



BC425

Enhancements and Modifications

THE BEST-RUN BUSINESSES RUN SAP



© SAP AG 2010

- System requirements: SAP_BASIS 6.10 or later
- Version 2010
- Material No: 50099924

Copyright 2010 SAP AG. All rights reserved.

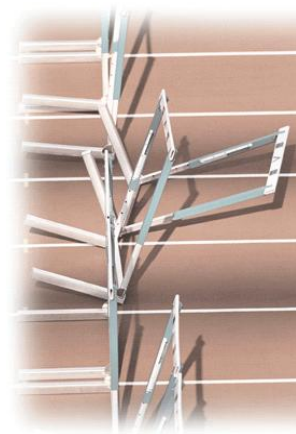
No part of this publication may be transmitted or reproduced in any way and for any purpose without the express written permission of SAP AG. The information contained in this publication is subject to change without prior notice.

© SAP AG

- Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.
- Microsoft, Windows, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.
- IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, and Informix are trademarks or registered trademarks of IBM Corporation in the United States and/or other countries.
- Oracle is a registered trademark of Oracle Corporation.
- UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.
- Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.
- .HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.
- Java is a registered trademark of Sun Microsystems, Inc.
- JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.
- MaxDB is a trademark of MySQL AB, Sweden.
- SAP, R/3, mySAP, mySAP.com, xApps, xApp and other SAP products and services as well as their logos are trademarks or registered trademarks of SAP AG in Germany and other countries worldwide. All other names of products and services are trademarks of the respective companies. The statements made herein are non-binding and are for informational purposes only. Products may differ by country.

- The information contained in this publication is subject to change without prior notice. The statements herein have been prepared by SAP AG and group companies ("SAP Group") and are for informational purposes only. The SAP Group assumes no responsibility for any errors or omissions in this publication. The SAP Group only assumes responsibility for products and services as per the express terms of the agreement covering the respective products and services. No further liability is implied by the information in this publication.

- ***BC400 – ABAP Workbench: Foundations and Concepts***
- ***Knowledge of ABAP Objects is useful***
- ***Experience with Customizing projects is also helpful***



- **Participant**

- Project managers and team members responsible for adjusting SAP functions to the requirements of the enterprise

- **Course Length: 3 days**



© SAP AG 2006

- **Notes to the User**

- These training materials are not a teach-yourself program. They complement the explanations provided by your course instructor. There is space for you to write down additional information on the sheets.

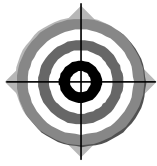


At the conclusion of this course, you will be able to:

- **Adjust the SAP standard to the customer's requirements**
 - **Personalization techniques**
 - **Enhancements to the SAP standard**
 - **Modifications to the standard and modification adjustments**

- **General Goals**
- **Course Objectives**
- **Table of Contents**
- **Course Overview Diagram**
- **Main Business Scenario**
- **Introduction**

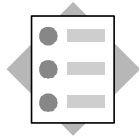
© SAP AG 2009



This course will prepare you to:

- **Make qualified changes to the SAP standard**
- **Evaluate the different methods for modification and choose the right one for any given situation**

© SAP AG 2009



At the end of the course, you will be able to:

- **Describe the different kinds of change levels available in the SAP System**
- **Enhance ABAP Dictionary objects without having to modify them**
- **Implement enhancements to the R/3 standard using user exits, customer exits, Business Transaction Events, and Business Add-Ins**
- **Make and adjust modifications using the Modification Assistant**

© SAP AG 2009

Preface

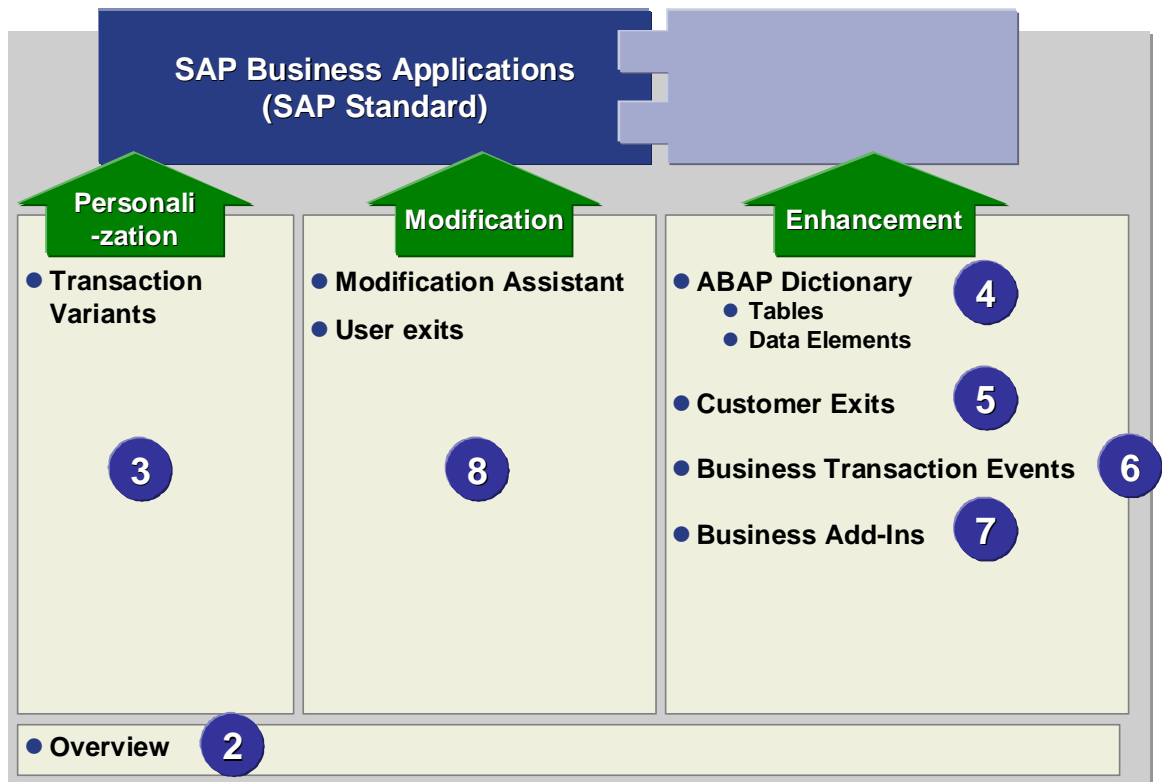
Unit 1	Course Overview
Unit 2	Changing the SAP Standard
Unit 3	Personalization
Unit 4	Enhancing Dictionary Elements
Unit 5	Enhancements Using Customer Exits
Unit 6	Business Transaction Events
Unit 7	Business Add-Ins
Unit 8	Modifications
Unit 9	Summary
Unit 10	Appendix

Exercises and their solutions can be found at the end of each chapter

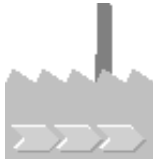
© SAP AG 2009

Course Overview Diagram

SAP



© SAP AG 2009



- You are employed by a major travel agency as a computer specialist. It is your responsibility to enhance the SAP programs that your customer service specialists use when helping clients
- There are a number of options available to assist you in this task

© SAP AG 2009

Contents:

- Overview of Adjustment Options and the Decision Diagram for Selecting an Adjustment Technique
- Enhancement Types

© SAP AG 2009

Internal Use SAP Partner Only

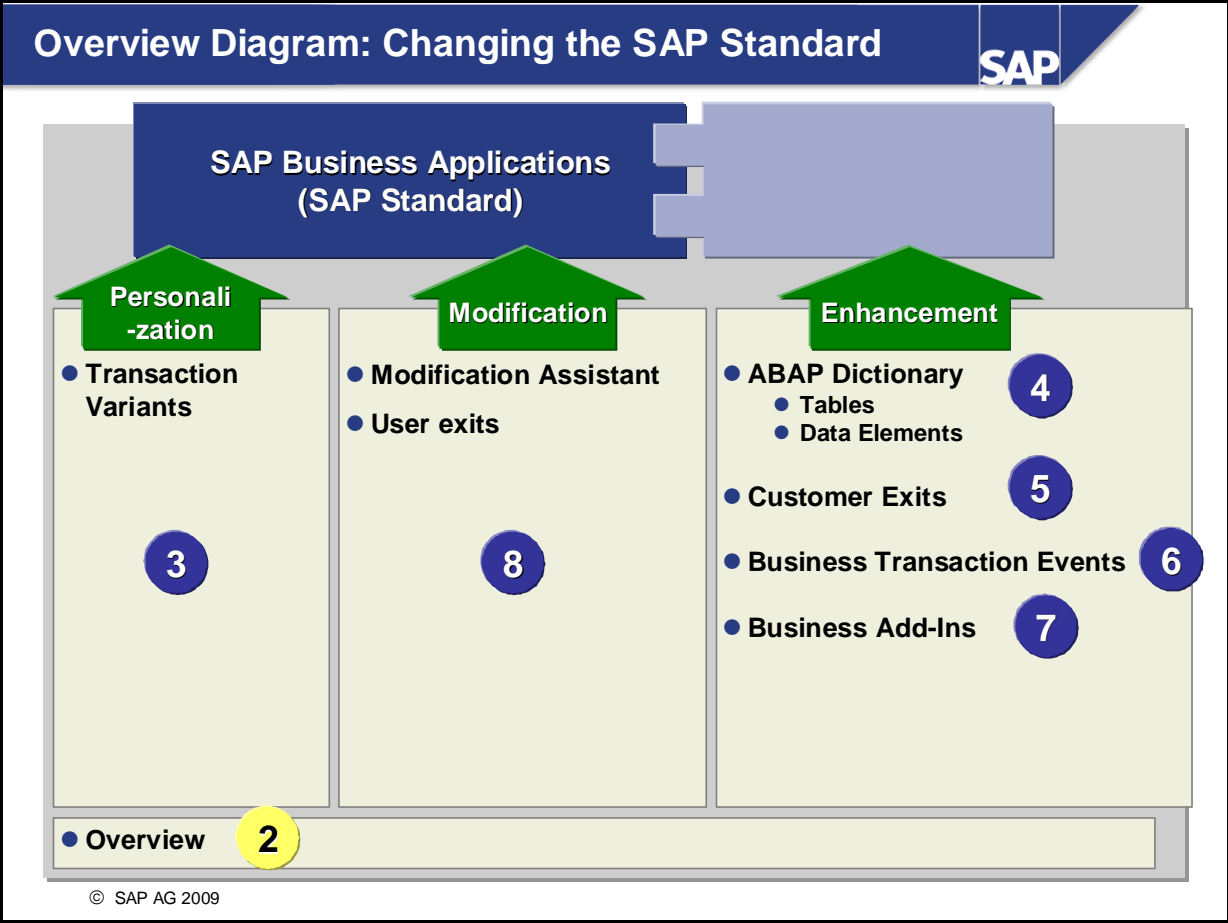
Internal Use SAP Partner Only

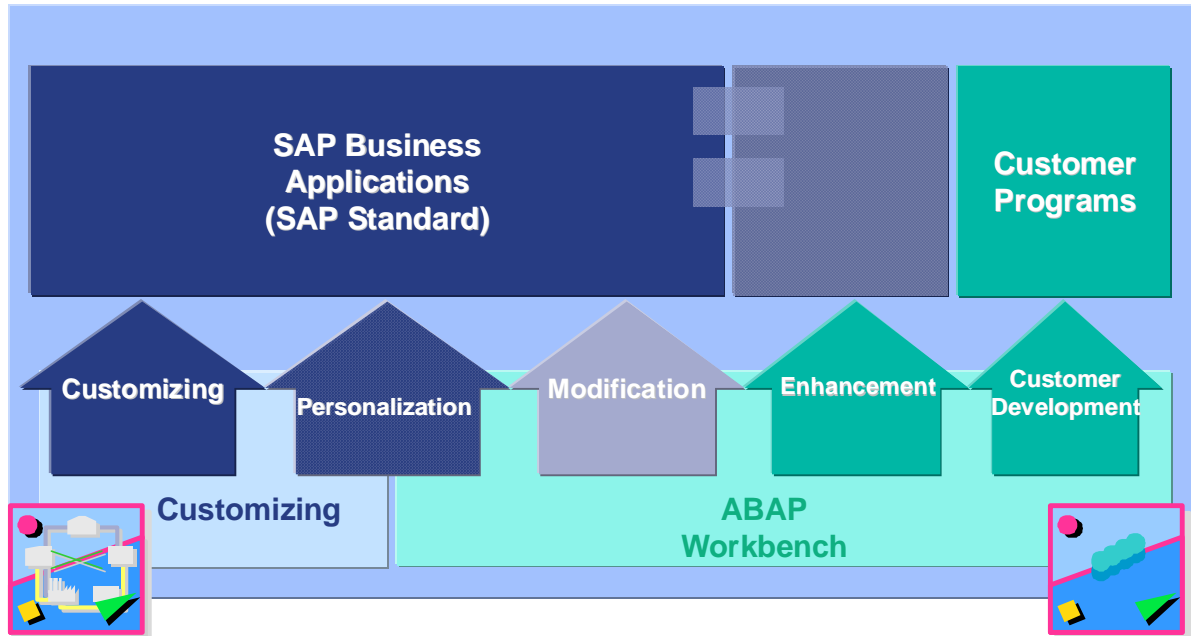


At the end of this unit, you will be able to:

- **Describe the different levels at which you can make changes to the standard delivered by SAP.**
- **Choose the most suitable method for changing the standard**
- **List the available enhancement types and explain their uses**

© SAP AG 2009



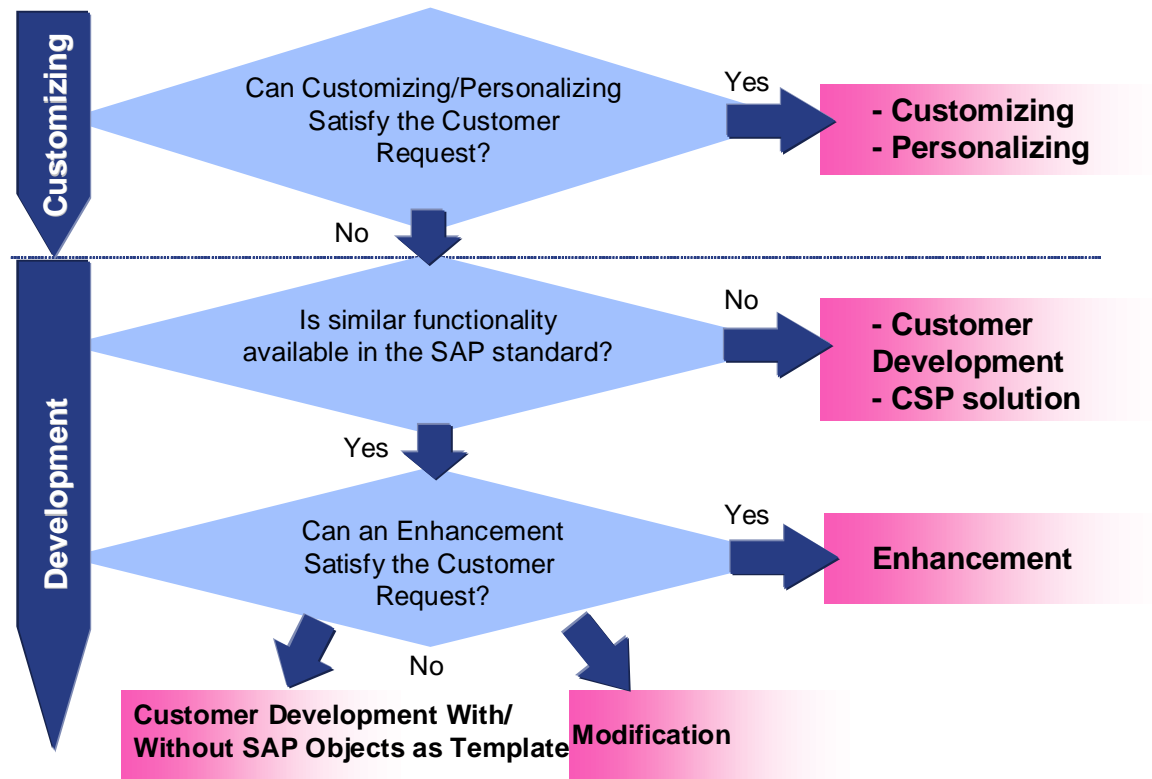


© SAP AG 2009

- You can adjust the ERP System to meet your needs in the following ways:
 - **Customizing:** Setting up specific business processes and functions for your system according to an implementation guide. Therefore, all possible changes have been thought through and organized.
 - **Personalization:** Making changes to certain fields' global display attributes (setting default values or hiding fields) as well as creating user-specific menu sequences.
 - **Modification:** These are changes to SAP Repository objects made at the customer site. If SAP delivers a changed version of the object, the customer's system must be adjusted to reflect these changes. Prior to Release 4.0B these adjustments had to be made manually using upgrade utilities. As of Release 4.5A, this procedure has been automated by the Modification Assistant.
 - **Enhancement:** This means creating Repository objects for individual customers that refer to objects that already exist in the SAP Repository.
 - **Customer Enhancement:** This means creating Repository objects unique to individual customers in the customer namespaces.
- Customizing and most personalization is done using tools found in the SAP Business Engineer. Customer developments, enhancements, and modifications are all made using the tools available in the ABAP Workbench.

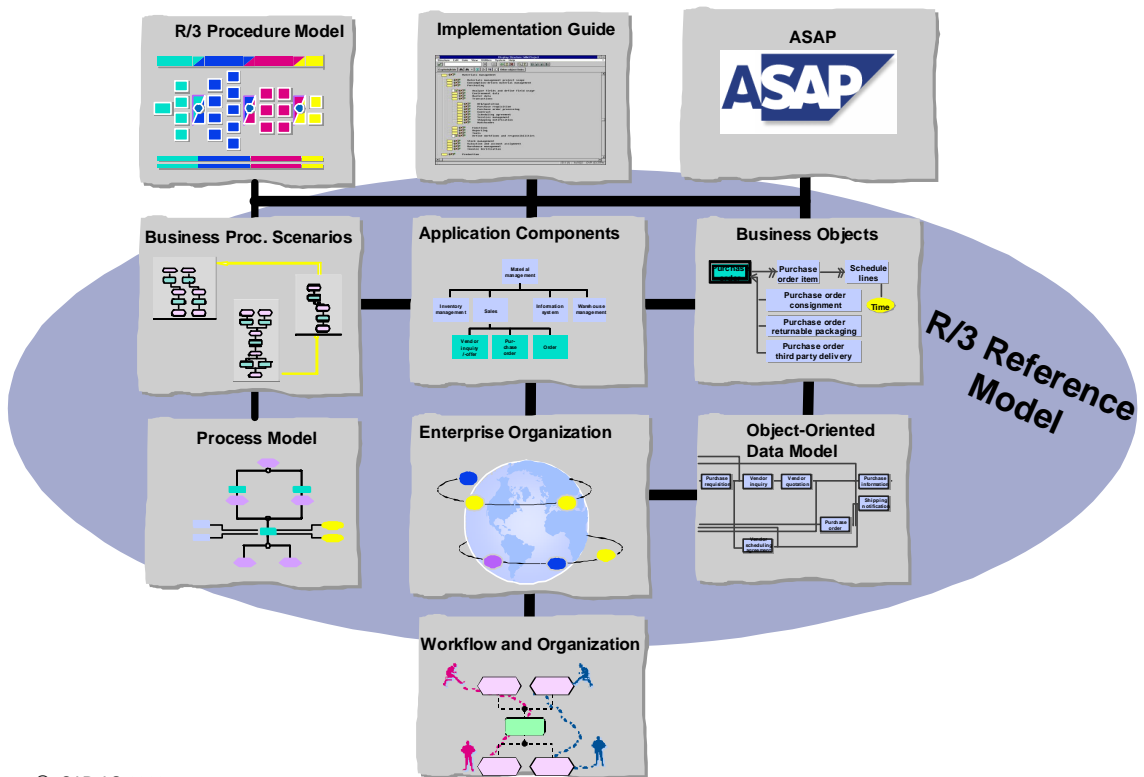
Procedure for Changing the Functionality

SAP



© SAP AG 2009

- If your requirements cannot be filled by Customizing or personalization, you can either start a development project or use a Complementary Software Product (CSP) solution, if available. A list of CSP solutions certified by SAP is available in SAP Service Marketplace under the alias /softwarepartner.
- A development project falls into the customer development category if the SAP standard does not already contain functions similar to the one you are trying to develop. However, if a similar SAP function exists, try to include it in your development project by either enhancing or modifying it, by using a user exit, or simply by making a copy of the appropriate SAP program.
- Modifications can cause problems: After an upgrade, new versions of SAP objects must be compared to modified versions of SAP objects you have created and adjusted if necessary. Prior to Release 4.0B, these adjustments had to be made manually using upgrade utilities. As of Release 4.5A, this procedure has been automated by the Modification Assistant.
- Therefore, you should only make modifications if:
 - Customizing or personalizing cannot satisfy your requirements.
 - Similar enhancements or user exits are not planned.
 - It would not make sense to copy the SAP object to the customer namespace.



© SAP AG 2002

- The Business Engineer is made up of all SAP **implementation tools**. These include:
 - **The R/3 Reference Model**
Contains all of the models used to describe R/3 (the process model, the data model, and the organization model)
 - **The Implementation Guide (IMG)**
 - A complete list of all customizing changes



You can often simplify an application without having to use the ABAP Workbench.

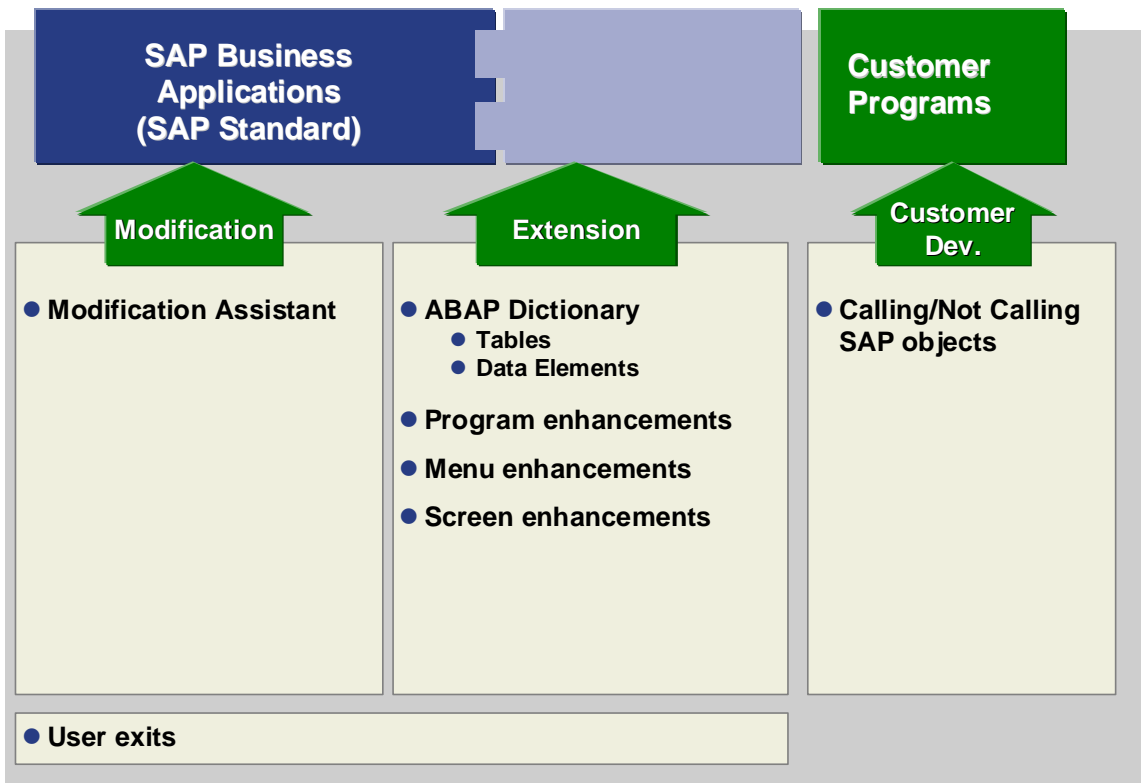
- **Global display attributes of fields**
 - Variant transactions
 - SET/GET parameter
 - User-Dependent or Client-Dependent Table Control Settings
- **Personalized menus**
 - Role-based menu
 - Favorites
 - Shortcuts on your desktop

© SAP AG 2009

- **Personalization** accelerates and simplifies the ERP System's processing of business cases. During personalization, individual application transactions are adjusted to meet the business needs of your company as a whole or even to the needs of specific user groups within your company. All unnecessary information and functions found in the transaction are switched off.
- Global display attributes allow you to define default values for specific screen fields. You can also suppress individual fields or table control columns in a particular transaction, or even a whole screen.
- Role-based menus, favorites, and shortcuts on the Desktop allow you to adjust menu sequences.
- Role-based menus, favorites, and shortcuts on the Desktop allow you to adjust **menu sequences** to reflect the needs of different user groups within your company.

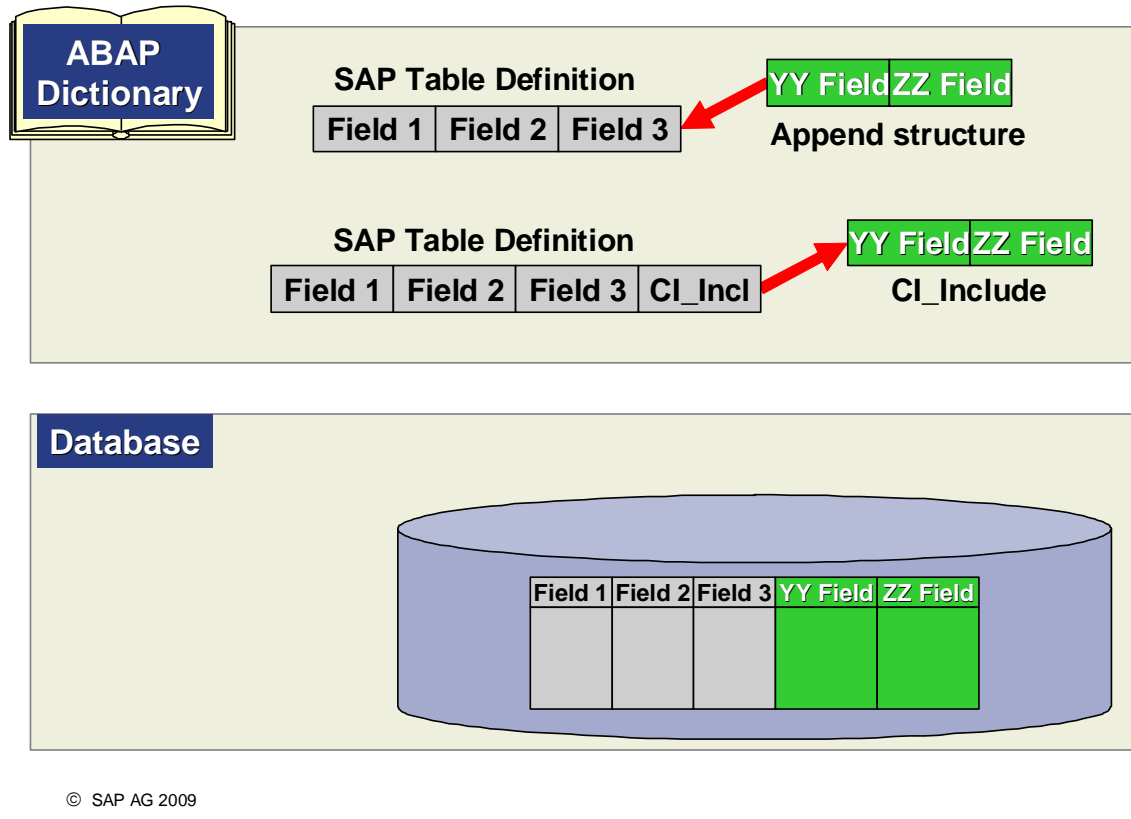
Change Levels using the ABAP Workbench

SAP

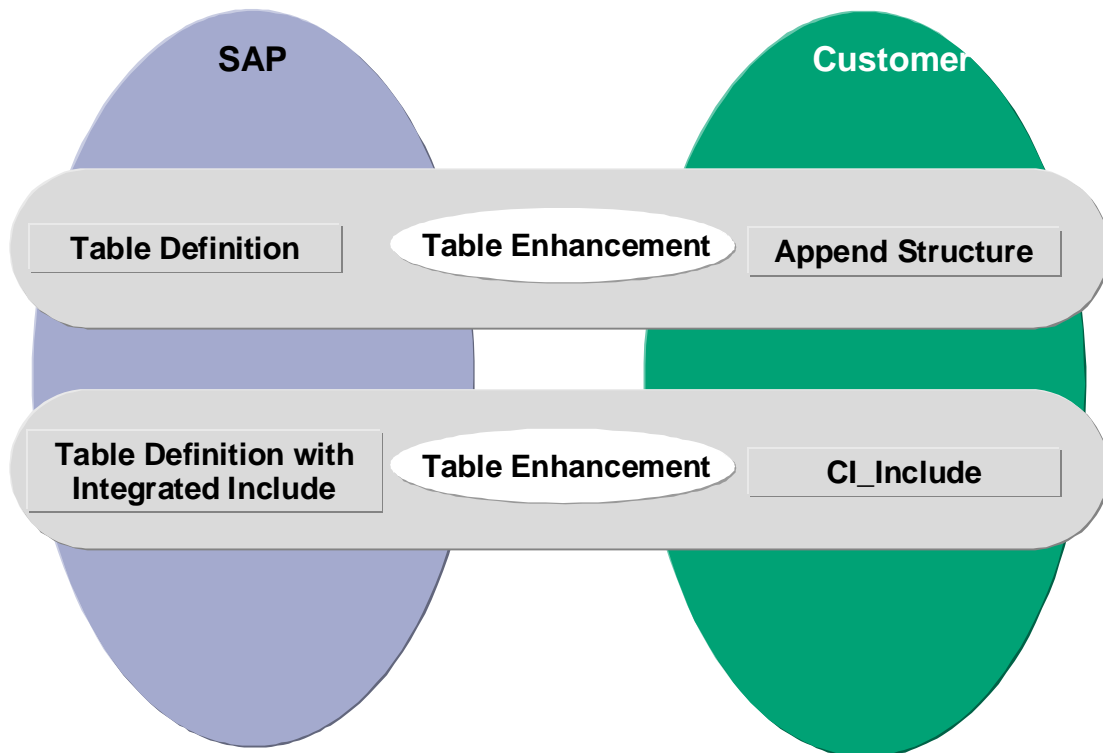


© SAP AG 2009

- Modifications are changes to SAP objects in customer systems. They are:
 - Executed with the help of user exits (these are subroutines reserved for customers within objects in the SAP namespace)
 - 'Hard-coded' at various points within SAP Repository objects
- Customer developments are programs developed by customers that can call SAP Repository objects. For example, customers often create programs that call SAP function modules.
- The role distribution is inverted in the enhancement concepts: SAP programs call Repository objects that you, as a customer, have created or changed. For example: You use a function module exit called by an SAP program. You can enhance your system at the following levels:
 - In ABAP programs (**function module exits**)
 - On GUI interfaces (**menu exits**)
 - On screens, by inserting a subscreen in an area specified by SAP (**screen exits**)
 - On screens by processing customer code that refers to a specific field on the screen (**field exits**)
 - In ABAP Dictionary tables or structures (**table enhancements**)

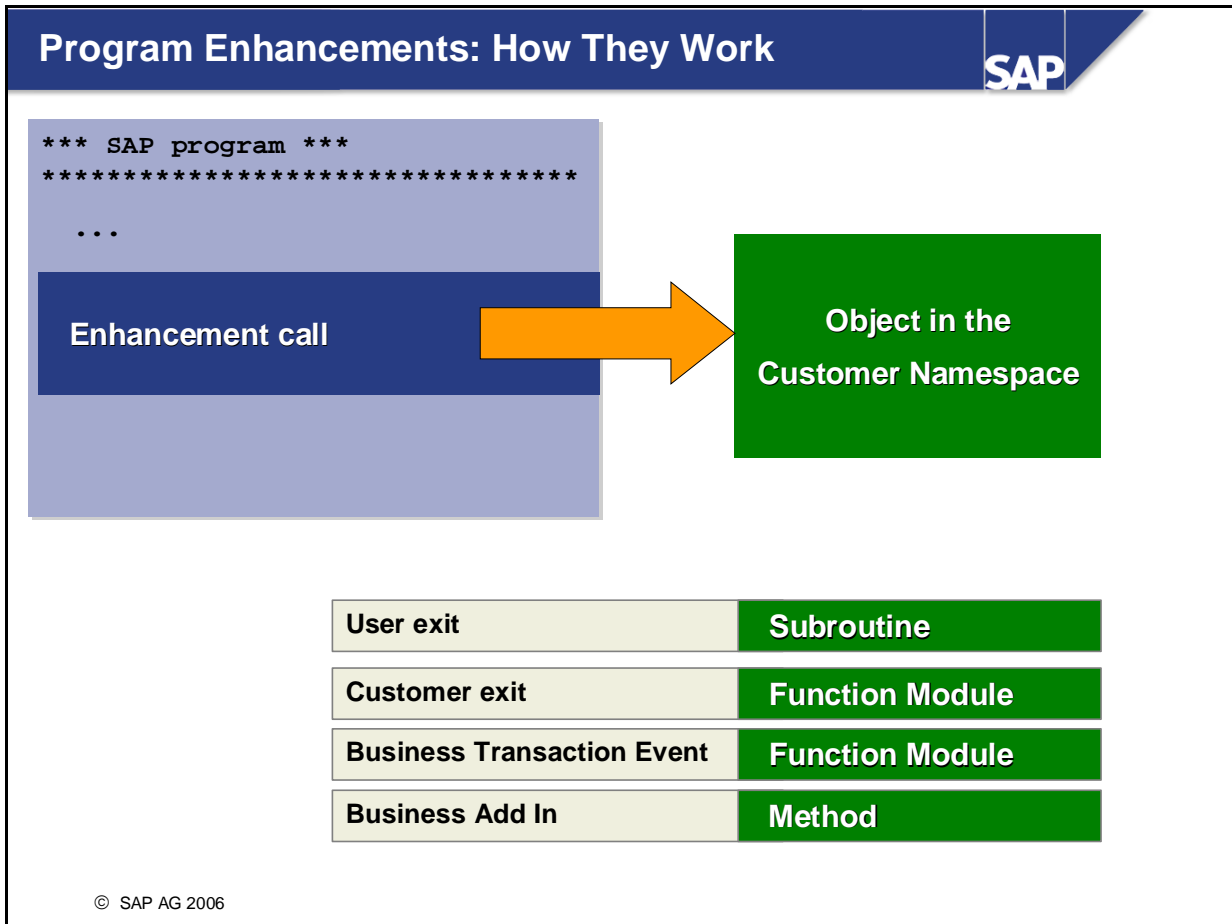


- SAP provides two ways to enhance tables and structures with fields.
 - Append structures
 - Customizing includes ("CI includes")
- Both techniques allow you to attach fields to a table without having to actually modify the table itself.
- An append structure is a structure which is assigned to exactly one table. There can be several append structures for a table. During activation, the system searches for all active append structures for that table and attaches them to the table.
- Append structures differ from include structures in how they refer to their tables. To include fields from an include structure in a table, you must add an '. INCLUDE...' line to the table. In this case, the table refers to the substructure. Append structures, on the other hand, refer to their tables. In this case, the tables themselves are not altered in any way by the reference.

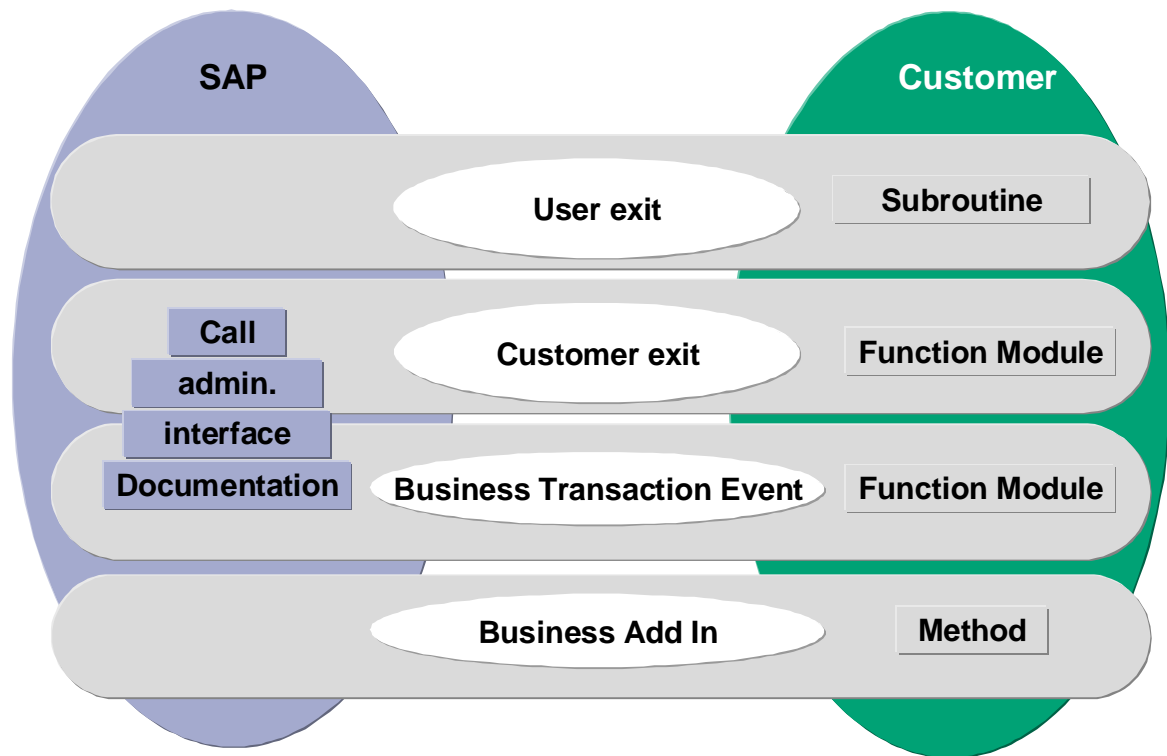


© SAP AG 2009

- Append structures allow you to attach fields to a table without actually having to modify the table itself. Table enhancements using append structures therefore do not have to be planned by SAP developers. An append structure can only belong to one table.
- In contrast, CI_includes allow you to use the same structure in multiple tables. The include statement must already exist in the SAP table or structure. Table enhancements using CI_includes do, however, have to be planned by SAP developers.



- The purpose of a program enhancement is always to call an object in the customer namespace. You can use the following techniques here:
 - **Customer Exits:**
A special exit function module is called by the SAP application program. The function module is part of a function group that is handled in a special manner by the system.
 - **Business Transaction Events**
The SAP application program dynamically calls a function module in the customer namespace.
 - **Business Add Ins**
The application program calls a method of a class or instance of a class. This class lies in the customer namespace.

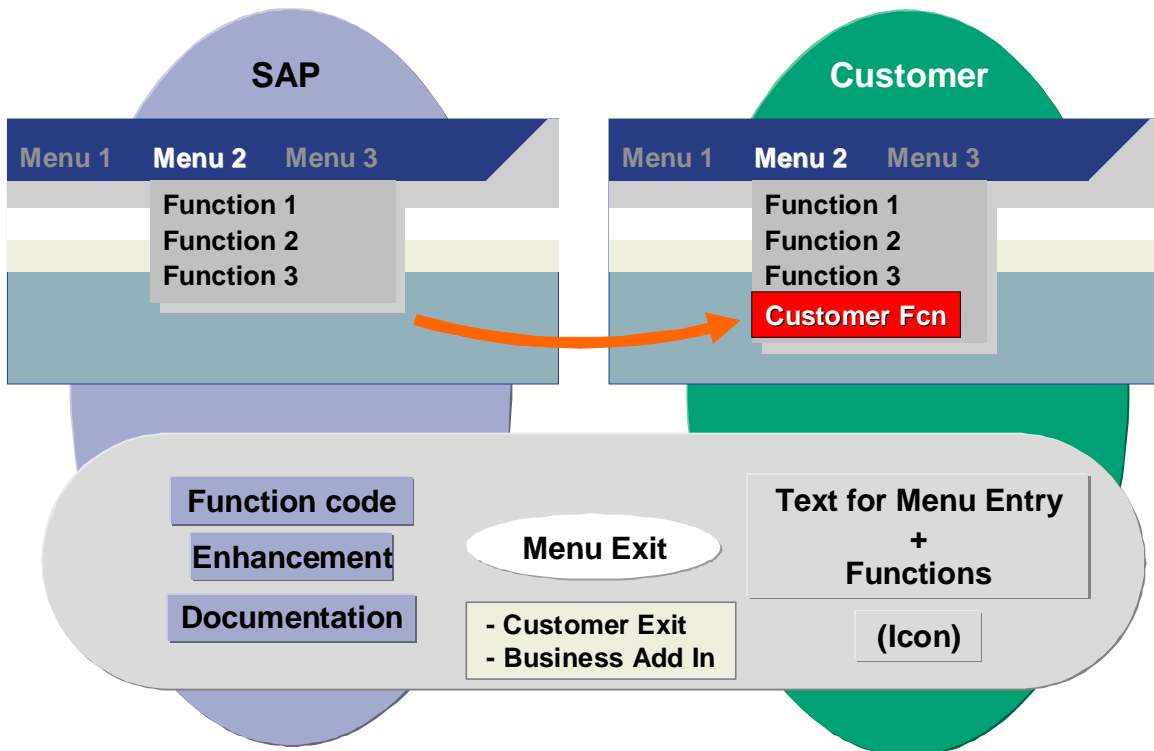


© SAP AG 2006

- Program enhancements permit you to execute additional program logic for an SAP application program. SAP currently provides the techniques outlined above.
- The advantages and restrictions of the particular enhancement techniques will be discussed in detail in later units.

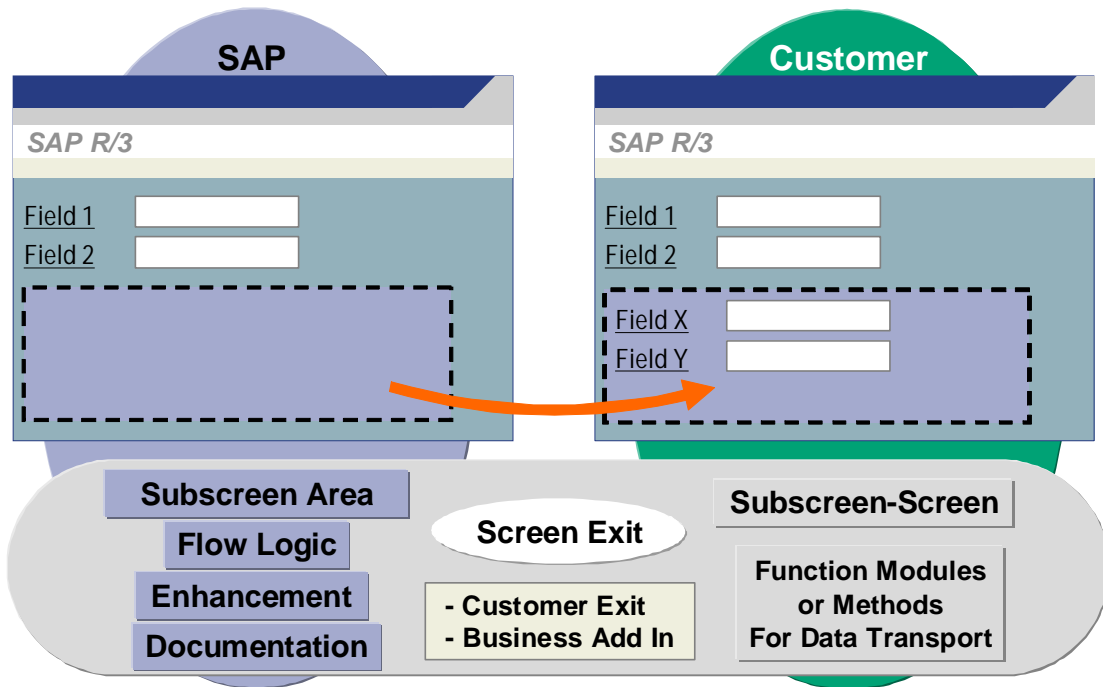
Menu Enhancements: SAP and Customer

SAP



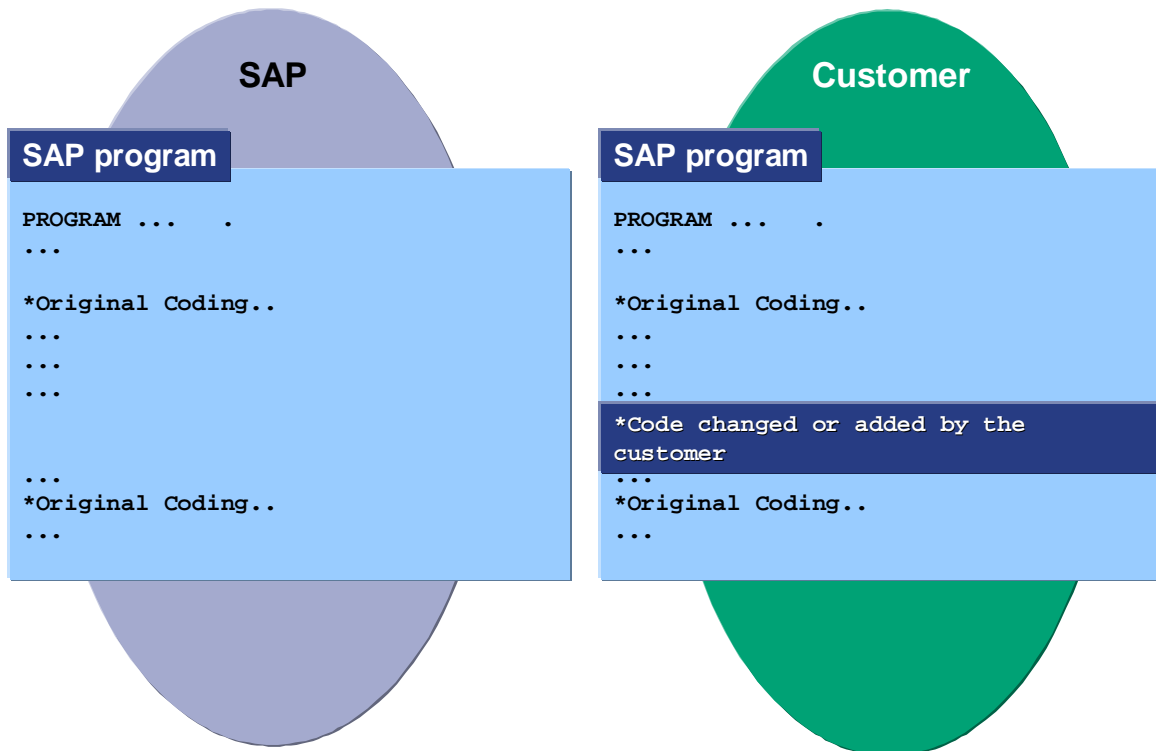
© SAP AG 2006

- Menu enhancements permit you to add additional menu entries to an SAP standard menu. The system currently provides two options:
 - Customer Exits
 - Business Add-Ins
- The additional menu entries are merged into the GUI interface.
- When the function code is implemented, you can change the text of the menu entry and change the icons - if the SAP developer provided for it.



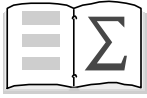
© SAP AG 2006

- Screen exits are a type of customer exit. They allow you to display additional objects in an SAP application program screen. The SAP developer must:
 - Define the subscreen areas
 - Specify the corresponding calls in the flow logic
 - Provide the framework for the data transport
 - Include the screen exit in an enhancement
 - Maintain the documentation!
- As of SAP Web Application Server 6.20, Business Add-Ins can also contain screen exits.
- You can implement screen exits by creating subscreens, possibly with flow logic. You also have to implement the data transport.
- How you can implement screen exits will be discussed in two later units in this course: Enhancements using Customer Exits (for classical screen exits) and Business Add-Ins for the new screen exits).



© SAP AG 2009

- Any change that you make your system to an object that has been delivered by SAP is known as a modification.
- Modifications can lead to complications at upgrade. When SAP delivers a new version of the object, you must decide whether the new object should be used, or whether you want to continue using your old object.
- Prior to Release 4.0B, modifications were only recorded at Repository object level (for example, an include program).
- Since Release 4.5A, the granularity for recording modifications has been finer. This has been made possible by the Modification Assistant, which is discussed in the Modifications unit of this course.
- The modification adjustment process has also been overhauled. How modifications are adjusted is also discussed in the Modifications unit.









