting each entry and choosing Conflict. See the long text display for more information.
6. Apply the necessary adjustments for each entry in the split screen editor, or revert the affected source code plug-in.
7. Set the conflict to adjusted by choosing the Adjust Conflict pushbutton and save your changes.
8. Remove all name clashes by renaming or deleting the respective enhancement implementation element.
9. Mark the enhancement implementation as adjusted by choosing the Adjust Enhancement Implementation pushbutton.
   All conflicts that can be solved automatically (green traffic light) are adjusted automatically.
10. Save and activate the enhancement implementation.

4.3.3.4 Adjustment Status

The adjustment status shows what kinds of conflicts occurred between the enhanced repository object and the enhancement during the upgrade.

Status	Description
Automatic adjustment (green traffic light)	The conflicts of the enhancement implementation can be adjusted automatically by using an adjustment tool.

Tool-aided adjustment (yellow traffic light)	The enhancement implementation must be adapted. An appropriate adjustment tool exists, but not all of the conflicts can be adjusted automatically.
Manual adjustment (red traffic light)	Apart from the existing adjustment tool (if any), you must use other tools to adjust the enhancement implementation. Additionalinformation about the conflict may be offered. The enhancement implementation cannot be adjusted automatically.
The enhancement implementation is adjusted (green tick)	The enhancement implementation has no more conflicts. It has either been adjusted, or another upgrade has neutralized all conflicts.
Semantic changes (grey traffic light)	Changes have been found in the enhanced repository object that are not directly connected with the enhancement. Such changes can have semantic effects on the enhancement implementation which cannot be displayed.
Enhanced repository object was deleted (trash can)	The enhanced repository object has been deleted. The enhancement no longer has a reference object and can be deleted as well, or must be reassigned.
Empty enhancement implementation (white flag)	The enhancement implementation contains no elements, it can be deleted.
Unknown upgrade status (question mark)	The adjustment state is not calculated or is unknown. To calculate it, you can use program ENH_SARUPGM.

4.3.3.5 Adjustment Without Tools

If, for one of the enhancement technologies delivered by SAP, no appropriate adjustment tool exists yet, or if in a customer system an independent enhancement technology has been developed for which no adjustment tool has been implemented yet, then the adjustment has to be performed in the editor of the respective enhancement. Use the adjustment status to find out what must be adjusted.

To remove the enhancement from the set of objects to be adjusted,after the adjustment you must set the enhancement in the Adjustment tab page to Adjusted,.

5 Enhancements of Component Configurations

Definition

It is now possible to create enhancements of component configurations.

To do this, follow the procedure below:

1. Start the configurator.
2. Press the Enhance button to go to the enhancement mode.
3. In the popup that appears you can select an existing enhancement or create a new one.

In the enhancement mode you can make the same changes as in the configuration mode. The changes are saved in the enhancement, the configuration itself is not changed. The finality of existing elements cannot be canceled in the enhancement mode, but you can still create enhancements for configurations that are final.

You can then display the configurations and enhancements in the configurator. In the display mode all activated enhancements are merged as at runtime, and without using shared objects.

More Information

For more information about the order of implementing multiple enhancements, see Implementation Order of Enhancements of Component Configurations.

5.1 Implementation Order of Enhancements of Component Configurations

Definition

You can save and implement as many configuration enhancements as you like. The order in which these enhancements are implemented is defined as follows:

Firstly, all the activated enhancements are identified. These are added in a three-step process to the results set:

1. All enhancements that are in the same software component as the configuration which they relate to.
   Sort key: release, name
2. All other enhancements whose software component does not begin with $.
   Sort key: software component, release, name
3. All other enhancements whose software component does begin with $.
   Sort key: software component, release, name
   The release is usually empty for local objects in TADIR.

In the display mode all activated enhancements are merged as at runtime, and without using shared objects. In NW 7.32 it is also possible to identify retrospectively the order in which enhancements have entered the shared object. In transaction SHMM see the root object of the area instance (area CL_WDR_CFG_COMP_SHM_TRANS). The order is read from bottom to top.

More Information

You can find the source code for sorting in the system in class CL_WDR_CFG_ENH_SORTER->GET_ORDER.