You are now able to:

- **Describe the different levels at which you can make changes to the standard delivered by SAP.**
- **Choose the most suitable method for changing the standard**
- **Enumerate the available enhancement types and explain their uses**

© SAP AG 2006

Data Used

Key to icons in the exercises and solutions

	Exercises
	Solutions
	Objectives
	Business scenarios
	Hints and tips
	Warning or caution

Data used in exercises

Type of data	Data in training system
Tables	SFLIGHT00 .. SFLIGHT18
Data elements	S_CARRID00 .. S_CARRID18
Programs	SAPBC425_EXIT_00 .. SAPBC425_EXIT_18
Transaction codes	BC425_00 .. BC425_18
Programs	SAPBC425_BOOKING_00 .. SAPBC425_BOOKING_00

Contents:

- **Personalizing Transactions
Using Transaction Variants**

© SAP AG 2009

Internal Use SAP Partner Only

Internal Use SAP Partner Only



At the conclusion of this unit, you will be able to:

- **Personalize transactions using**
 - **Transaction variants**
 - **GuiXT**

© SAP AG 2006

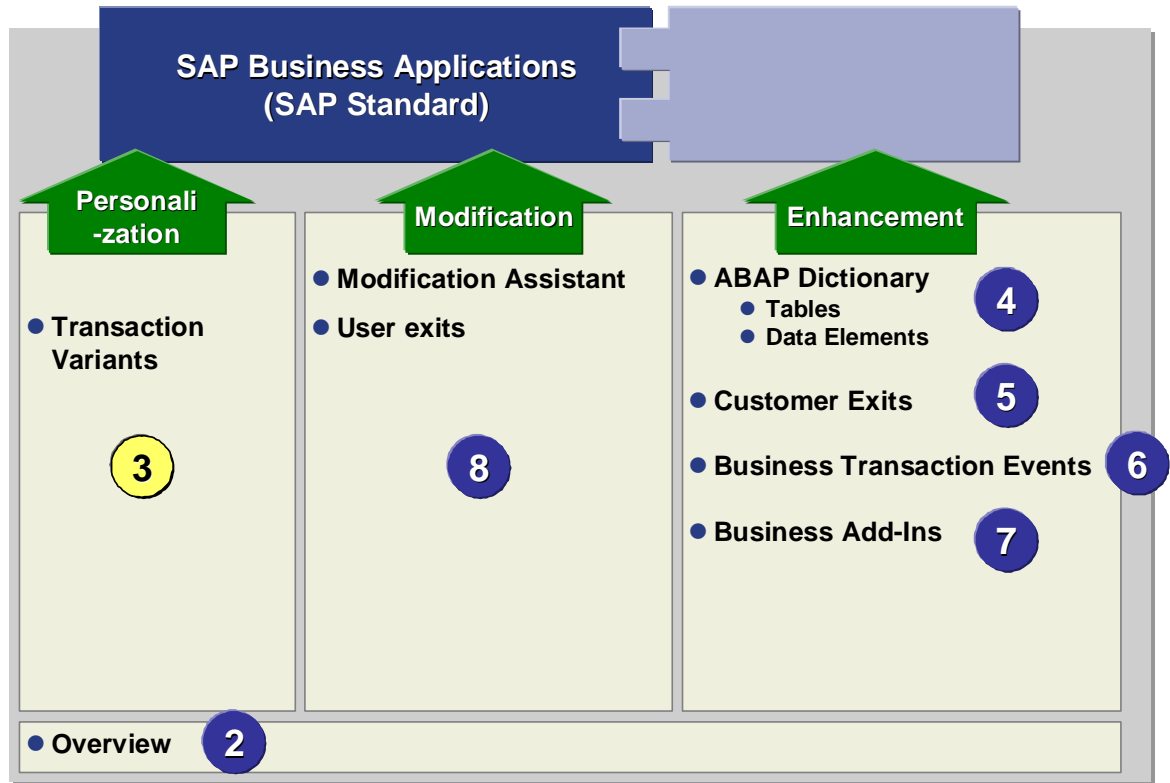


- **Your end users only use a small part of the functions provided by the SAP System. To simplify their daily work, you should set up their workplace to correspond to their needs.**
- **Your colleagues should be able to execute simplified transactions. This can usually be done with transaction variants.**

© SAP AG 2006

Overview Diagram: Personalization

SAP



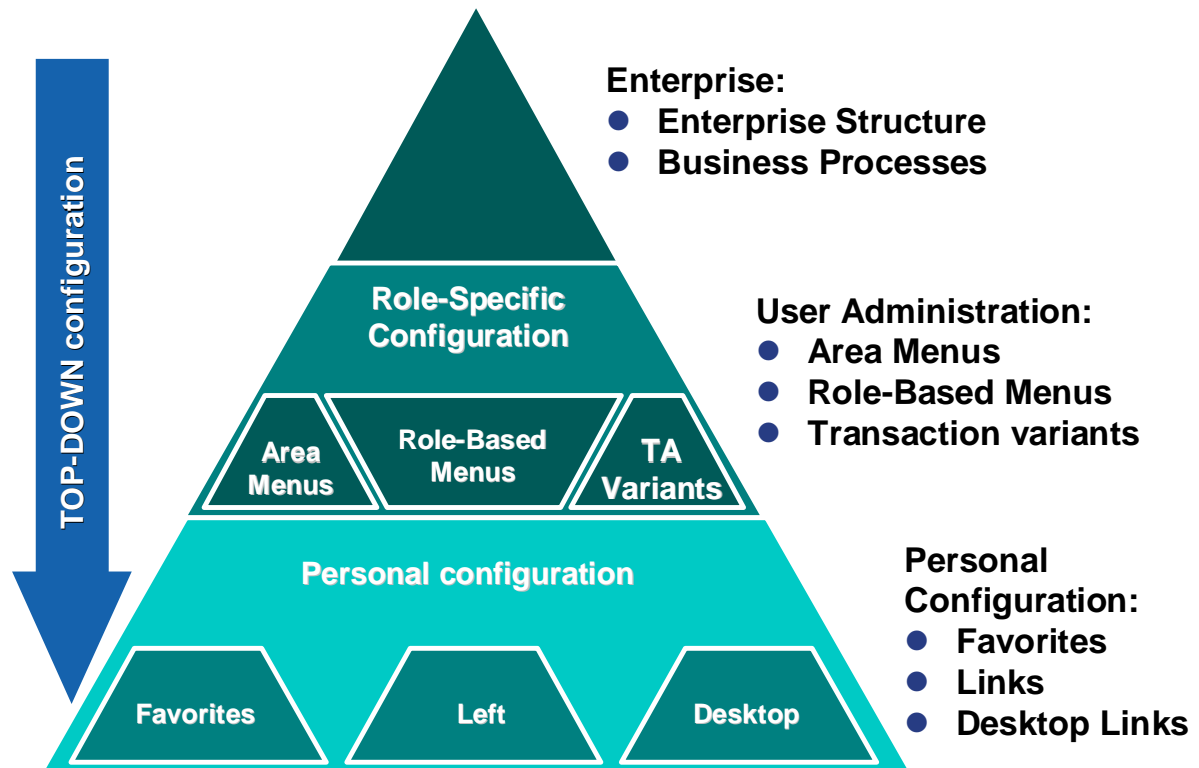
© SAP AG 2006

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Personalization Levels

SAP



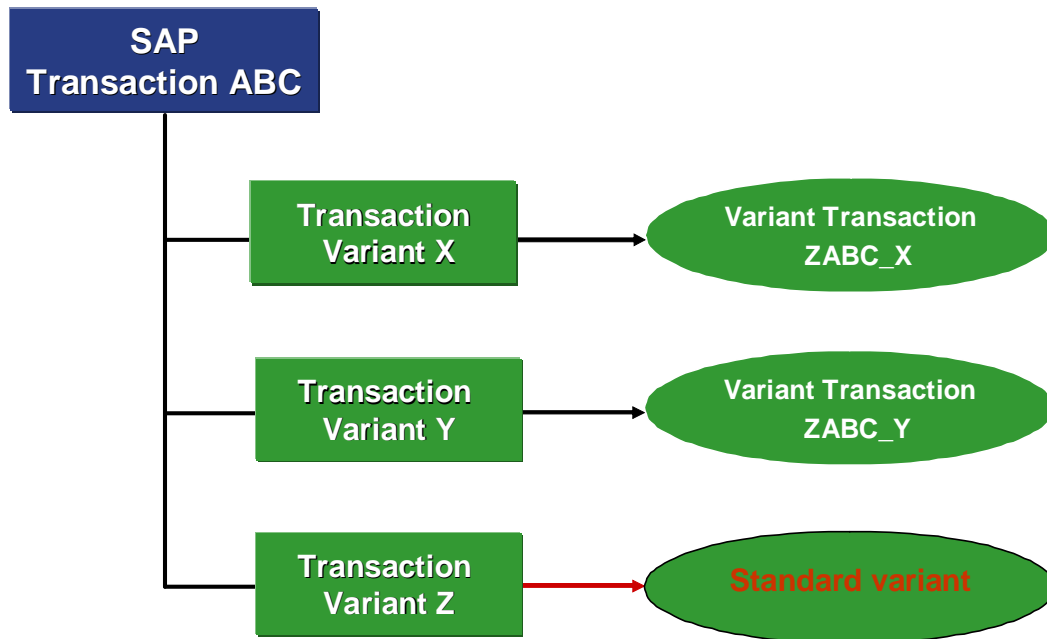
© SAP AG 2006

- The SAP system adjusts itself to the user's working style: When the system starts, it only offers the user those functions that are used in everyday work. There is no unnecessary navigating through functions that are not used. In the past, user menus could be called in the Session Manager or in the dynamic menu in R/3. As of Release 4.6A, role-based menus appear in the form of a tree at logon.
- When a function is selected, it is started in the same session. The executed function replaces the role-based menu. The role-based menu appears again automatically when you leave a transaction or when you start a new session.
- On the role maintenance screen (transaction PFCG), the administrator can combine the menu structure for a role (consisting of transactions, reports, and Internet/Intranet links) with a user menu. You can choose any structure and description for the functions contained.
- The enterprise menu is no longer available as of Release 4.6A.

- **Simplification of transactions by:**
 - **Predefining screen fields**
 - **Revoking ready for input status**
 - **Suppressing screen elements that are not needed (fields, subscreens, screens)**
- **WYSIWYG maintenance with special recording function**
- **Individual Variants and/or Standard Variants**

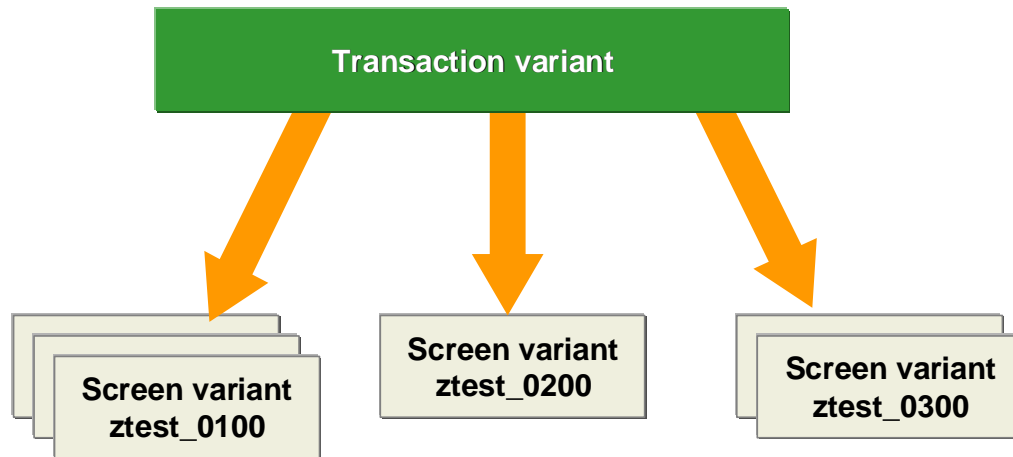
© SAP AG 2006

- You can reduce the complexity of the system by suppressing functions that are not required. Transaction variants allow you to reduce the complexity of complete transactions for users. You can
 - Predefine fields with values This predefinition can be overwritten by the user.
 - Remove the ready for input status from fields that require no user input;
 - Suppress screen elements that are not required This may include input fields or other screen elements.
 - Suppress entire screens.
- We will now discuss the maintenance of transaction variants.



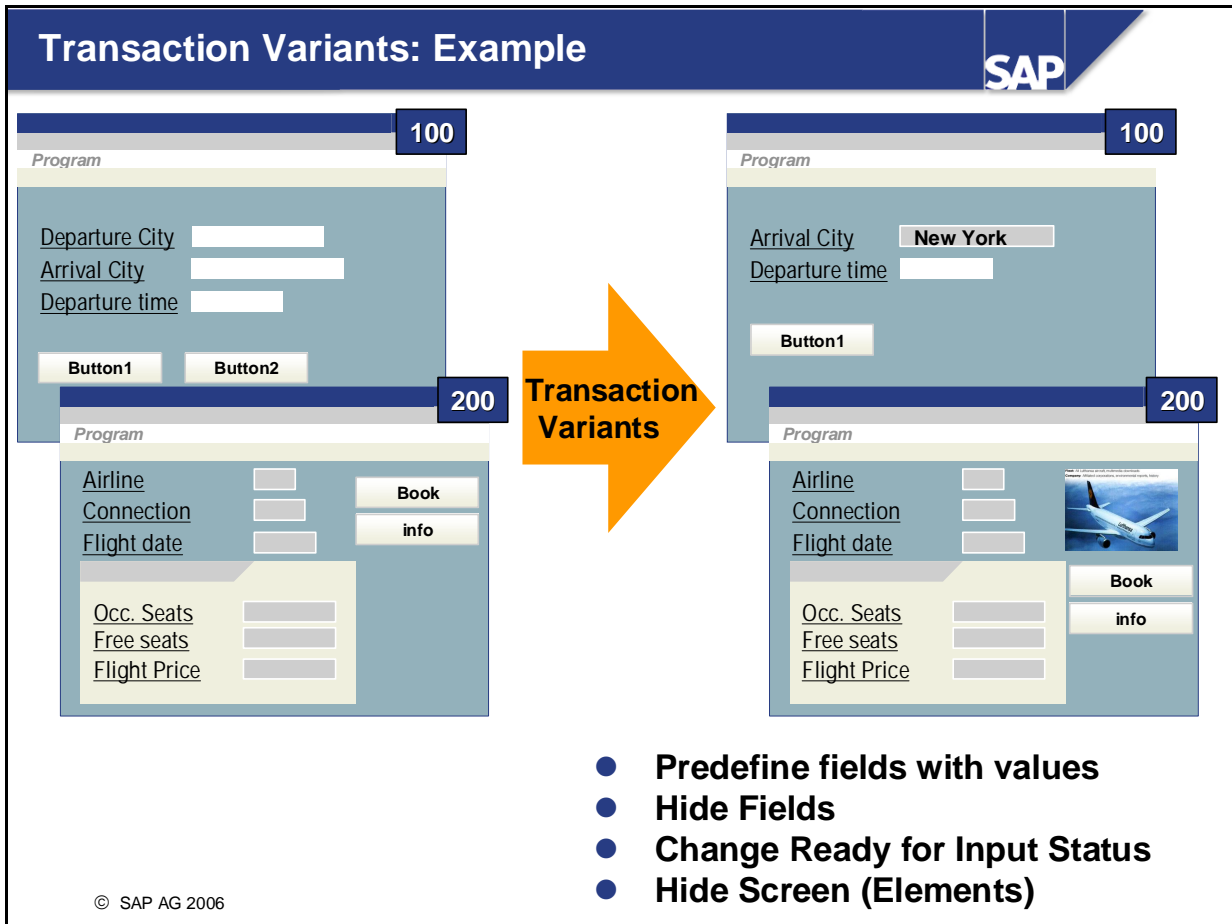
© SAP AG 2006

- You can use a variant defined for an SAP transaction as follows:
 - a) Create a variant transaction, which consists of the SAP transaction and the variant.
or
 - b) Declare the variant as a standard variant of the SAP transaction.
- For a) the user must call the variant transaction specifically, to access the variant.
- For b) on the other hand, the variant is "attached" to the SAP transaction so that when the SAP transaction is called, the system automatically uses the variant.
You can define exactly one standard variant for any given SAP transaction.



© SAP AG 2006

- A transaction variant is a reference to a set of screen variants.
- You can create any number of screen variants for a screen. The transaction variant consists of these screen variants.



- In this example you see two screens of an SAP transaction that you want to redesign using a transaction variant.
- Screen 100 is changed as follows: Fields are hidden; Field attributes are changed; Buttons are suppressed or hidden.
- Screen 200 shows the following changes: buttons moved and screen inserted (with GuiXT). We will discuss the use of GuiXT further below.

Transaction variants

Transaction Code

Transaction Variant

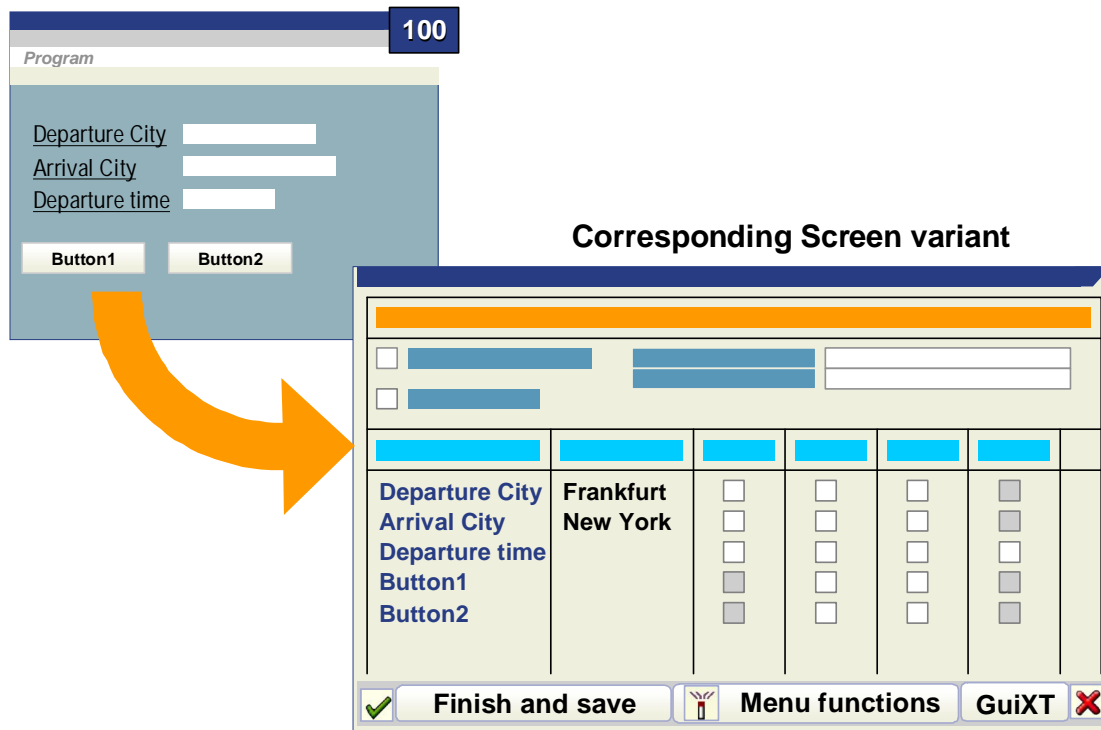
Transaction SHD0

Name of the SAP Transaction

Variant Name

© SAP AG 2006

- To create transaction variants, choose the component *Personalization* from the entry *AcceleratedSAP* in the SAP menu and then choose *Transaction Variant*. You go to the transaction for maintaining transaction variants.
- Enter the name of the transaction from which you want to create a variant. The name of the variant must be unique in the system and be in the customer namespace.
- You can use the Goto menu option to choose if you want to create a client-specific or a cross-client transaction variant.
- To create the variant, choose the corresponding button in the application toolbar.



© SAP AG 2006

- Pressing "Screen entries" starts the transaction in CALL mode.
- Triggering a dialog also triggers PAI of the current screen. The system sends another screen in which you can evaluate the fields that are on the screen.
- Also read the online documentation about transaction variants.
- The screen that was evaluated is stored as a screen variant when you continue. This will be discussed in the following slides.

Attributes of a Screen Variant

SAP

Screen variant

Screen Variant Name		Short Text		Attributes of the Screen Elements	
Departure City	Frankfurt	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Arrival City	New York	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Departure time		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Button1		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Button2		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

☒ **Finish and save**
☐ **Menu functions**
☐ **GuiXT**
☒ **Deactivate Menu Functions**

© SAP AG 2006

- A screen variant is an independent Repository object, which has a unique name in the system. The name is constructed as follows:
 - Variant name
 - Client (only for client-specific transaction variants)
 - Screen number
- You define here if field contents should be copied to the screen variant. You can set various attributes for the individual fields: You can remove the ready for input status of a field, or make it invisible. You can find a detailed list of the possibilities in the online documentation about transaction variants.

Testing Variants:

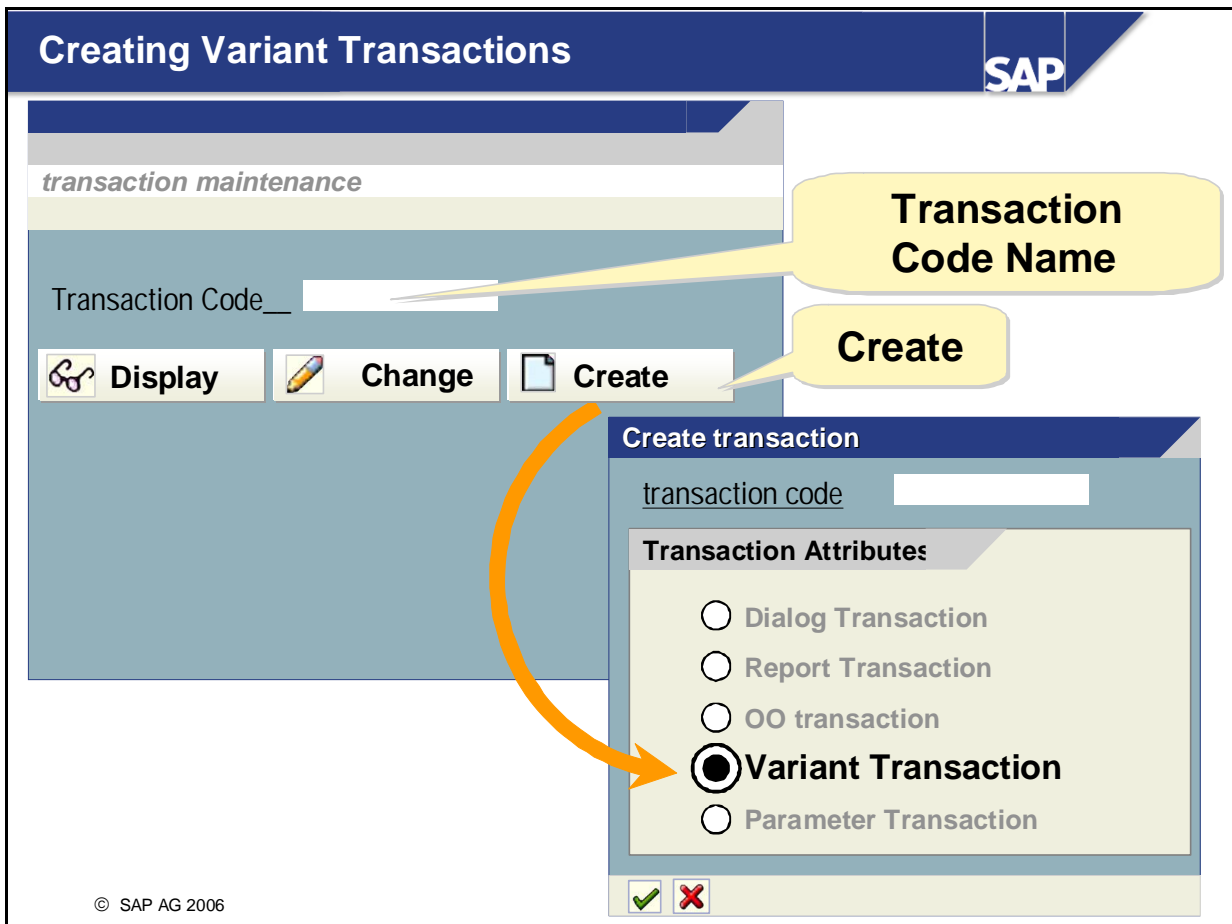
1. Enter the SAP transaction and variant name in SHD0
2. Press F8

Using a Variant:

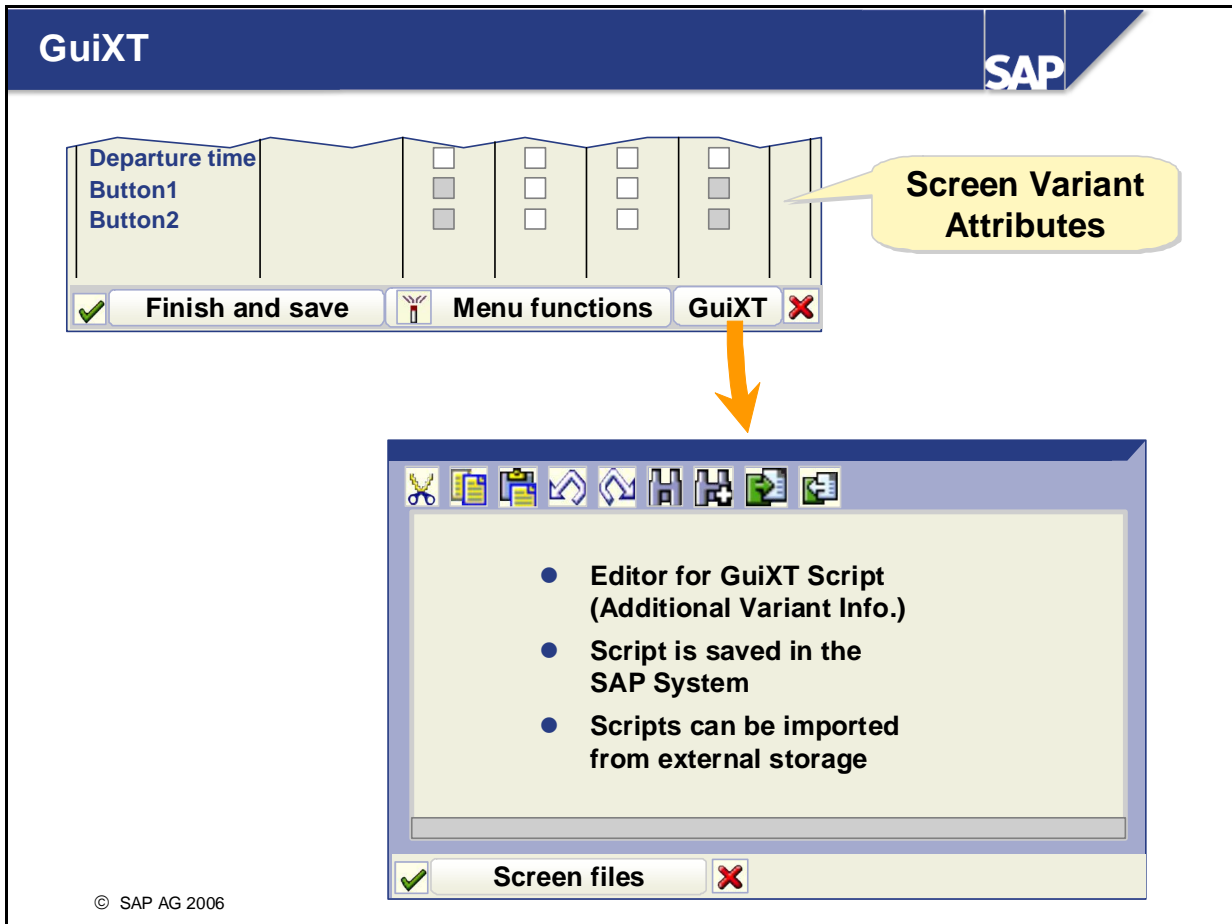
1. Create the variant transaction using SHD0
(Menu Goto → Create Variant Transaction)
2. Call the variant transaction

© SAP AG 2006

- You have the following options for starting a transaction variant:
 - Test Environment
 - Transaction code of type "variant transaction"
 - User Menu
- You can test the process flow of the transaction in the test environment of transaction variant maintenance. This is intended primarily for developers creating transaction variants.
- To hang a variant transaction in a user menu or role, you must create a transaction code of type "variant transaction".



- To start a transaction variant from a menu, you must create a transaction code of the type "variant transaction". You can find a link there directly in the maintenance screen for the transaction variants, under the menu "Goto". Alternatively you can create a transaction code from the ABAP Workbench. If you choose this approach, remember to choose the correct initial object, Variant Transaction.



- The supplementary tool GuiXT permits you to design the individual screens in a more flexible manner. GuiXT uses a script language to
 - Position objects on the screen,
 - Set attributes,
 - Include new objects.
- If you press "GuiXT", an editor window appears for entering the script. You can also choose graphic files stored on your local machine.
- You can also import scripts created on the local machine and export them there.

```
// Version: 19990921151118
```

Comment

```
IMAGE (1,1) "C:\temp\sap.jpg"
```

Inserting a Graphic

```
BOX (10,20) (16,44) "Frame"
```

Insert frame

```
POS [Element] [Element]+(10,0)
```

Move element

```
POS [Area] [Area]+(10,0)
```

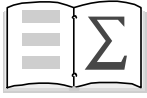
Move area

```
Pushbutton (10,50) "Text" "fcode"
```

Pushbutton with text and function code.

© SAP AG 2009

- You can use the GuiXT scripting language to change the layout of a screen. You can
 - Move objects
 - Insert pictures
 - Insert pushbuttons
 - Insert value helps
 - Change the input attributes of fields
 - Delete screen elements
- You are provided with a complete documentation of GuiXT with the installation. You can find more information on the homepage of the GuiXT vendor (<http://www.synactive.com>).



You are now able to:

- **Personalize transactions using**
 - **Transaction variants**
 - **GuiXT**

© SAP AG 2006

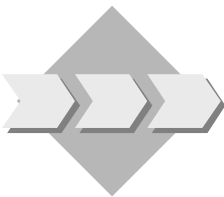
Exercises



Unit: Personalization Topic: Creating a Package



For correct development you need a package.



- 1-1 Create a package.
 - 1-1-1 The package should be named ZBC425_## (where ## is your group number)
 - 1-1-2 Assign the package to a change request.

Exercises

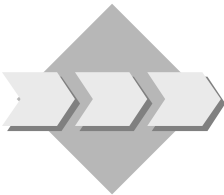


Unit: Personalization Topic: Transaction Variants



At the conclusion of these exercises, you will be able to:

- Use screen variants and transaction variants to simplify the use of a transaction.



Your users complain that Transaction **BC425_TAVAR** is much too difficult to use (despite the Enjoy initiative). In fact, they only need to fill in a few of the many fields. You now want to simplify the use of the transaction, by hiding the fields that are not used.

- 2-1 Create a transaction variant for transaction **BC425_TAVAR**.
 - 2-1-1 How do you get to the maintenance screen for transaction variants?
Include the corresponding transaction in your list of favorites.
 - 2-1-2 Give the variant a name: **ZBC425##** (where ## is your group number).
- 2-2 Go through the transaction screen by screen and create a screen variant for each of the screens. You should make the following changes:
 - 2-2-1 Initial screen: Initialize the first two fields with "DE", "Frankfurt" and set them to "Output only".
 - 2-2-2 Second screen: Set column "Air" of the table control to *invisible*.
Deactivate menu function "BACK".
 - 2-2-3 Third screen: Deactivate menu function "BACK".
- 2-3 Create a transaction code for the variant.
Call your transaction **ZBC425##**.
- 2-4 Test your results.



Unit: Personalization Topic: Creating a Package

- 1-1 You create a package by choosing the following menu path in the SAP menu:
Tools → ABAP Workbench → Overview → Object Navigator.
- 1-1-1 Alternatively you can choose transaction SE80.
 - 1-1-2 Choose the menu path ***Workbench → Edit Object...***
 - 1-1-3 On the *Development Coordination* tab, choose *Package* and enter the name of the package you want to create in the appropriate field.
 - 1-1-4 Choose *Create*.
 - 1-1-5 On the next screen, enter a meaningful short description for your package. Assign the package to the software component Home. Confirm your entries.
 - 1-1-6 On the next screen, assign a change request and confirm it.
 - 1-1-7 You will assign all your subsequent development objects to this package that you have just created.



Unit: Personalization

Topic: Transaction Variants

- 2-1 Create a transaction variant for transaction **BC425_TAVAR**:
- 2-1-1 You maintain transaction variants in Transaction SHD0
In Transaction SHD0, choose
System → User profile → Expand favorites to include the transaction in your list of favorites.
(Alternatively, you can add SHD0 to your favorites using the context menu of the favorites-folder on the initial screen.)
 - 2-1-2 Enter the name of the transaction from which you want to create a variant in the field *Transaction*. Enter the name of the variant in the field *Variant*: **ZBC425##** (where ## is your group number).
- 2-2 Execute the transaction screen by screen. Enter the corresponding values in the input fields. Leave the screen with the appropriate pushbutton and create a screen variant for each of the screens.
- 2-2-1 Initial screen: Assign the first two fields the values "DE" and "FRANKFURT". Leave the screen by choosing the appropriate pushbutton. In the next popup window select the checkbox *With content* and *Output only* in the corresponding checkboxes for the screen objects. Give the screen variant a short description: Save the screen variant.
 - 2-2-2 Second screen: Leave the screen by choosing the appropriate pushbutton. Choose *With content* again in the next dialog box. Select *Output only* for the column "Airport" of the table control. Choose the pushbutton for menu functions and deselect function code "BACK". Save the screen variant.
 - 2-2-3 Third screen: Leave the screen using the *Save* function. Deactivate menu function "BACK" as in 1-2-2. Save the screen variant.
 - 2-2-4 A list with a summary of all the screen variants that were created appears. You can now check your entries again. Save them to finally create the transaction variant.
- 2-3 Choose the menu path *Goto → Create transaction* from transaction SHD0.

Contents:

- Append structures
- Customizing includes
- Text enhancements

© SAP AG 2002



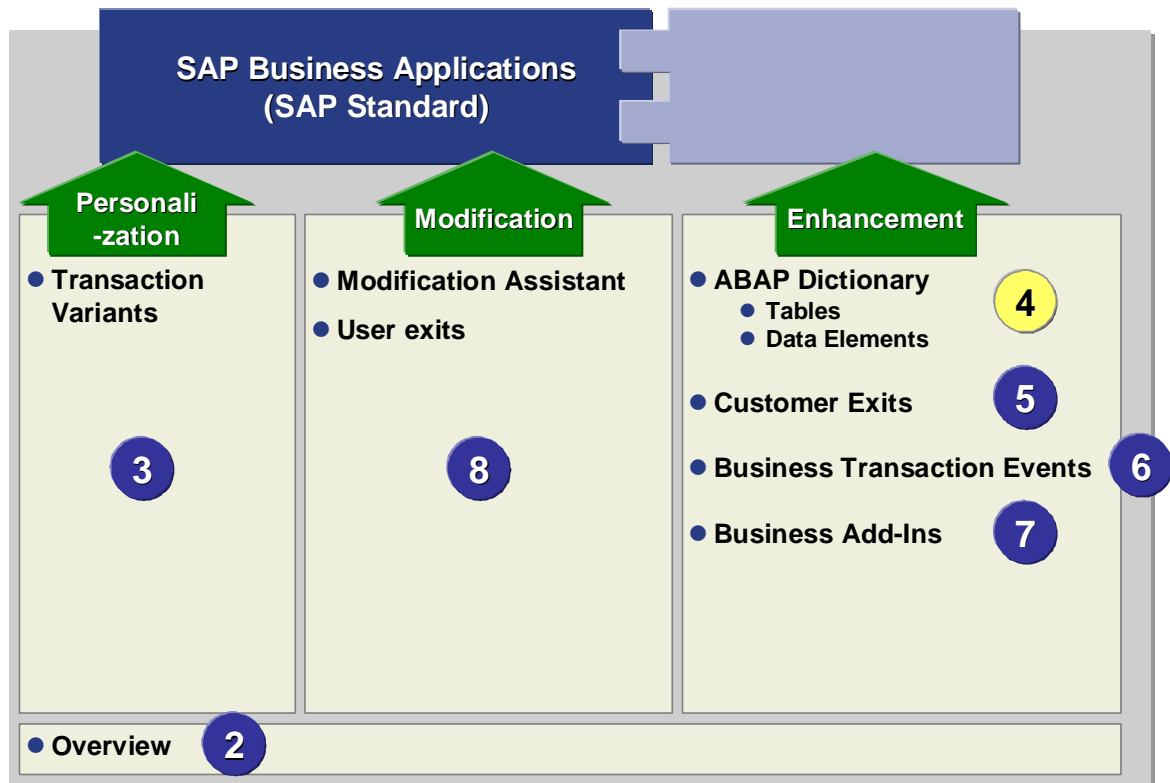
At the end of this unit, you will be able to:

- Enhance tables using append structures
- Enhance tables using Customizing includes
- Change field labels and documentation for SAP data elements without carrying out a modification

© SAP AG 2009

Enhancements to the ABAP Dictionary: Overview Diagram

SAP



© SAP AG 2006



At the conclusion of this topic, you will be able to:

- Enhance tables using append structures
- Enhance tables using Customizing includes

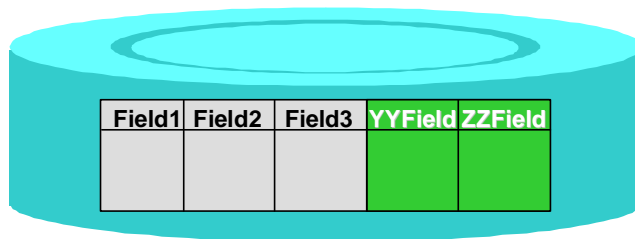
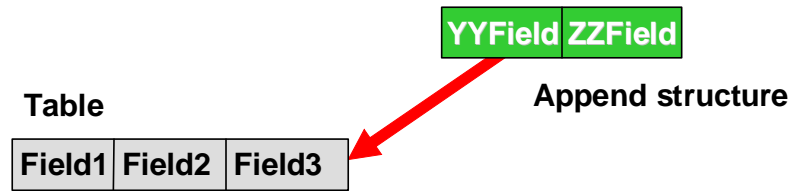
© SAP AG 2002

There are two ways that you can add extra fields to an SAP table without a modification:

- **Append structure**
 - Customers can create an append structure for an SAP table (without SAP preparation)
 - Multiple append structures can be used with a single SAP table
 - They can be used as normal structures in programs
- **Customizing include**
 - Is already integrated into SAP tables by SAP
 - Customer fills it with the desired additional fields
 - May contain source code or screen exits provided by SAP for processing or displaying the fields

© SAP AG 2009

- Tables and structures can be expanded in one of two different ways:
- Append structures allow you to enhance tables by adding fields to them that are not part of the standard. They allow you to add customer-specific fields to any table or structure you want.
- Append structures are created for use with a specific table. However, a table can have multiple append structures assigned to it.
- If it is known in advance that one of the tables or structures delivered by SAP needs to have customer-specific fields added to it, the SAP application developer includes these fields in the table using a Customizing include statement.
- The same Customizing include can be used in multiple tables or structures. This ensures consistency in these tables and structures whenever the include is extended.
- Nonexistent Customizing includes do not lead to errors.

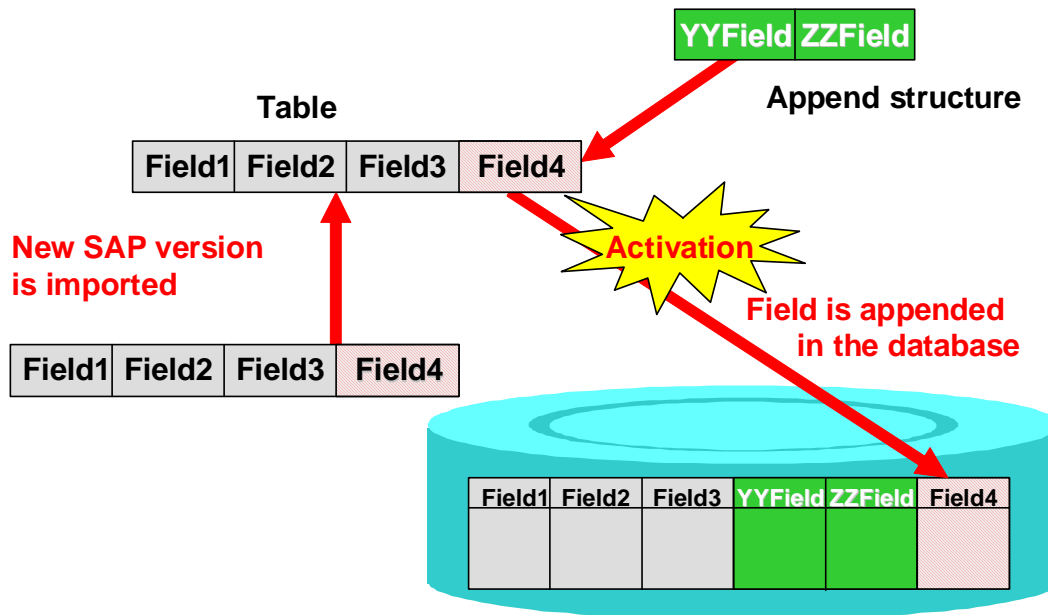


© SAP AG 2009

- Append structures allow you to attach fields to a table without actually having to modify the table itself.
- Append structures may only be assigned to a single table. A table may, however, have several append structures attached to it. Whenever a table is activated, the system searches for all active append structures for that table and attaches them to the table. If an append structure is created or changed and then activated, the table it is assigned to is also activated, and all of the changes made to the append structure take effect in the table as well.
- You can use the fields in append structures in ABAP programs just as you would any other field in the table.
- If you copy a table that has an append structure attached to it, the fields in the append structure become normal fields in the target table.

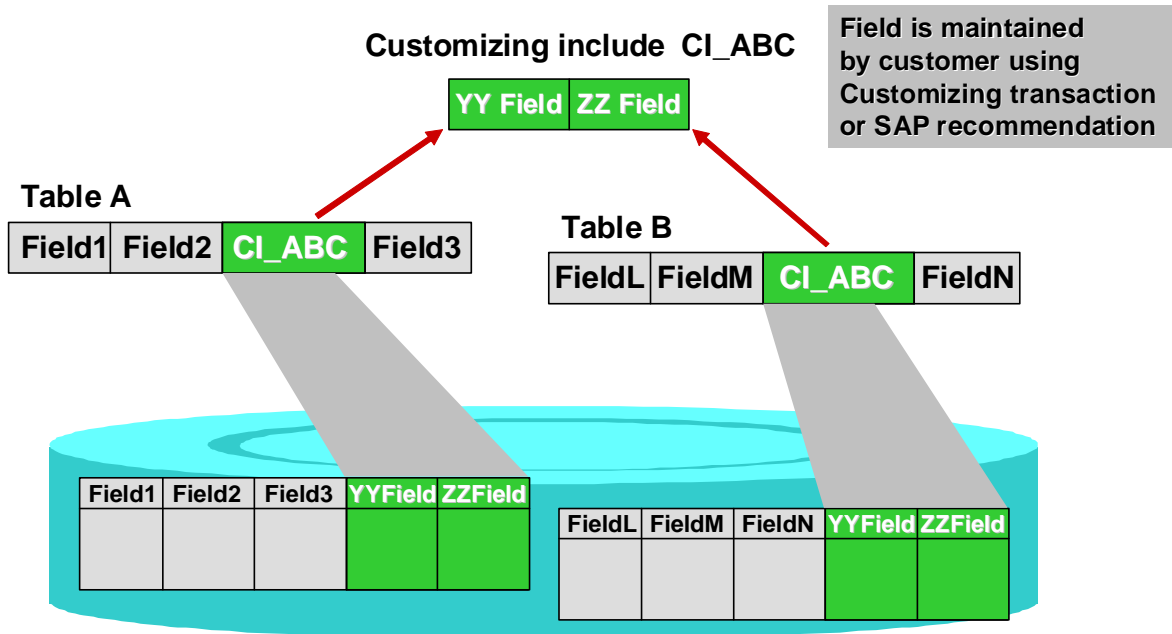
Append Structures at Upgrade

SAP



© SAP AG 2002

- You create append structures in the customer namespace. This protects them from being overwritten at upgrade or during release upgrade. New versions of standard tables are loaded during upgrades. The fields contained in active append structures are then appended to the new standard tables when these new standard tables are activated for the first time.
- From Release 3.0, the field sequence in the ABAP Dictionary can differ from the field sequence in the database. Therefore, no conversion of the database table is necessary when adding an append structure or inserting fields into an existing one. All necessary structure adjustment is taken care of automatically when you adjust the database catalog (**ALTER TABLE**). The table's definition is changed when it is activated in the ABAP Dictionary and the new field is appended to the database table.
- Pay attention to the following points when using append structures:
 - You cannot create append structures for pool and cluster tables.
 - If a table contains a long field (either of data type **LCHR** or **LRAW**), then it is not possible to expand the table with an append structure. This is because long fields of this kind must always be the last field in their respective tables. No fields from an append structure may be added after them.
 - If you use an append structure to expand an SAP table, the field names in your append structure must be in the customer namespace, that is, they must begin with either **YY** or **ZZ**. This prevents naming conflicts from occurring with any new fields that SAP may insert in the future.



© SAP AG 2009

- Some of the tables and structures delivered with the R/3 standard contain special include statements calling Customizing includes. These are often inserted in those standard tables that need to have customer-specific fields added to them.
- In contrast to append structures, Customizing includes can be inserted into more than one table. This provides for data consistency throughout the tables and structures affected whenever the include is altered.
- Customizing include programs are part of the customer namespace: all of their names begin with 'CI_'. This naming convention guarantees that nonexistent Customizing includes do not lead to errors. No code for Customizing includes is delivered with the R/3 standard.
- You create Customizing includes using special Customizing transactions. Some are already part of SAP enhancements and can be created by using project management (see the unit on 'Enhancements using Customer Exits').
- The Customizing include field names must lie in the customer namespace, just like field names in append structures. These names must all begin with either 'YY' or 'ZZ'.
- When adding the fields of a Customizing include to your database, adhere to same rules you would with append structures.



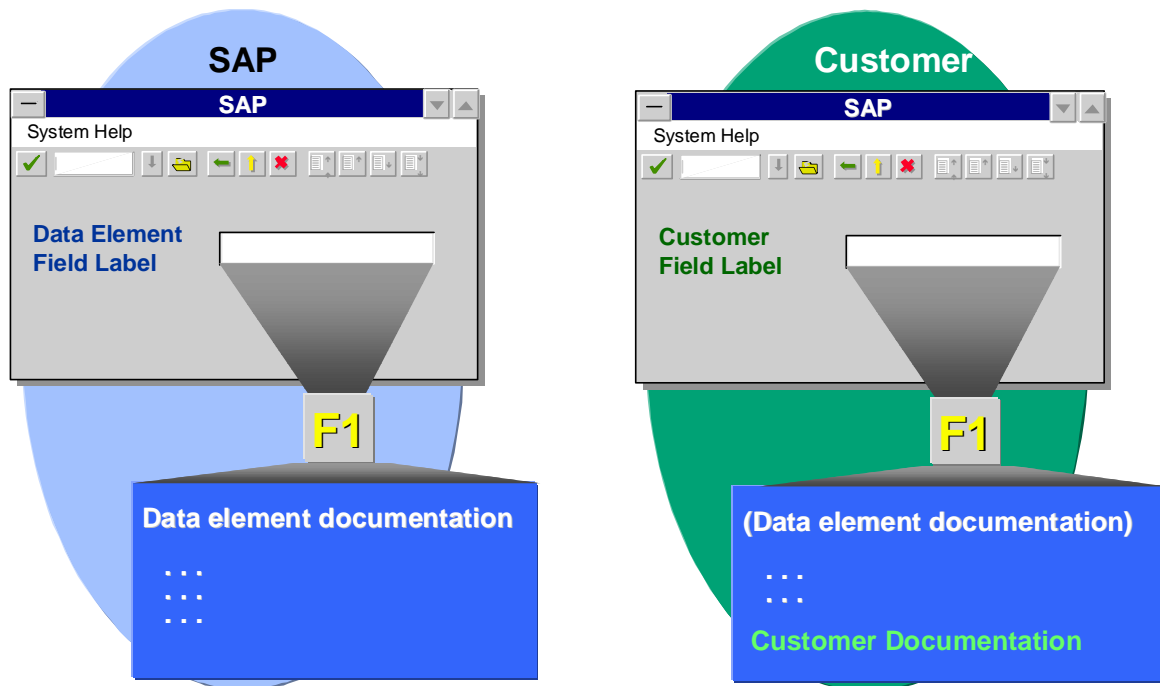
At the conclusion of this topic, you will be able to:

- **Change field labels and documentation for SAP data elements without carrying out a modification**

© SAP AG 2006

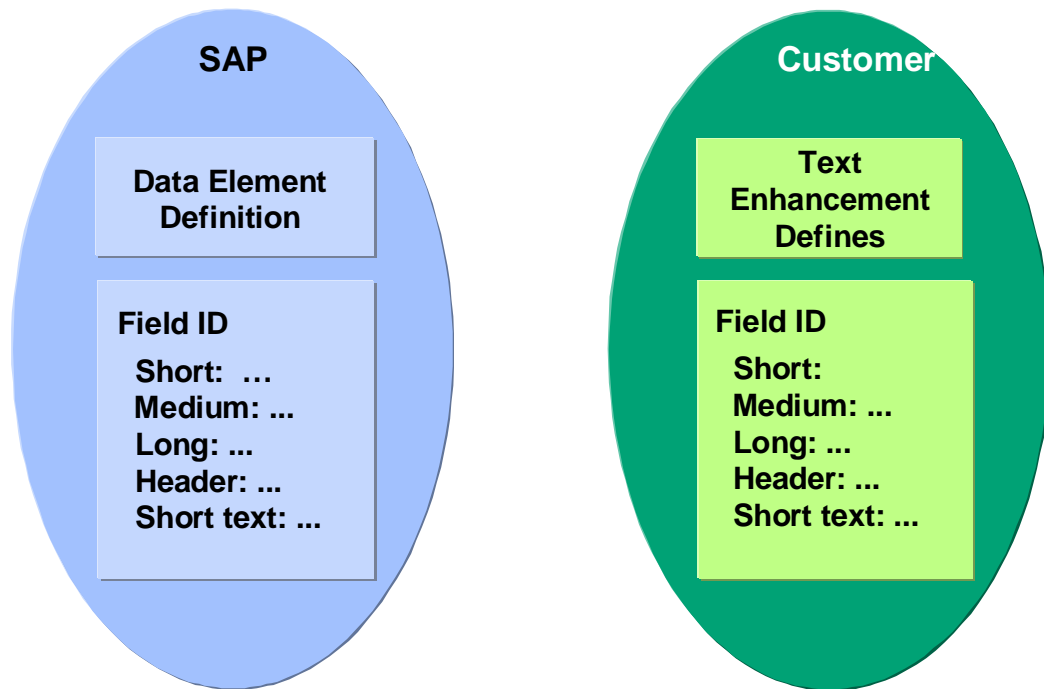
Text Enhancements: Overview

SAP



© SAP AG 2009

- Text enhancements comprise customer-specific field labels and documentation for SAP data elements.
- Text enhancements function in all SAP applications that use the affected data element (they are global enhancements)

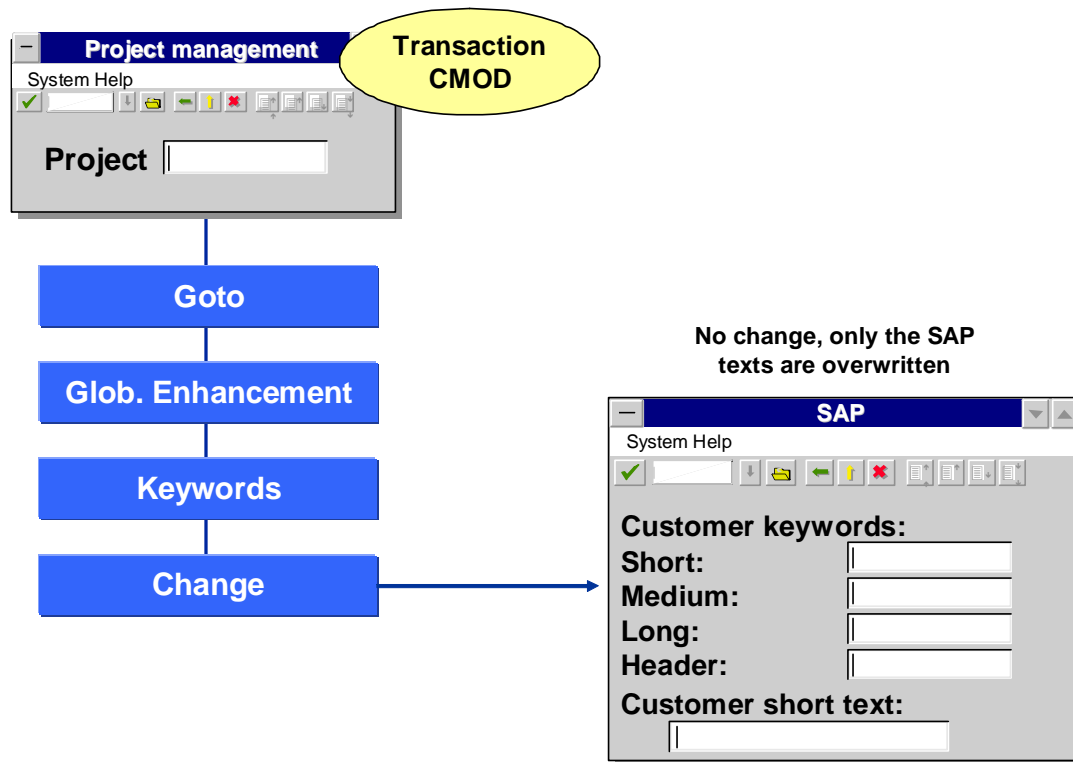


© SAP AG 2009

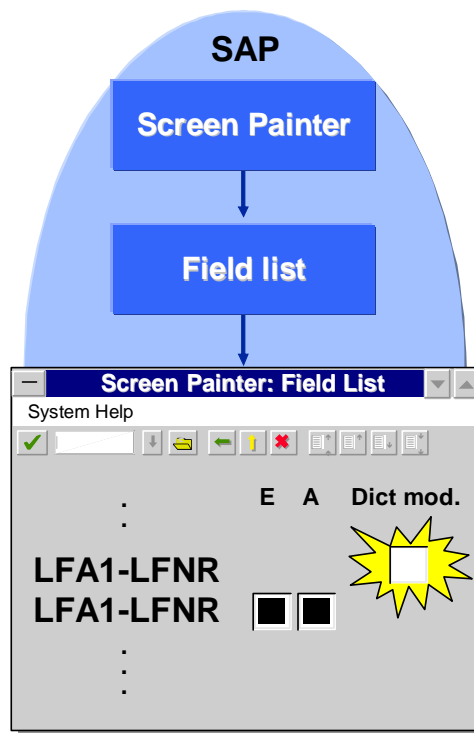
- SAP application programmers define field labels in different lengths and a short descriptions for each data element. Customers can overwrite these texts with customer-specific texts.
- New field labels can be provided for all screen fields in this way.

Overwriting SAP Field Labels (2)

SAP



- You edit text enhancements using the project management (ABAP Workbench → Utilities → Enhancements → Project Management) (that is, transaction CMOD).
- Choose "Goto -> Global Enhancements" to open the field label and documentation enhancement (overwriting) for SAP data elements.

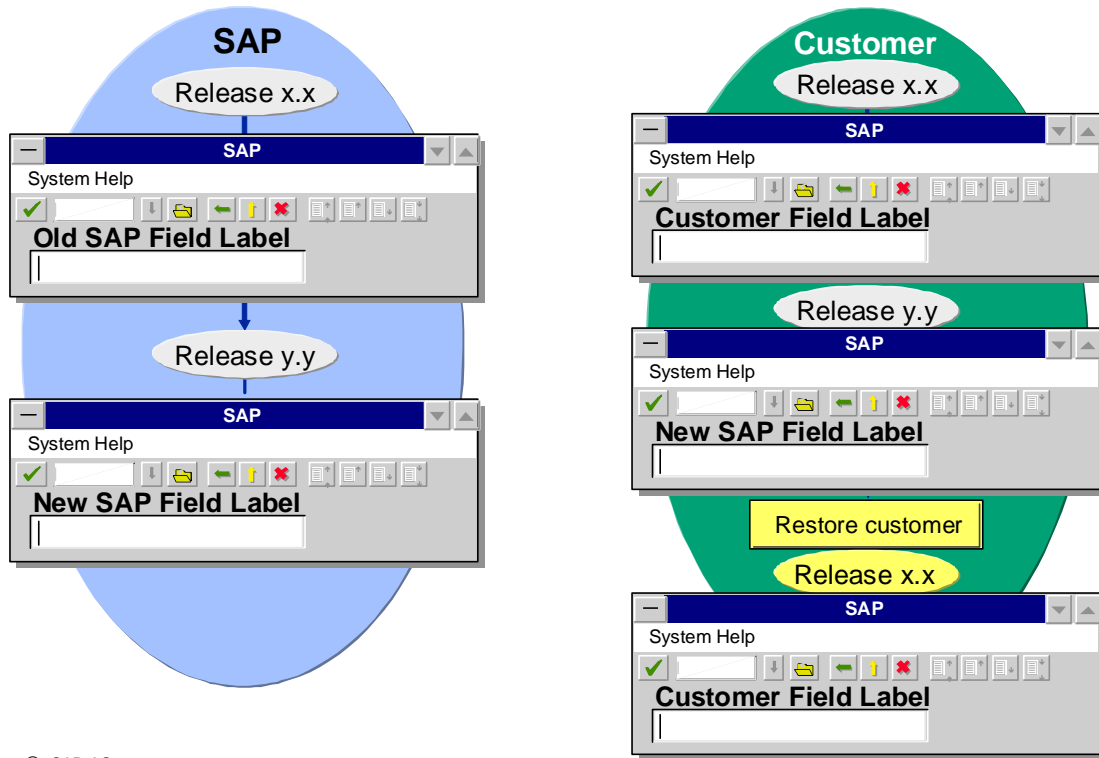


© SAP AG 2009

- You can only overwrite field labels for screen fields that are not explicitly assigned a description text by the screen's developer.
In such cases, the screen field's "Dict. modified" attribute has the value "F".
- The "Dict. modified" attribute can have the following values:
 - SPACE: The field label that best fits the field length
 - 1: Short field label
 - 2: Medium field label
 - 3: Long field label
 - 4: Field label for header
 - V: Variable text transfer from the dictionary (as SPACE)
 - F: Fixed - no text transfer from the dictionary.

Overwritten Texts in Upgrades

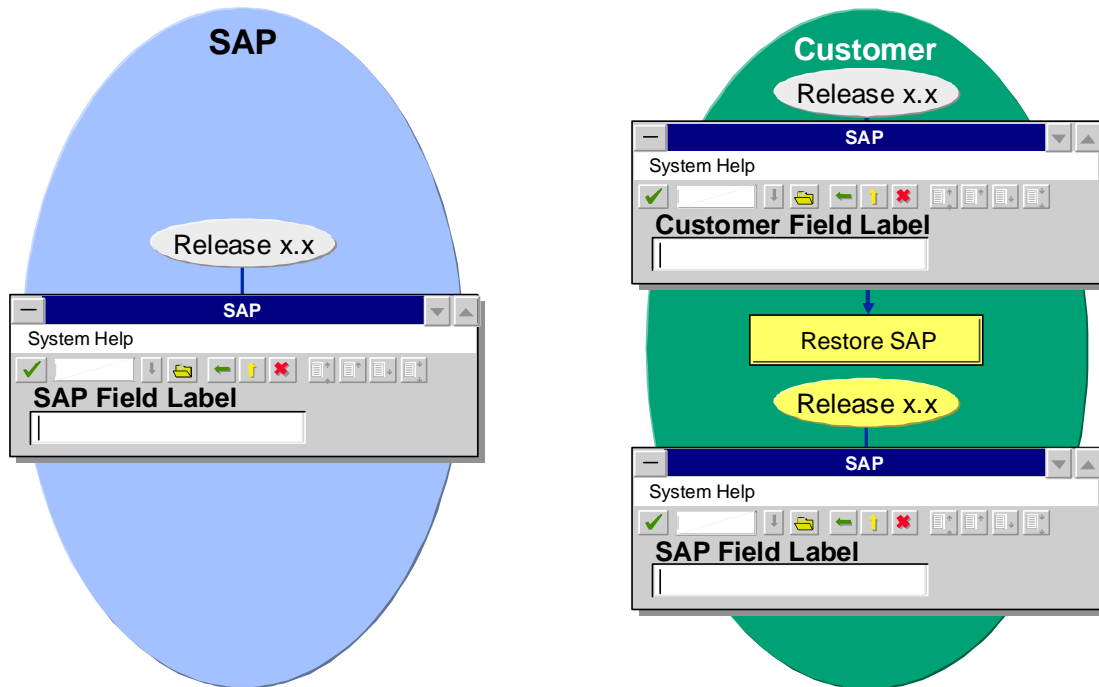
SAP



- If SAP has redelivered existing field labels, as part of a release upgrade or you import new corrections, you need to restore your customer field labels. If you want to retain your own field labels from the previous release, choose the menu option that restores customer field labels. SAP recommends that you always restore your field labels after a release upgrade.
- Field labels are restored by a program that runs in the background. This program checks all of the data elements that you have edited and restores their field labels if necessary.
- For central field labels such as the data elements BUKRS, MANDT and so on, we recommend that you start the restoration at a time when the tables that use BUKRS and MANDT are not being changed. Otherwise, activation of the data element may fail, changing the status to only partially active.

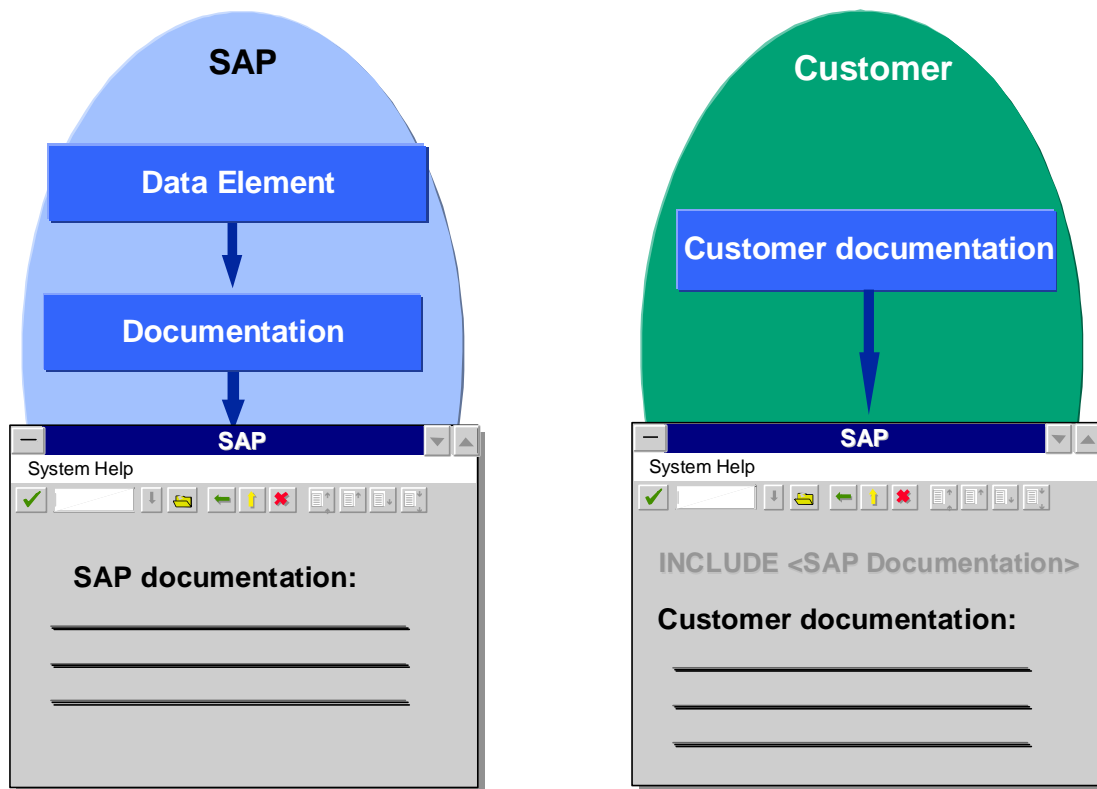
Restoring SAP Field Labels

SAP



© SAP AG 2009

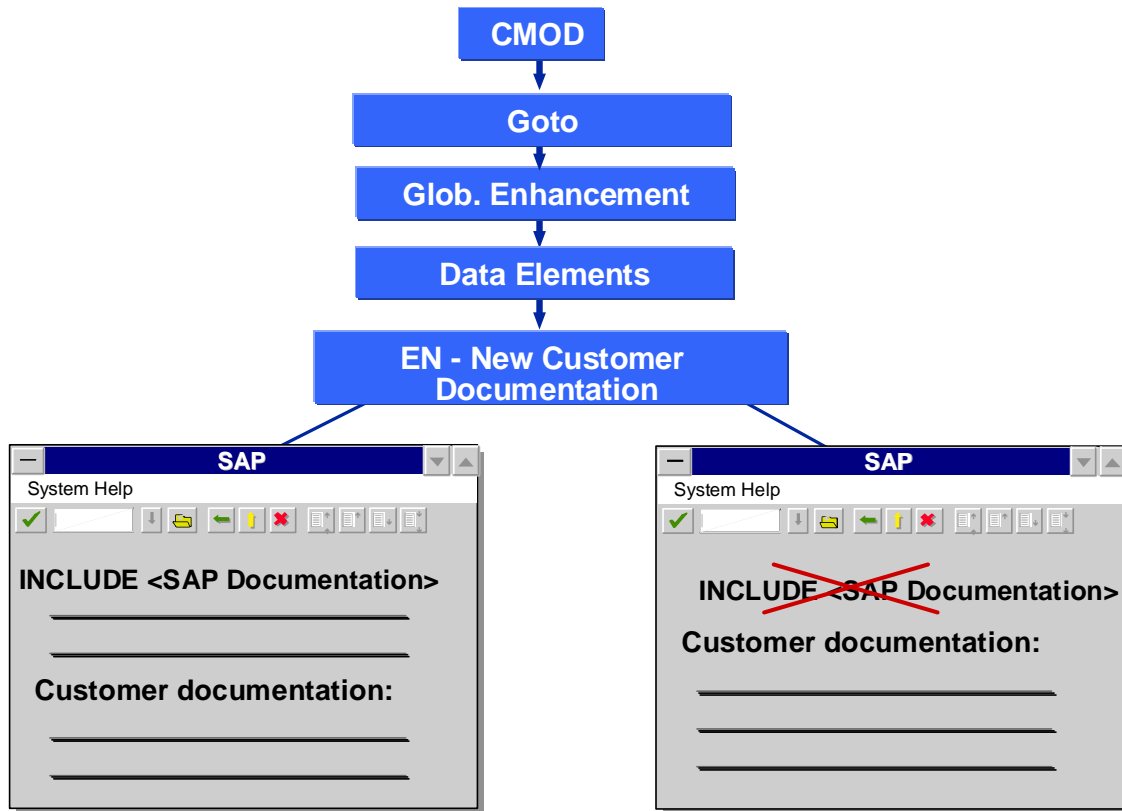
- To undo field label changes, choose the menu option Restore SAP.
- Field labels are restored by a program that runs in the background. This program checks all of the data elements that you have edited and restores their field labels if necessary.
- For central field labels such as the data elements BUKRS, MANDT and so on, we recommend that you start the restoration at a time when the contents of the tables that use BUKRS and MANDT are not being changed.



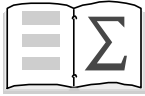
- When you enhance the data element documentation, you can copy the SAP documentation as well as your own. In this case, when the user presses F1 in the corresponding screen field, the system displays the SAP documentation as well as the customer-specific documentation.
- You can generate a list of data elements that have been changed and edit the customer documentation by selecting the elements' corresponding lines in the list.
- Simply delete your own documentation if you want original SAP documentation to be displayed.

Creating Customer Documentation

SAP



- The menu path described in the graphic above opens a dialog window in which you must choose between "Original text" and "Template". Choose the first option if you wish to add to the SAP Documentation. Choose the second option if you wish to create customer-specific documentation without including the text of SAP documentation.



You are now able to:

- Enhance tables using append structures
- Enhance tables using Customizing includes
- Change field labels and documentation for SAP data elements without carrying out a modification

© SAP AG 2006

Exercises



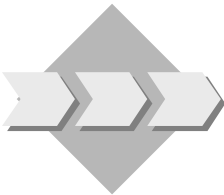
Enhancements to ABAP Dictionary Objects

Topic: Table enhancements



At the conclusion of these exercises, you will be able to:

- Enhance tables with append structures.



You work as a computer specialist for a large travel agency. Your fellow employees use transaction **BC425_##** to display flight information when helping customers. They would like more information about a flight, for example the pilot's name or the main meal.

Your flight data is stored in table **SFLIGHT##**. You need to add two columns to this table without modifying it.

- 1-1 How can you add these two fields to table **SFLIGHT##** without modifying it?
 - 1-1-1 How do you go about enhancing table **SFLIGHT##**?
 - 1-1-2 Enhance table **SFLIGHT##** with a technique that does not require modifications.
- 1-2 Create an append structure for table **SFLIGHT##** (name proposal **ZASFLIGHT##**).
 - 1-2-1 Include the following fields in the structure: Pilot Name (name proposal: **YYPILOT**, type assignment using data element **S_PILNAME**) Meal (name proposal **YYMEAL**, type assignment using data element **S_MEAL**)
- 1-3 Choose a category of Enhancement for the append structure before activating.



Enhancements to ABAP Dictionary Objects

Topic: Table enhancements

- 1-1 The only method available to enhance the transparent table **SFLIGHT##** is an append structure, since it contains no Customizing Includes.
 - 1-1-1 How do you go about enhancing table **SFLIGHT##**?

You can work with append structures just like with "normal" structure definitions. They are created from a table (or structure). Call the ABAP Dictionary (Transaction SE11 or the Object Navigator *Edit object* → *Edit Dictionary objects*). Enter table name **SFLIGHT##** and choose *Display*.
 - 1-1-2 Enhance table **SFLIGHT##** with the append technique. The detailed procedure is described below:
- 1-2 Create your append structure using either the menu option *Goto* → *Append structure...* or its corresponding pushbutton. Use ZASFLIGHT## (## means your group number) as the name for it. Give the append structure a short description and save it under the package you created.
 - 1-2-1 Include two fields in the structure:

The field names must start with YY or ZZ.
For example **YYPILOT** and **YYMEAL**.
- 1-3 Choose a category of enhancement for the append structure.

Use the menu option *Extra* -> *Enhancement category* and choose *Cannot Be Enhanced*. Activate your append structure. If an error occurs, you can find details in the activation log.

Contents:

- Introduction
- Enhancement Management
- Function Module Exits
- Menu Exits
- Screen Exits

© SAP AG 2009



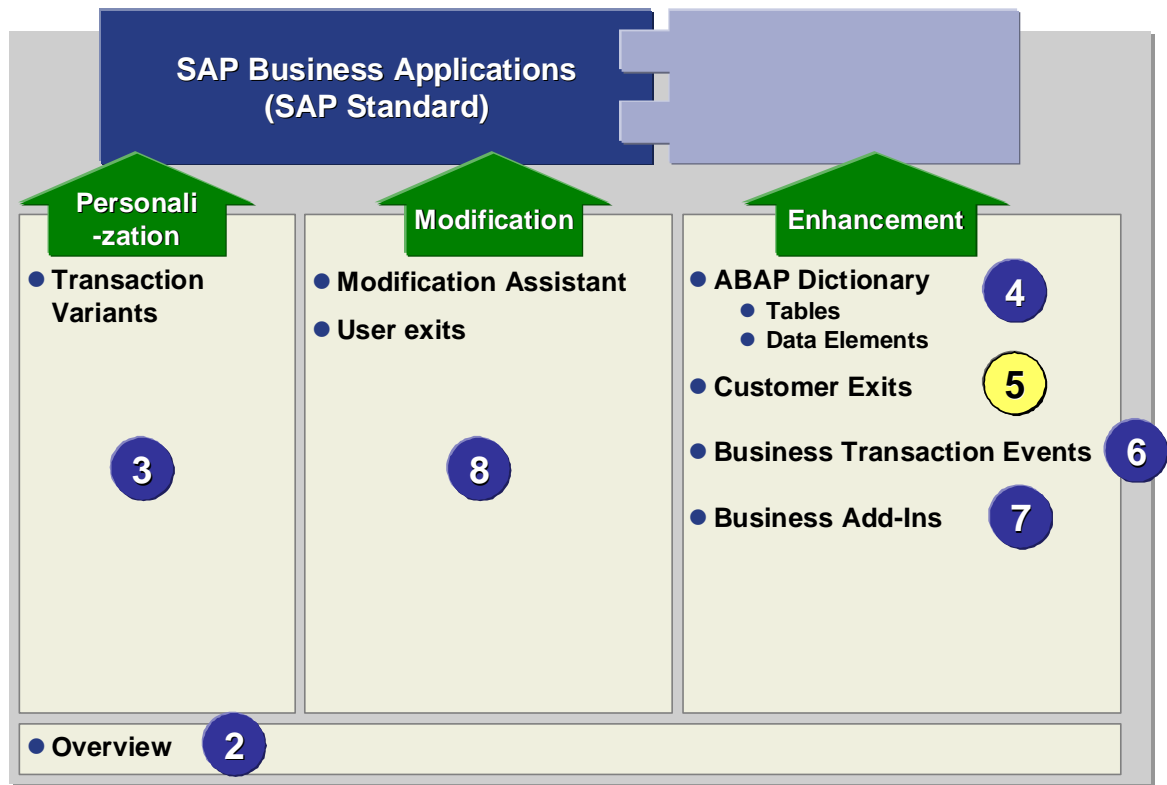
At the conclusion of this unit, you will be able to:

- **Use program, menu, and screen exits that are created using customer exit techniques**
- **Explain what components, enhancements and enhancement projects are**
- **Create enhancement projects and edit enhancements and their components**
- **Describe the connection to the Workbench Organizer and the transport system**
- **Transport enhancement projects**

© SAP AG 2006

Enhancements Using Customer Exits: Overview Diagram

SAP





Overview

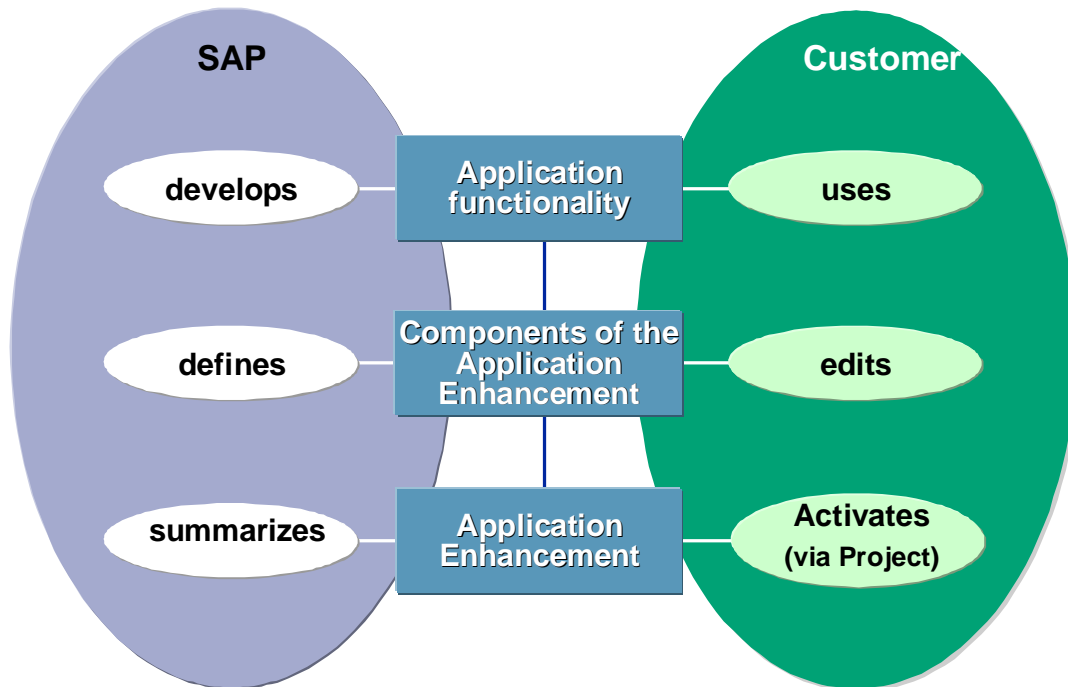
Enhancement Management

Program Exit

Menu Exit

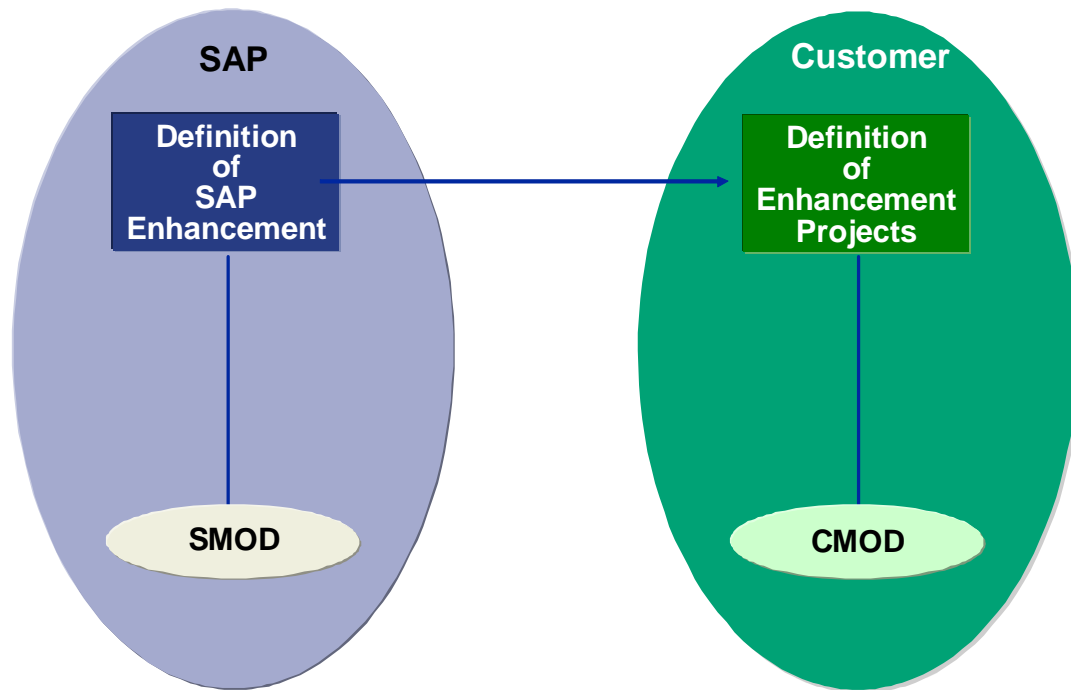
Screen Exit

© SAP AG 2009



© SAP AG 2009

- Application enhancements allow customers to enhance their application functions. Customer exits are preplanned by SAP and generally consist of several components.
- Application enhancements are inactive when delivered and can be completed and activated by customers as they are needed.
- Application enhancement characteristics:
 - Each enhancement provides you with a set of preplanned, precisely defined functions.
 - The interface between SAP and customer functions is clearly defined.
 - As a customer, you do not need in-depth knowledge of how to implement SAP applications.
 - You do not need to adjust enhancements at upgrade because of new functions that SAP has developed.



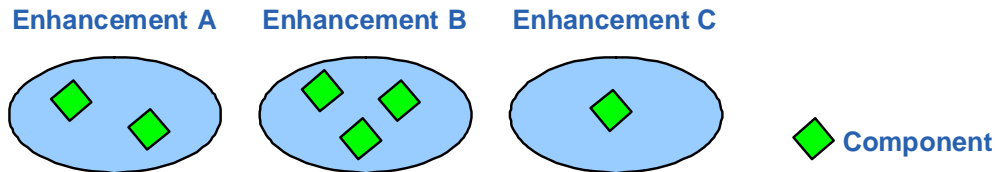
© SAP AG 2009

- SAP application programmers create SAP enhancements from function module exits, menu exits and screen exits. A management function is provided for this purpose (transaction code SMOD).
- Customers are given a catalog containing an overview of existing SAP enhancements. They can then combine the SAP enhancements they want into an enhancement project using transaction CMOD.

Enhancement components:

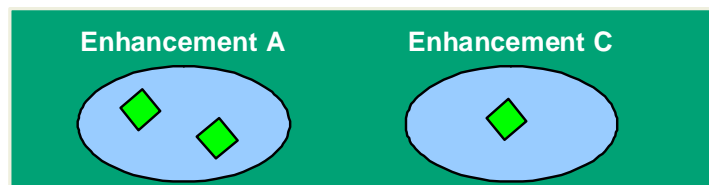
Program Exits, Menu Exits, Screen Exits

SAP combines all Enhancement components which belong together to one Enhancement



The customer has to edit all Enhancements that are to be activated together within one Enhancement project

Enhancement project



© SAP AG 2009

- SAP enhancements are made up of component parts. These components include program exits, menu exits, and screen exits. A specific component may be used only once in a single SAP enhancement (this guarantees the **uniqueness of SAP enhancements**). SAP combines all Enhancement components which belongs together to one enhancement.
- Customer enhancement projects consist of SAP enhancements. Each individual SAP enhancement may be used only once in a single customer enhancement program (this guarantees the **uniqueness of a customer project**). The customer has to edit all enhancements that are to be activated together within one Enhancement project.



Overview

Enhancement Management

Program Exit

Menu Exit

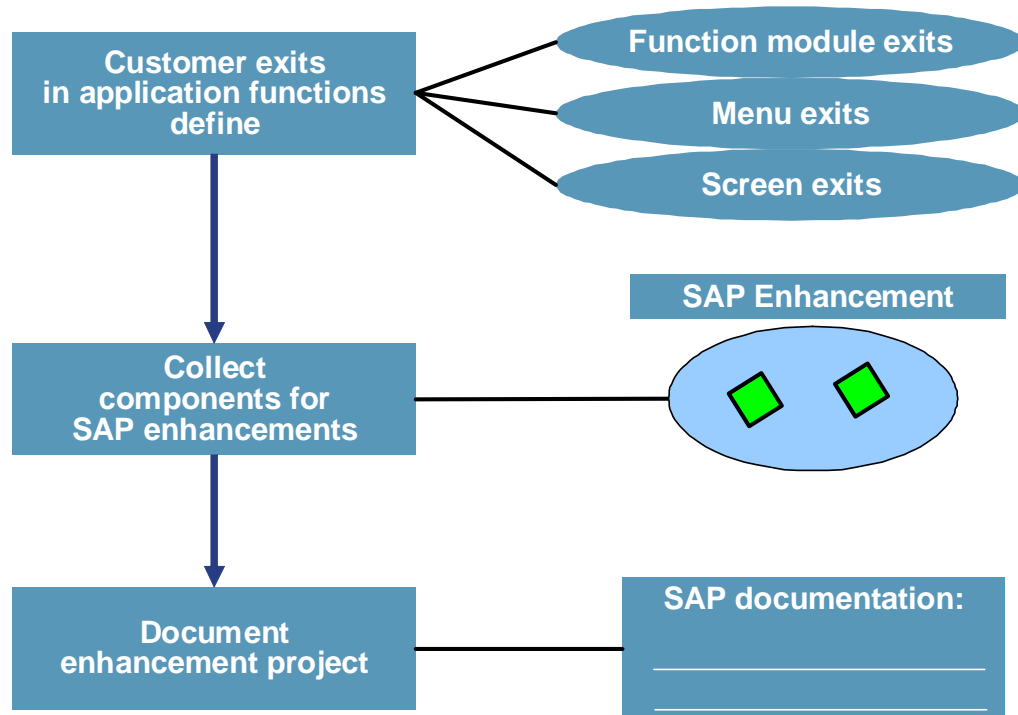
Screen Exit

© SAP AG 2009

Internal Use SAP Partner Only

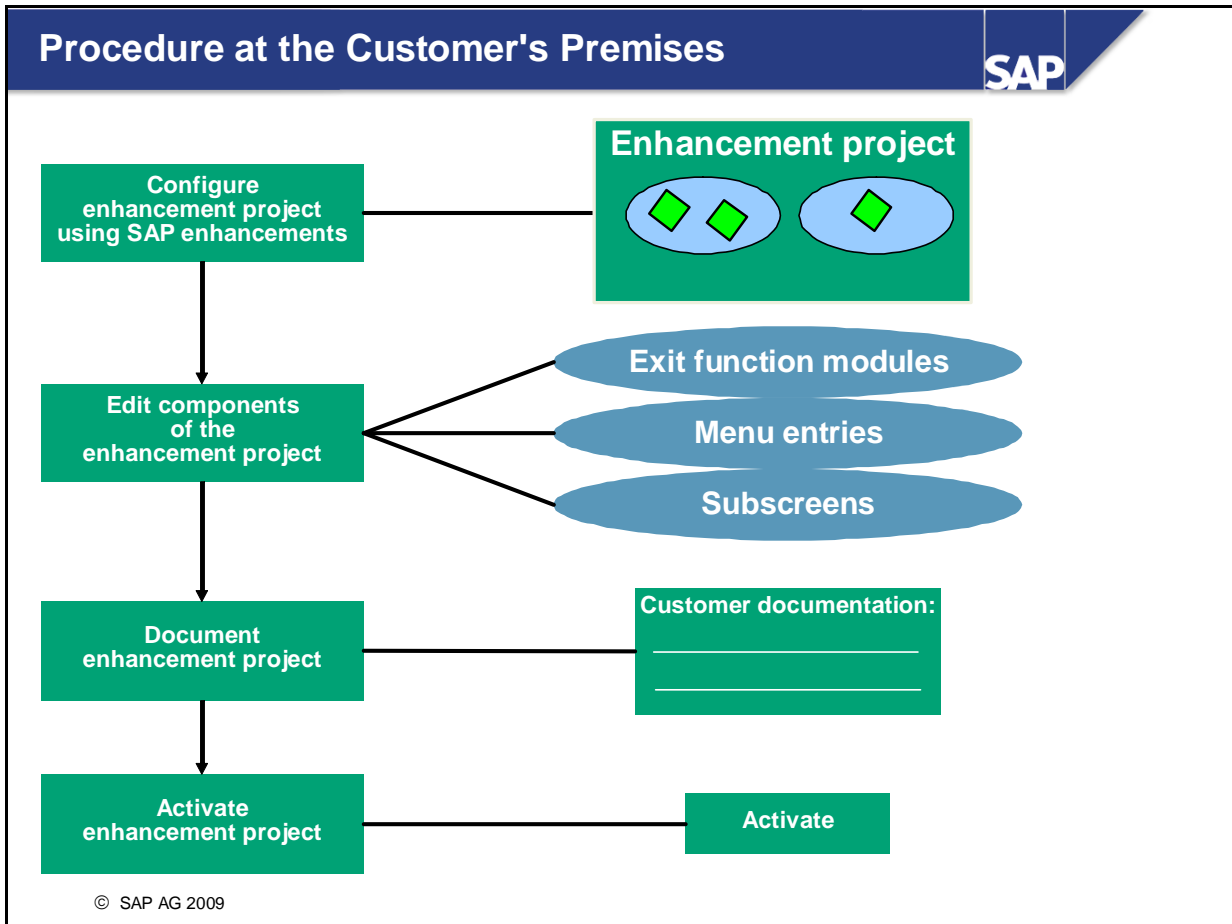
Internal Use SAP Partner Only

The SAP Enhancement Creation Procedure



© SAP AG 2009

- The SAP application programmer plans possible application enhancements in an application and defines the necessary components. These components are combined in SAP enhancements.
- The programmers document their enhancements as best they can, so that customers can implement the enhancements without having to analyze program source code or screen source code.



- First, create an enhancement project and then choose the SAP enhancements that you want to use.
- Next, edit your individual components using the project management function and document the entire enhancement project.
- Finally, activate the enhancement project. This activates all of the project's component parts.

Creating a Customer Enhancement Project

SAP

The screenshot shows the 'SAP Enhancement Project Administration' window. At the top, a yellow oval highlights the text 'Transaction CMOD'. Below this, the 'project' field is followed by a 'Create' button, which is circled in red. Under the 'Sub-objects' section, the 'Attributes' radio button is selected and circled in orange. Other sub-objects listed are 'Enhancement assignment', 'Components', and 'Documentation'. At the bottom of this section are 'Display' and 'Change' buttons. An orange arrow points from the 'Attributes' selection to a secondary window titled 'Enhancement Project Attributes'. This window shows the 'project' field with the value '<project>' and a 'Short Text' input field.

© SAP AG 2009

- You start the project management function (transaction CMOD). The first thing to do is to give your enhancement project a name. SAP recommends that you think up a naming convention for all of your projects. You can, for example, include the project's transaction or module pool in its name. The project name uniquely identifies the enhancement in the system.
- Next, go to the project's attributes and enter a short text describing the enhancement project. The system enters the remaining attributes (name stamp and time stamp for creating and changing, and status).

Assigning SAP Enhancements to Customer Projects

SAP

Project Management of SAP Enhancements

Project  **Create**

Subobjects

- ☐ Attributes
- ☒ **Enhancement assignment**
- ☐ Components
- ☐ Documentation

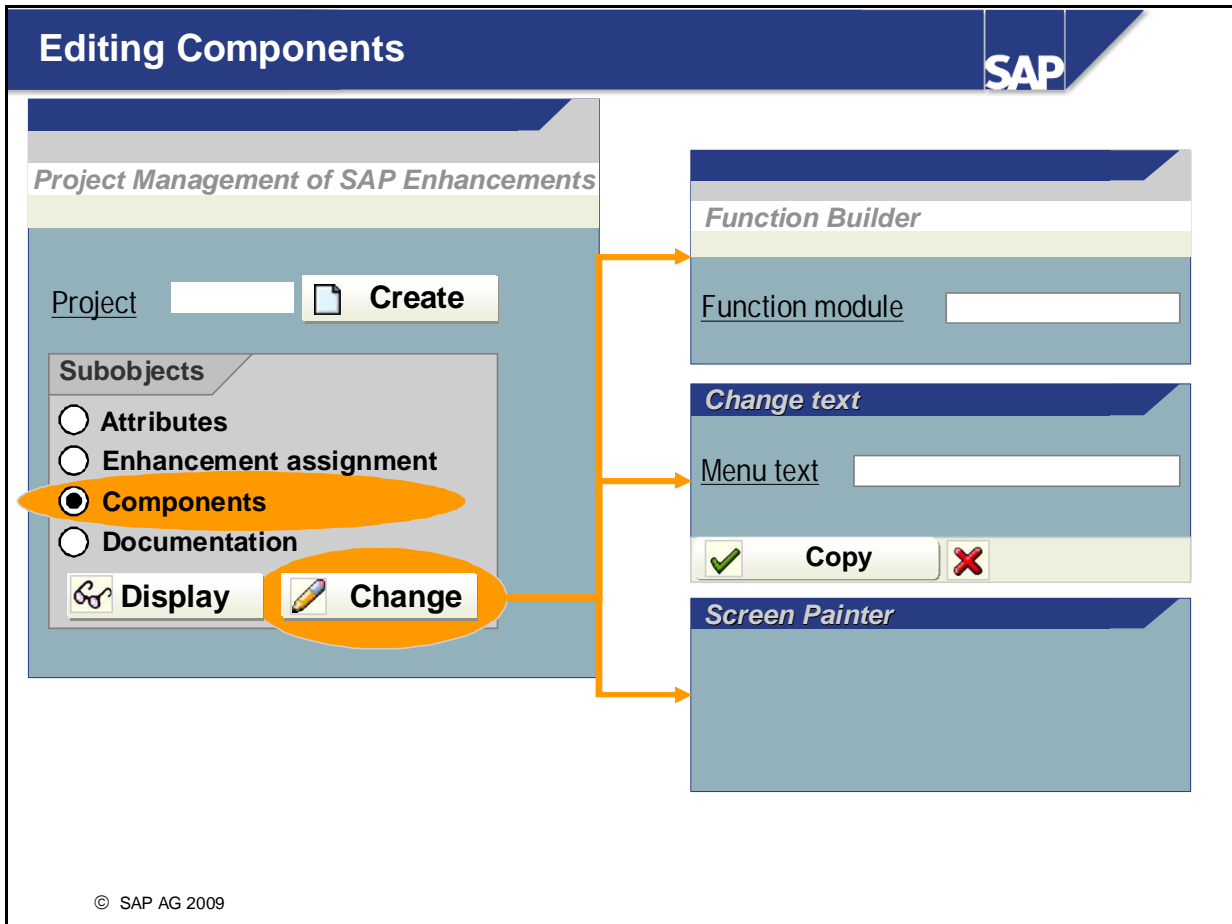
 **Display**  **Change**

SAP Enhancements in Project

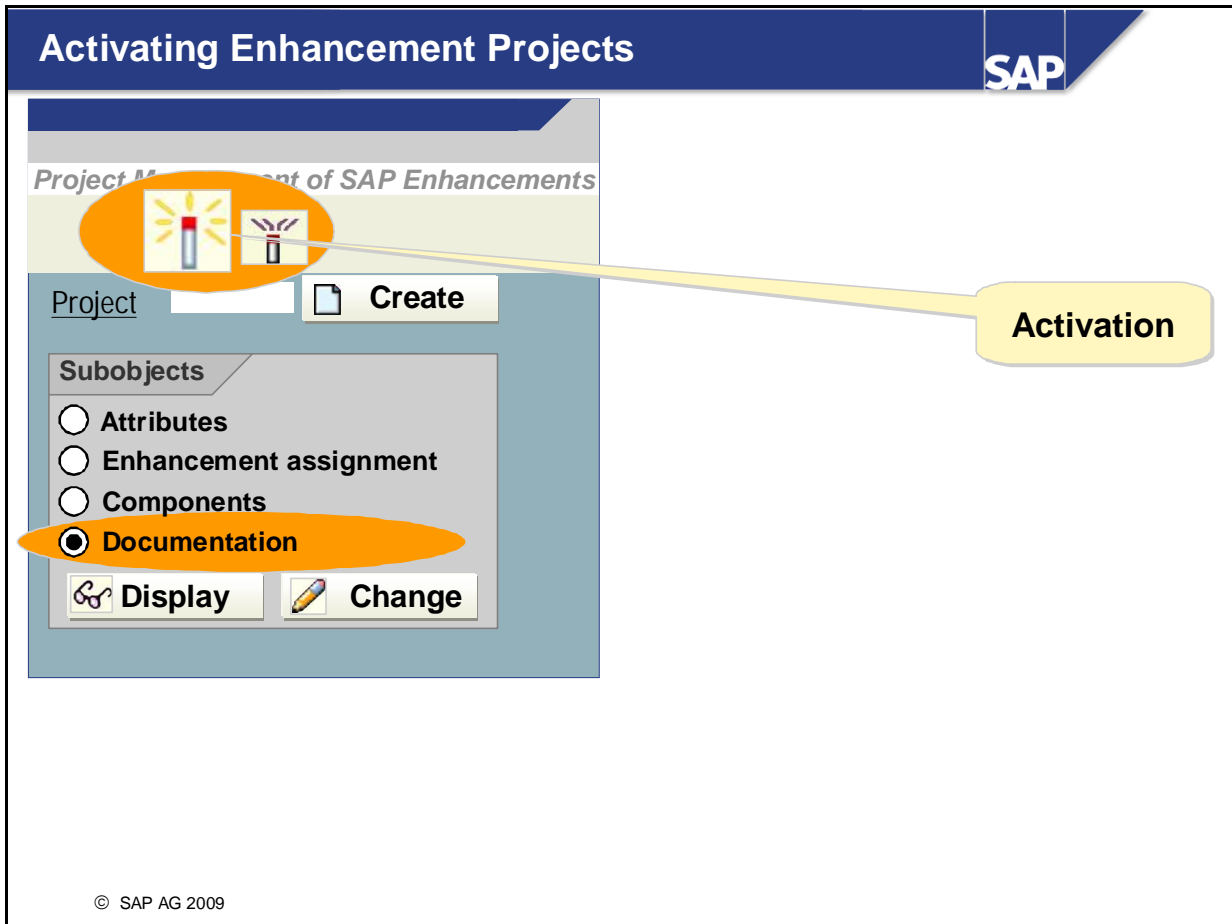
Enhancement	Description
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>

© SAP AG 2009

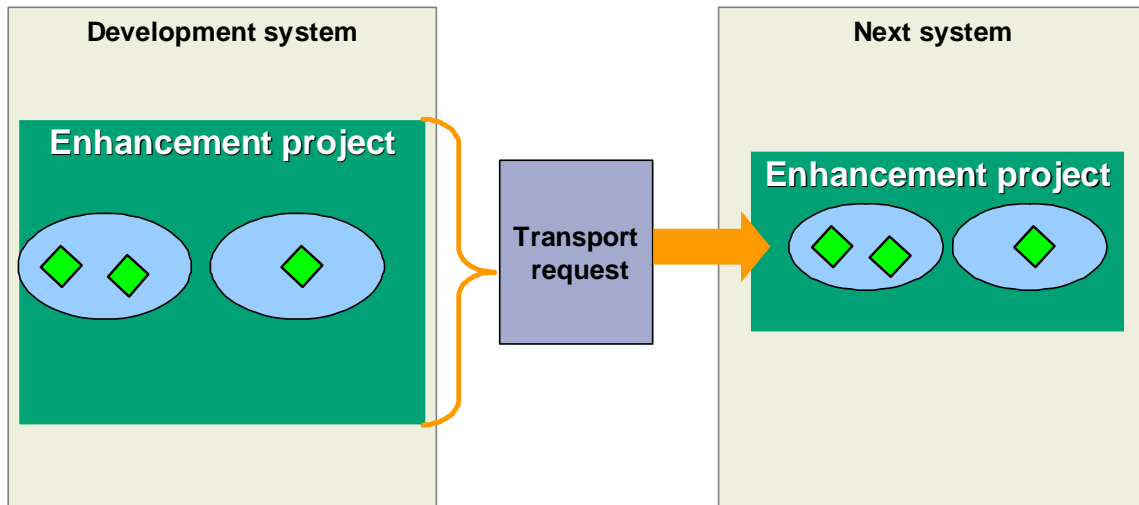
- Use the project management function to assign SAP enhancements to customer enhancement projects. Enter the names of the SAP enhancements you want to use on the appropriate screen.
- The search function gives you a catalog-like overview of existing SAP enhancements. From there you can select those enhancements that are of interest to you.



- Use the product management function to edit the components of your enhancement project.
- Depending on whether the component you are editing is a function module, a menu entry, or a subscreen, you branch to either the Function Builder, a dialog box for entering menu entries, or to the Screen Painter.



- Activation of an enhancement project affects all of its components. After it has been activated successfully, the project has the status *active*.
- During activation, all programs, screens, and menus containing components that belong to the project are regenerated (programs at the time they are executed). After activation, you can see the effect of the enhancements in your application functions.
- The *Deactivate* function allows you to reset an active enhancement project's status to *inactive*.



© SAP AG 2009

- When the enhancement project was created, you should have assigned it to a change request. Each of the component pieces (include programs, subscreens, menu exits, and so on) should be assigned to the same change request. Using the same change request allows you to transport the entire enhancement at the same time.

Overview

Enhancement Management



Program Exit

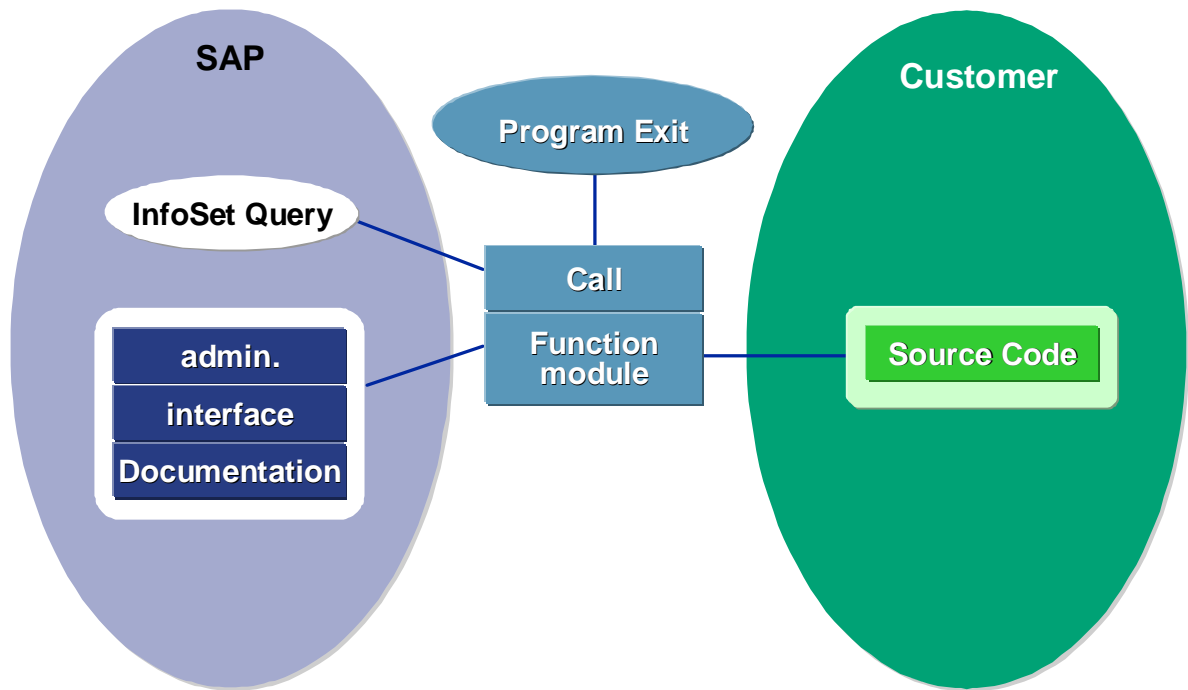
Menu Exit

Screen Exit

© SAP AG 2009

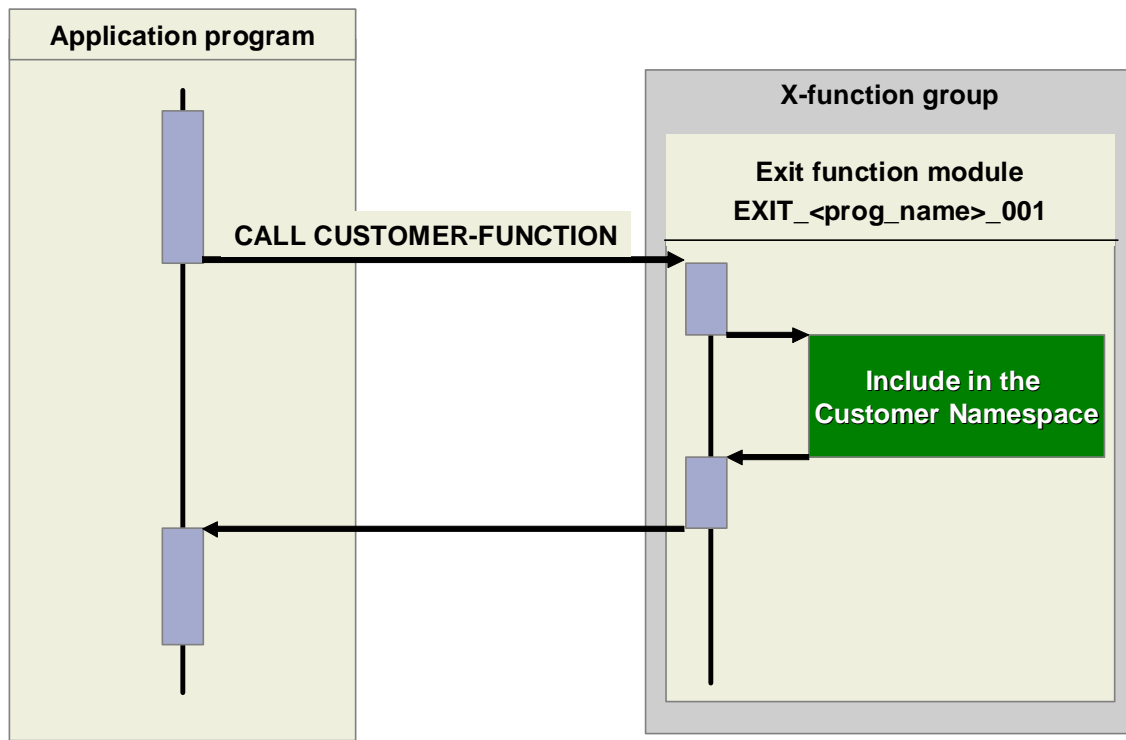
Internal Use SAP Partner Only

Internal Use SAP Partner Only



© SAP AG 2009

- Program exits allow customers to implement additional logic in application functions. SAP application programmers define where program module exits are inserted and what kind of data they transfer. SAP programmers also create an exit's corresponding function modules complete with short text, interface, and documentation, as well as describing each program exit's intended purpose in the SAP documentation.
- You write the source code for the function modules yourself. If need be, you must also create your own screens, text elements, and includes for the function group.
- The system processes your ABAP code for the first time whenever the enhancement project (of which your function module is a component) is activated as a whole. Program exits have no effect prior to enhancement project activation.

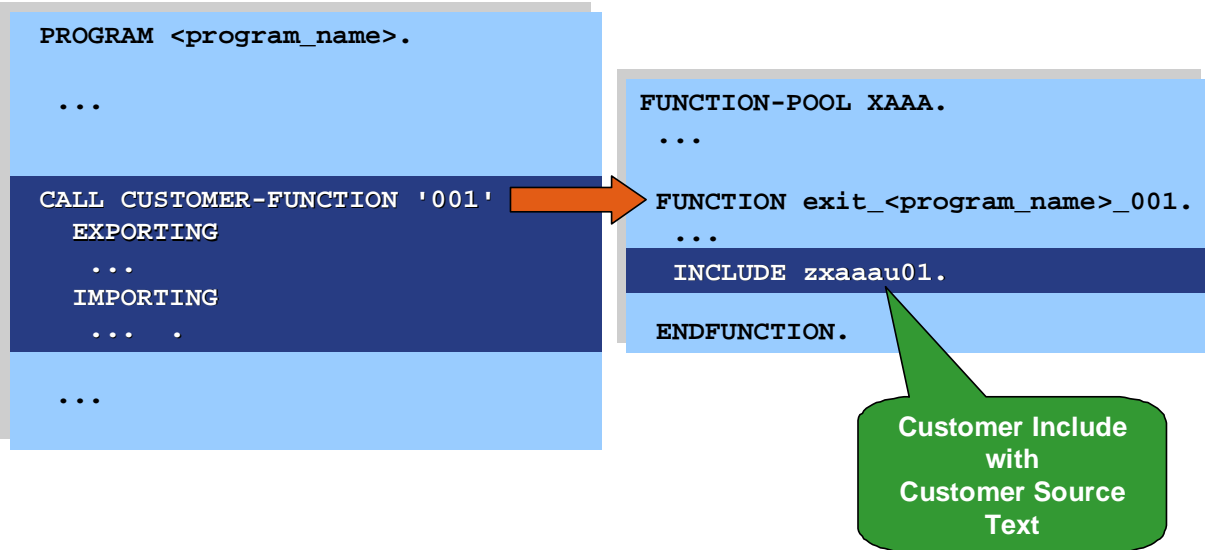


© SAP AG 2006

- This graphic shows the flow of a program providing an enhancement in the form of a program exit.
- The exit function module is called at a point in the source text defined by the SAP application developer. Within the function module, users can add a function to the customer namespace using an include.

Program Exits: Syntax

SAP



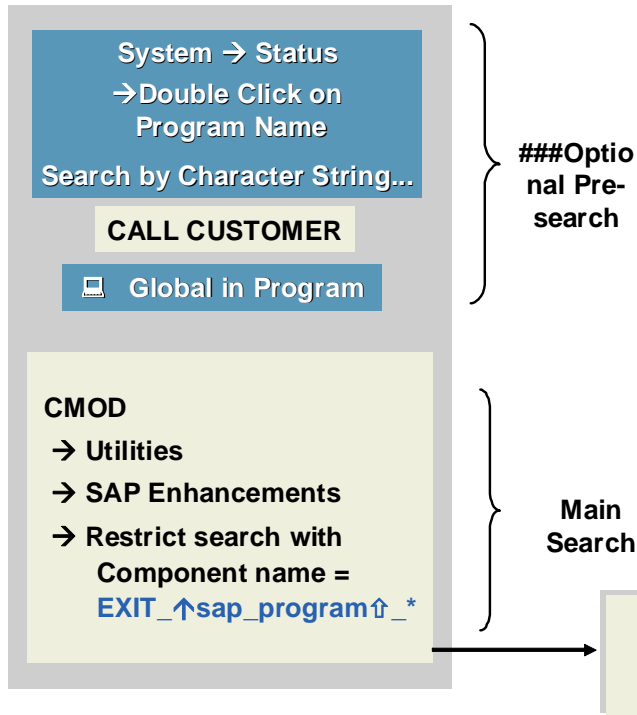
© SAP AG 2009

- SAP application programmers define the function module call using the ABAP statement `CALL CUSTOMER-FUNCTION 'nnn'`, where 'nnn' is a three-digit number. SAP application programmers also create the corresponding function module and function group.
- These function modules always belong to function groups whose names begin with X (X function group).
- The following naming convention applies to function modules:
 - Prefix: EXIT
 - Name of the program that calls the function module
 - Suffix: Three-digit number
 - The three parts of the name are separated by an underscore.
- The `CALL CUSTOMER-FUNCTION` statement is only executed once the enhancement project has been activated. If the same function module is called several times, the activation is valid for all calls.

Finding Program Exits

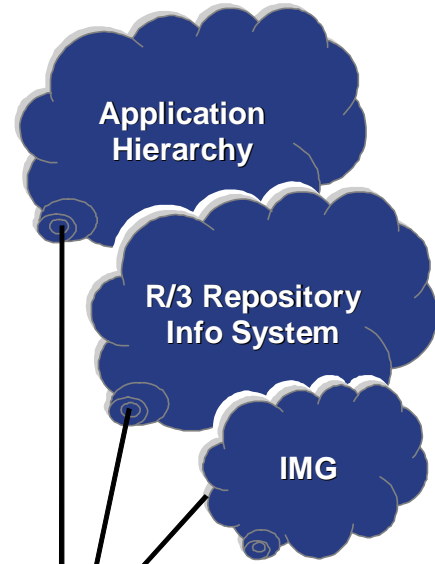
SAP

● Search by Program



© SAP AG 2006

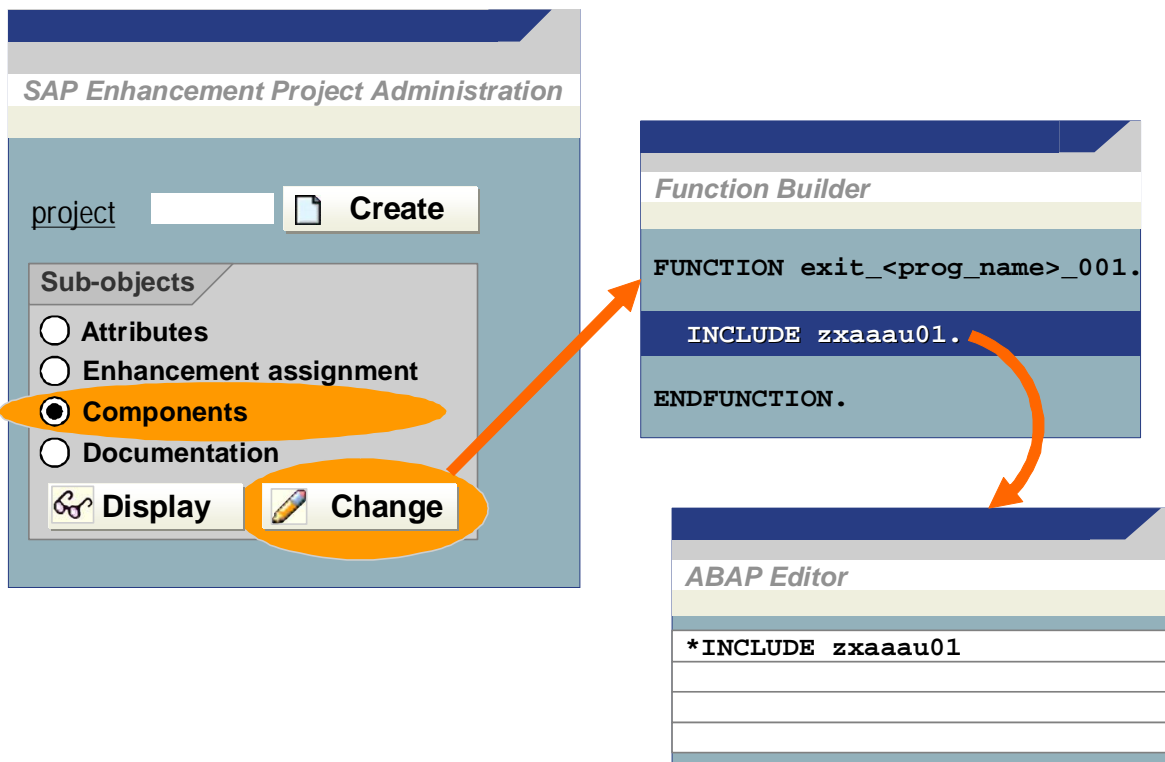
● Using tools



- The most frequently asked question concerning enhancements is: How can you see if an application program offers a program exit? This is a central question for every kind of enhancement. There are a number of ways to find the answer to this question.
- To quickly determine if an application program offers a program exit, you can follow the path on the left-hand side of the graphic: (The menu path *System* → *Status* always gives you the name of the application program.) In our example a suitable character string would be "CALL CUSTOMER". Use the Find icon and search globally in the program. If your search is unsuccessful, you can extend the search domain: Run an environment analysis for the corresponding program, and search for the specific character string in the program environment.
- The right side of the graphic shows you how to find the name of the required enhancement using search tools. You can restrict the search in the R/3 Repository Information System using different criteria. Important criteria are:
 - Package (also try generic entries)
 - technical name of the enhancement

Editing Program Exits

SAP

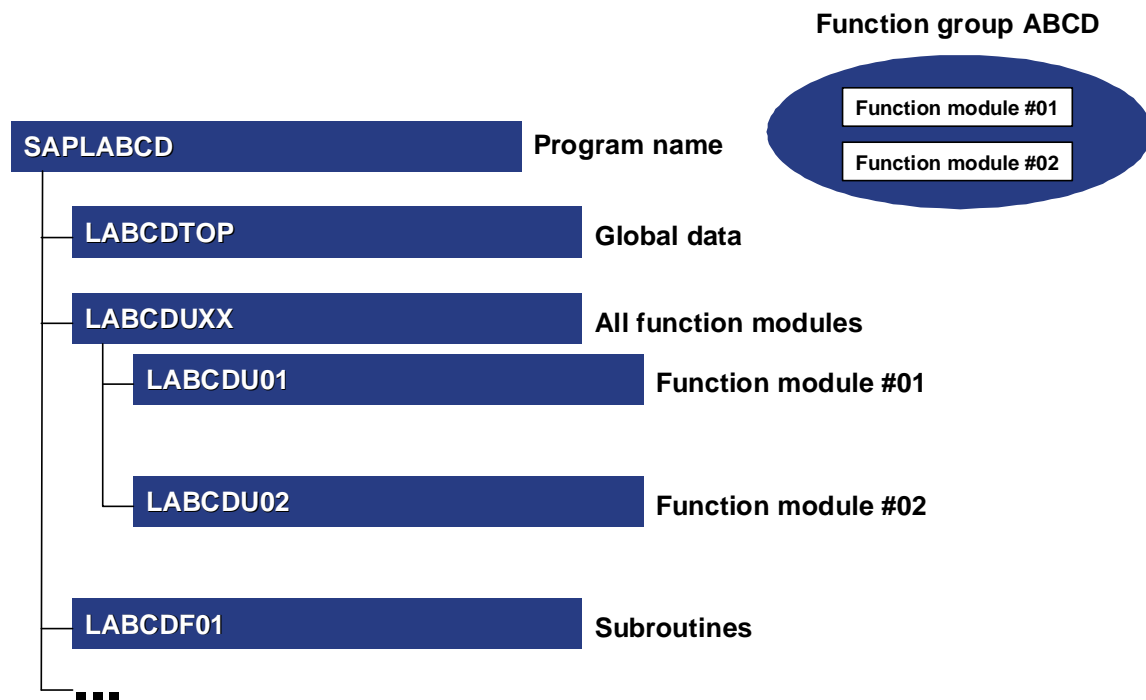


© SAP AG 2009

- Use the project management (transaction: CMOD) function to edit function modules for program exits.
- Use the button for editing components to go directly to the function module editor (display mode).
- DO NOT change the function module itself, particularly the interface. The function module, however, contains an `INCLUDE` statement for an include program that you have to create in the customer namespace.
- Double-click on the object to select it. This automatically takes you to the editor of the include program, where you can enter your source code.

Structure of a Function Group

SAP

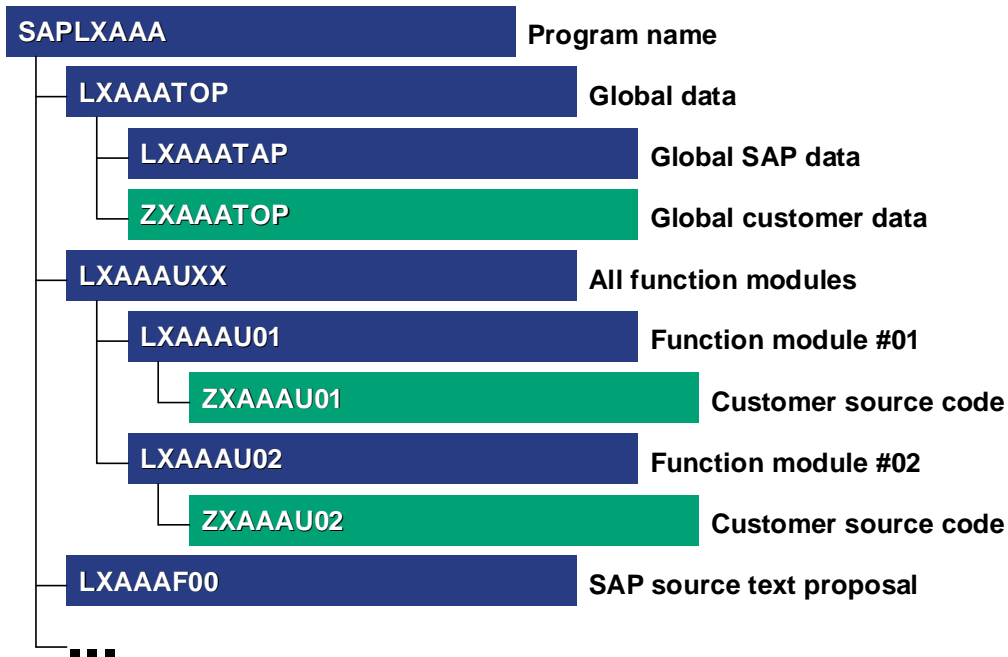


© SAP AG 2009

- To understand how an X function group works, you need to understand how a normal function group works:
 - A function group consists of includes. The system assigns unique names to the includes for different objects. Some of the include names are simply proposals and some cannot be changed.
 - Global data is stored in the TOP include. This include is generated automatically when a function group is created.
 - Function modules are stored in includes with sequential numbering, and they in turn are all stored in an include ending with UXX.
 - While you can freely choose the names of the includes for all other objects (subroutines, modules, events, and so on), we advise you to accept the proposed names.

Structure of an Exit Function Group

SAP

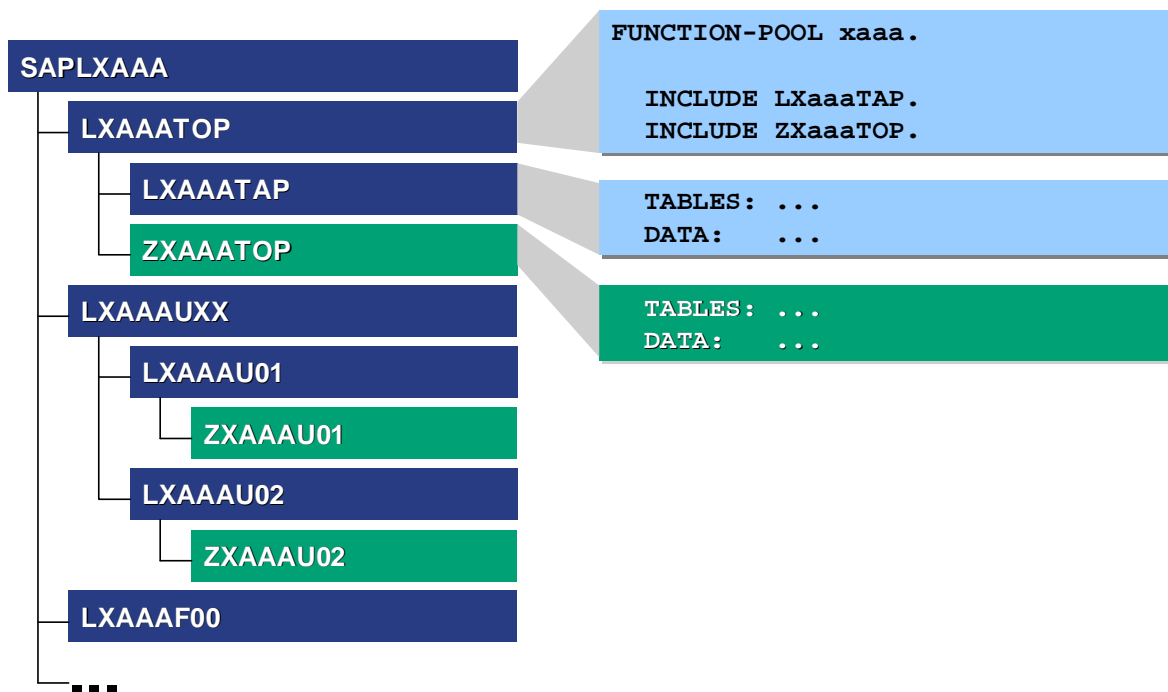


© SAP AG 2002

- Exit function groups created by SAP application programmers for enhancement exits contain include programs that begin with either LX or ZX. You can only edit includes beginning with a Z, since they are stored in the customer namespace.
- No further function modules may be added to the function group.
- The include program **ZxaaaUnn** contains the source code for the function modules of a function module exit.

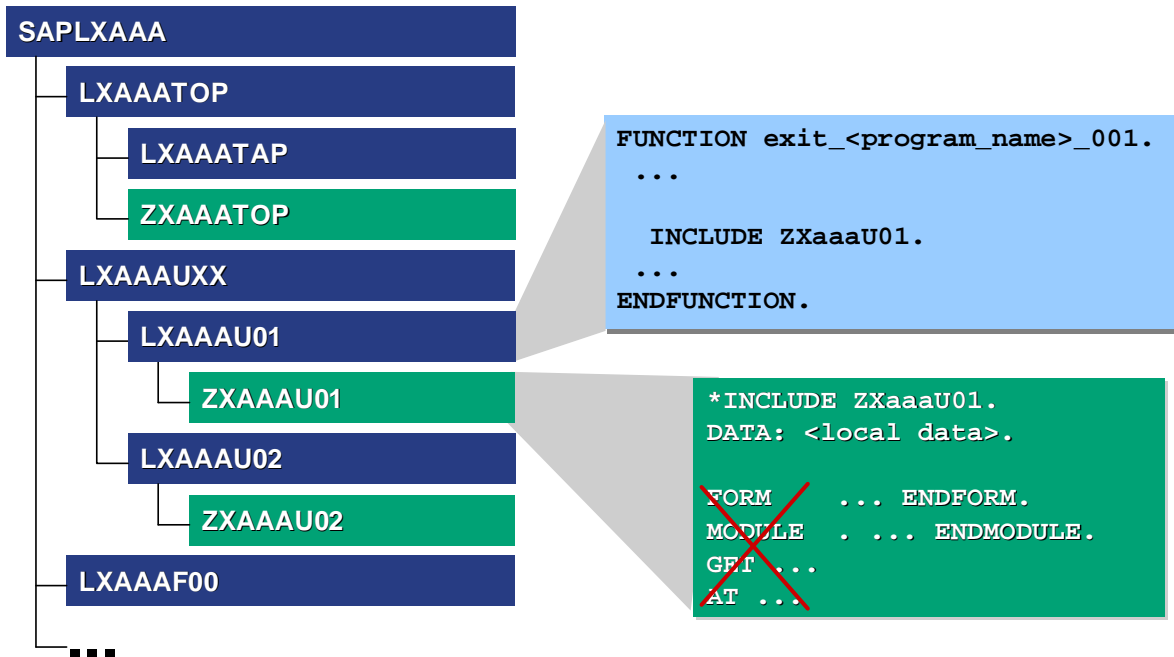
Global Data of an Exit Function Group

SAP



© SAP AG 2006

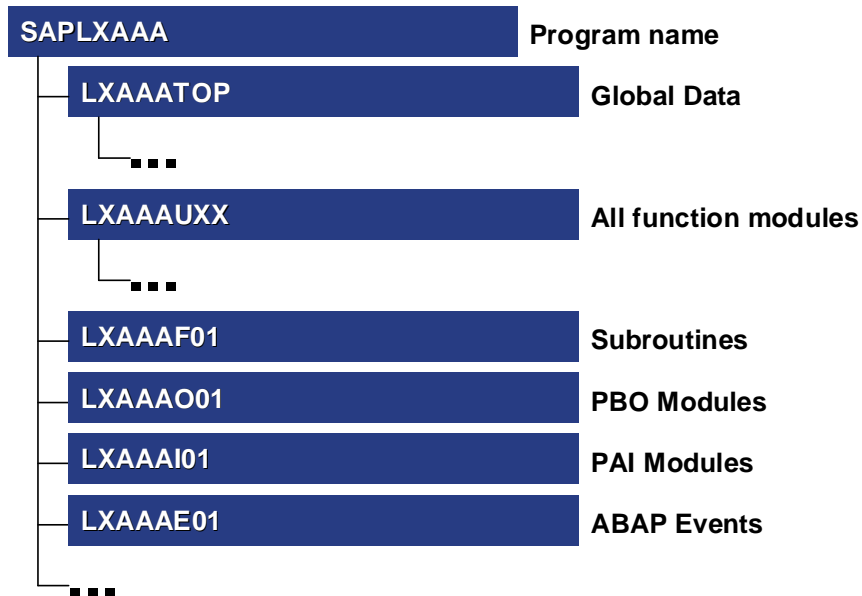
- SAP application programmers can declare global data in include program LXaaaTAP.
- You can declare your global data in include ZXaaaTOP.
- Include program LXaaaTOP also contains the FUNCTION-POOL statement, which may not be changed. Therefore, you must always include the message class in parentheses when outputting messages - for example, MESSAGE E500 (EU).



© SAP AG 2009

- The INCLUDE statement for program ZXaaaUnn is in a FUNCTION – ENDFUNCTION block. Because of this, neither events, nor subroutines (FORM) nor modules (MODULE) are permitted here. They need to be created in separate includes, a process that will be discussed below. Data declarations made here with DATA are valid locally in this function module.
- The SAP application programmer can also make a proposal for the source text. In this case, an INCLUDE LXaaFnn is created (where nn is the internal number for the function module in the include LXaaaUXX). Documentation is also provided within the SAP enhancement. You can copy the source code from this include into your own customer include program ZXaaaUnn using the project management transaction.
- You can create your own text elements for the function group.

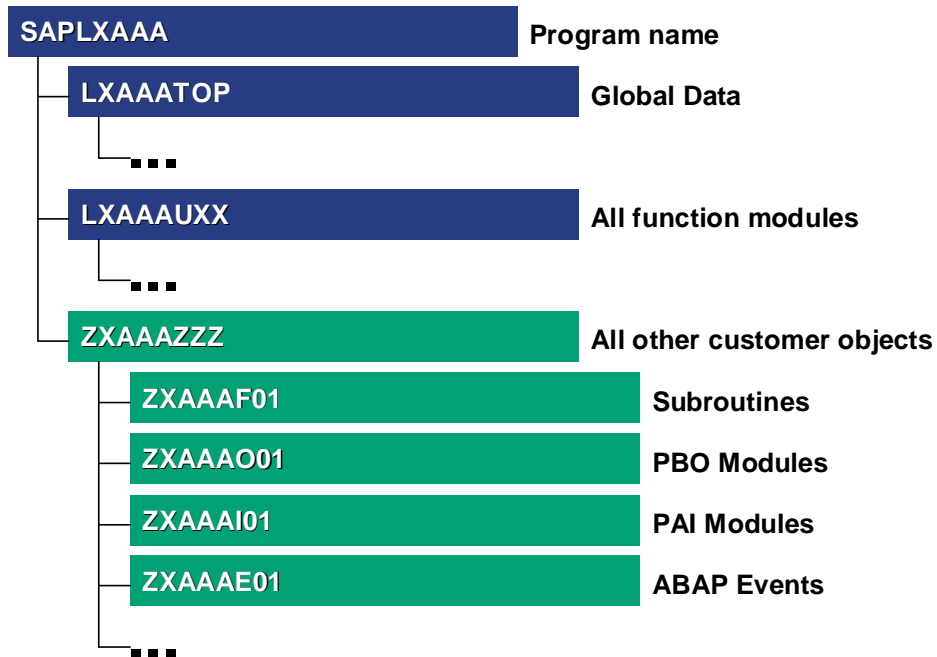
Other SAP Objects in an Exit Function Group



© SAP AG 2006

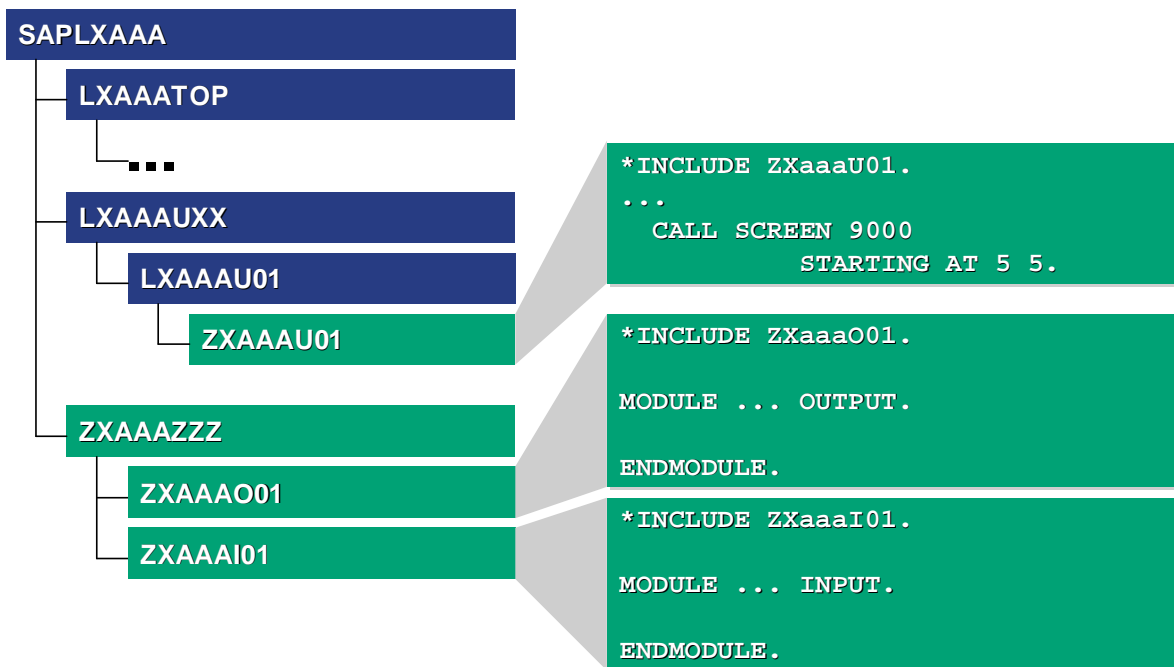
- SAP application programmers can supply you with default subroutines in include LXaaaF01.
- There may be further includes containing specific sub-objects.
 - LX...F01 contains subroutines delivered by SAP.
 - LX...E01 contains the events belonging to the X function group.
 - LX...O01 contains PBO modules for screens to be delivered.
 - LX...I01 contains the corresponding PAI modules.

Customer Objects in an Exit Function Group



© SAP AG 2006

- Subroutines, modules, and interactive events (AT...) are created as include programs and included enhancements using include program **ZXaaaZZZ**.
- Additional includes must adhere to the following naming convention:
 - **ZXaaaFnn** for subroutines
 - **ZXaaaOnn** for PBO modules
 - **ZXaaaInn** for PAI modules
 - **ZXaaaEnn** for events

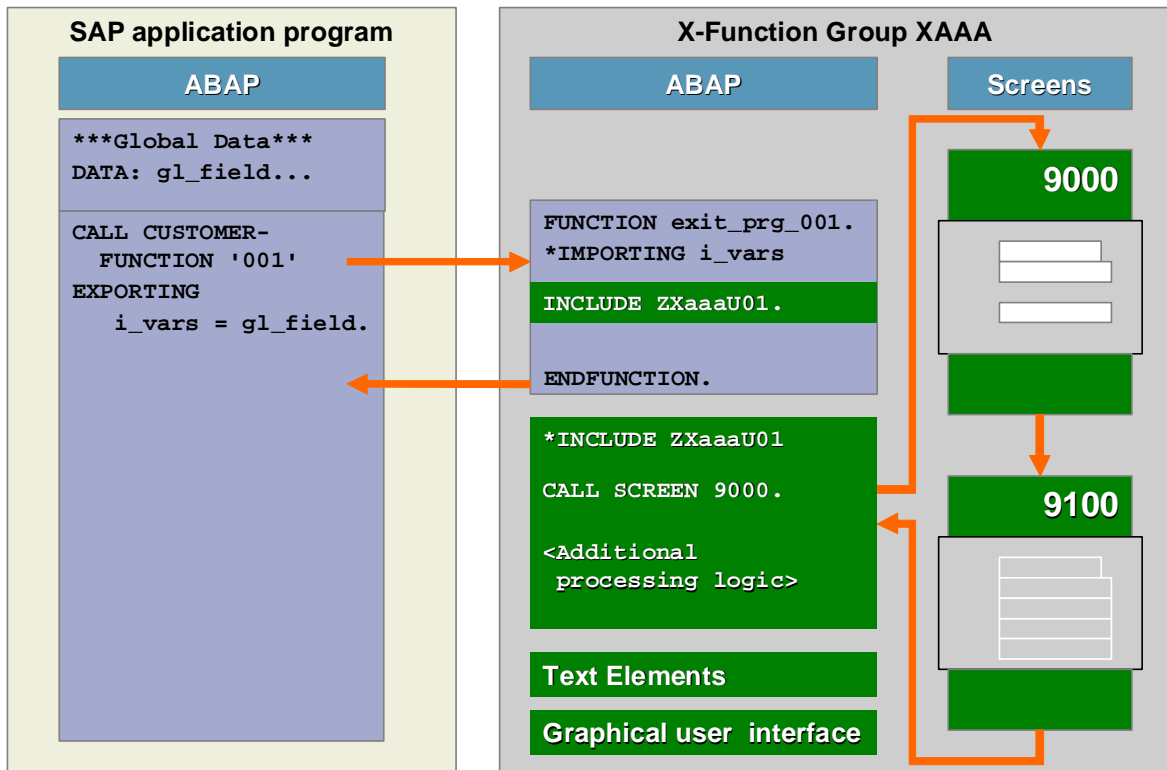


© SAP AG 2009

- You can use **CALL SCREEN** to call your own screens. Create the related include programs for the Process Before Output (PBO) and Process After Input (PAI) modules in include program **ZXaaaZZZ**.
- Use forward navigation (select an object and then double-click on it) to create your own screens and modules.
- Screens created in this manner are automatically given the name of the function module's main program (**SAPLXaaa**). The PBO modules for these screens can be found in include **ZXaaaO01** , the PAI modules in include **ZXaaaI01** .

Summary: Program Exits

SAP



© SAP AG 2009

- You can enhance SAP applications by adding your own processing logic at predefined points.
- Such enhancements can include your own screens with their corresponding processing logic and graphical user interface, as well as text elements created by customers.

Overview

Enhancement Management

Program Exit



Menu Exit

Screen Exit

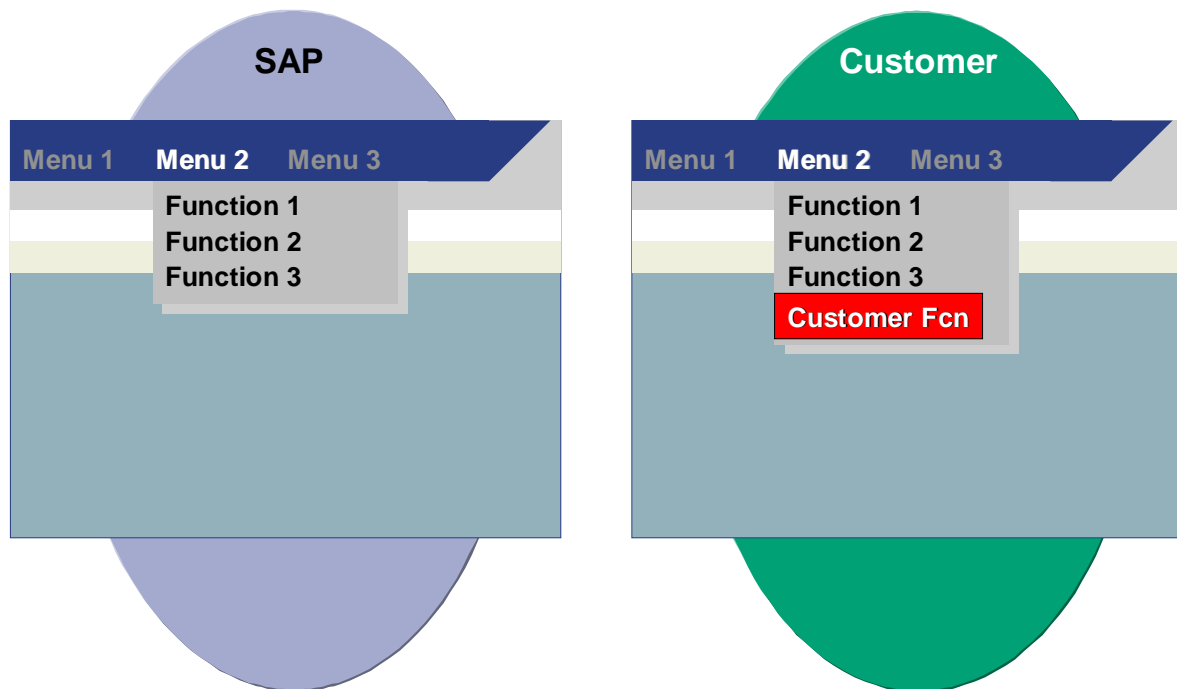
© SAP AG 2009

Internal Use SAP Partner Only

Internal Use SAP Partner Only

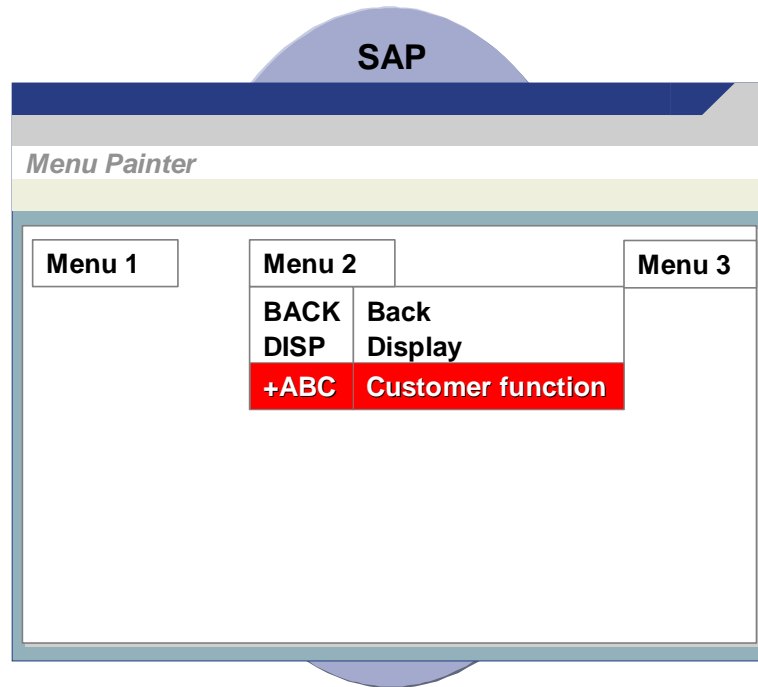
Menu Exits Overview

SAP



© SAP AG 2009


- Menu exits allow you to attach your own functions to menu options in SAP menus. SAP application programmers reserve certain menu entries in your GUI interface for this. You can specify the entry text yourself.
- Once you activate menu exits, they become visible in the SAP menu. When you choose the corresponding menu option, the system changes to a program exit that contains your customer-specific functions.



© SAP AG 2009

- For you to be able to implement menu exits, SAP application programmers must equip the GUI interface with function codes that begin with a plus sign (+).
- These function codes are inactive at first and do not appear in the GUI until you activate them. They do not appear on the screen.

```
PROGRAM <program_name>.  
DATA ok_code LIKE sy-ucomm.  
...  
* PAI-Modul :  
CASE ok_code.  
  WHEN 'DISP'.  
    ...  
  WHEN '+ABC'.  
    CALL CUSTOMER-FUNCTION '001'  
      EXPORTING  
        <i_variables>  
      IMPORTING  
        <e_variables>.  
    ...  
ENDCASE.
```



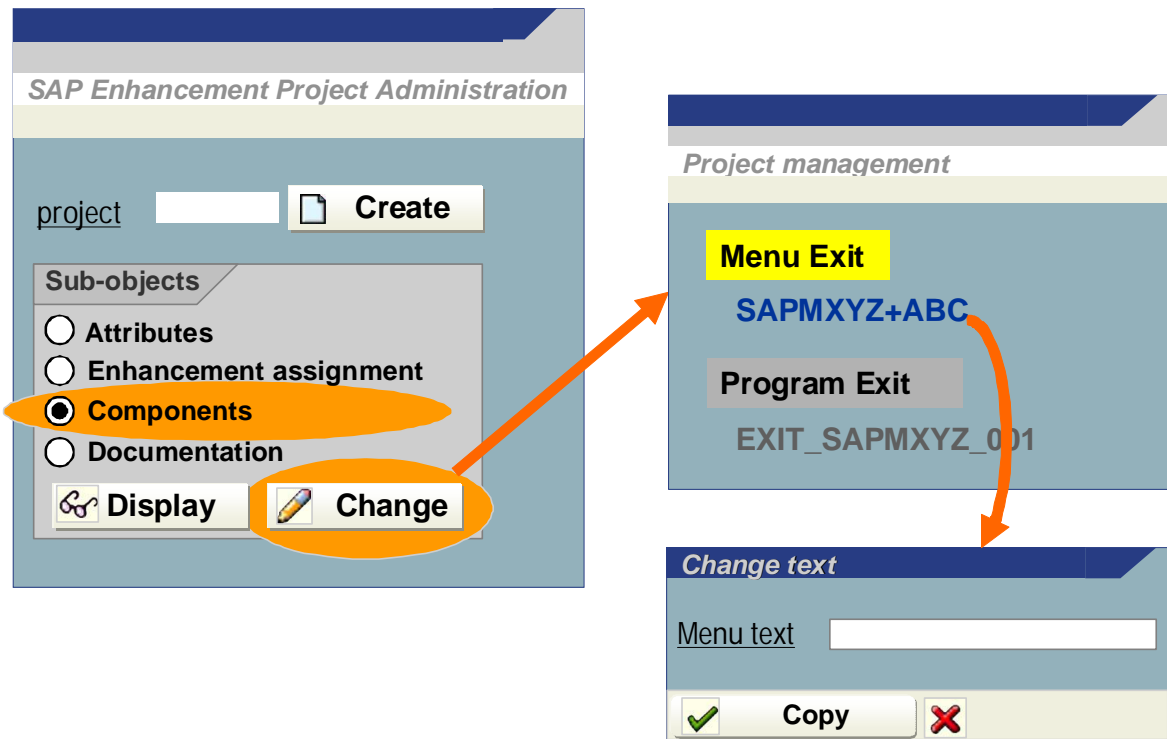
Customer
Functions

© SAP AG 2009

- SAP application programmers determine where a program reads additional function codes and how it reacts, either with a program exit or with a predefined function.

Naming and Editing Menu Exits

SAP



© SAP AG 2009

- Menu exits are edited with the project management transaction (CMOD).
- The pushbutton for editing components calls a dialog box where you can enter language-dependent short descriptions for each additional menu entry.
- You may not make any changes to the GUI interface.

Overview

Enhancement Management

Program Exit

Menu Exit

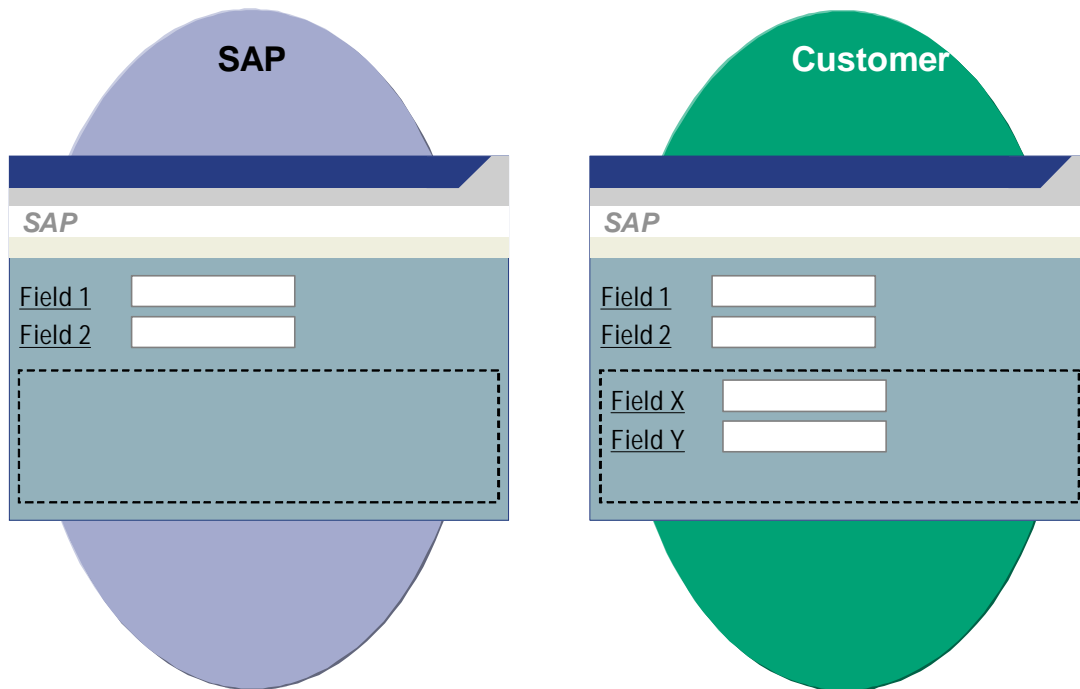


Screen Exit

© SAP AG 2009

Internal Use SAP Partner Only

Internal Use SAP Partner Only

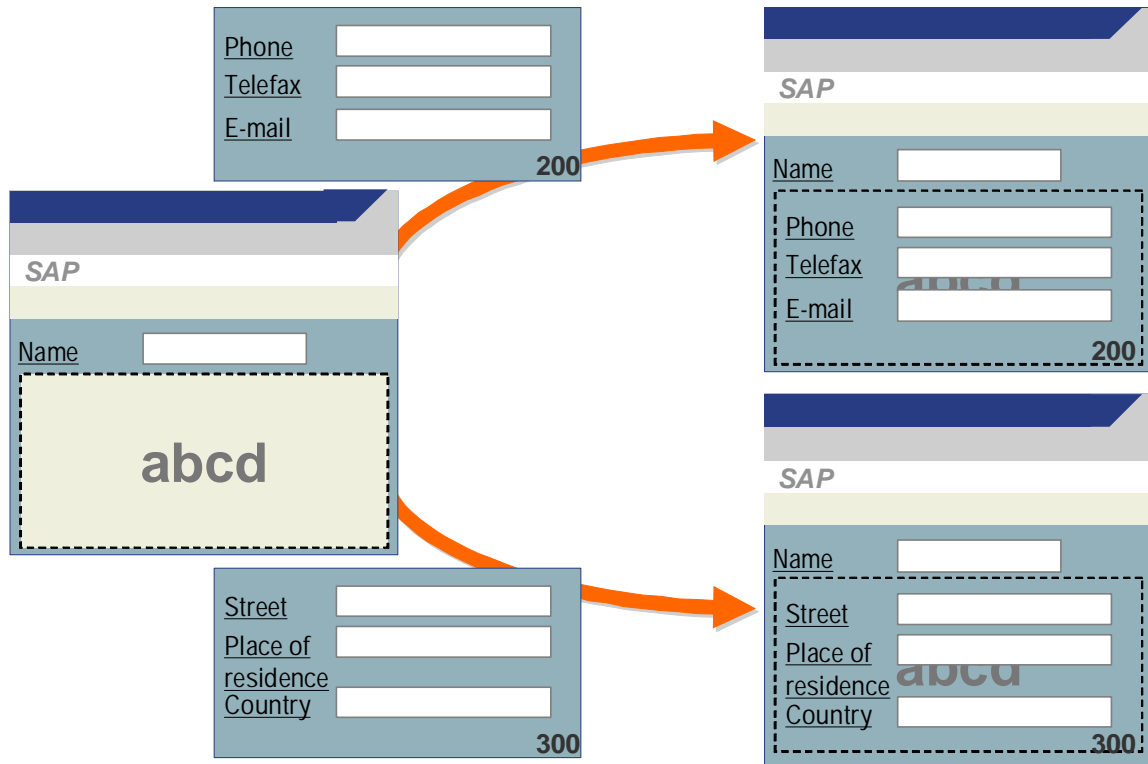


© SAP AG 2009

- Screen exits allow you to make use of reserved sections of a main screen (subscreen areas). You can either display additional information in these areas or input data. You define the necessary input and output fields on a customer screen (subscreen).

Subscreens (General)

SAP



© SAP AG 2009

- Subscreens are rectangular areas on your screen that are reserved for displaying additional screens at runtime. Another screen (of the subscreen type) can be displayed in each subscreen area at runtime.

- **Controlling the main screen**

```
PROCESS BEFORE OUTPUT.  
  MODULE ...  
  
  CALL SUBSCREEN abcd.  
    INCLUDING sy-cprog '1234'.  
  
  MODULE ...
```

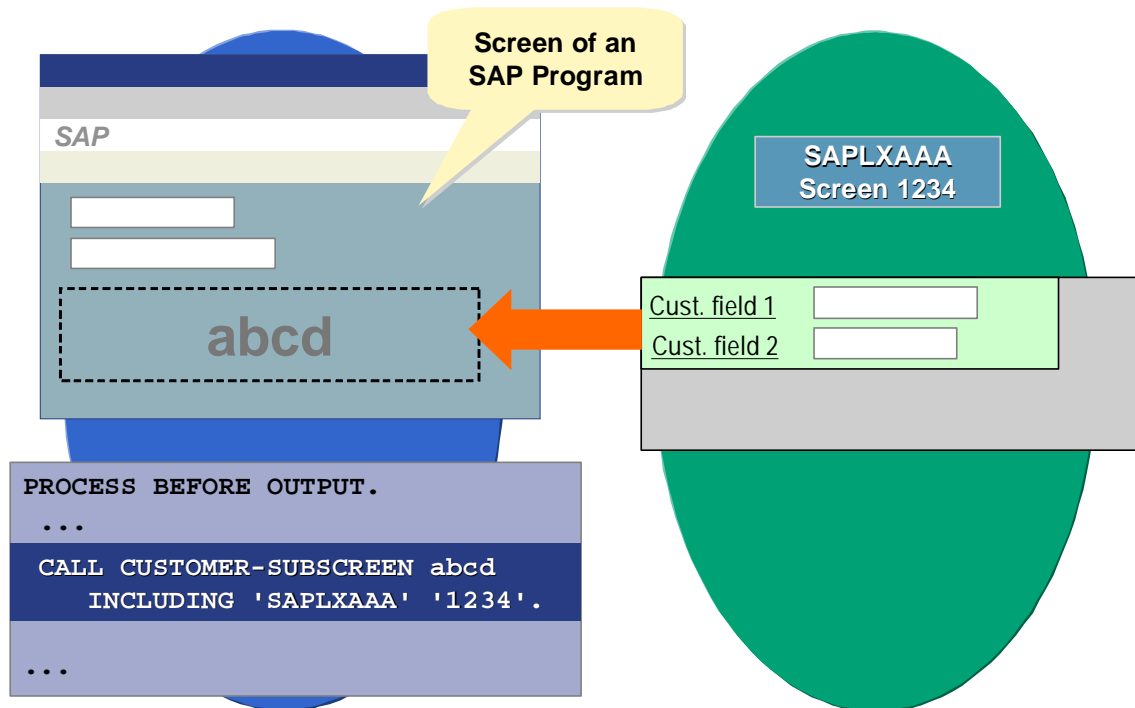
```
PROCESS AFTER INPUT.  
  MODULE ...  
  
  CALL SUBSCREEN abcd.  
  
  MODULE user_command_0100.
```

© SAP AG 2009

- Your system determines which screen will be displayed in a subscreen area at the PBO event. The general syntax is:
`CALL SUBSCREEN <subscreen-area> INCLUDING <prg> <dynpro_no>.`
- For each subscreen, PAI and PBO events are processed just as if the subscreen were a normal screen.
- The sequence of "CALL SUBSCREEN" calls in the flow logic of the main screen determines the sequence in which the flow logic of individual subscreen screens is processed.
- Important:
 - The function code can only be processed via the main screen
 - You are not allowed enter a name for a subscreen's command field.
 - You are not allowed to define GUI statuses for subscreens.
 - No value for next screen may be entered in a subscreen's flow control.

Defining Screen Exits

SAP

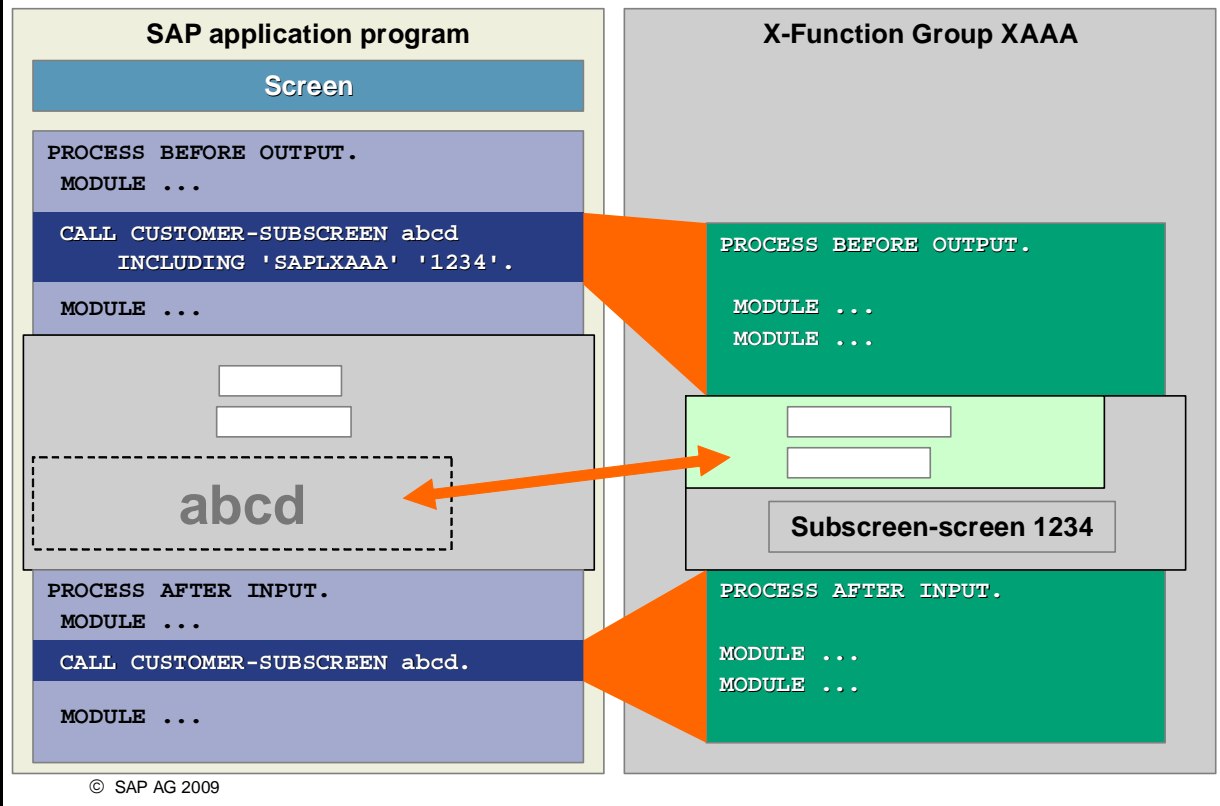


© SAP AG 2009

- The SAP application programmer can reserve multiple subscreen areas for a screen.
- The subscreen is called during flow control of the main screen with the CALL CUSTOMER-SUBSCREEN statement. The name of the subscreen area must be defined without apostrophes. The function group to which the subscreen belongs must be defined statically in apostrophes, but the screen number can be kept variable by using fields; it must always have four places.
- Screen exit calls are inactive at first, and are skipped when a screen is processed.
- Only after a corresponding subscreen has been created in an enhancement project, and this project has been activated, will the system process the screen exit.
- You create subscreens in X function groups. Normally, these function groups also contain program exits.

Calling Customer Subscreens

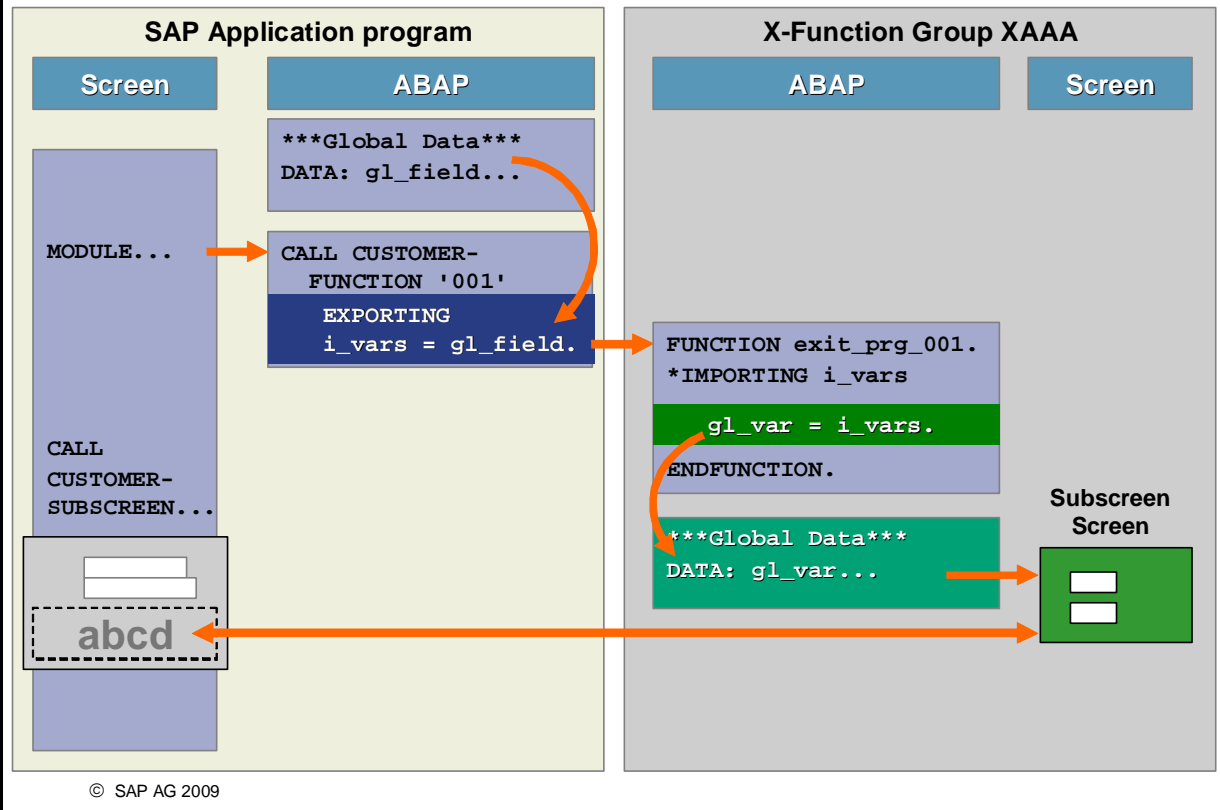
SAP



- Whenever the statement
`CALL CUSTOMER-SUBSCREEN <area> INCLUDING <X-function-pool>`
`<screen_number>`
 occurs at PBO in the flow control of a screen, a subscreen is included in the subscreen area defined by SAP application programmers. At this point, all modules called during the PBO event of the subscreen are also processed.
- The PAI event of a subscreen is processed when the calling screen calls the subscreen during its PAI event using the statement
`CALL CUSTOMER-SUBSCREEN <area>.`

Transporting Data to Subscreens

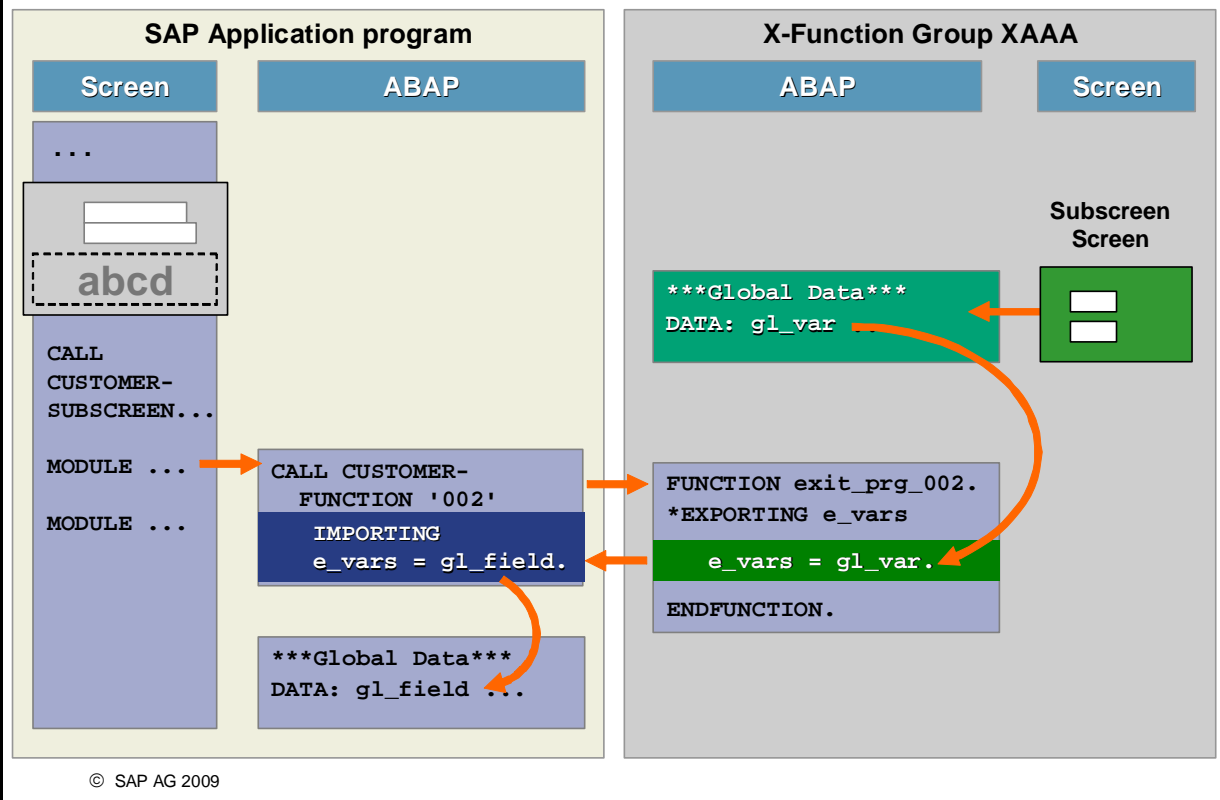
SAP



- The global data of the calling program is not known to the X function group that contains your subscreen; SAP application programmers use program exits to explicitly provide this data to subscreens.
- To facilitate data transport, modules are called in the flow control of the calling program that contain program exits for transferring data via interface parameters.
- The corresponding exit function modules are in the same X function group in which you must create the customer subscreen.

Transporting Data from Subscreens

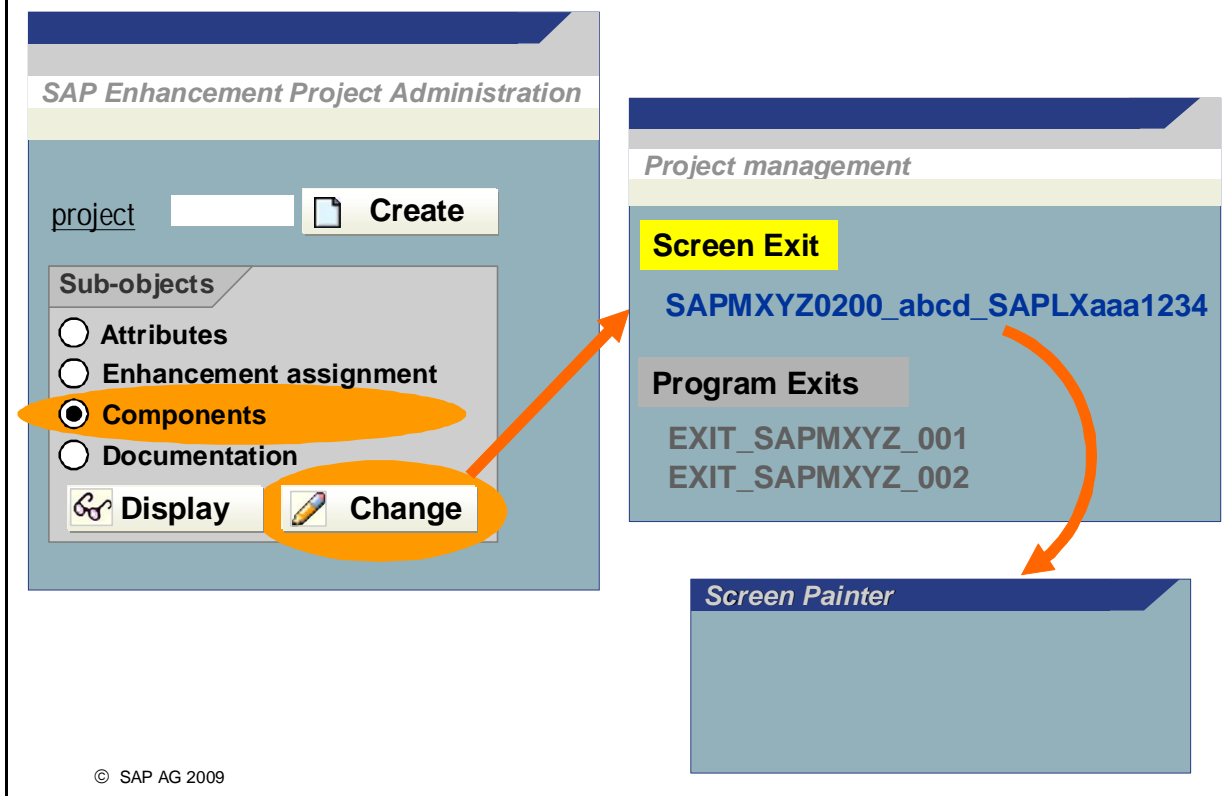
SAP



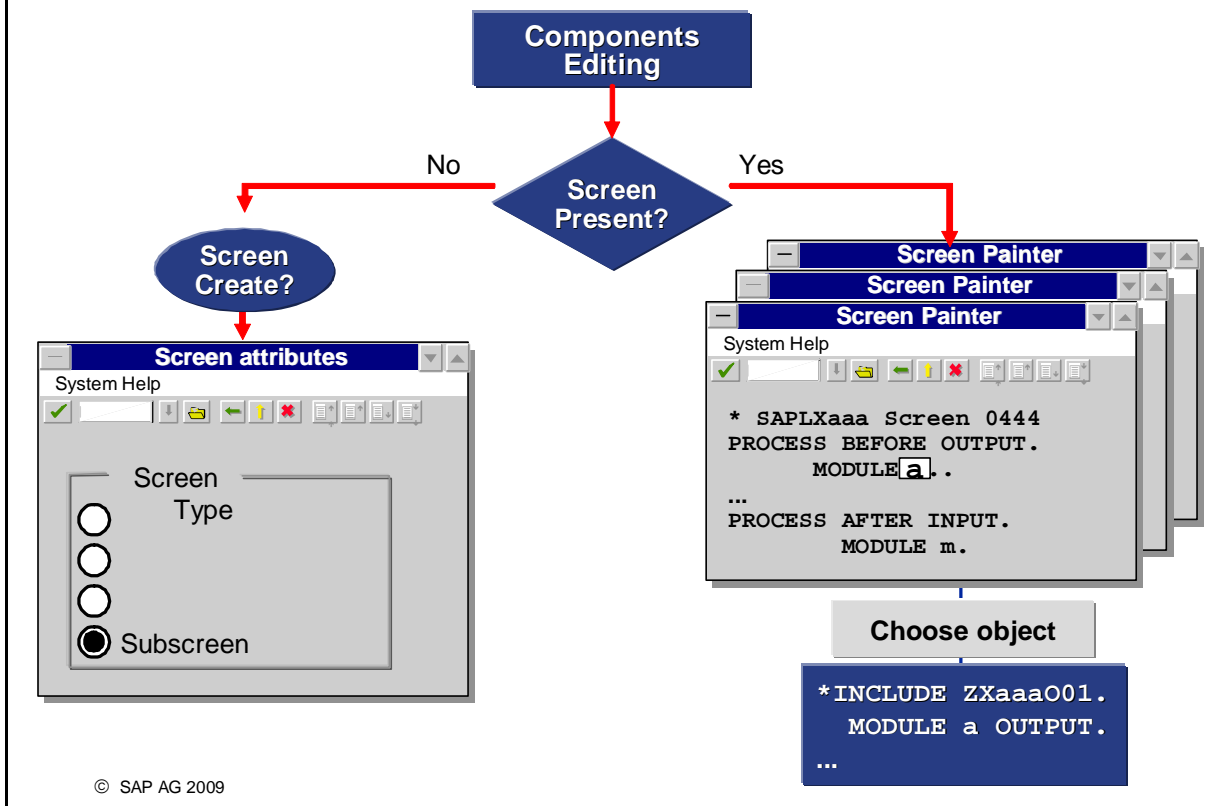
- Data must be transported in the other direction as well, since global data from the X function group that contains the user entries on your subscreen is not known to the calling program either. For this reason, SAP application programmers use program exits to return any data that was changed in the subscreen to the calling program.
- To do this, the system calls a module at the PAI event of the main screen, which contains a program exit that can receive the relevant global data for the X function group.

Naming and Editing Screen Exits

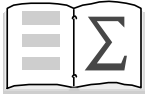
SAP



- Screen exits are edited with the project management transaction (CMOD).
- The technical names of screen exits consist of the name of the calling program, the four-digit screen number for the main screen, and the name of the subscreen area, followed by the name of the X function group's program and the number of the subscreen.



- The SAP development environment supports the creation of customer subscreens and the respective PBO and PAI modules as a part of forwards navigation.
- Ensure when creating the subscreen that it has the screen type **Subscreen**.
- You are not allowed to change any of the interfaces in the X function group that the subscreen and the program exits belong to, nor are you allowed to add any of your own function modules.



You are now able to:

- **Use program, menu, and screen exits that are created using customer exit techniques**
- **Explain what components, enhancements and enhancement projects are**
- **Create enhancement projects and edit enhancements and their components**
- **Describe the connection to the Workbench Organizer and the transport system**
- **Transport enhancement projects**

© SAP AG 2006

Exercises



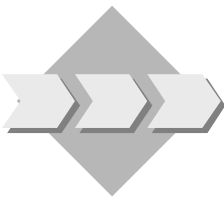
Unit: Customer Exits

Topic: Program Exits



At the conclusion of these exercises, you will be able to:

- Implement an enhancement using a program exit.



Your co-workers have asked you to alter Transaction **BC425_##** so that every time they try to display the details of a flight in the past, a warning message is displayed.

Adjust the program so that there is a warning when a flight in the past is selected. Do not modify the SAP program while doing so.

- 1-1 Check if it is possible to enhance the transaction.
 - 1-1-1 Did the SAP developer implement a customer exit for the given transaction that you can use to add the required functionality?
 - 1-1-2 What is the name of its corresponding enhancement? Choose the enhancement with which you can implement a supplementary check when you leave the first screen of the transaction.
- 1-2 Implement the enhancement.
 - 1-2-1 Name the enhancement project **TG##CUS1**.
 - 1-2-2 Program the following check:

Check if the date that was entered is before today's date (i.e. lies in the past). If this is the case, issue a warning containing an appropriate text. To do this, use message **011** from message class **BC425**
 - 1-2-3 Check your results.

Exercises



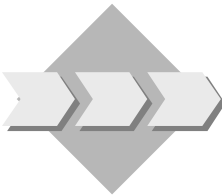
Unit: Customer Exits

Topic: Menu Exits



At the conclusion of these exercises, you will be able to:

- Implement an enhancement with a menu exit in combination with a program exit.



You colleagues are pleased: The new warning messages in transaction **BC425_##** help them to avoid selecting flights from the past. However, they want more...

They want you to create a link in the transaction they use for viewing flights that will allow them to display a list of bookings for their current flight from within the flight display transaction. You have always used a suitable list-generating program.

However, the user must call it separately and enter the current data into the program selection screen: It has the name **SAPBC425_BOOKING_##**.

- 2-1 Examine transaction **BC425_##**. Is it possible to call the program using a menu entry in the transaction?
 - 2-1-1 Did the SAP developer implement a customer exit for the given transaction that you can use to add the required functionality?
 - 2-1-2 What is the name of their corresponding enhancement? Choose the enhancement with which you can implement a menu enhancement.
- 2-2 Implement the enhancement.
 - 2-2-1 Name the enhancement project **TG##CUS2**.
 - 2-2-2 Edit the components of the enhancement: Start the specified program by choosing the supplementary menu entry. Return to Transaction **BC425_##** again when you leave the list.
 - 2-2-3 Transfer the current transaction data to the selection screen of the program that you want to call. To do this, use the data that is available to you in the program exit.
- 2-3 Check your results.

Exercises



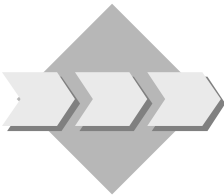
Unit: Customer Exits

Topic: Screen Exits



At the conclusion of these exercises, you will be able to:

- Display further fields in the screen of an SAP transaction and fill them.



"It would be really great if the details list of Transaction **BC425_##** for displaying flights also displayed more data...".

You accept this new challenge from your co-workers and try to solve the problem without having to modify the transaction. Specifically, you start looking for a way to add a couple of new fields to the second screen of this transaction (screen number 200).

- 3-1 What kind of possibilities are there to place additional fields on a screen? Take a closer look at screen 200 in Transaction **BC425_##** and see if this is possible.
 - 3-1-1 Is there a screen exit for enhancing the screen?
 - 3-1-2 If this is the case, what is the name of the corresponding enhancement?
- 3-2 Implement the enhancement to do the following (project name: **TG##CUS3**) :
 - 3-2-1 Add three fields to the screen. The following should appear:
 - Pilot's name
 - Meal
 - The number of free places on the current flight.
 - 3-2-2 Ensure that the data is correctly transported to the subscreen.
- 3-3 Check your results.



Unit: Customer Exits

Topic: Program Exits

- 1-1 You can check if the transaction offers customer exits as follows:
- 1-1-1 *System* → *Status* gives you the name of the corresponding program (SAPBC425_FLIGHT##)
 - 1-1-2 To find a customer exit: You can either search for the character string **CALL CUSTOMER-FUNCTION** globally in the main program or you can use the Repository Information System to search for enhancements containing the program name in the technical name of the component (restrict the search with **EXIT_SAPBC425_FLIGHT##_*** in the component name).

The enhancement you were looking for has the name **SBC##E01**. The documentation for the enhancement shows that it is intended for supplementary checks of the first screen of the transaction.
- 1-2 Choose transaction **CMOD** to implement the enhancement.
- 1-2-1 You can go to transaction CMOD with the menu path *Tools* → *ABAP Workbench* → *Utilities* → *Enhancements* → *Project management*. Create a project named **TG##CUS1** here and save it.
 - 1-2-2 Include enhancement **SBC##E01** that you found in your project.
 - 1-2-3 Edit the components. You open the exit function module's source code. Create the include by double-clicking. The source code you created might look something like this:
IF flight-fldate < sy-datum.
MESSAGE w011(bc425) WITH sy-datum.
ENDIF.

Activate your include. Go back to project management and activate your enhancement project.



Unit: Customer Exits

Topic: Menu Exits

- 2-1 Examine the transaction as in the last exercise. It is useful to search using the Repository Information System or transaction CMOD.
 - 2-1-1 Search for an enhancement that has a component name that contains the program name (restrict your search using the component name **SAPBC425_FLIGHT##***)
 - 2-1-2 The enhancement you were looking for has the name **SBC##E02**.
- 2-2 Choose transaction **CMOD** to implement the enhancement.
 - 2-2-1 You can go to transaction CMOD with the menu path **Tools → ABAP Workbench → Utilities → Enhancements → Project management**. Create a project named **TG##CUS2** here and save it.
 - 2-2-2 Include enhancement **SBC##E02** that you found in your project. Edit the enhancement's components. Assign a menu text. Double-click on the program exit to edit it. Create the customer include using forward navigation.
 - 2-2-3 The source text of the include should be as follows for group ##:
SUBMIT sapbc425_booking_##
WITH so_car = flight-carrid
WITH so_con = flight-connid
WITH so_fld = flight-fldate
AND RETURN.
Activate the include program. Activate the enhancement project.



Unit: Customer Exits

Topic: Screen Exits

- 3-1 Look at the transaction screens in the Screen Painter. You will see that screen 200 of transaction **BC425_##** offers a screen exit.
- 3-1-1 Examine the flow logic of the screens for character string **CALL CUSTOMER-SUBSCREEN**. You will see that screen 200 of transaction **BC425_##** offers a screen exit.
- 3-1-2 You can get the name of the enhancement for example by searching in the Repository Information System (see previous exercises). The name of the enhancement is **SBC##E03**.
- 3-2 Implement the enhancement in the same way as described in the previous exercises. Create a project called **TG##CUS3** in transaction CMOD. Include enhancement **SBC##E03** in your project. Edit the enhancement's components.
- 3-2-1 Use the screen exit to enhance the screen. You can create screen 0500 by double-clicking on the corresponding enhancement component. Make sure that you choose screen type "Subscreen". Copy the fields from the corresponding structure **SFLIGHT##** of the Dictionary. You have two options for positioning a field for the free spaces on the screen:
- You declare a corresponding variable in the TOP include of the X function group, and generate the program. You can then place this program field on the screen. Or, you can enhance your append structure. You should not do this in the exercise since the trainer uses a program to fill the fields of the append structure. Enhancing the append structure could result in errors in this program.
- 3-2-2 Use the enhancement's program exit for correct data transport. Create the customer include and enter the following source code:
- ```
MOVE-CORRESPONDING flight TO sflight##.
seatsfree = flight-seatsmax – flight-seatsocc.
TOP include:
TABLES: sflight##.
DATA: seatsfree type s_seatsocc.
Activate the program and then your enhancement project.
```
- 3-3 Execute transaction **BC425\_##** and check your results.



### Contents:

- What are business transaction events (BTE)?
- Different kinds of BTEs
- Finding Business Transaction Events
- Using Business Transaction Events
- Comparison: Customer Exits and Business Transaction Events

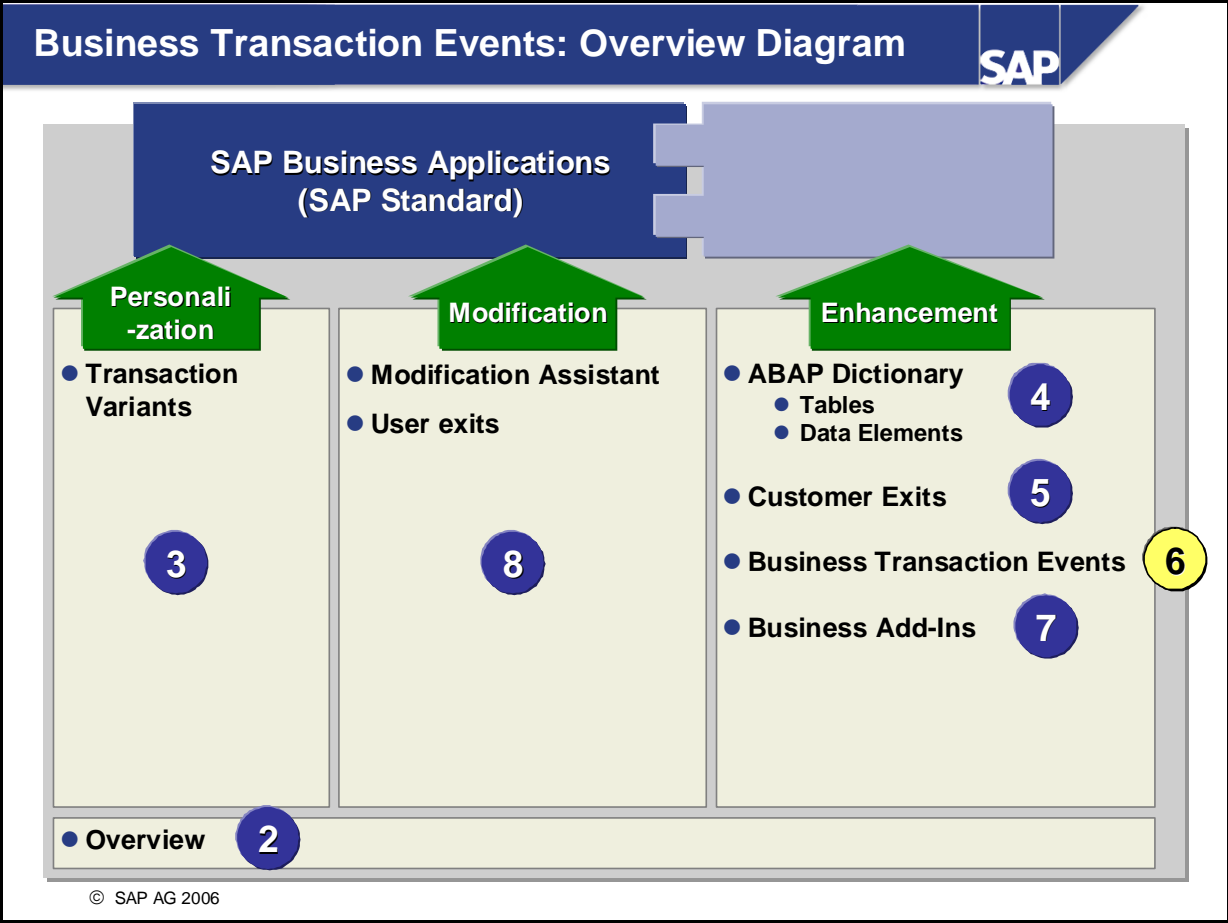
© SAP AG 2006

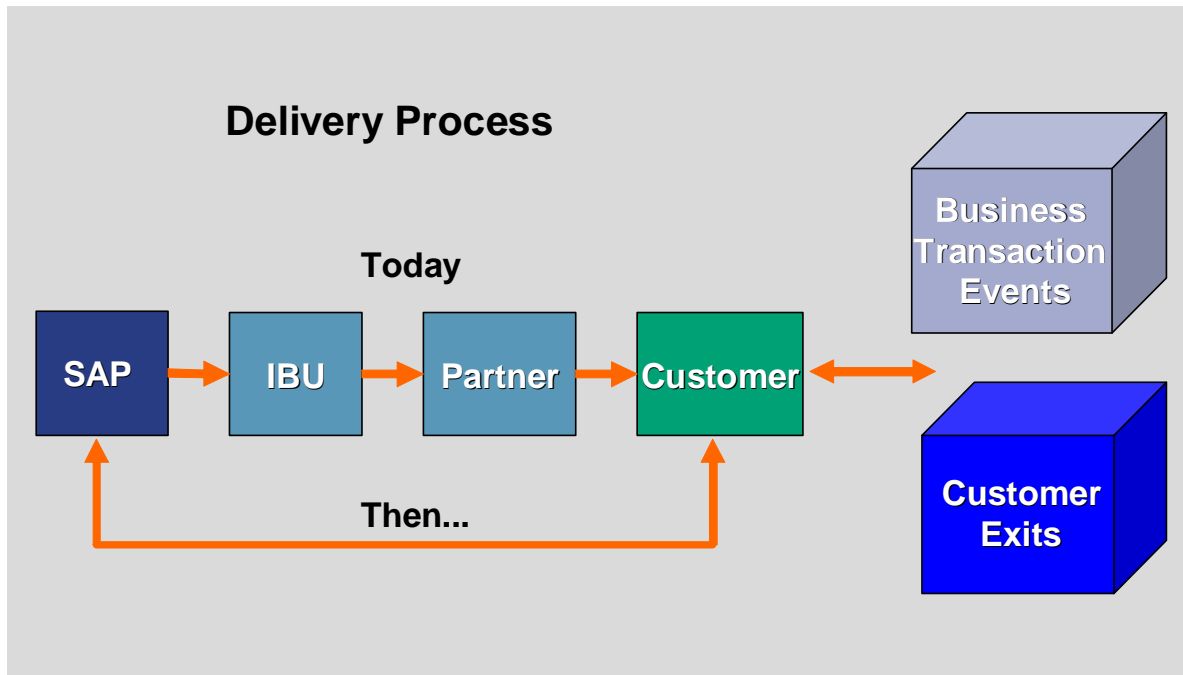


**At the conclusion of this unit, you will be able to:**

- **Describe what Business Transaction Events (BTEs) are**
- **Find a suitable Business Transaction Event**
- **Use Business Transaction Events to enhance SAP software**

© SAP AG 2006





© SAP AG 2006

- Software delivery has changed considerably from the earlier process: Previously, only two participants were involved - SAP (the producer) delivered the software directly to the end-user. Customers could enhance this standard using customer exits.
- Due to strong component-orientation, today many more participants are involved in the software deliver process: SAP delivers the R/3 Standard as base software to an Industrial Business Unit (IBU), who then develop and offer encapsulated functions. The next link in the chain might be a partner firm, which builds its own Complementary Software Program (CSP) solution based on R/3. The last link in the chain is the customer, as before.
- All of the parties involved in this process are potential users and providers of enhancements. This requirement cannot be satisfied by customer exits, which can only be used once. Consequently, SAP developed a new enhancement technique in Release 4.0, which allows enhancements to be reused.



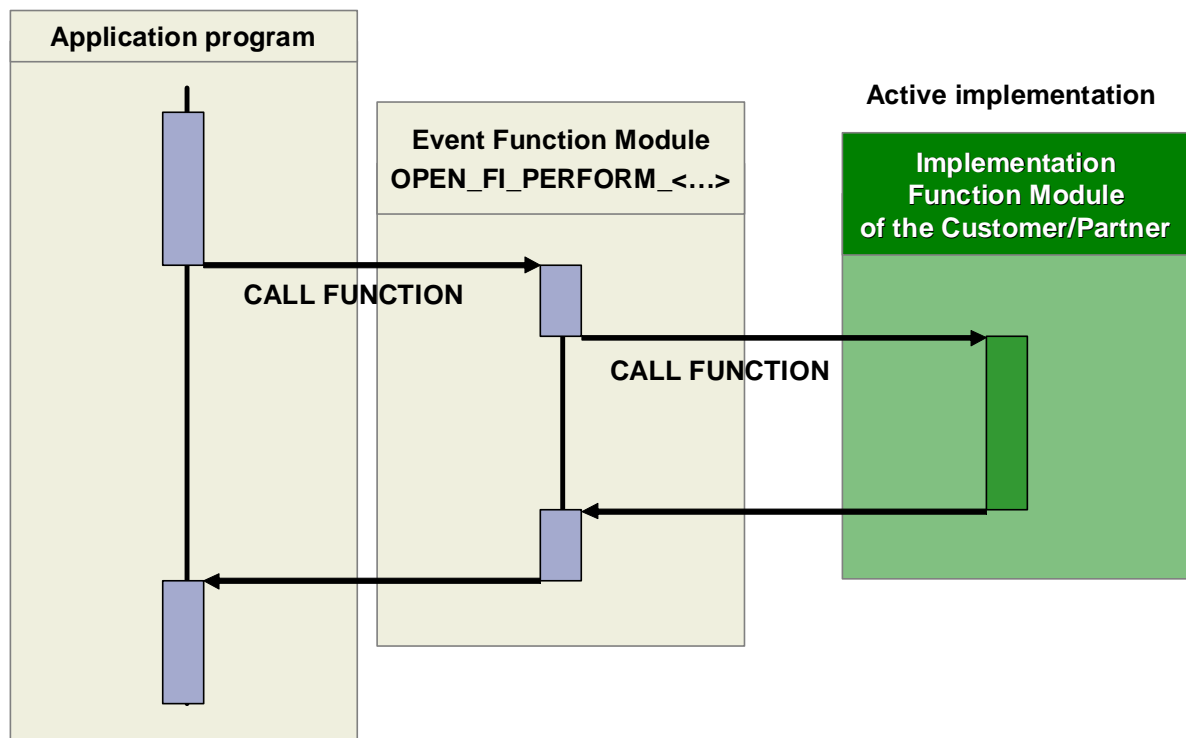


- **Technique for Implementing Program Exits  
(Developed from SAP Financial Accounting)**
- **Have predefined interfaces allowing customer access**
- **Have one of the following types of interfaces (BTEs):**
  - **Publish & Subscribe**
  - **Process interfaces**

© SAP AG 2009

- Business Transaction Events (BTE) allow you to attach additional components, in the form of a function module, for example, to the R/3 system.
- Two types of interface are available for this purpose:
  - **Publish & Subscribe interfaces:**  
These interfaces inform external software that certain events have taken place in an SAP standard application and provide them with the data produced. The external software returns no data to the SAP Standard System.
  - **Process interfaces:**  
These interfaces are used to control a business process differently than the way in which it is handled in the standard R/3 System. They intervene in the standard process, and return data to the SAP application.
- You can attach various external developments to the R/3 System. You can create additional developments using the ABAP Workbench.

## BTE Functions: How They Work / Flow Diagram



© SAP AG 2006

- The graphic shows the process flow of an SAP program. The program contains an enhancement in the form of a Business Transaction Event. A function module is called in the SAP program, which determines and processes the active implementations. The names of the event function modules begin with "OPEN\_FI\_PERFORM\_" or "OUTBOUND\_CALL\_".
- The event function module OPEN\_FI\_PERFORM\_<...> or OUTBOUND\_CALL\_<...> determines the active implementations for each enhancement and stores them in an internal table. Function modules are implemented in the sequence defined by the internal table. At this point the system also considers the conditions under which the function module will be processed in the customer namespace. For example, a country or an application can be entered as a condition. These conditions are also referred to as filter values.

## BTE: Call Syntax in the SAP Program

SAP

```
REPORT ...
```

```
...
```

```
CALL FUNCTION
 'OPEN_FI_PERFORM_00001350_E'
 EXPORTING
 <i_variables>.
```

```
FUNCTION-POOL <sap_fp>.
```

```
...
```

```
FUNCTION open_fi_perform_00001350_e.
```

```
...
```

```
LOOP AT itab INTO wa.
```

```
 CALL FUNCTION wa-fb_name
 EXPORTING
 <i_variables>.
```

```
ENDLOOP.
```

```
...
```

```
ENDFUNCTION.
```

wa

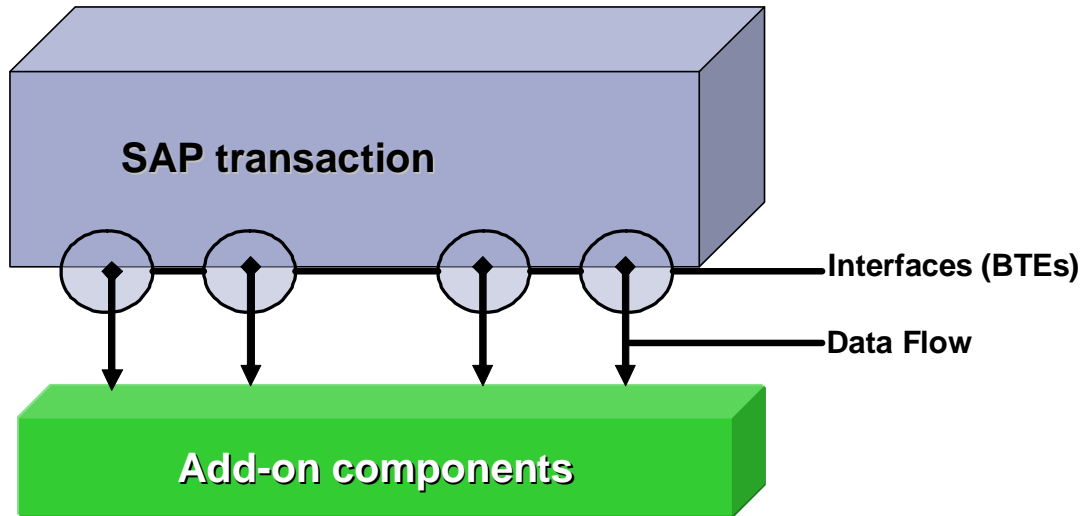
|  |  |
|--|--|
|  |  |
|--|--|

itab

| Business transaction event (BTE) | FB_NAME        |
|----------------------------------|----------------|
| 00001350                         | Z_FUBA001      |
| 00001350                         | /ABC/FUBA002   |
| 00001350                         | /CUSNR/FUBA003 |

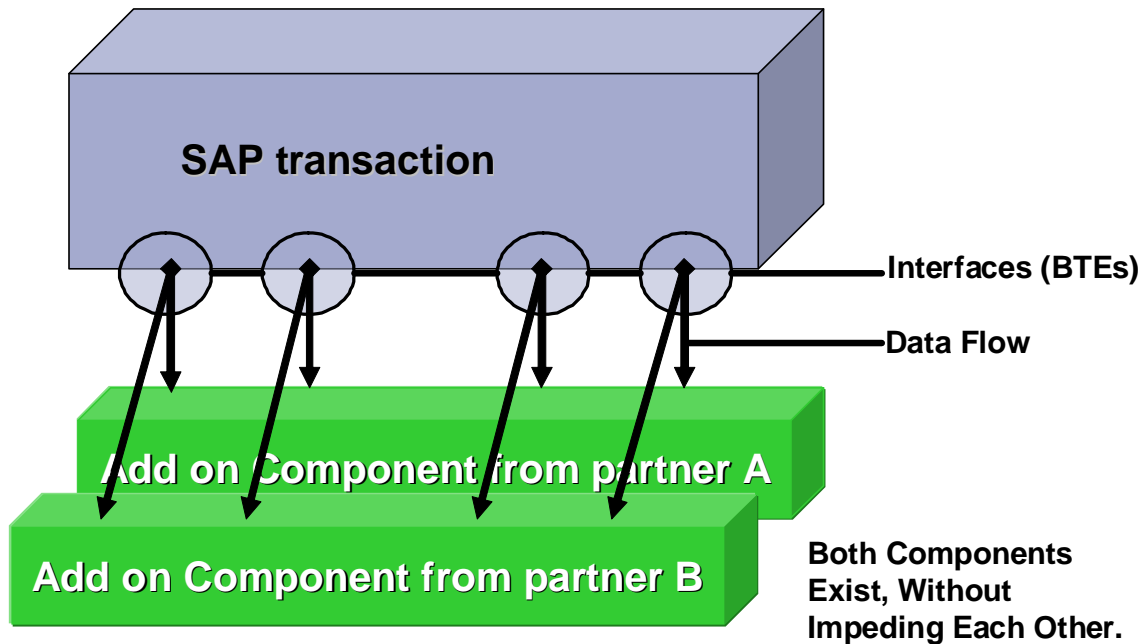
© SAP AG 2006

- This graphic shows the syntax used to call a program enhancement using a business transaction event.
- In the SAP application program, function module "OPEN\_FI\_PERFORM\_<...>" (or OUTBOUND\_CALL\_<...>") is called. The application transfers the interface data to this service function module. The interface is predefined by the SAP developer.
- The service function module also searches for active implementations and enters these in an internal table. The implementations it finds are then processed in a loop.



© SAP AG 2009

- The example above relates to Publish & Subscribe interfaces. In this case, data only flows in one direction - from the SAP application to the additional component.
- SAP application developers make interfaces available to you at certain callup points in a transaction. You can define additional logic at these points.
- In a very basic scenario, SAP partners and customers can use the interfaces themselves. In this case business transaction events function in much the same manner as customer exits (see the unit on "Enhancements using Customer Exits").



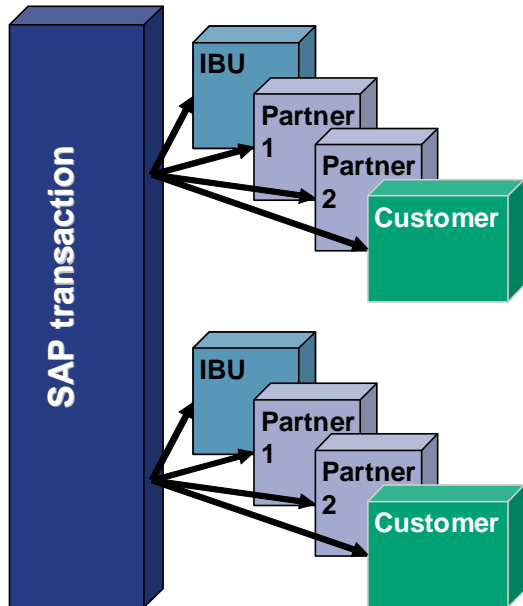
© SAP AG 2009

- The above scenario also pertains solely to Publish & Subscribe interfaces.
- In contrast to customer exits, business transaction events allow you to use an interface for multiple types of additional logic.
- If this is the case, you can decide if all enhancements or only specific enhancements should be executed for a given BTE.
- Therefore, the two enhancements coexist, without impeding each other.

## Publish & Subscribe Interfaces; Process Interfaces

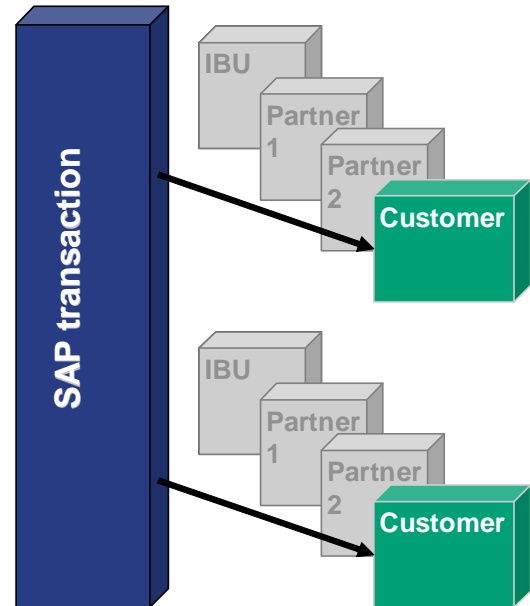


### ● P/S interfaces



**Executing Multiple Implementations Possible**

### ● Process interfaces



**Exactly One Add-On Component is Executed**

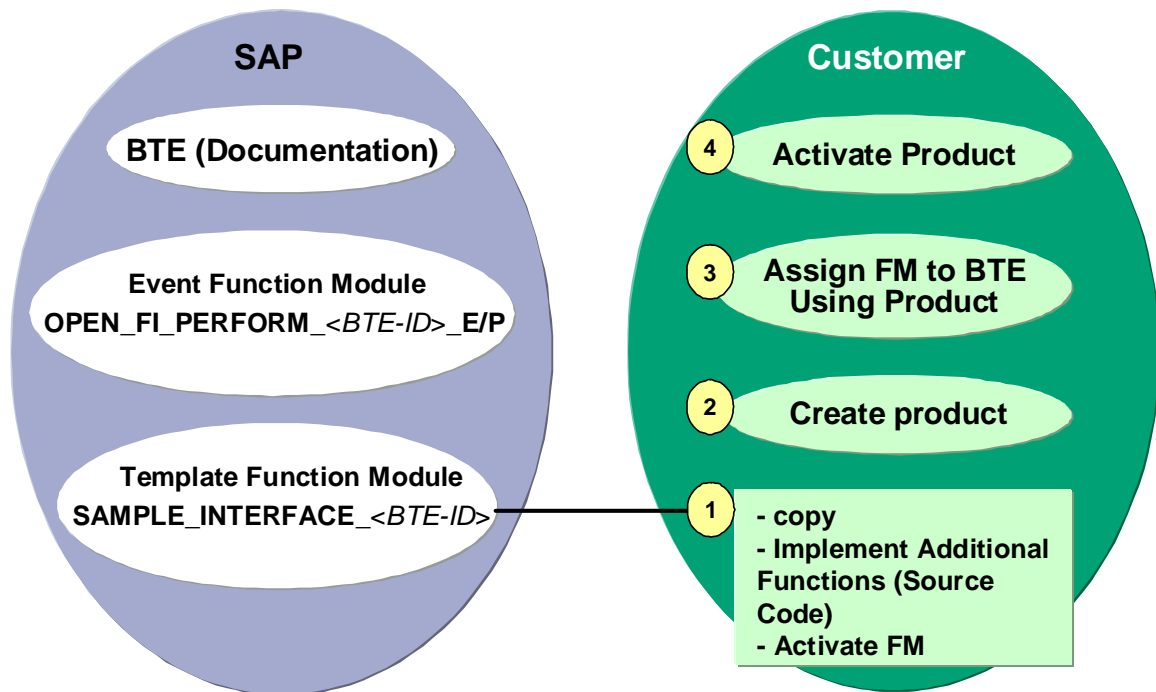
© SAP AG 2006

#### ■ Publish & Subscribe interfaces:

- Allow you to start one or more (multiple) **additional operations** when a particular event is triggered. They do not influence the standard R/3 program in any way.
- Multiple operations do not interfere with each other.
- Add on components can only import data.
- Additional checks (authorizations, existing duplicates, and so on)

#### ■ Process interfaces

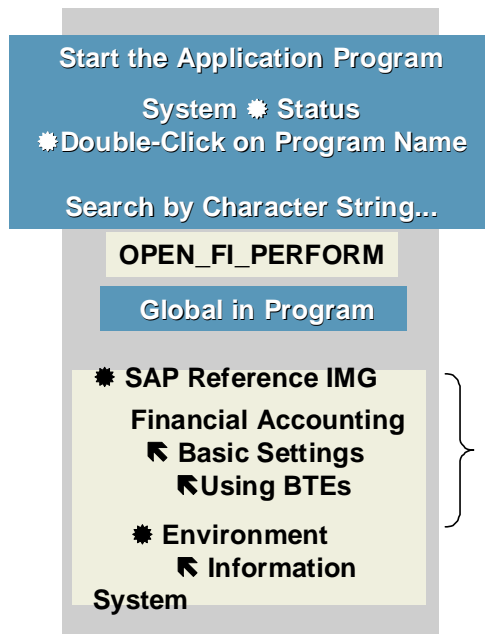
- In contrast to Publish & Subscribe interfaces, data exchange takes place in both directions with process interfaces. This influences the number of additions that can be attached to the interface.
- When an event is triggered, a process in the standard program can only be replaced by **one** external process using the process interface.
- If you are using an add on from an SAP partner that uses a process interface, this enhancement is processed at runtime. If you choose to use this same process interface for one of your own developments, the partner enhancement is discarded and your own enhancement is processed at runtime instead.



© SAP AG 2009

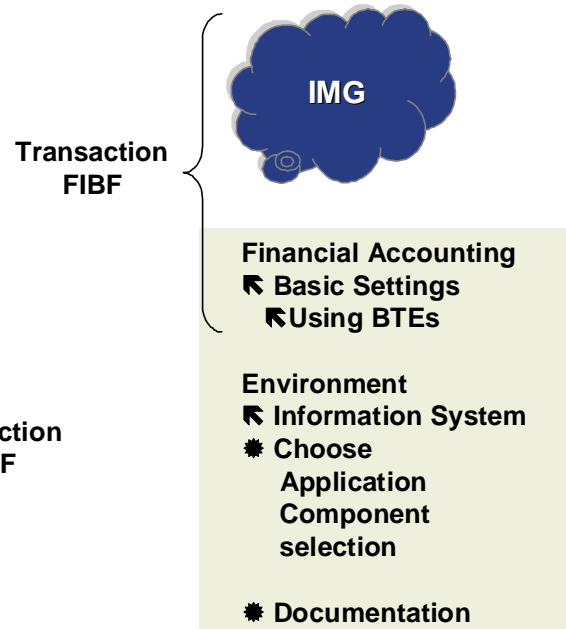
- Business transaction events allow you to implement additional logic in a task function, similar to program exits. SAP application programmers determine where to place business transaction events in a task function and what data should be transferred at each point. They also create sample function modules complete with short texts, an interface, and documentation, and describe the functions possible with the enhancement in the accompanying SAP documentation.
- First, SAP application programmers assign a business transaction event an eight digit number by which it can be identified. These numbers should observe a particular convention. For example, the names of events that are integrated into the same SAP application program should be identical at the fifth and sixth characters.
- The SAP developer registers the event and creates a template function module, `sample_interface_<BTE-ID>`, which establishes the interface for the user.

## ● Search by Program



Transaction FIBF

## ● Using tools



© SAP AG 2006

- To find out directly whether an application transaction offers Business Transaction Events, you can use the procedure described on the left-hand side of the graphic as follows: Search the source code of the application program for the character string "OPEN\_FI\_PERFORM". The number that completes the name of the function module is also the name of the event.
- In the SAP Customizing Implementation Guide (IMG), you will find the entry "Use business transaction events " under the "Financial Accounting Global Settings" node of the Financial Accounting area. Choosing this entry calls transaction FIBF, where you can execute all of the actions necessary for using Business Transaction Events.
- Under Environment, you will find search functions that you can use to identify appropriate business transaction events. You can view the documentation for the event from the list.



Event ... Settings Environment

SAP Business Framework: Bu

InformSystem (P/S)

InfoSystem (Processes) ts

Business Transaction Events: P/S Interfaces

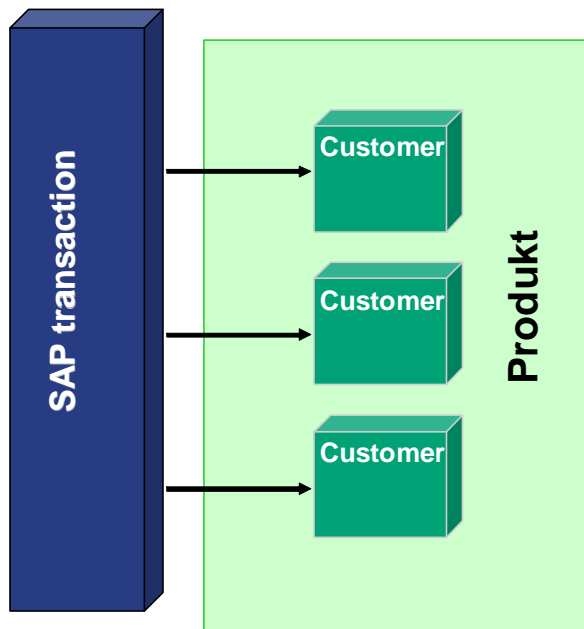
| Key | Description |
|-----|-------------|
| ... | ...         |
| ... | ...         |
| ... | ...         |

\*\*\* List of Selected BTEs \*\*\*

- Template FM
- FM Interface
- Documentation

© SAP AG 2006

- The "Environment" menu of the service transaction FIBF contains two programs that you can use to search for BTEs. You can restrict the search by using various parameters.
- The BTEs that the system finds are displayed in a list. You can then:
  - Display the model function module (start the Function Builder and copy it, for example)
  - Display the interface
  - Display the documentation
- The documentation provides a clear explanation of how to use the enhancement and any restrictions that apply to it.



- The product groups enhancements together
- It can be activated or deactivated as a whole

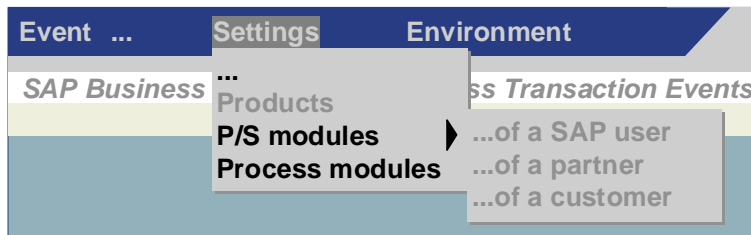
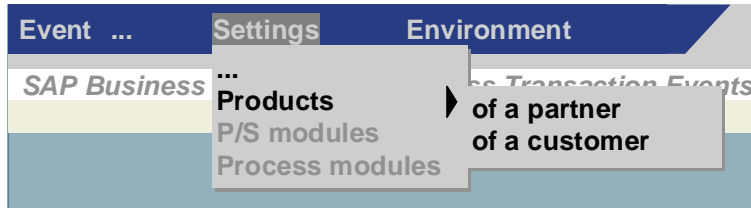
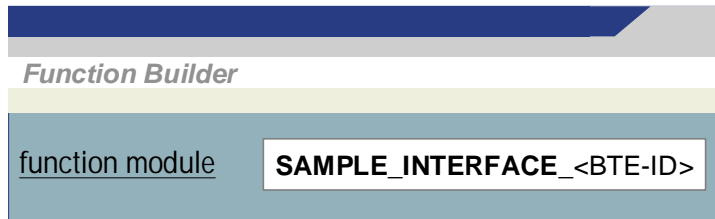


© SAP AG 2006

- Use service transaction FIBF to create a product. A product groups together a collection of enhancements.
- You can create products for various layers in the delivery chain. They define a sequence for processing the implementations of a business transaction event.
- You can only switch each product on or off as a whole entity. This allows the user to control which enhancements should be processed and which should not. It also ensures the integrity of the whole enhancement.

## Implementing a BTE

SAP



© SAP AG 2009

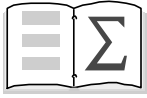
- Copying Template FMs as Customer FMs (Z\_\*)
  - Maintain Source Code
  - Activate
- Create product
- Assign Customer FM to BTE Using Product
- Activate Product

- You can use transaction FIBF (called when you selected Use business transaction events from the financial accounting hierarchy) to carry out all necessary activities prior to using a business transaction event.
- First, choose an interface to which you would like to attach your function module. The Interface button displays the parameter structure for the interface you have selected. You can also use the documentation to determine what functions each interface allows you to perform.
- Use the ABAP Workbench to copy the sample function module sample\_interface\_<n> to the customer namespace (z\_\*) of a customer function group. You must not change the interface. You can fill the module with any source text except COMMIT WORK. Do not use a COMMIT WORK! Do not forget to activate the function module.
- Create a product in the administration screen.
- Assign a number to your function module and product.

| Comparison: Customer Exits and BTEs |                |                             | SAP |
|-------------------------------------|----------------|-----------------------------|-----|
|                                     | Customer Exits | Business Transaction Events |     |
| Program Exit                        | +              | +                           |     |
| Menu Exit                           | +              | -                           |     |
| Screen Exit                         | +              | -                           |     |
| Append Fields On Screens            | +              | -                           |     |
| Administration Level                | +              | -                           |     |
| Reusable                            | -              | +                           |     |
| Client-Dependent                    | -              | +                           |     |
| Filter-specific                     | -              | +                           |     |

© SAP AG 2009

- In contrast to customer exits, business transaction events are client-specific. This means that the same event can be used in different clients for different purposes.
- Business transaction events may also be used more than once.



**You are now able to:**

- **Describe what Business Transaction Events (BTEs) are**
- **Find a suitable Business Transaction Event**
- **Use Business Transaction Events to enhance SAP software**

© SAP AG 2006



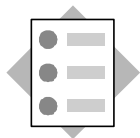
### Contents:

- Searching for Business Add-Ins
- Implementing Business Add-Ins
- (Creating Business Add-Ins)

© SAP AG 2006

Internal Use SAP Partner Only

Internal Use SAP Partner Only



**At the conclusion of this unit, you will be able to:**

- **Search for Business Add-Ins**
- **Implement Business Add-Ins**
- **(Create Business Add-Ins)**

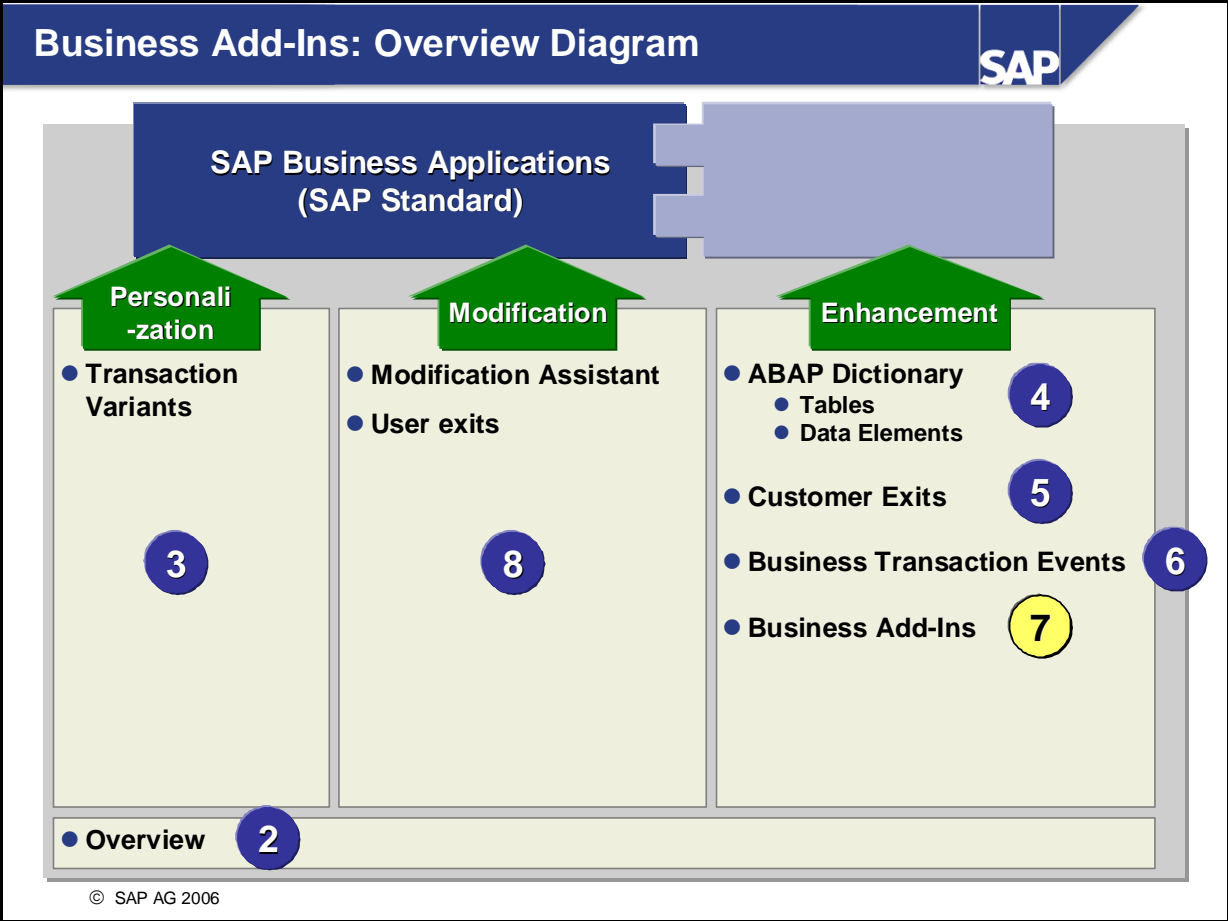
© SAP AG 2006





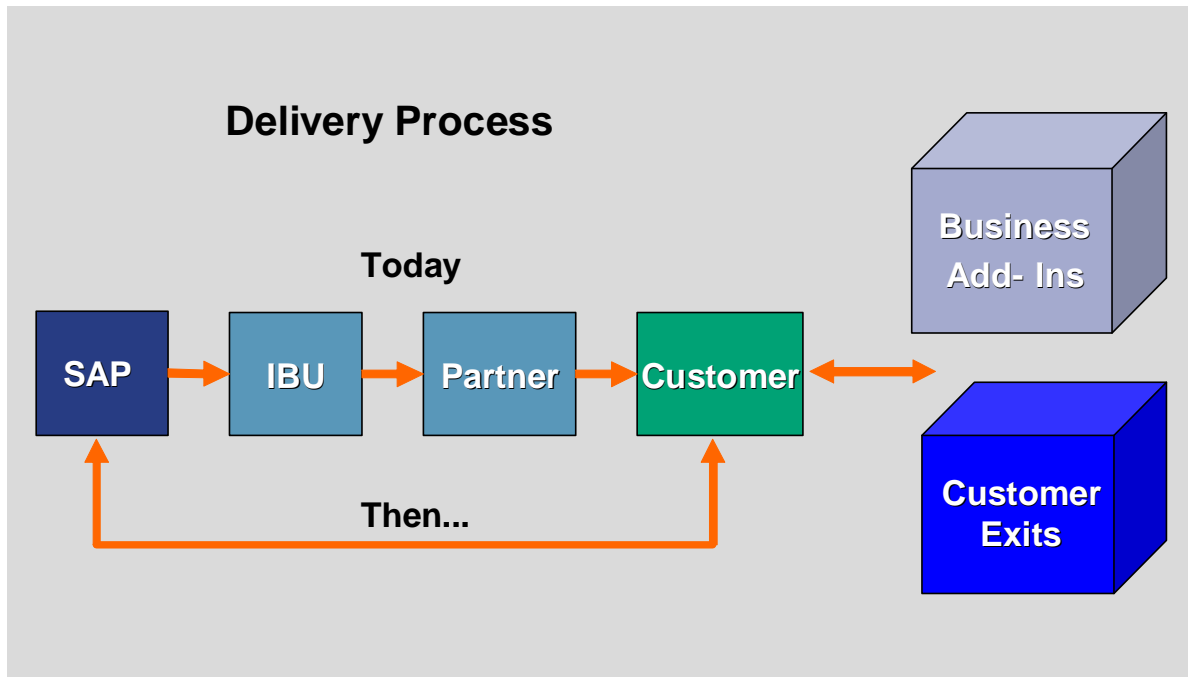
- You want to add new functions to a flight maintenance transaction. To minimize the adjustment effort during the next upgrade, you want the implementation to contain as few modifications as possible. In particular, you want to use BADIs made available by SAP - where available.

© SAP AG 2006



- **Disadvantages of earlier enhancement techniques**
  - Could only be used once (customer exits)
  - No screen enhancement (business transaction events (BTEs))
  - No menu enhancement (BTEs)
  - No administration level (BTEs)
- **Requirements for new enhancement techniques:**
  - Reusable
  - All enhancement types (program / menu / screen exit)
  - Administration level
  - Implemented using latest technology

© SAP AG 2006



© SAP AG 2006



**At the conclusion of this topic, you will be able to:**

- **Describe an ABAP Objects interface**

### Class (description of objects, e.g. class of bookings)

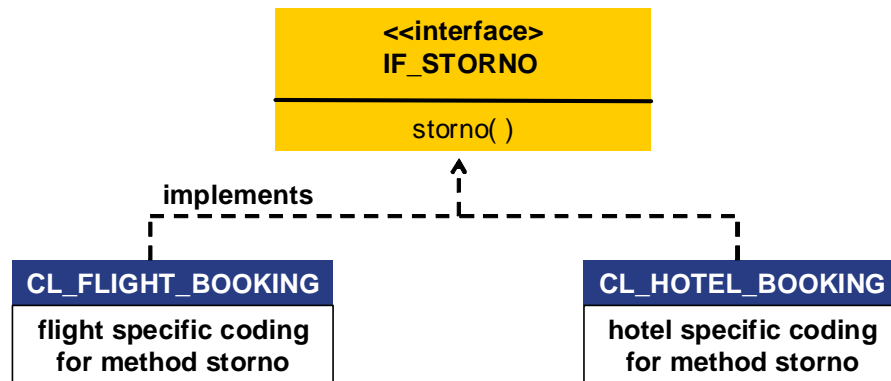
- **Class attributes** (instance independent  
e.g. total number of bookings)
- **Class methods** (possibility to access class attributes,  
e.g. determine total number of bookings)

### Object (Instance) (representative of a class, e.g. special booking)

- **Instance attributes** (instance dependent, e.g. booking number,  
status of a booking)
- **Instance methods** (possibility to access instance attributes,  
e.g. cancelation of a booking)

© SAP AG 2009

- The basic approach of object oriented programming is to think in objects. For example a special booking with all the attributes and their special functionality which only work on that attributes.
- The technical description of an object is given by its class and the object is an instance of a class. Each object only exists while the program is running.
- There are not only object dependent descriptions in the class, but also components which are valid in general and have no special dependence to a concrete object. These components are called static components (class attributes, class methods).



© SAP AG 2009

- An interface is a collection of formal defined methods without any coding, that means name of methods and their signature.
- An interface can be integrated to a class. The class is then called as the implementing class of the interface. In the class you are able to implement all interface methods with their class specific implementation.
- If you have several interface implementing classes with their class specific coding you have reached the following: there is only one method definition (in the interface), but several different functionalities (in the implementing classes). In the next but one slide we will see how we can get a special effect using this feature.

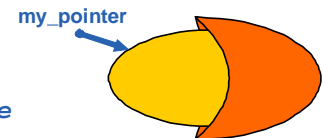
*\* Definition of a reference variable*

```
DATA my_pointer TYPE REF TO cl_flight_booking.
```

my\_pointer →

*\* Creating an object*

```
CREATE OBJECT my_pointer [EXPORTING ...].
```



*\* Calling an instance method using reference variable*

```
CALL METHOD my_pointer->get_flight_booking [...].
```

*\* Calling a class method using class prefix*

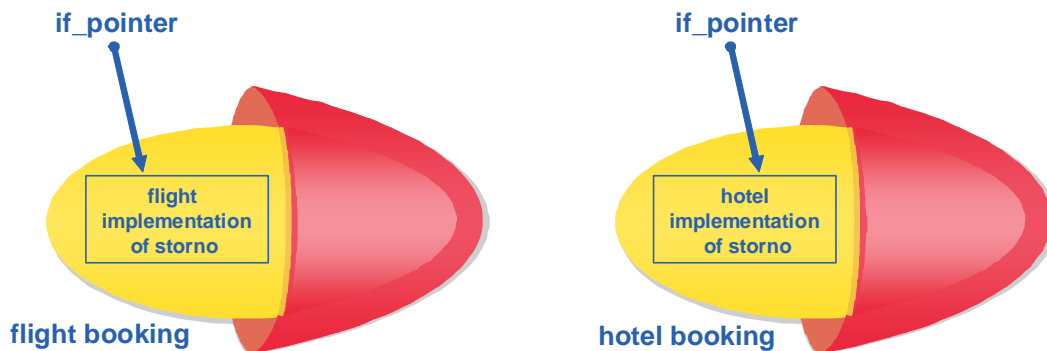
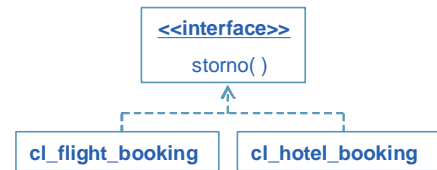
```
CALL METHOD cl_flight_booking=>get_n_o_bookings [...].
```

© SAP AG 2009

- Instances were created during runtime and therefore they have no name. That means you have to call an instance over its reference variable (pointer) which refers to the instance.
- First of all you have to define a reference variable in the declaration block of a program with the additional specification REF TO <class>. That means that this reference variable shows during runtime to a specific instance of this class which is not yet created. It doesn't mean that the pointer shows to the class, because a class is not anything physical. It is only a type description.
- To create an object of the class, use the defined reference variable. The reference variable then refers automatically to this object.
- Now you can call (instance) components of the object using the reference variable as prefix.
- To call class components (for example class methods), you don't need an object, because the class components are independent from any object of this class. Use therefore the class prefix to call a class method for example.



```
DATA if_pointer type ref to <interface>.
...
CALL METHOD if_pointer->storno.
```



© SAP AG 2009

- There is no way to instantiate an interface, because interfaces doesn't have any coding. That's why the statement CREATE OBJECT to a reference that ist defined to an interface does't make sense.
- The implementation of an interface is given by a class which is hierarchically under the interface. So a reference which is defined to an interface is able to refer to an object of the implementing class.
- Interface references are only able to call the interface components of an object of the implementing class. It depends on their type definition (static type). The functionality then runs in the context of this implementing class. So, if you have several implementing classes of an interface, you can use the same semantic of calling an interface method for the objects of these classes. But for each object runs different functionality.



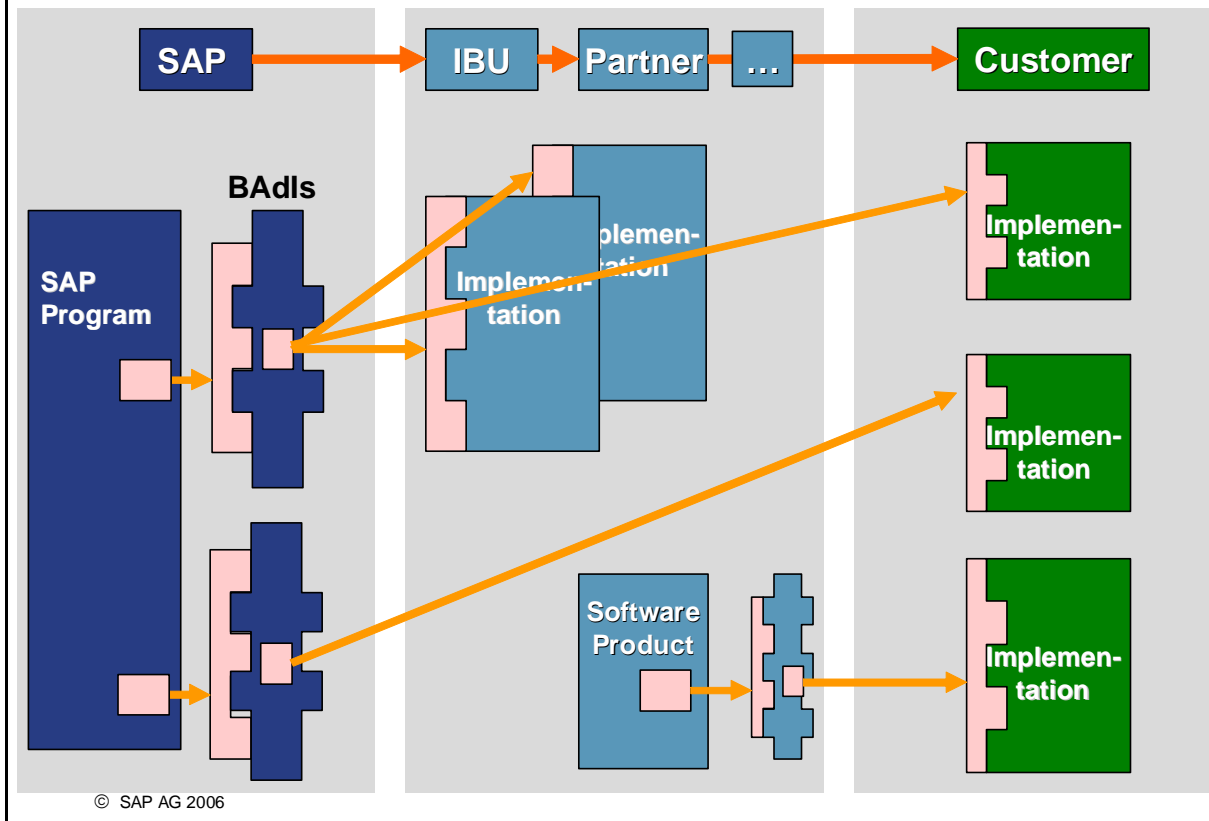
**At the conclusion of this topic, you will be able to:**

- **Search for available Business Add-Ins in SAP Programs**
- **Use Business Add-Ins to implement enhancements to programs**

© SAP AG 2006

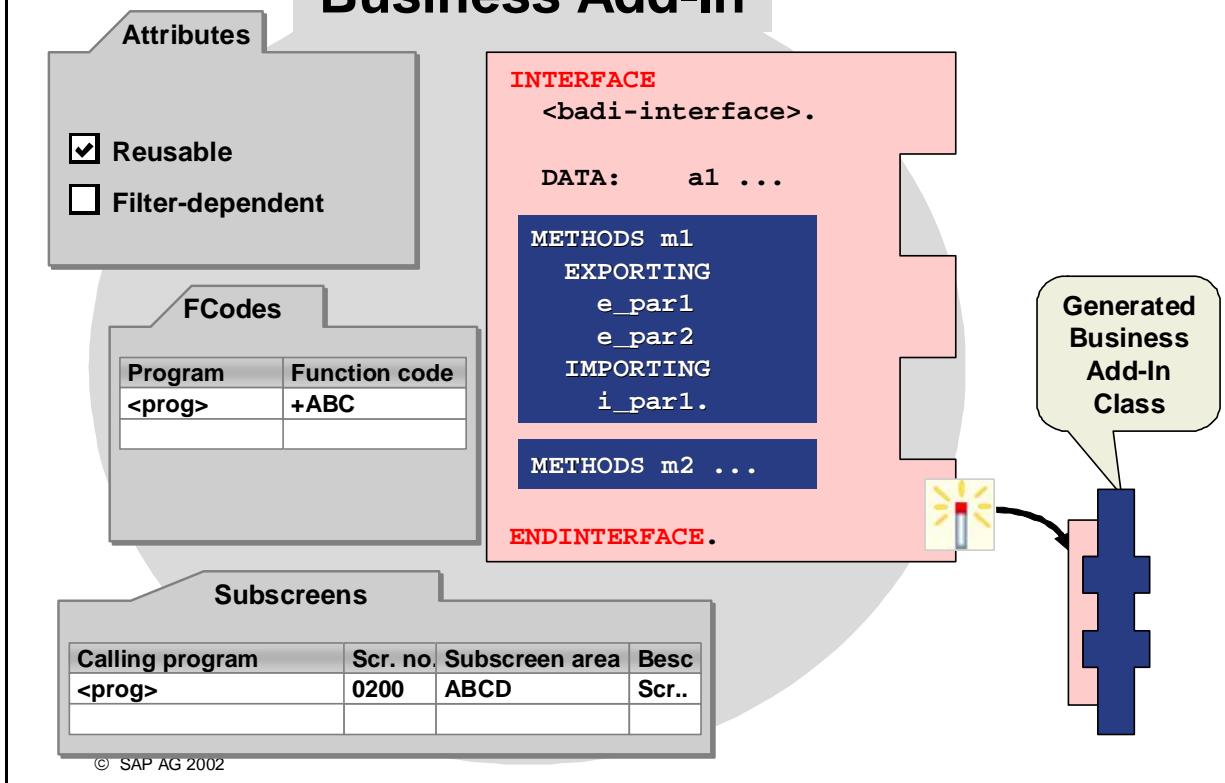
## Business Add-Ins: Architecture

SAP

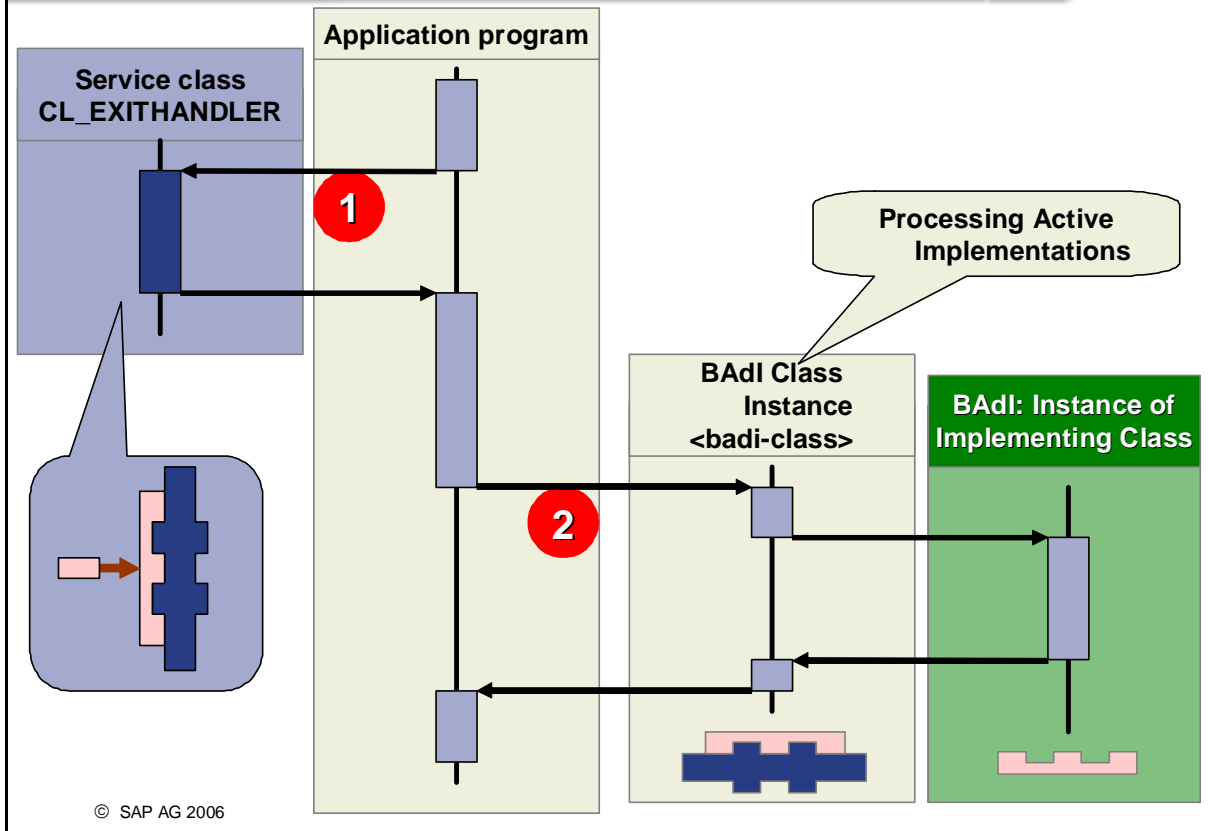


- Business Add-Ins, unlike customer exits, take into account the changes to the software delivery process. The upper section of the screen displays the typical delivery landscape: It no longer consists only of provider and user. Instead, it can now contain a whole chain of intermediate providers.
- The lower section of the screen explains how Business Add-Ins function: An SAP application program provides the enhancement option. This requires at least one interface and a BAdI class that implements it. The interface is implemented by the user.
- The strength of this concept lies in its reusability: A BAdI can be implemented many times, even by parties further towards the end of the software delivery chain.
- Business Add-Ins also enable each party to the software delivery chain to offer enhancements.

## Business Add-In



- A Business Add-In contains the components of an enhancement. Each Business Add-In can contain the following components:
  - Program enhancements: In the Business Add-In, the interfaces for program enhancements are defined in the form of interface methods. This interface is used to implement the enhancement. The SAP program calls the interface methods of the generated Business Add-In class.
  - Menu enhancements: As with customer exits, you can enter function codes in a Business Add-In. These menu entries are available in the GUI definition and are made visible when the Business Add-In is implemented.
  - Screen enhancements: As with customer exits, you can define screen enhancements in a Business Add-In, which you can then implement.
- Several components are created when you define a Business Add-In:
  - Interface
  - Generated class (Business Add-In class) that implements the interface
- The generated class (Business Add-In class) performs the following tasks:
  - Filtering: If you implement a filter-dependent Business Add-In, the Add-In class ensures that only the relevant implementations are called.
  - Control: The adapter class calls the active implementations.



- This graphic shows the process flow of a program that contains a BAdI call. It enables us to see the possibilities and limitations inherent in Business Add-Ins.
- Not displayed: You must declare a reference variable referring to the BAdI interface in the declaration section.
- An object reference is generated in the first step. This replaces the service class CL\_EXITHANDLER provided by SAP. We will discuss the precise syntax later on. This generates the conditions for calling methods of program enhancements.
- When you define a Business Add-In, the system generates a BAdI class, which implements the interface. In call (2), the interface method of the BAdI class is called. The BAdI class searches for all of the active implementations of the Business Add-In and calls the implemented methods.

```
REPORT <sap_program>.
```

```
DATA r_var TYPE REF TO <badi-interface>.
```

...

```
CALL METHOD
 cl_exithandler=>get_instance
 CHANGING
 instance = r_var.
```

1

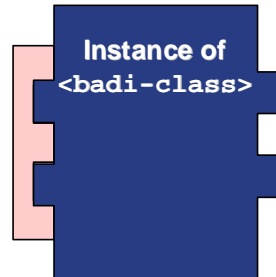
r\_var

...

```
CALL METHOD r_var->meth_abc
 EXPORTING
 <i_variables>
 IMPORTING
 <e_variables>.
```

2

...



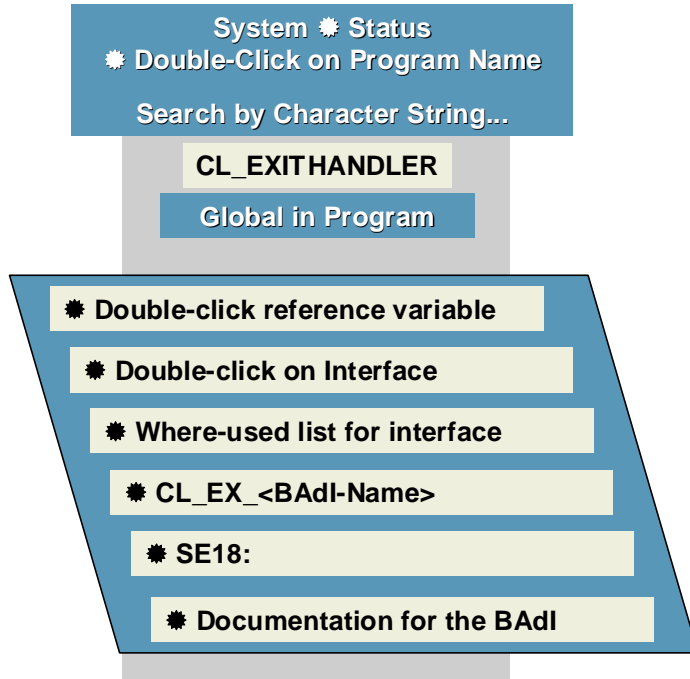
© SAP AG 2006

- This graphic displays the syntax with which you call a Business Add-In. The numbered circles correspond to the calls from the previous page.
- First, a reference variable must be defined that refers to the BAdI interface. The name of the reference variable does not necessarily have to contain the name of the BAdI.
- In the first call (1), an object reference is created. This creates an instance of the generated BAdI class. Only the methods of the interface can be contacted using this object reference.
- You can use this object reference to call the required methods available with the enhancement (2).

## Finding a Business Add-In

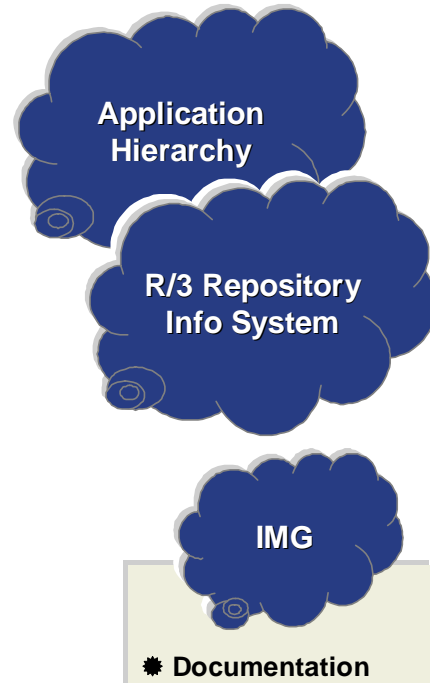
SAP

### • Search by Program



© SAP AG 2006

### • Using tools



- There are various ways of searching for business add-ins:
- You can search in a relevant application program for the string "CL\_EXITHANDLER". If a Business Add-In is called from the program, the "GET\_INSTANCE" method of this class must be called.
- You can then reach the definition of the business add-in using forward navigation. The definition also contains documentation and a guide for implementing the Business Add-In.
- Use the application hierarchy to restrict the components in which you want to search. Start the Repository Information System, then choose Enhancements -> Business Add-Ins to start the relevant search program.
- Alternatively, you can use the entries in the relevant component of the IMG.

## Business Add-Ins: Implementation Maintenance

BAdI-Name \_\_\_\_\_

Implementation Name \_



Display



Change



Create

© SAP AG 2006

- To implement Business Add-Ins, use transaction SE19 (*Tools -> ABAP Workbench -> Utilities -> Business Add-Ins -> Implementation*).
- Enter a name for the implementation and choose Create. A dialog box appears. Enter the name of the Business Add-In. The maintenance screen for the Business Add-In then appears.
- Alternatively, you can use the Business Add-In definition transaction (SE18) to reach its implementations. The menu contains an entry "Implementation", which you can use to get an overview of the existing implementations. You can also create new implementations from here.
- Note: As of *SAP NetWeaver Application Server 7.0* there are new BAdIs, in addition to the older "classical" BAdIs (see the attachment). The initial screen of transaction SE19 has been adjusted appropriately. To create an implementation for a classical BAdI, select "Classical BAdI" in the lower input area of the initial screen, enter the BAdI name in the corresponding field, and choose the button "Create Impl.".



## Implementing Business Add-Ins: Methods

SAP

The screenshot shows the SAP Class Builder interface. At the top, there are three tabs: 'Attributes', 'Interface', and 'FCodes'. The 'Interface' tab is selected. Below the tabs, there are two input fields: 'Interface name' with the value '<badi-interface>' and 'Name of implementing class' with the value '<impl-class>'. Below these fields is a table with two columns: 'Method' and 'Description'. The table contains one row with the method name '<meth\_abc>' and the description 'Business add-in method'. An orange arrow points from this method name to the 'Edit Method' dialog box below. The dialog box has a title bar that says 'Class Builder: Edit Method <badi-interface>~<m...>'. Inside the dialog, there is a 'Signature' button. A yellow callout bubble points to this button with the text 'Shows the method interface'. Below the 'Signature' button, the method signature is displayed: 'METHOD <badi-interface>~<meth\_abc>.' followed by a line with '\* ...' and then 'ENDMETHOD.'. There is also a small icon of a lightbulb in the bottom right corner of the dialog.

© SAP AG 2006

- You can assign any name to the implementing class. However, it is a good idea to observe the proposed naming convention. The suggested name is constructed as follows:
  - Namespace prefix, Y, or Z
  - CL\_ (for class)
  - IM\_ (for implementation)
  - Name of the implementation (without namespace prefix)
- To implement the method, double-click its name. The system starts the Class Builder editor.
- When you have finished, you must activate your objects.

## Implementing Business Add-Ins: Private Methods

SAP

**Class Builder: Edit Method <badi-interface>~<m...>**

METHOD if <badi-interface>~<method>.

CALL METHOD <priv\_method> EXPORTING ... .

END METHOD.

**Class Builder: Change class <impl-class>**

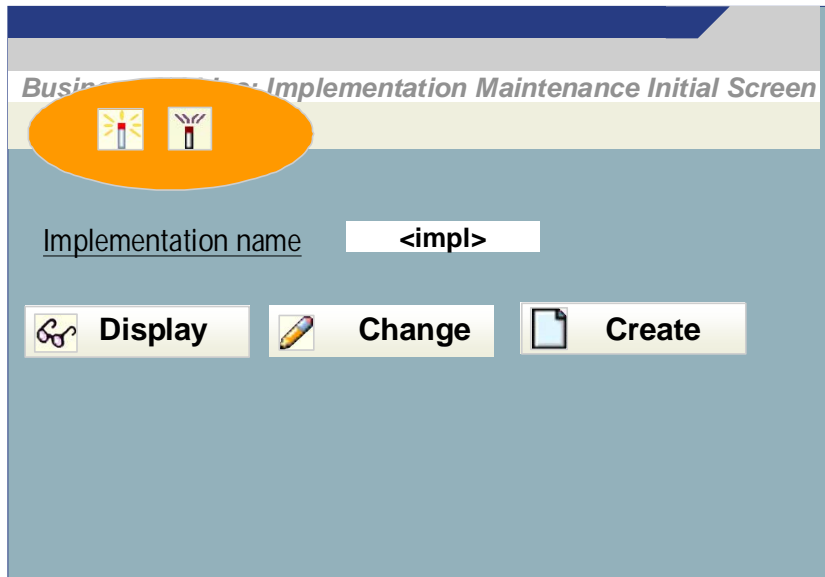
Attributes Methods Events

Parameters Exceptions

| Method        | Type | Description                  |
|---------------|------|------------------------------|
| <priv_method> |      | new method in implementation |

© SAP AG 2002

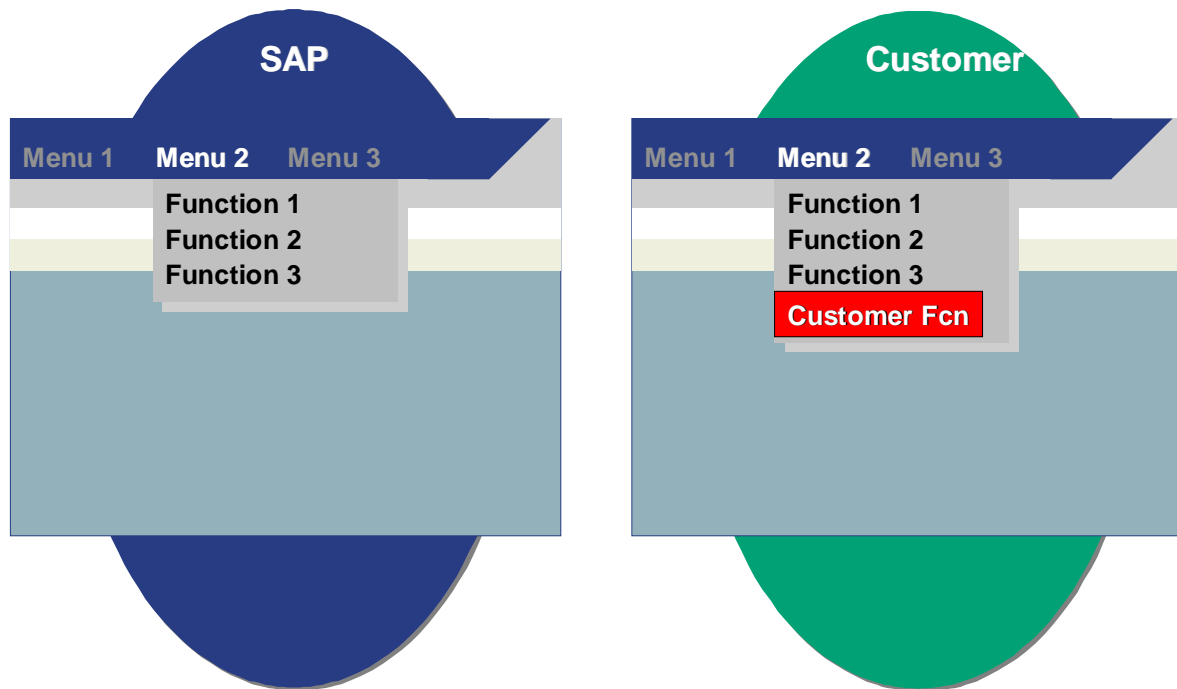
- In the implementing class, you can create private methods that you then call from the interface method.
- To do this you must edit the implementing classes directly in the Class Builder. You create the private methods including interfaces. Specify a visibility level for the method, and implement it.



- Activating
- Deactivating

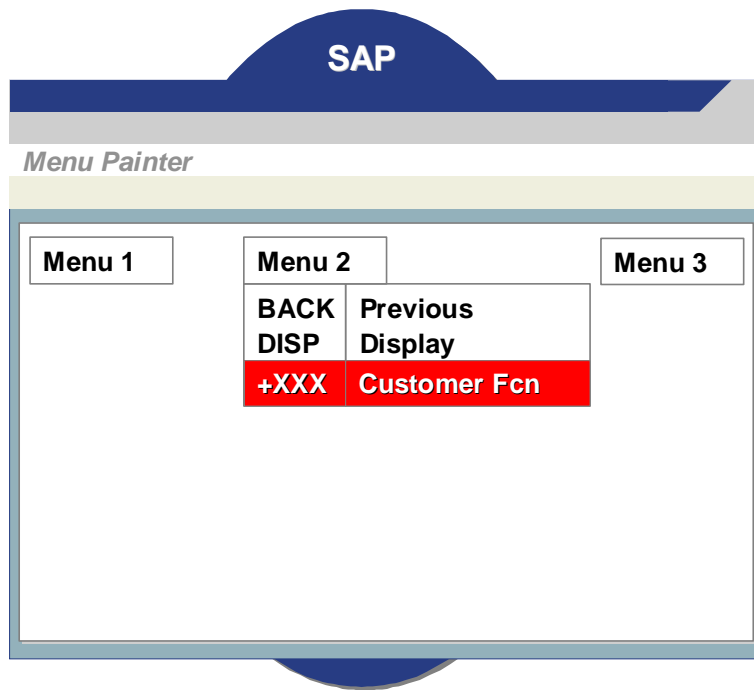
© SAP AG 2006

- Use the "Activate" icon to activate the implementation of a Business Add-In. From now on, the methods of the implementation will be executed when the relevant calling program is executed.
- If you deactivate the implementation, the methods will no longer be called. However, the corresponding calls in the application program are still processed. The difference is that the instance of the adapter class will no longer find any active implementations. Unlike "CALL CUSTOMER-FUNCTION", the "CALL METHOD CL\_EXITHANDLER=>GET\_INSTANCE" call is still executed even if there are no implementations. The same applies to the method call that calls the method of the adapter class.
- You can only activate or deactivate an implementation in its original system without modification. The activation or deactivation must be transported into subsequent systems.
- If a Business Add-In can only have one implementation, there can still be more than one implementation in the same system. However, only one can be active at any time.



© SAP AG 2006

- As with customer exits, you can use menu enhancements with Business Add-Ins. However, the following conditions must be met:
  - The developer of the program you want to enhance must have planned for the enhancement.
  - The menu enhancement must be implemented in a BAdI implementation.



© SAP AG 2006

- Function codes of menu enhancements begin with a plus sign '+'. The menu entry will only appear if there is an active BAdI implementation containing the corresponding enhancement.

```
PROGRAM <sap_program>.
DATA ok_code LIKE sy-ucomm.
DATA r_var TYPE REF TO <badi-interface>.
...

CASE ok_code.
 WHEN 'DISP'.
 ...

 WHEN '+XXX'.
 CALL METHOD r_var-><meth_abc>
 EXPORTING
 <i_variables>
 IMPORTING
 <e_variables>.

 ...

ENDCASE.
```

© SAP AG 2006

- If the user chooses the menu entry in the program to which the function code "+<exit>" is assigned, the system processes the relevant method call.
- The method call and the menu enhancement belong inseparably to one another. Having the former without the latter would make no sense. For this reason, it is important that the two enhancement components are combined in a single enhancement - the business add-in.

The screenshot shows the SAP Business Add-Ins configuration interface. The 'FCodes' tab is selected. A table with three columns: 'Program', 'Function code', and 'Description' is visible. The 'Function code' column contains the value '+xxx', which is highlighted with an orange oval. Below the table, there is an 'Attributes' section with two checkboxes: 'Reusable' and 'Filter-dependent', both of which are unchecked.

| Program        | Function code | Description |
|----------------|---------------|-------------|
| /namespace/... | +xxx          |             |

**Attributes**

☐ Reusable

☐ Filter-dependent

© SAP AG 2006

- You can only create function codes for single use Business Add-Ins Moreover, the Business Add-In must not be filter-dependent.
- These restrictions are necessary to ensure that there are no conflicts between two or more implementations. ("Which menu entry should be displayed?")



**At the conclusion of this topic, you will be able to:**

- **Create Business Add-Ins that will serve as enhancements for later developments in your programs**

© SAP AG 2006



### Business Add-Ins: Definition Maintenance Initial Screen

Definition name\_\_



**Display**



**Change**



**Create**

© SAP AG 2006

- To create a Business Add-In, use the BAdI Builder (*Tools -> ABAP Workbench -> Utilities -> Business Add-Ins -> Definition*) or transaction SE18.
- Prior to *SAP NetWeaver Application Server 7.0*, the initial screen of transaction SE18 appears as displayed above.
- As of *SAP NetWeaver Application Server 7.0*, there also exist new BAdIs, which are contained in so-called Enhancement Spots (see the attachment). To create an older (classical) BAdI from the new initial screen of Transaction SE18, you need to choose the menu option "Utilities→ Create classical BAdI".

The screenshot shows a dialog box with three tabs: 'Attributes', 'Interface', and 'FCodes'. The 'Attributes' tab is active. It contains two checkboxes: 'Reusable' (unchecked) and 'Filter-dependent' (checked). To the right of the 'Filter-dependent' checkbox is a text field labeled 'Filter type' containing the placeholder text '<filter\_type>'.

© SAP AG 2006

- A business add-in has two important attributes that you must define:
  - Reusable
  - Filter-dependent
- If you select the "Reusable" checkbox, then many implementations can exist for the Business Add-In. The sequence in which the implementations will be processed is not defined. Even if the business add-in does not support multiple use, you can still have more than one implementation for it. However, only one implementation can be **active** at a time.
- If you make a business add-in filter-dependent, you can make calls to it depending on certain conditions. You must specify the filter type in the form of a data element or a structure present in the ABAP Dictionary. The value table of the domain used by the data element contains the valid values for the implementation. If you use a structure as the filter type, this is also true for the single fields of the structure.
- When the enhancement method is called, a filter value must be passed to the interface.

## BAdI Definition: Restrictions for Menu Exits

SAP

The screenshot shows the SAP BAdI Definition interface. The 'FCodes' tab is selected, displaying a table with columns: Program, Function code, and Description. The 'Function code' column contains the value '+xxx', which is highlighted with an orange oval. Below the table, the 'Attributes' dialog is open, showing two checkboxes: 'Reusable' and 'Filter-dependent', both of which are unchecked.

| Program        | Function code | Description |
|----------------|---------------|-------------|
| /namespace/... | +xxx          |             |

**Attributes**

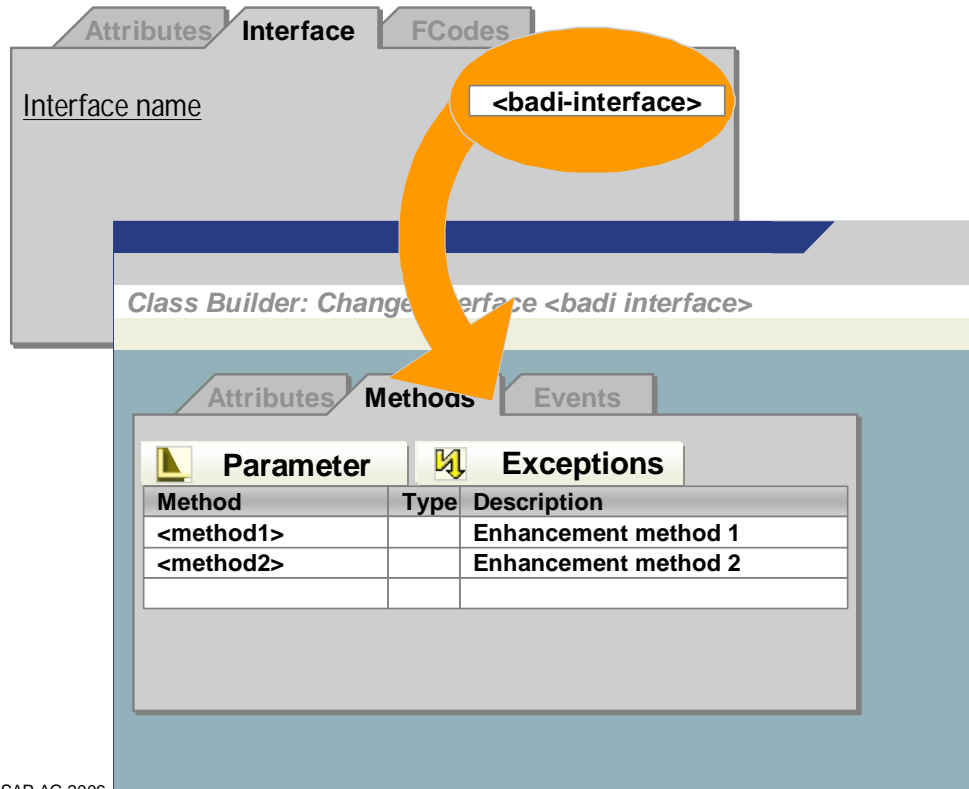
- ☐ Reusable
- ☐ Filter-dependent

© SAP AG 2006

- You can include function codes in a Business Add-In definition (like menu exits in customer exits). To do this, enter the program name and function code, and a short description on the relevant tab page.
- Restrictions:
  - It is not currently possible to create BAdIs that consist **only** of menu enhancements (function codes).
  - If you use menu enhancements, you cannot reuse a BAdI or make it filter-dependent.

## BAdI Definition: Defining Interface Methods

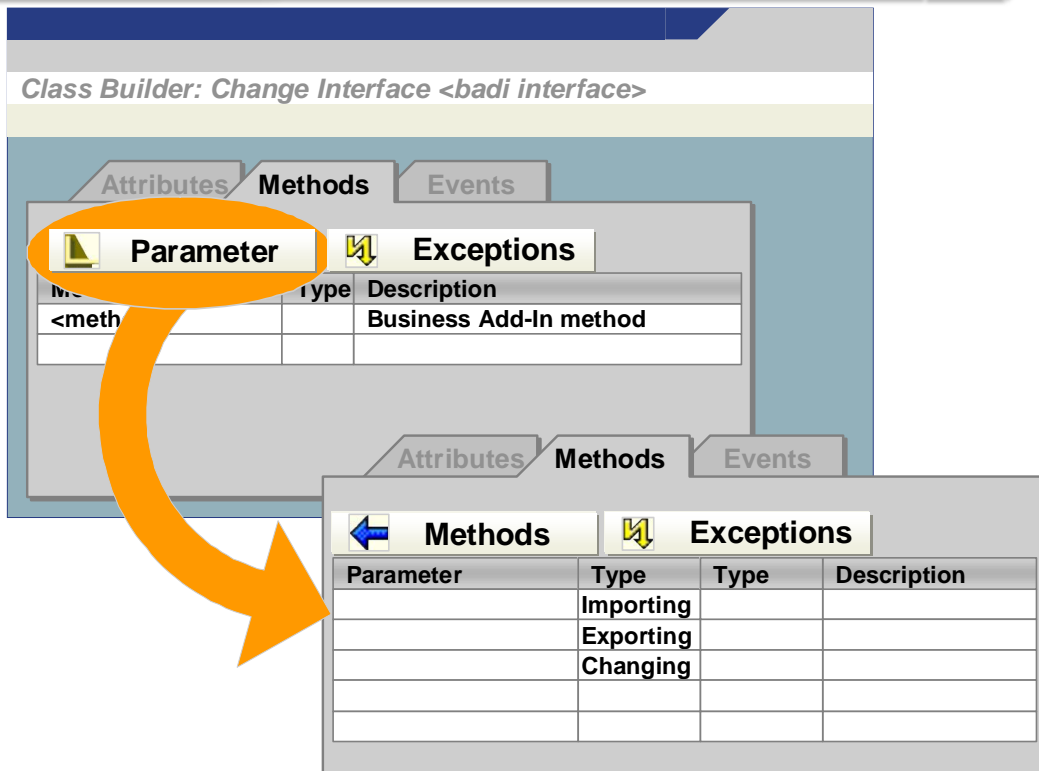
SAP



- The system proposes a name for the interface and the generated class. You can, in principle, change the name of the interface to anything you like. However, your BAdI will be easier to understand if you retain the proposed name.
- The name of the generated class is composed as follows:
  - Namespace prefix
  - CL\_ (to signify a class in general)
  - EX\_ (stands for "exit")
  - Business Add-In name (without namespace prefix)
- If you double-click on the interface name, the system switches to the Class Builder, where you can define the interface methods.
- A Business Add-In interface can have several interface methods.

## BAdI Definition: Method Interface Parameters

SAP

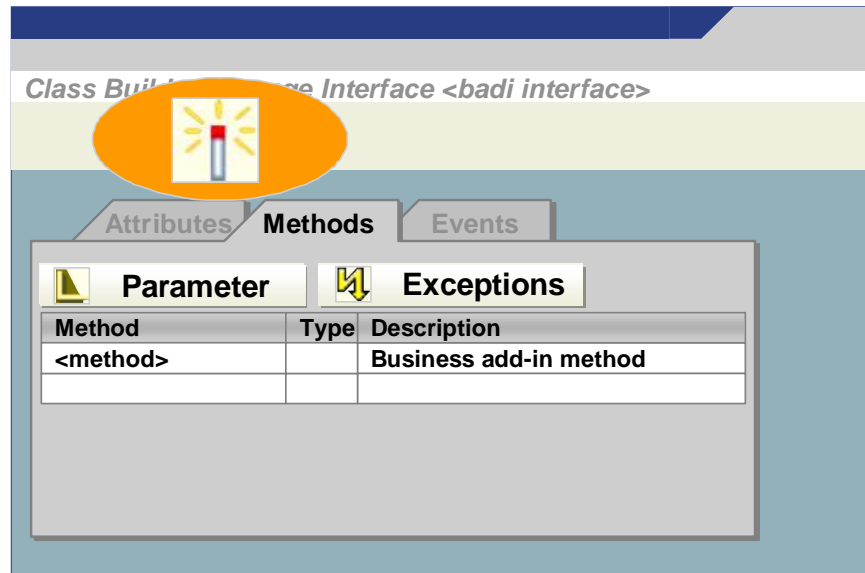


© SAP AG 2006

- You can use all of the normal functions of the Class Builder. For example, you can:
  - Define interface methods
  - Define interface parameters for the methods
  - Declare the attributes of the interface
- If the Business Add-In is filter-dependent, you must define an import parameter **flt\_val** for each method. Otherwise, you define the interface parameters that you need for the enhancement.

## BAdI Definition: Activating the BAdI Interface

SAP



© SAP AG 2006

- Once you have finished working on your interface, you must activate it. This generates the BAdI class for the Business Add-In.
- If you change the interface, the BAdI class is automatically regenerated.
- You can also generate the BAdI class explicitly at any time by choosing *Utilities --> Regenerate* from the initial screen of the Business Add-In maintenance transaction.

## BAdI Definition: Calls in the Program

SAP

```
REPORT <sap_program>.
```

```
DATA r_var TYPE REF TO <badi-interface>. 1
```

```
...
```

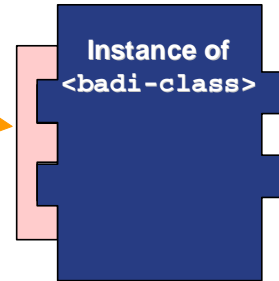
```
CALL METHOD cl_exithandler=>get_instance
CHANGING
 instance = r_var. 2
```

r\_var

```
...
```

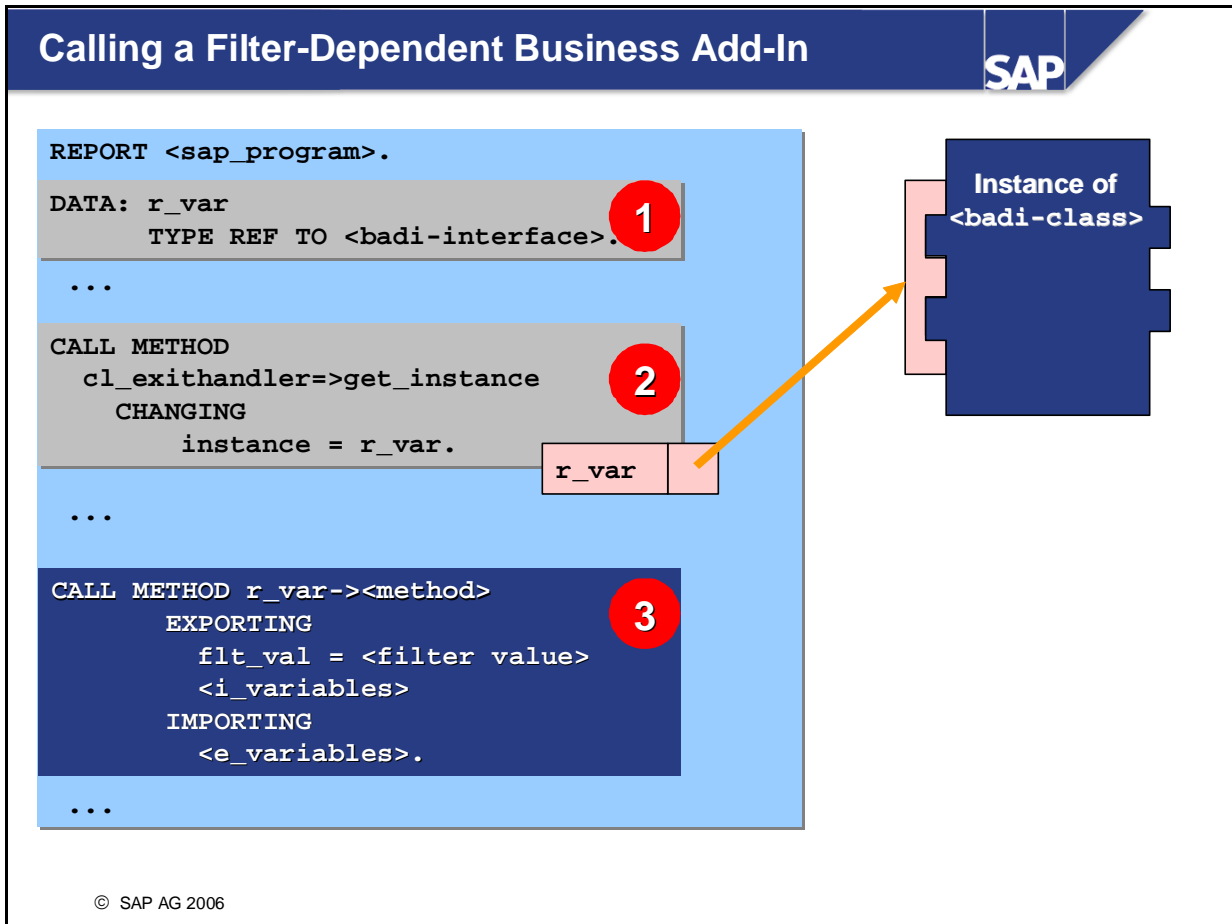
```
CALL METHOD r_var-><meth_abc>
EXPORTING
 <i_variables>
IMPORTING
 <e_variables>. 3
```

```
...
```



© SAP AG 2006

- To call Business Add-In methods in an application program, you must include three statements in the program:
- Declare a reference variable (1) with reference to the Business Add-In interface (in our example, "r\_var").
- Call the static method GET\_INSTANCE of the service class CL\_EXITHANDLER (2). This returns an instance of the required object. This involves an implicit down cast, so that only the interface methods of the object with the reference variable "r\_var" can be called.
- You can now call all of the methods of the Business Add-In (3). Ensure you specify the method interfaces correctly.



- If your Business Add-In is filter-dependent, you must pass an appropriate value to the parameter **flt\_val**.



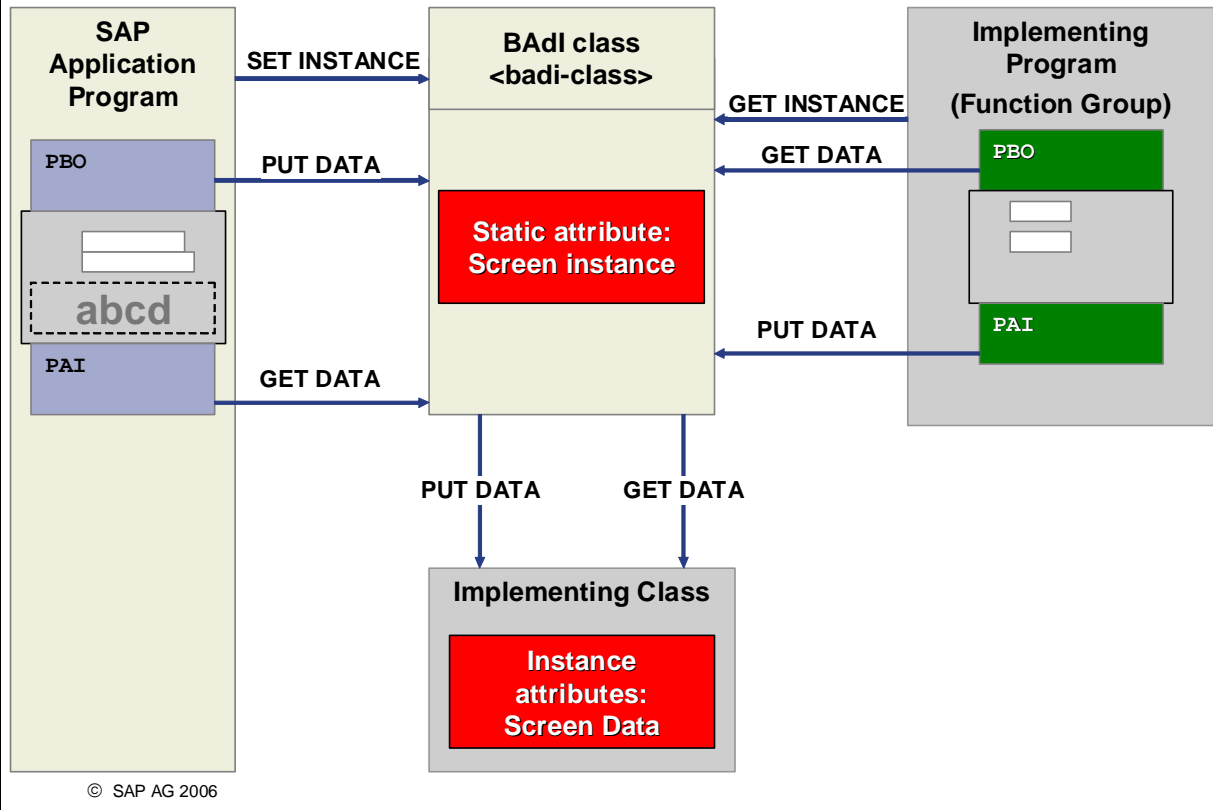


**At the conclusion of this topic, you will be able to**

- **Create a program that provides a BAdI screen enhancement (screen exit)**

## BAdI Screen Exits: Basic Principles

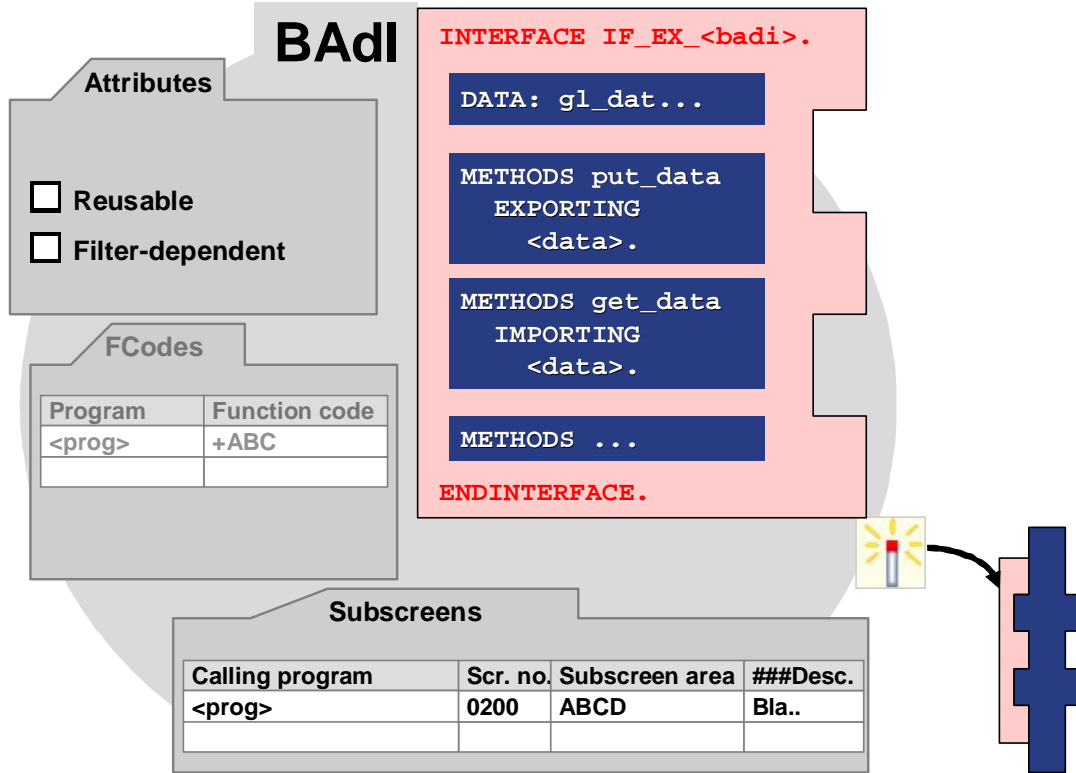
SAP



- The ABAP virtual machine does not recognize screens bound to classes. Thus, only "classical" programs (of types 1, F, or M) can be used as containers for screens. Screen enhancements must take this into consideration.
- When you create a Business Add-In screen enhancement, the provider reserves a subscreen area on the application program screen, which is then filled with the subscreen of the implementing class (similar to customer exits). However, the application program and the subscreen container program do not communicate directly. They communicate using the generated BAdI class.
- The following slides show this communication process step by step.

## BAdI Screen Exits: Components

SAP



© SAP AG 2009

- If a BAdI contains one or more screen enhancements, it cannot be flagged as reusable. If it also contains menu enhancements, then it also may not be filter-dependent.
- In addition to the interface definition you enter the calling program, screen number, and subscreen area on the Subscreens tab. The name of the implementing program and number of the subscreen screen are specified later by the implementing developer.

- **In the PBO:**
  - **Generate Business Add-In class instance**
  - **Publish the instance, so that implementing classes can access it**
  - **Specify the program name and screen number of the implementation**
  - **Make available the data for the subscreen screen**
  - **Integrate the subscreen into the subscreen area**

© SAP AG 2006

- To use a BAdI to provide a screen enhancement, you must implement the steps listed above in the application screen's PBO.

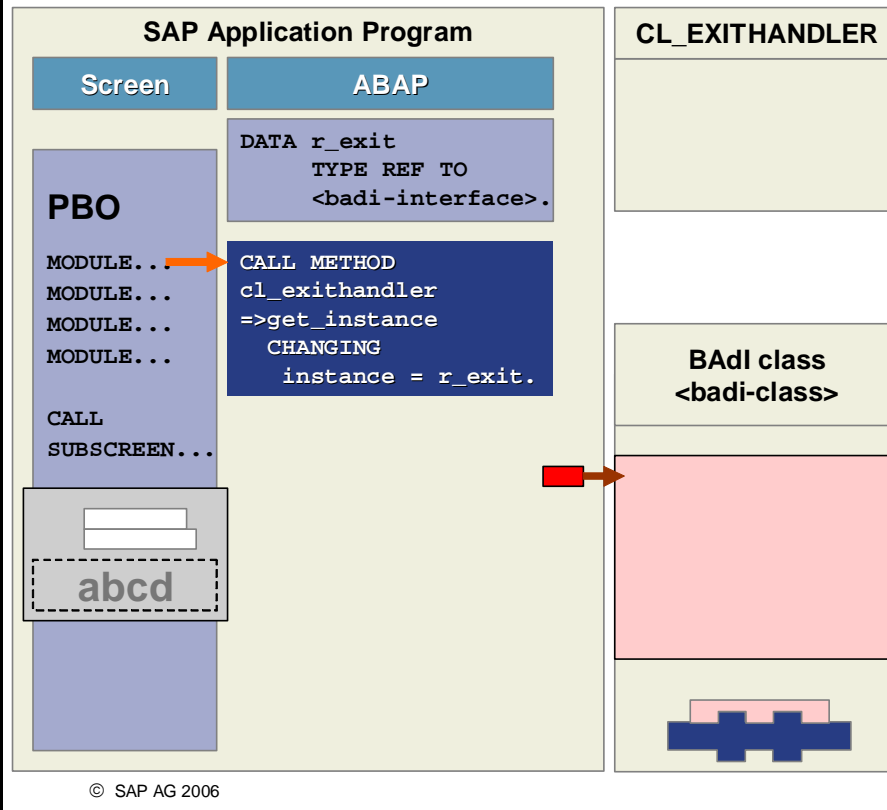
- **In the PAI:**
  - **Call the subscreen**
  - **Take user entries from the subscreen**

© SAP AG 2006

- In the subscreen container screen's PAI, the subscreen must be called again and - depending on the enhancement design - the data changed by the user on the subscreen must be loaded into the application program.

## PBO Steps (1): Create BAdI Instance

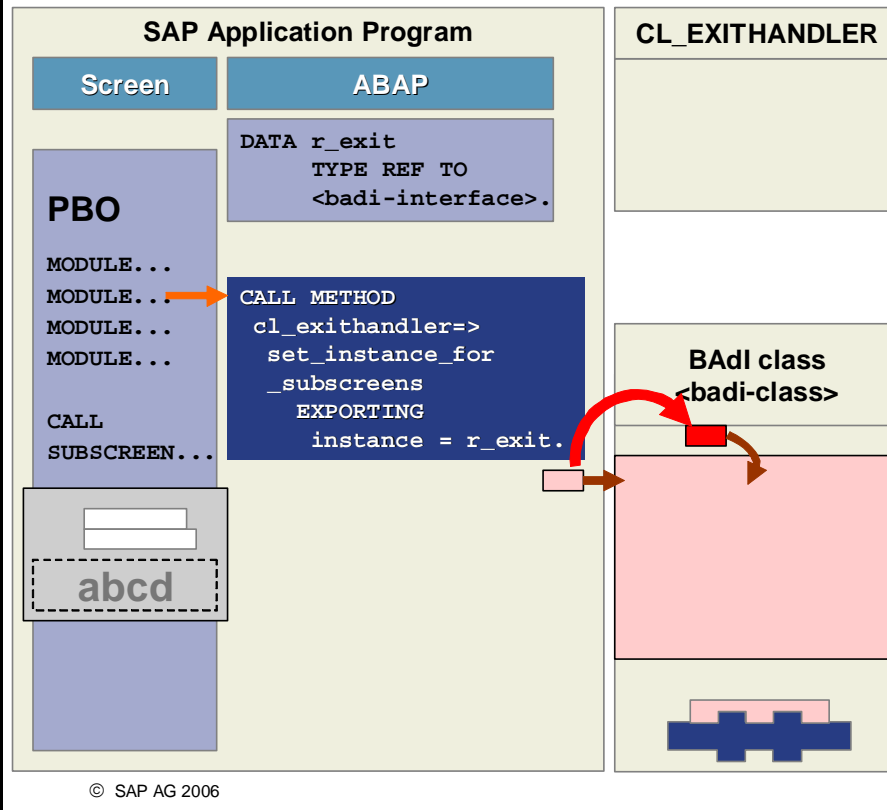
SAP



- In the first step, an instance of the Add-In class is generated, just as in functional enhancements. You do this by calling the factory method ""get\_instance"" from the class CL\_EXITHANDLER.

## PBO Steps (2): Publish Instance

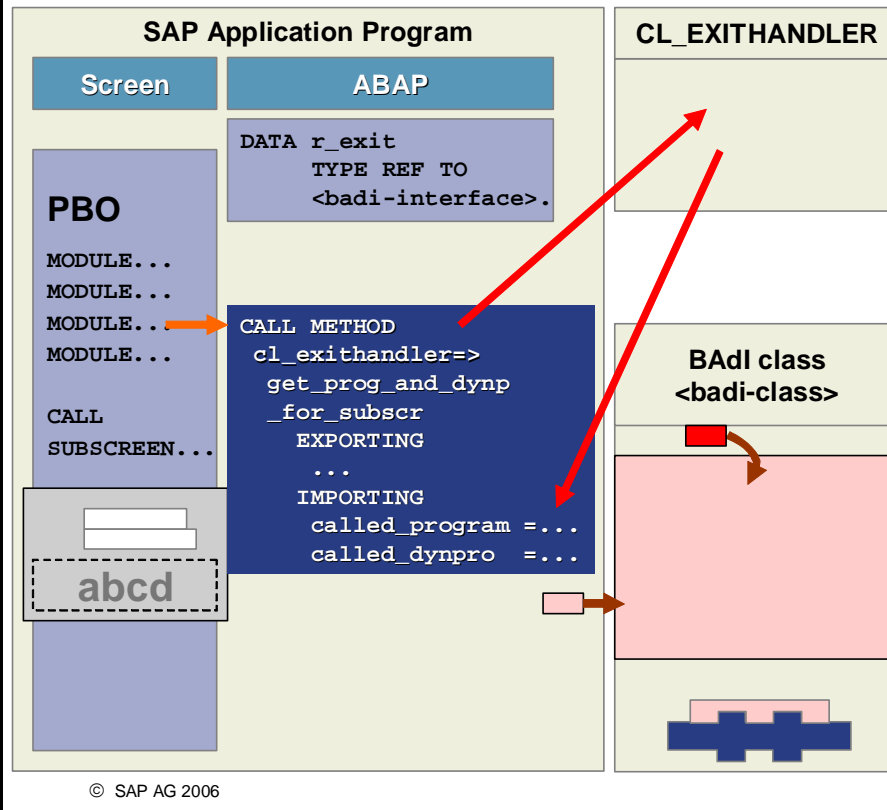
SAP



- Later, the implementation will have to access the BAdI class instance. Therefore, it is necessary to "publish" the instance. To do this, call the static method `SET_INSTANCE_FOR_SUBSCREENS` of the standard class `CL_EXITHANDLER`.

## PBO Steps (3): Provide Subscreen Number

SAP

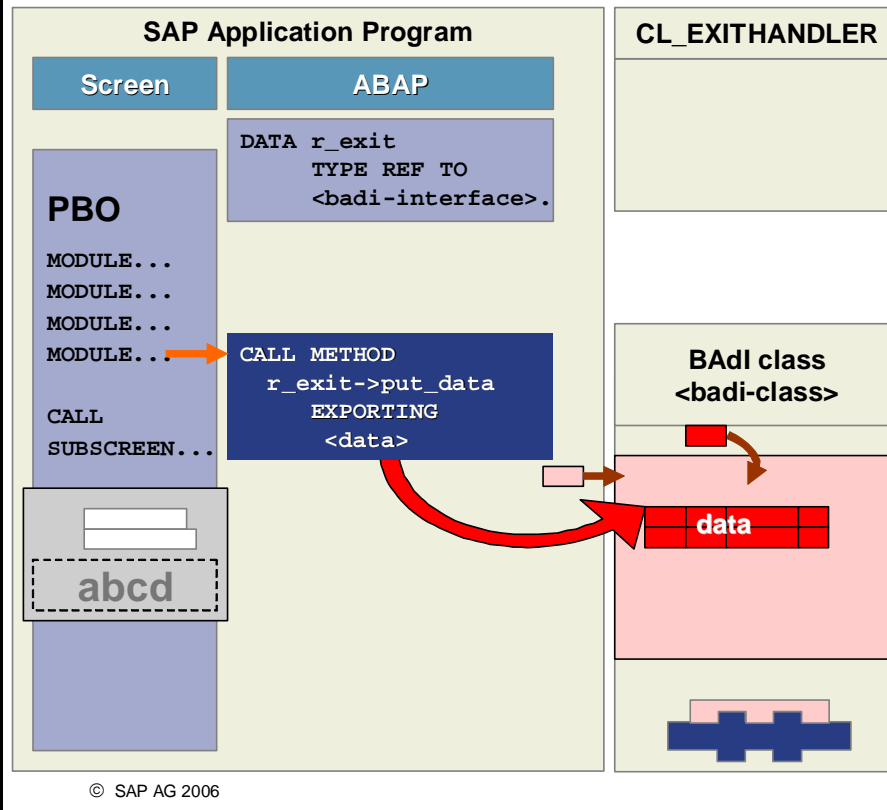


- You must determine the subscreen screen number and the corresponding program for the implementation for the statement CALL SUBSCREEN in the container screen's PBO. To do this, call the method  
CALL METHOD cl\_exithandler=>get\_prog\_and\_dynp\_for\_subscr
- If there is no active implementation yet, the method returns an empty dummy subscreen screen (0200) from an existing dummy function group (SAPLSEX).
- If, however, there is an active implementation for the BAdI definition you want to use, the subscreen screen specified in the implementation is used.



## PBO Steps (4): Provide Data

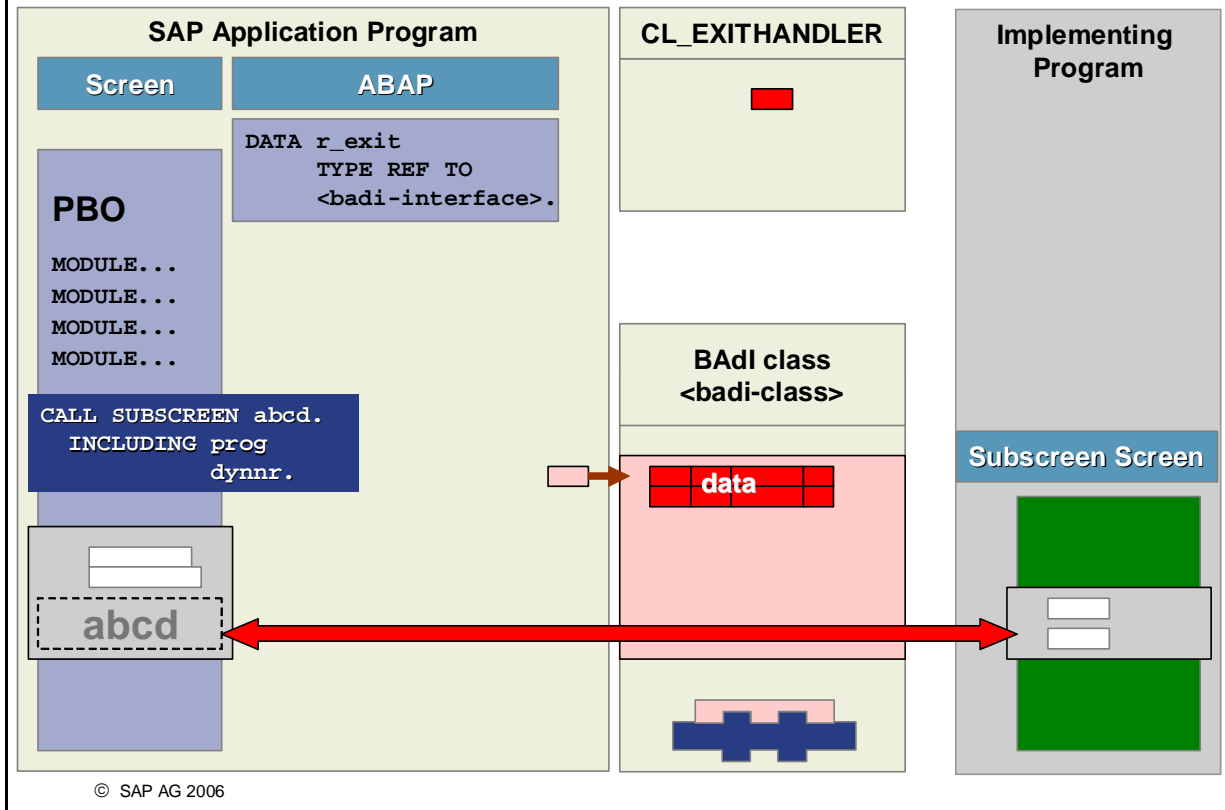
SAP



- To make data to the implementation, you need to pass it in two steps:
- Pass the data to the Business Add-In class. This passed data is stored in global attributes in the implementation of this method. We also strongly advise you to provide sample code at this point. The data is passed by calling the method defined in the BAdI.
- The data stored in the Add-In class in global attributes is automatically passed to the global attributes of the implementing class, if there is an active implementation.

## PBO Steps (5): Integrate Subscreen

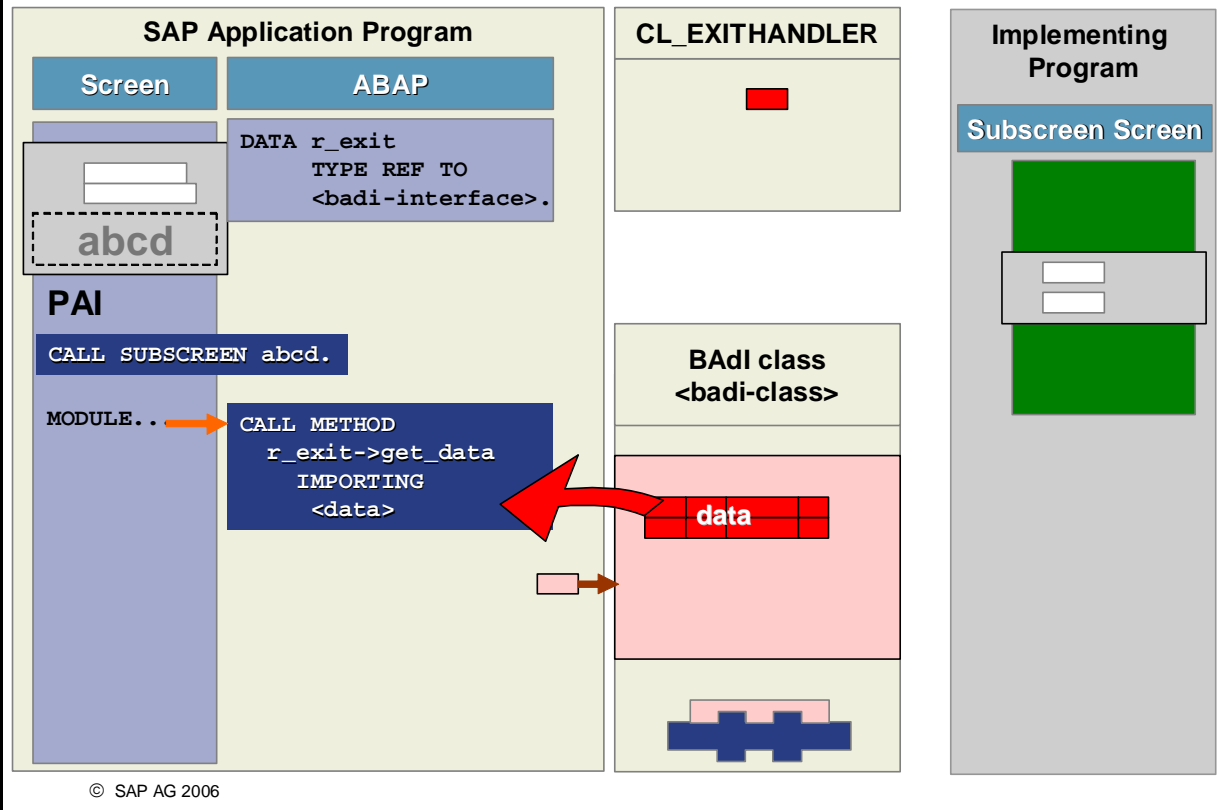
SAP



- Finally, the application program calls the subscreen screen.

## PAI Steps: Call Subscreen and Load Data

SAP



- If the entries made by the user on the subscreen need to be loaded into the application program for further processing, you must call the corresponding method (here, it is GET\_DATA).



**At the conclusion of this topic, you will be able to**

- **Implement BAdI screen enhancements (screen exits)**

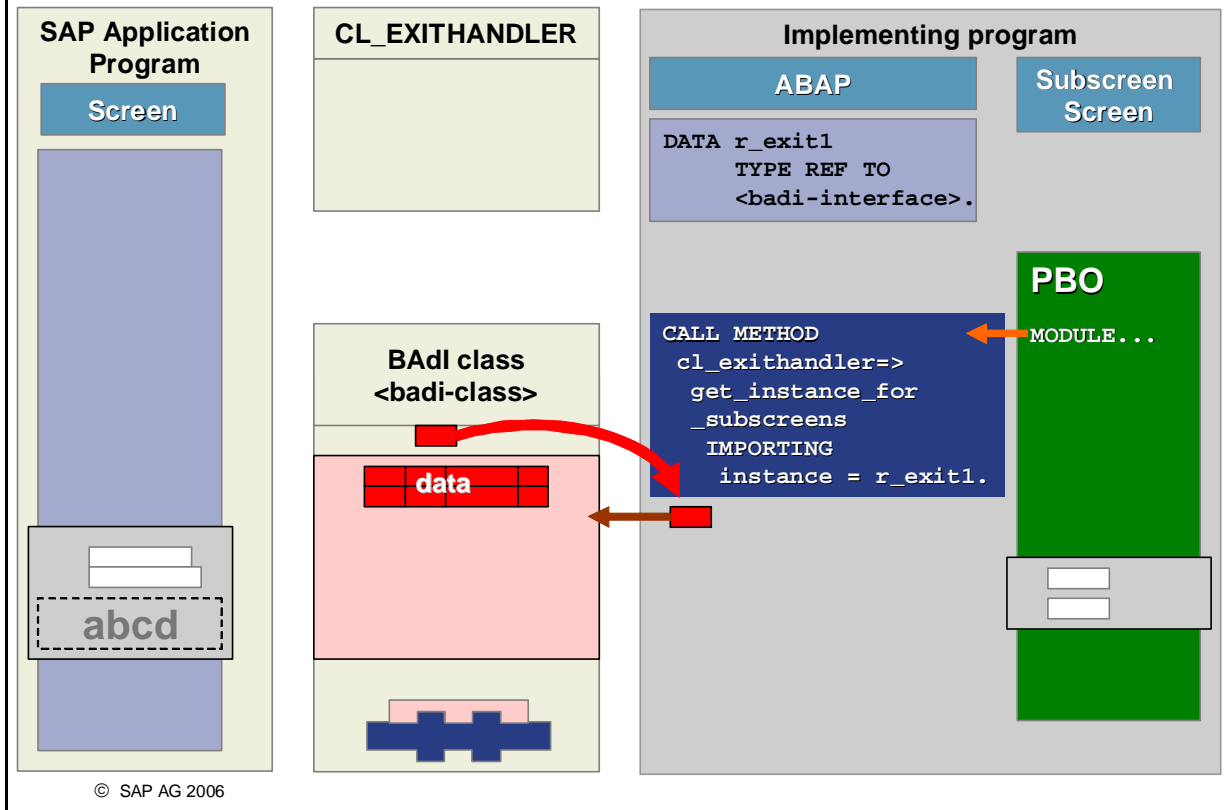
- **Create the program (function group)**
- **Create the subscreen screen**
  - **PBO:**
    - **Generate BAdI class instance**
    - **Provide subscreen screen data**
  - **PAI:**
    - **Load data changed on the subscreen**
- **Create BAdI implementation**

© SAP AG 2006

- To implement a Add-In screen enhancement:
  - Create the implementation for the Add-In definition. Specify the program containing the subscreen screen and the subscreen number.
  - Create the program specified in the BAdI implementation.
  - Create and lay out the subscreen screen.
- If you want data that is changed on the subscreen to be returned to the application program, you must call the corresponding method in the subscreen screen's PAI.

## PBO Steps (1): Provide BAdI Instance

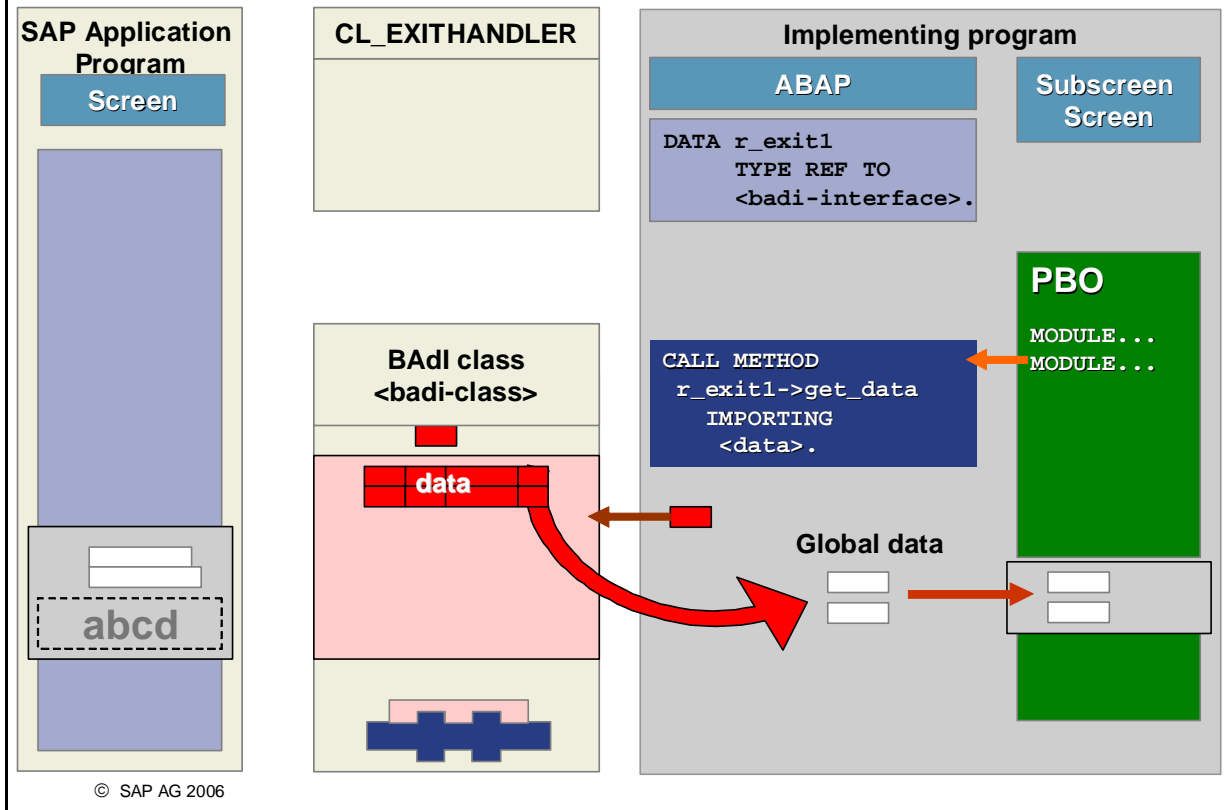
SAP



- To get the reference to the instance of the Add-In class, there is a method `GET_INSTANCE_FOR_SUBSCREENS`, in the class `CL_EXITHANDLER`.

## PBO Steps (2): Provide Subscreen Data

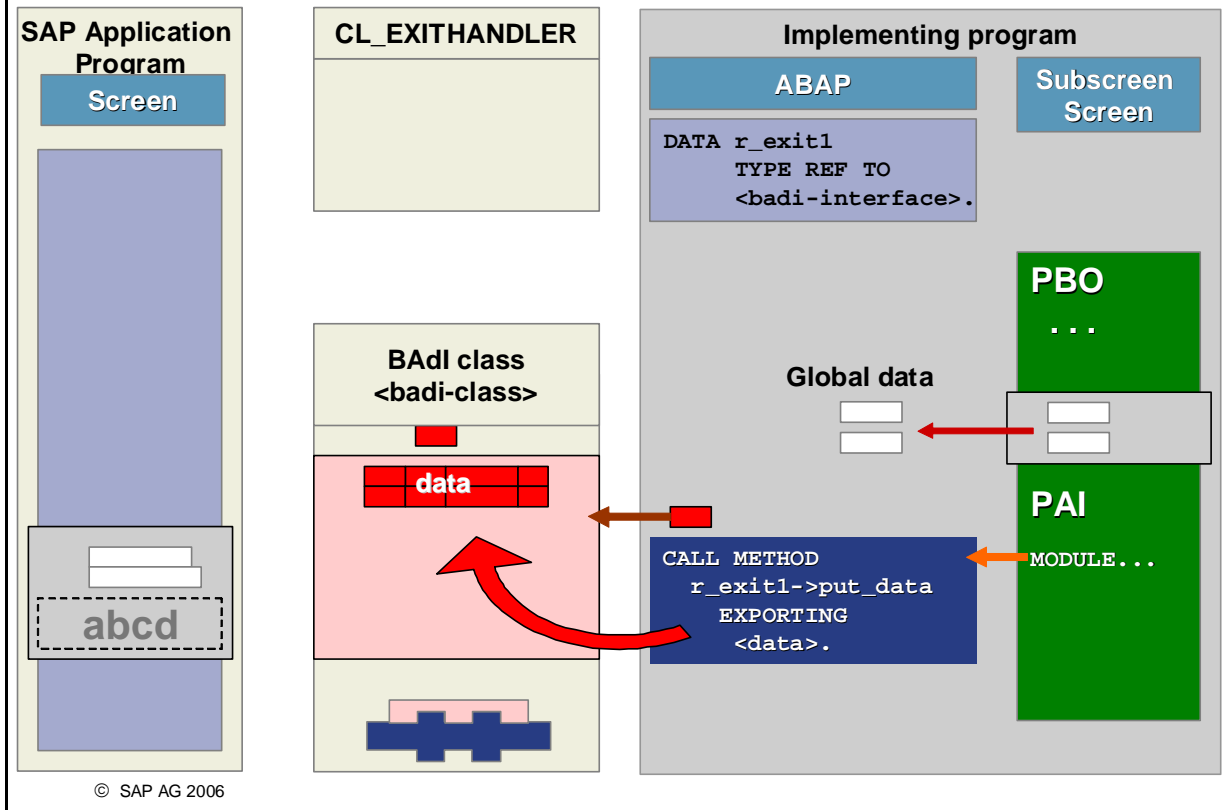
SAP



- The implementing program obtains the data to be displayed on the subscreen by calling the corresponding method (here, that is GET\_DATA). The data is then stored in the implementing program's global variables, which fill corresponding subscreen fields at the end of the PBO.

## PAI Step: Returning Data

SAP



- If you want the data on the subscreen to be changeable, then you should ensure that the changed data are returned to the application program so that they can be used for further processing. You do this by calling the appropriate Add-In method (here, that is PUT\_DATA).





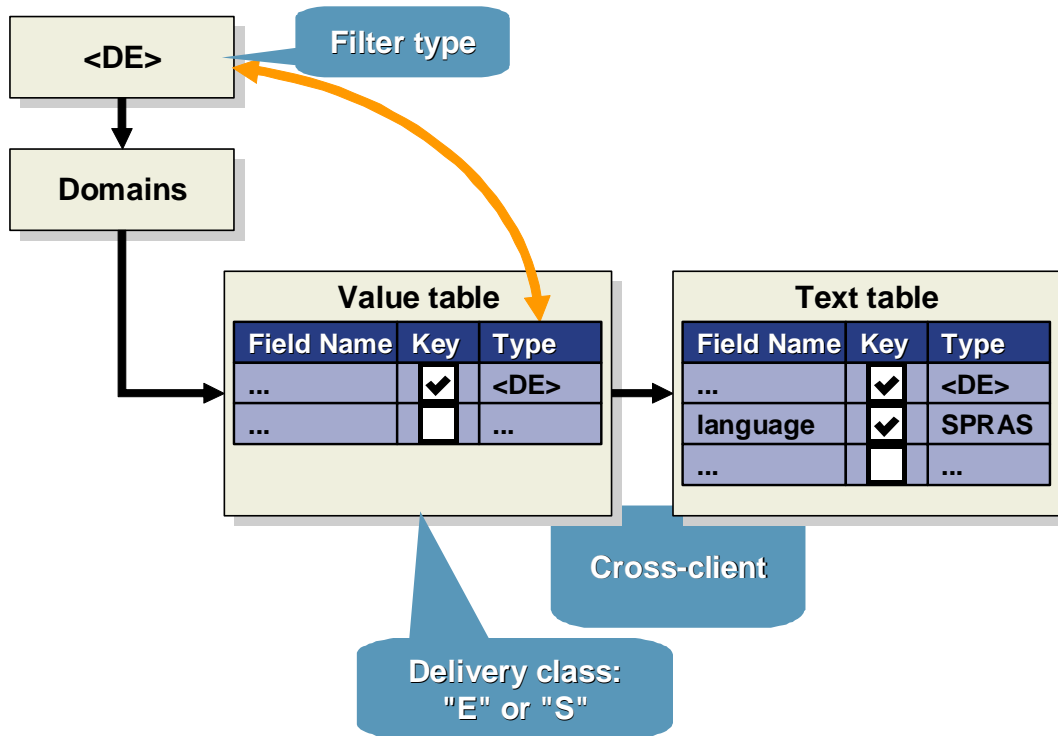
**At the conclusion of this topic, you will be able to:**

- **Explain what an extensible filter type is**
- **Explain the default and sample code**

© SAP AG 2006

## Extendible Filter Types: Prerequisites

SAP

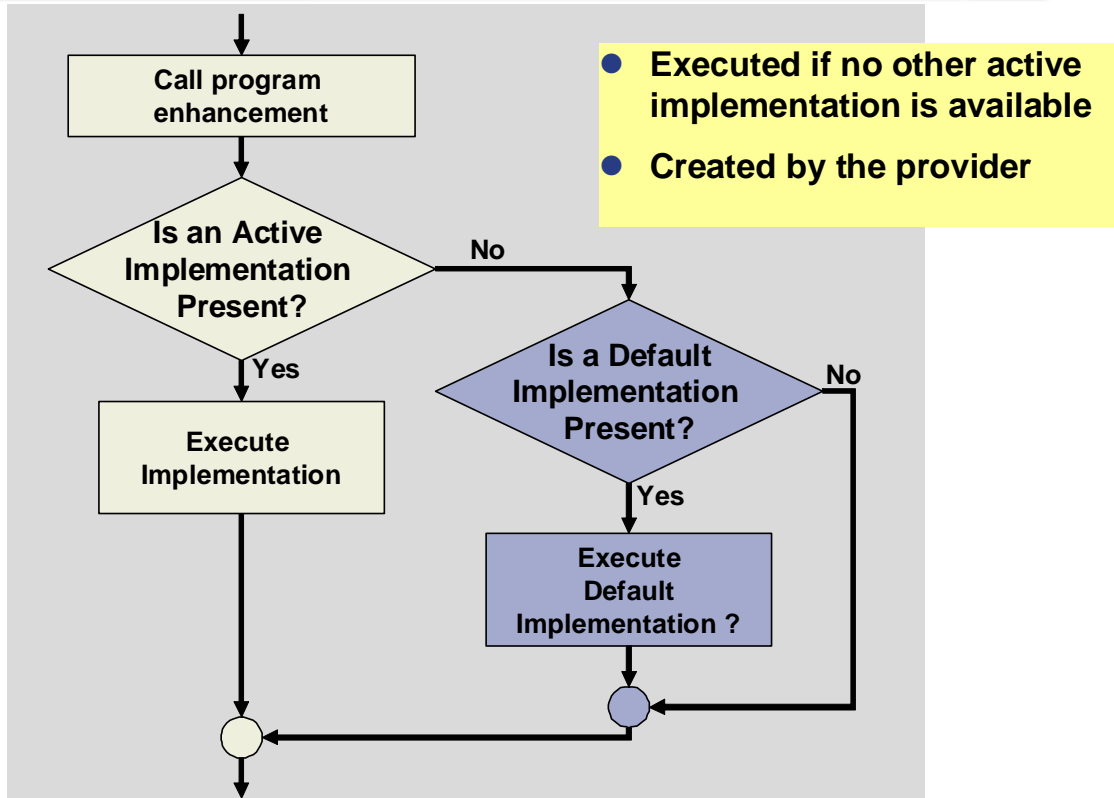


© SAP AG 2002

- The assignment of the **extendible** attribute is subject to the following restrictions:
- The domain to which the extendible filter type refers must have the following properties:
  - The domain is linked to a **cross-client** value table. The value table has exactly one key field which has the data element of the filter type as its field type.
  - The domain has a text table with two key fields. A key field has the filter type as its field type, and a key field is a language field. To mark a field as a text field, the field must exist in this table that contains the string **TEXT** or **TXT** as a partial string. In the ABAP Dictionary, the text table must be assigned to the value table.
  - The delivery class of both tables must be **E** or **S**.
- All filter values that are created in the context of an extendible filter-dependent Business Add-In must not yet occur in the value field and are added to the value table when the data is saved. Correspondingly, the values are removed from the value table when the implementation or the entire Business Add-In is deleted. The same applies to text tables.

## Default Implementation

SAP



- A *default implementation* is executed whenever no active implementation of a Business Add-In exists. The default implementation is created by the enhancement provider.
- To create a default implementation in the BAdI definition choose *Goto --> Default code*. The system automatically generates a class with a predefined name. Implement the methods so that the required default behavior is generated.
- You can also create a *sample implementation*. This is a template that can be inserted into the methods of the implementation.
- To create sample implementations choose *Goto --> Sample Code*. The system creates a class that implements the methods of the interface. The sample code is displayed for the user as a template.

| Comparison With Other Enhancement Techniques |                |                             |                  |
|----------------------------------------------|----------------|-----------------------------|------------------|
|                                              | Customer Exits | Business Transaction Events | Business Add-Ins |
| Program Exit                                 | +              | +                           | +                |
| Menu Exit                                    | +              | —                           | +                |
| Screen Exit                                  | +              | —                           | +                |
| Append Fields On Screens                     | +              | —                           | (+)              |
| Administration Level                         | +              | —                           | +                |
| Reusable                                     | —              | +                           | +                |
| Filter-specific                              | —              | +                           | +                |

© SAP AG 2006

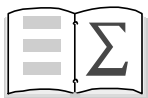
- Business add-ins are a natural further development of the conventional enhancement technique. They have taken over the administration layer from customer exits, along with the availability of the various enhancement components.
- Add-Ins adopted the idea of reusability from business transaction events, and have been implemented using a consistent object-oriented approach.

- **Business Add-In definition**
  - `<badi>` or `Z<badi>` or `/../<badi>`  
(Choose any; comply with namespace)
- **Interface**
  - `IF_EX_<badi>` or `ZIF_EX_<badi>` or `/../IF_EX_<badi>`  
(Choose any; comply with namespace)
- **Methods**
  - Choose any name you want
- **Generated Business Add-In Class (Adapter Class)**
  - `CL_EX_<badi>` or `ZCL_EX_<badi>` or `/../CL_EX_<badi>`  
(Not changeable)

© SAP AG 2009

- **Business Add-In implementation**
  - `<impl>` or `Z<impl>` or `./.<impl>`  
(Choose any; comply with namespace)
- **Interface**
  - `IF_EX_<badi>` or `ZIF_EX_<badi>` or `././IF_EX_<badi>`  
(Specified by the BAdI definition)
- **Methods**
  - Defined in Business Add-In definition
- **Implementing class**
  - `CL_IM_<impl>` or `ZCL_IM_<impl>` or `././CL_IM_<impl>`  
(Choose any; comply with namespace)

© SAP AG 2009



**You are now able to:**

- **Search for Business Add-Ins**
- **Implement Business Add-Ins**
- **(Create Business Add-Ins)**

© SAP AG 2006





# Exercises



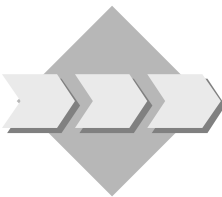
## Unit: Business Add-Ins

### Topic: Using Business Add-Ins



At the conclusion of this exercise, you will be able to:

- Implement an enhancement with Business Add-Ins



The customer service personnel in the agency want the list of bookings that you implemented in the exercise on menu exits to contain more information. The list should contain the name of the customer in addition to his customer number.

- 1-1 Check if program **SAPBC425\_BOOKING\_##** (## = group number) can be enhanced.
  - 1-1-1 Check the program for ways in which it can be enhanced.
  - 1-1-2 Check if an enhancement option is suitable for outputting further information in the list.
- 1-2 Implement the enhancement you found. Call your implementation **ZBC425IM##**.
  - 1-2-1 What data is passed to the interfaces of the methods? Are there already fields here that should be displayed in the list?
  - 1-2-2 Table **SCUSTOM** contains the information about the customers. Get the customer's name from his customer number. Output the name.
- 1-3 Format the list.
  - 1-3-1 How can you move the vertical line so that the additional fields are displayed within the frame?
  - 1-3-2 Is the **CHANGE\_VLINE** method suitable for changing the position of the vertical line? If so, use it.
- 1-4 Check your results.





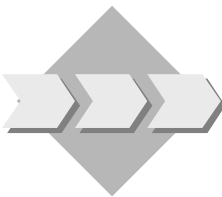
## Unit 7: Business Add-Ins

### Topic: Creating Business Add-Ins



At the conclusion of this exercise, you will be able to:

- Create a Business Add-In and offer an enhancement in a program with Business Add-In technology



Develop your own supplementary components for the R/3 System. You want to offer an enhancement that can implement subsequent software layers in a program.

You deliver a program that outputs a list of flight connections. You want to provide your customers with the following enhancement options using a Business Add In: When the user double-clicks a row, developers at the customer's site should be able to implement other actions. Your customers should be able to build a details list.

In the second part of the exercise, test your enhancement. The details list should show all the flights for a connection.

- 2-1 Create a program that outputs list of flight connections.
  - 2-1-1 To do so, copy program **SAPBC425\_TEMPLATE** to the name **ZBC425\_BADI\_##**.
  - 2-1-2 Assign your program to a package and a change request.
- 2-2 Create a Business Add-In.
  - 2-2-1 The name of the Business Add-In is **ZBC425##**.
  - 2-2-2 Create a method. Define the interface.
  - 2-2-3 Which parameter do you have to pass to the interface?
- 2-3 Edit the program so that a user can double-click on a line to output the details list.
  - 2-3-1 Implement event **AT LINE-SELECTION**.
  - 2-3-2 Insert the appropriate statements in your program to call a Business Add-In: Declare a reference variable; instantiate an object of the Business Add-In class; implement the call of the Business Add-In method at the right place in the program.

- 2-4 Implement the enhancement (name of the implementation: **ZB425##IM**).
  - 2-4-1 A details list should be output when you double-click on a line of the list of the application program. The flight dates of the selected connection should be output in the details list. Table **SFLIGHT##** contains the flight dates.
  - 2-4-2 Read the relevant data from table **SFLIGHT##** to an internal table with Array-Fetch. Then output selected fields of the internal table.
  - 2-4-3 Which variables (attributes of the implementing class) do you have to declare? How do you declare an internal table? Where can you declare a table type?
- 2-5 Check your results.

# Exercises



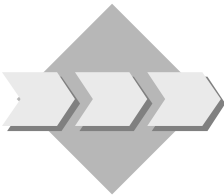
## Unit: Business Add-Ins

### Topic: Implement an Add-In screen enhancement



At the conclusion of this exercise, you will be able to:

- Implement a screen enhancement using a Business Add-In.



Use a transaction of your own to record and change flight data. At present, this transaction offers only standard input fields. Your enterprise wants to record additional data for each flight. The transaction offers an enhancement possibility for this purpose that you will implement.

- 3-1 Find out whether or not the program **SAPBC425\_FLIGHT\_CHNG##** offers an enhancement possibility, which you can use to extend the detail recording screen.
- 3-2 Look for Business Add-Ins with which you can implement the above requirements.
  - 3-2-1 Search the application hierarchy.
  - 3-2-2 Search the Repository Info System.
  - 3-3-3 Search the Implementation Guide.
  - 3-3-4 If you find a Business Add-In, read its documentation and decide whether or not it is suitable for your enhancement.
- 3-3 Implement the enhancement.
  - 3-3-1 Create the function group **ZBC425IM##**. You need this function group to create a subscreen screen.
  - 3-3-2 Create a subscreen screen and assign a number to it.
  - 3-3-3 Add the appropriate fields to the screen.
  - 3-3-4 Program the flow logic for this subscreen screen. At the PBO event, get the instance of the Add-In class and the data that is passed from the SAP application.  
At the PAI event, the changed data must be returned to the SAP application.

- 3-4 Create a Business Add-In implementation. Name your implementation **ZBC425SIN##**.
  - 3-4-1 Implement the interface method.
  - 3-4-2 Enter the necessary information on the *Subscreens* tab.
  - 3-4-3 Activate the implementation.
- 3-5 Test the enhanced application.



## Unit 7: Business Add-Ins

### Topic: Using Business Add-Ins

- 1-1 Check if program **SAPBC425\_BOOKING\_##** (## = group number) can be enhanced as follows:
- 1-1-1 Place the cursor in the list and choose **F1 → Technical Info**. Double-click on the program name (You can also start directly in the ABAP Editor.). Look for the character string **CL\_EXITHANDLER** in the program. Double-click the transfer parameter **exit\_book**. The name of the interface used to define the type of **exit\_book** is **if\_cl\_badi\_book##** and thus the name of the Business Add-In is **BADI\_BOOK##**.
  - 1-1-2 Start transaction **SE18** (Business Add-In definition). Read the documentation about Business Add-Ins.
- 1-2 In transaction **SE18**, choose radiobutton BAdI Name and enter the BAdI. Then go to the transaction for creating implementations of Business Add-Ins using **Enhancement Implementation → Create**. Call your implementation **ZBC425IM##**.
- 1-2-1 You can display the interface parameters by double-clicking the class in transaction **SE18**. In the Class Builder, place the cursor on the required method and choose "Parameters". The transfer structure does not contain the fields that you want to display in the list. You have to read the corresponding data separately.
  - 1-2-2 Double-click on the method name to go to the Editor. A proposal for implementing the methods is given below (group 00):  
METHOD if\_ex\_badi\_book00~output.  
DATA:  
    name TYPE s\_custname.  
SELECT SINGLE name  
    FROM scustom  
    INTO name  
    WHERE id = i\_booking-customid.  
WRITE: name.  
ENDMETHOD.

- 1-3 The **change\_vline** method is provided for formatting the list. You can move the right edge of the list here.
- 1-3-1 Parameter **c\_pos** defines the position of the right vertical line.
- 1-3-2 The method can be implemented as follows:
- ```
METHOD if_ex_badi_book00~change_vline.  
    c_pos = c_pos + 25.  
ENDMETHOD.
```




Unit 7: Business Add-Ins

Topic: Creating Business Add-Ins

- 2-1 Copy the template program as specified in the exercise.
- 2-2 To create Business Add-Ins, start transaction SE18 (in the ABAP Workbench: **Utilities → Enhancements → Business Add-Ins → Definition**).
- 2-2-1 Go to menu **Utilities(M) → Create Classic BAdI**. Choose **ZBC425##** as the name of the Business Add-In. Enter a short description and save your entries.
- 2-2-2 Choose the tab page "Interface". Double-click on the name of the interface. The Class Builder is started. Enter the name of a method. Give a short description. Choose **Parameters** to define the interface.
- 2-2-3 Define two importing parameters whose types are defined with **S_CARR_ID** (airline) and **S_CONN_ID** (connection number). Activate the interface. The adapter class is also generated.

- 2-3 Source code of the program with the Business Add-In:

```
*&-----*
```

```
*& Report SAPBC425_TEMPLATE*
```

```
*&-----*
```

```
REPORT sapbc425_badi.
```

```
DATA:
```

```
    wa_spfli TYPE spfli,
```

```
    it_spfli TYPE TABLE OF spfli WITH KEY carrid connid.
```

```
* Reference Variable for BAdI
```

```
DATA:
```

```
    exit_ref TYPE REF TO zif_ex_bc42500.
```

```
* Selection Screen
```

```
SELECTION-SCREEN BEGIN OF BLOCK carrier
```

```
    WITH FRAME TITLE text-car.
```

```
SELECT-OPTIONS: so_carr FOR wa_spfli-carrid.
```

```
SELECTION-SCREEN END OF BLOCK carrier.
```

```
*&-----*
*&  Event START-OF-SELECTION
*&-----*
START-OF-SELECTION.
```

```
CALL METHOD cl_exithandler=>get_instance
CHANGING
    instance = exit_ref.
```

```
SELECT *
  FROM spfli
  INTO CORRESPONDING FIELDS OF TABLE it_spfli
  WHERE carrid IN so_carr.
*&-----*
*&  Event END-OF-SELECTION
*&-----*
END-OF-SELECTION.
```

```
LOOP AT it_spfli INTO wa_spfli.
```

```
  WRITE: / wa_spfli-carrid,
          wa_spfli-connid,
          wa_spfli-countryfr,
          wa_spfli-cityfrom,
          wa_spfli-countryto,
          wa_spfli-cityto,
          wa_spfli-deptime,
          wa_spfli-arrtime.
```

```
  HIDE:  wa_spfli-carrid,
         wa_spfli-connid.
```

```
ENDLOOP.
```

```
CLEAR wa_spfli.
```

```
*&-----*
*&  Event AT LINE-SELECTION.
*&-----*
AT LINE-SELECTION..
```

CHECK NOT wa_spfli-carrid IS INITIAL.

CALL METHOD exit_ref->lineselection

EXPORTING

i_carrid = wa_spfli-carrid

i_connid = wa_spfli-connid.

clear wa_spfli.

- 2-4 Implement the Business Add-In. From transaction SE18 *choose Implementations* → *Create*. Give the implementation the name ZBC425##_IM. Choose the tab "Interface" and double-click on the name of the method. The Editor is started. Enter the source text here:

METHOD zif_ex_bc42500~lineselection.

DATA:

it_flights TYPE TABLE OF sflight00,

wa_flights TYPE sflight00.

FORMAT COLOR COL_HEADING.

WRITE: / text-hea, i_carrid, i_connid.

FORMAT COLOR COL_NORMAL.

SELECT *

FROM sflight00

INTO CORRESPONDING FIELDS OF TABLE it_flights

WHERE carrid = i_carrid AND

connid = i_connid.

LOOP AT it_flights INTO wa_flights.

/ wa_flights-fldate,

wa_flights-planetype,

wa_flights-price CURRENCY wa_flights-currency,

wa_flights-currency,

wa_flights-seatsmax,

wa_flights-seatsocc.

ENDLOOP.

ENDMETHOD.

Activate the implementation.



Unit: Business Add-Ins
Topic: Screen Enhancements

- 3-1 Find out whether or not the program **SAPBC425_FLIGHT_CHNG##** offers an enhancement possibility, which you can use to extend the entry screen.
- 3-1-1 Choose *System* → *Status* to get the name of the program. Double-click this program name. Choose *Goto* → *Object Catalog Entry*. This tells you the package to which the application is assigned. Double-click the package name to find out the application component containing this package.
- 3-2 In either the Repository Info System or the application hierarchy, search for suitable Business Add-Ins, using this package and application component as criteria.
- 3-2-1 In the Repository, choose *Enhancements* → *Business Add Ins* → *Definition* to search for Business Add-Ins.
- 3-2-2 In the application hierarchy, select the application component. Then choose *Info System* to navigate to the Repository Info System. All the packages contained in the application component you have selected are entered as search criteria.
- 3-2-3 The system returns a list of Business Add-Ins. Select the Business Add-In you are interested in, **BC425_##FLIGHT2**, and choose *Display*. You are now in the *Business Add-In Builder*, where you will find the documentation for this Add-In. Get to know the documentation.
- 3-3 Implement the enhancement.
- 3-3-1 Create the function group **ZBC425IM##** in the Object Navigator. To do this, display the object list for your package. Click the package name with the right mouse button and choose *Create* → *Function Group*. Enter **ZBC425IM##** in the *Name* field and create a short description. Save your entries and assign them to a change request.
- 3-3-2 Create a subscreen in the function group and assign a number to it. Select the function group and choose *Create* → *Screen* from the context menu. Enter a short description and choose the type *Subscreen*.
- 3-3-3 Launch the Layout Editor. Choose *Goto* → *Secondary Window* → *Dict/Program Fields*. Enter the ABAP Dictionary structure and choose *Get from Dict*. Select the fields you want and confirm your choice. Add the fields to the screen.

3-3-4 Navigate to the flow logic for this subscreen screen. This flow logic should include a call to the following module:

```

*-----
PROCESS BEFORE OUTPUT.

MODULE get_instance.
MODULE get_data.
*
*-----
PROCESS AFTER INPUT.

MODULE put_data.

*-----*
***INCLUDE LZBC425_IMO01 .
*-----*
*&-----*
*&  Module get_instance OUTPUT
*&-----*
MODULE get_instance OUTPUT.

CALL METHOD cl_exithandler=>get_instance_for_subscreens
  CHANGING
    instance          = r_var
  EXCEPTIONS
    OTHERS             = 6.
IF sy-subrc <> 0.
  MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
    WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
ENDIF.

ENDMODULE.          " get_instance OUTPUT
*&-----*
*&  Module get_data OUTPUT
*&-----*
MODULE get_data OUTPUT.

CALL METHOD r_var->get_data
  IMPORTING
    e_conn = sdyn_conn.

```

```

ENDMODULE.          " get_data OUTPUT
*-----*
***INCLUDE LZBC425_IMI01 .
*-----*
*&-----*
*&   Module put_data INPUT
*&-----*
*   text
*-----*

MODULE put_data INPUT.

CALL METHOD r_var->put_data
EXPORTING
  i_conn = sdyn_conn.

ENDMODULE.          " put_data INPUT

*-----*
***INCLUDE LZBC425_IMTOP
*-----*

FUNCTION-POOL kaura_im_bc425.      "MESSAGE-ID ..

TABLES:
  sdyn_conn.

DATA:
  r_var TYPE REF TO if_ex_bc425_##flight2.

```

- 3-4 Create a Business Add-In implementation. From the definition, choose **Implementation → Create**. Give the implementation the name **ZBC425SIN##** and confirm your entries.
- 3-4-1 To navigate to the ABAP Editor, choose the *Interface* tab and double-click the name of the method.

```

METHOD if_ex_bc425_##flight2~get_data .
  MOVE-CORRESPONDING wa TO e_conn.
ENDMETHOD.          "IF_EX_BC425_##FLIGHT2~GET_DATA

METHOD if_ex_bc425_##flight2~put_data .
  MOVE-CORRESPONDING i_conn TO wa.
ENDMETHOD.          " IF_EX_BC425_##FLIGHT2~PUT_DATA

```

- 3-4-2 On the *Subscreens* tab, enter the name of the program from your function group, **SAPLZBC425IM##**, and the number of the subscreen screen you created.
- 3-4-3 Activate the implementation by choosing either ***Implementation*** → ***Activate*** or the appropriate button.

- 3-5 Execute the application and check that your enhancement is processed.

Contents:

- What are modifications?
- Implementing modifications
- Modification Assistant
- Modification Browser
- User exits
- Modification adjustment

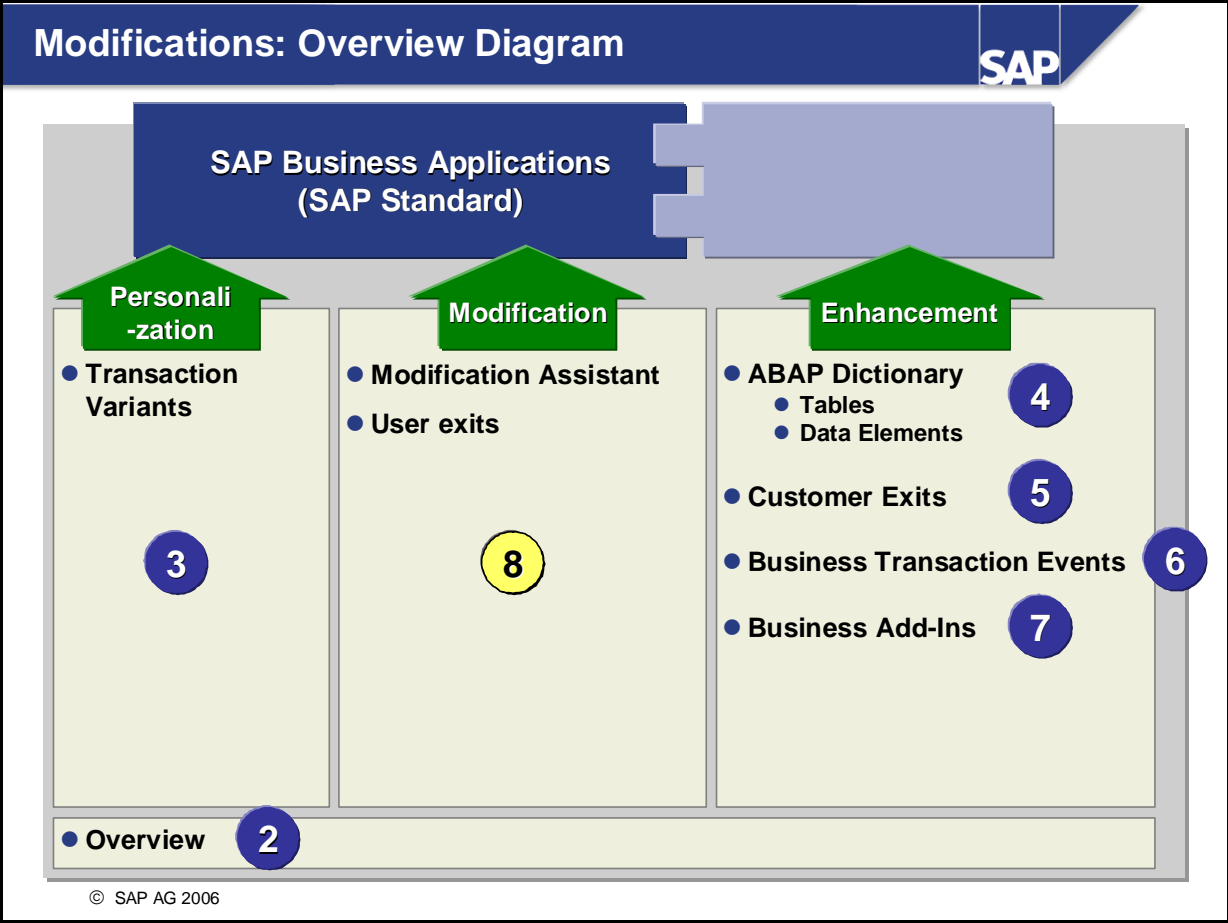
© SAP AG 2008



After completing this unit, you will be able to:

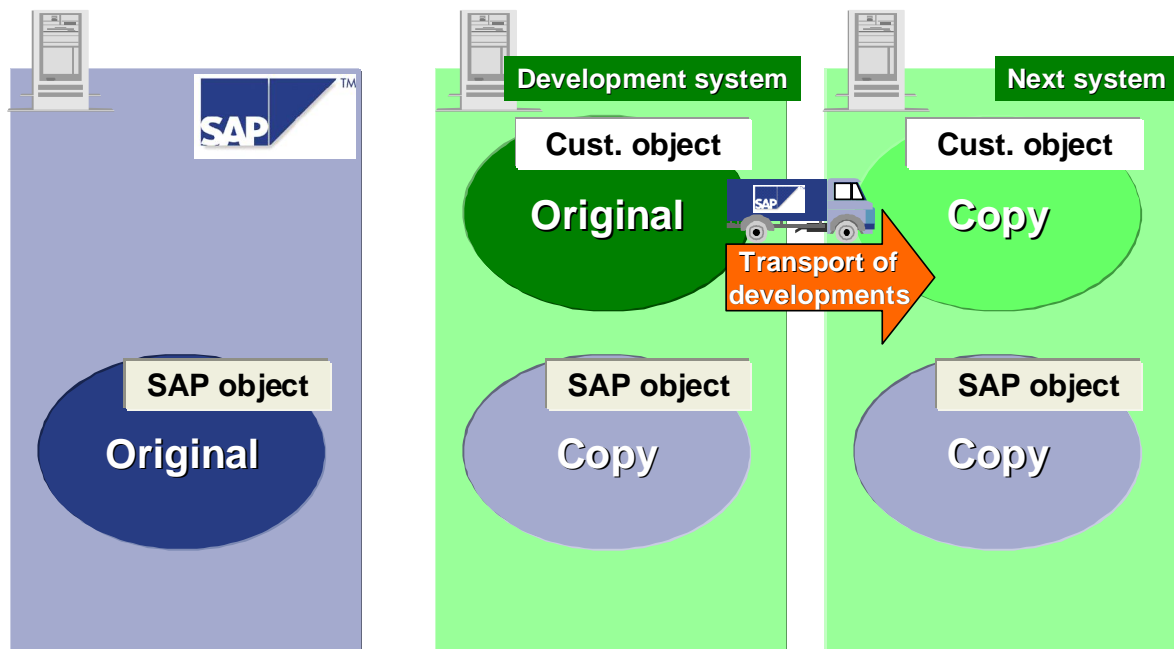
- **Describe what a modification is**
- **List the most important rules when making modifications**
- **Implement modifications using the Modification Assistant**
- **Deploy user exits to implement enhancements**
- **Adjust modifications**

© SAP AG 2008



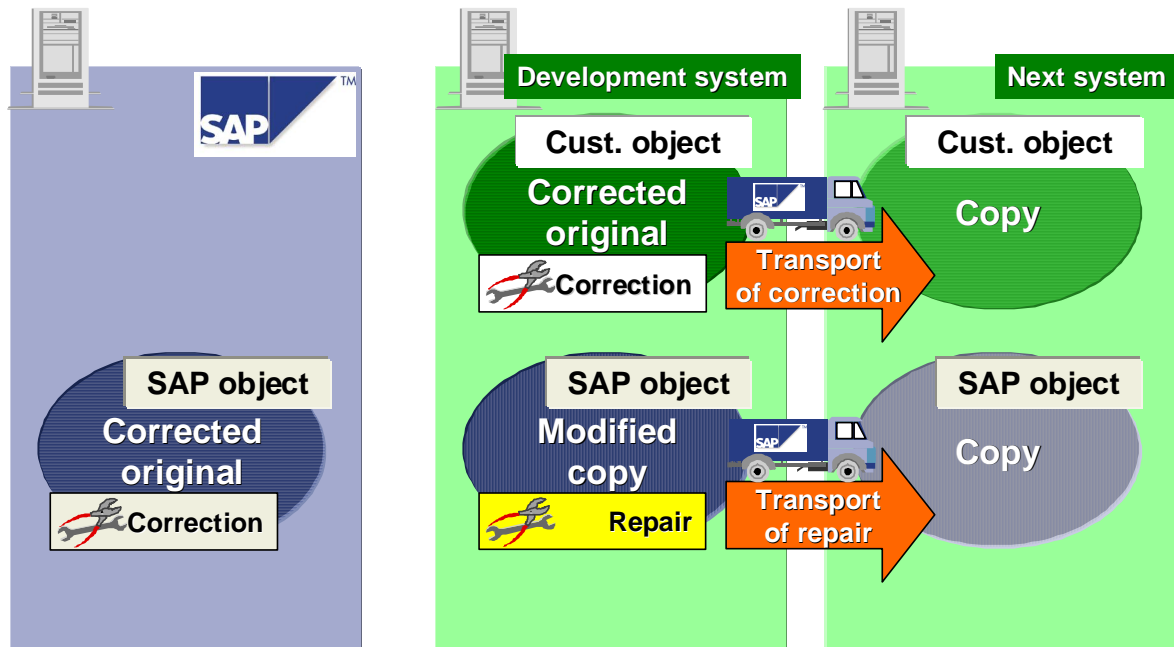
Originals and Copies

SAP



© SAP AG 2002

- An object is original in **only one system**. In the case of objects delivered by SAP, the original system is at SAP itself. In customer systems, these objects are then only available as copies. This applies to your development system and all other systems that come after it.
- If you write your own applications, the objects that you create are original in your development system. You assign your developments to a change request, which has the type **Development/Correction**.
- This request ensures that the objects are transported from the development system into the subsequent systems.

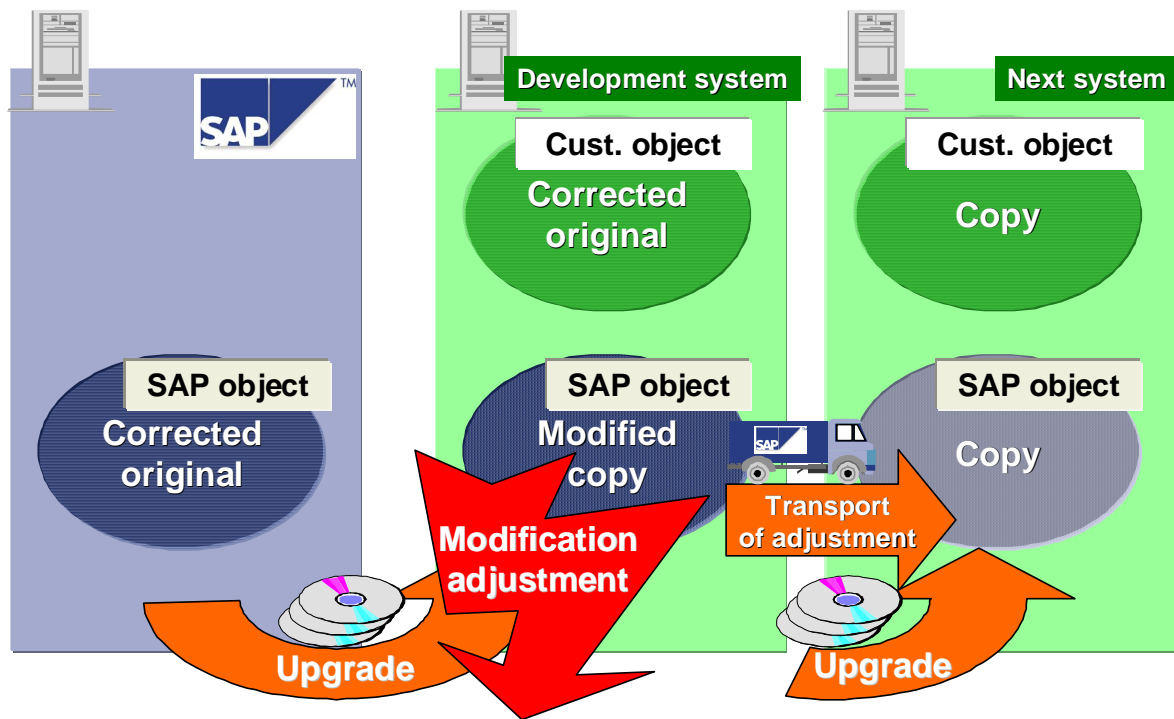


© SAP AG 2002

- Changes to an original are called **corrections**. They are recorded in a change request whose tasks have the type *Development/correction*.
- If, on the other hand, you change a copy (an object outside its own original system), the change is recorded in a task with the type **Repair**. Repairs to SAP objects are called **modifications**.
- When you repair your own objects (for example, if something goes wrong in your production system), you can correct the original in your development system straight away. **When you change copies, you must correct the original immediately!**
- However, you cannot do this with SAP objects, because they are not original in any of your systems.
- You should only modify the SAP standard if the modifications you want to make are absolutely necessary for optimizing workflow in your company. Be aware that good background knowledge of application structure and flow are essential prerequisites for deciding what kind of modifications to make and how these modifications should be designed.

Modifications and Upgrades

SAP



© SAP AG 2006

- Whenever you upgrade your system, apply a Support Package, or import a transport request, conflicts can occur with modified objects.
- Conflicts occur when you have changed an SAP object and SAP has also delivered a new version of it. The new object delivered by SAP becomes an active object in the Repository of your system.
- If you want to save your changes, you must perform a **modification adjustment** for the objects. If you have a lot of modified SAP objects, your upgrade can be slowed down considerably.
- To ensure consistency between your development system and subsequent systems, you should only perform modification adjustments in your development system. The objects from the adjustment can then be transported into other systems.



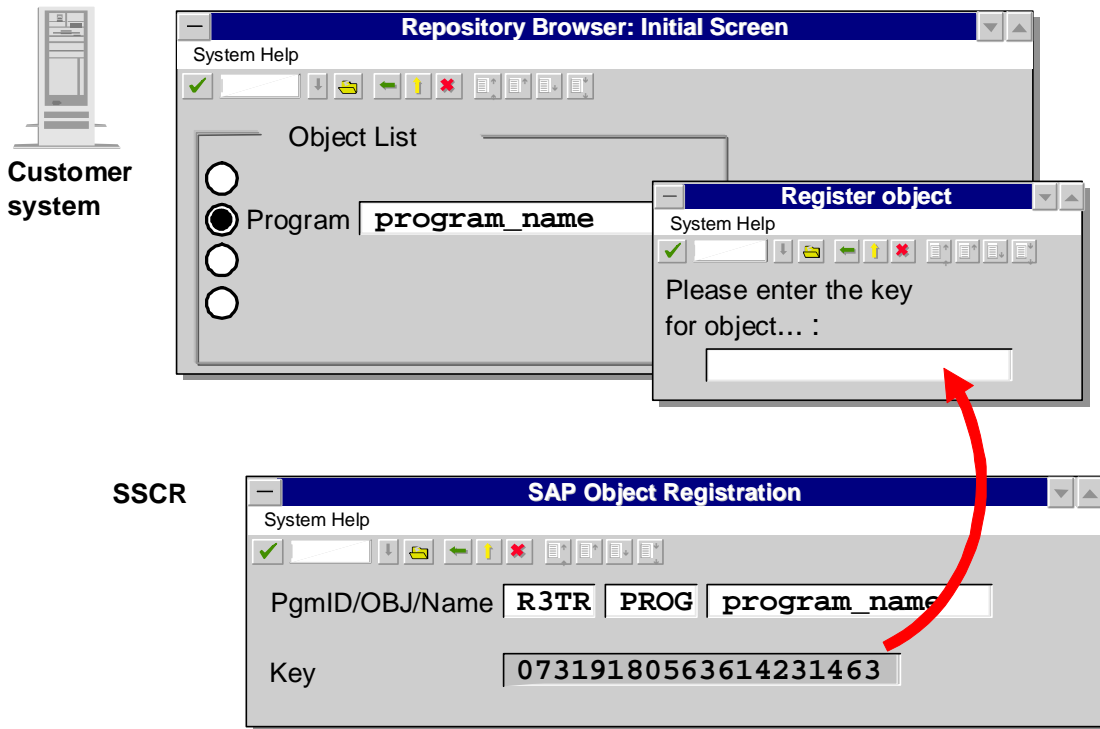
At the conclusion of this topic, you will be able to:

- **Describe what to pay attention to when modifying program objects**

© SAP AG 2002

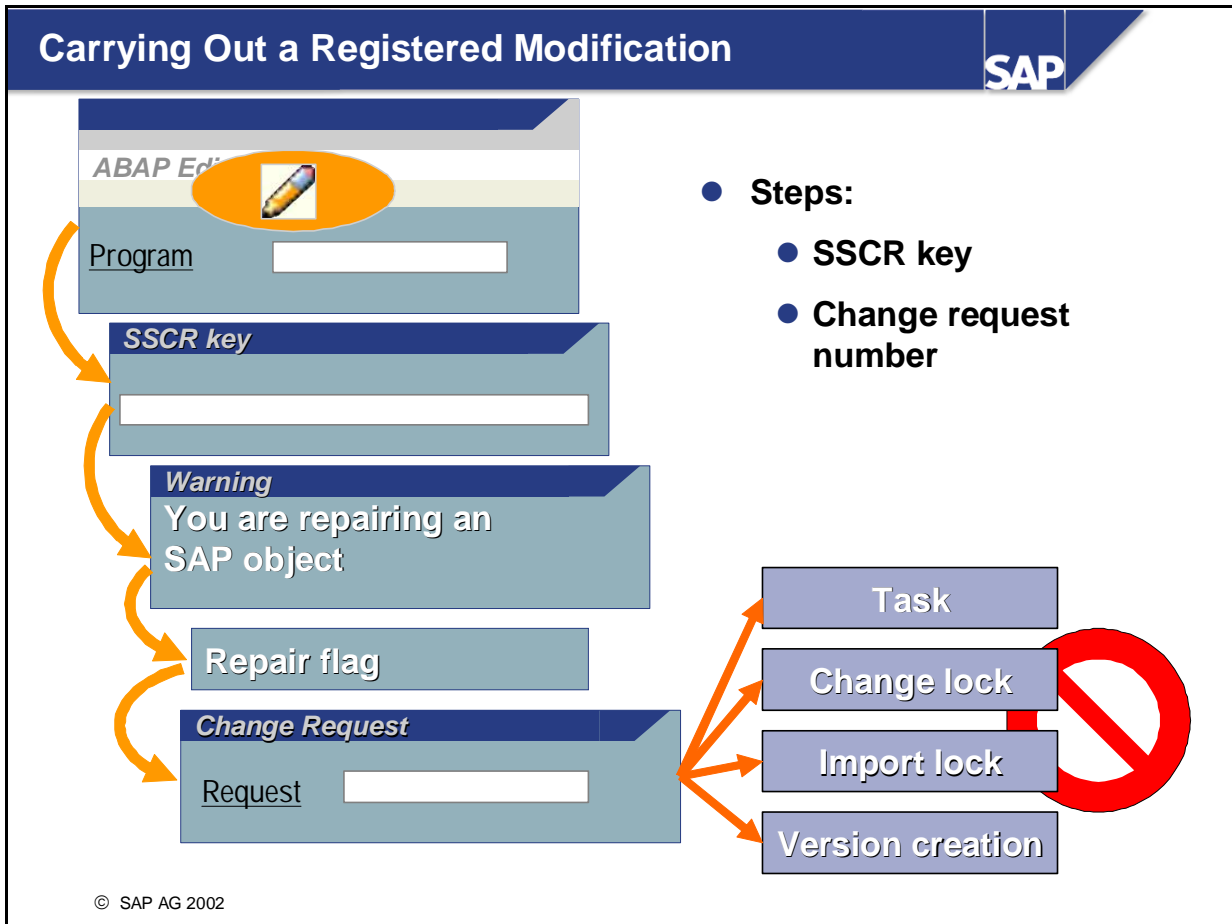
Registering Modifications in SSCR

SAP

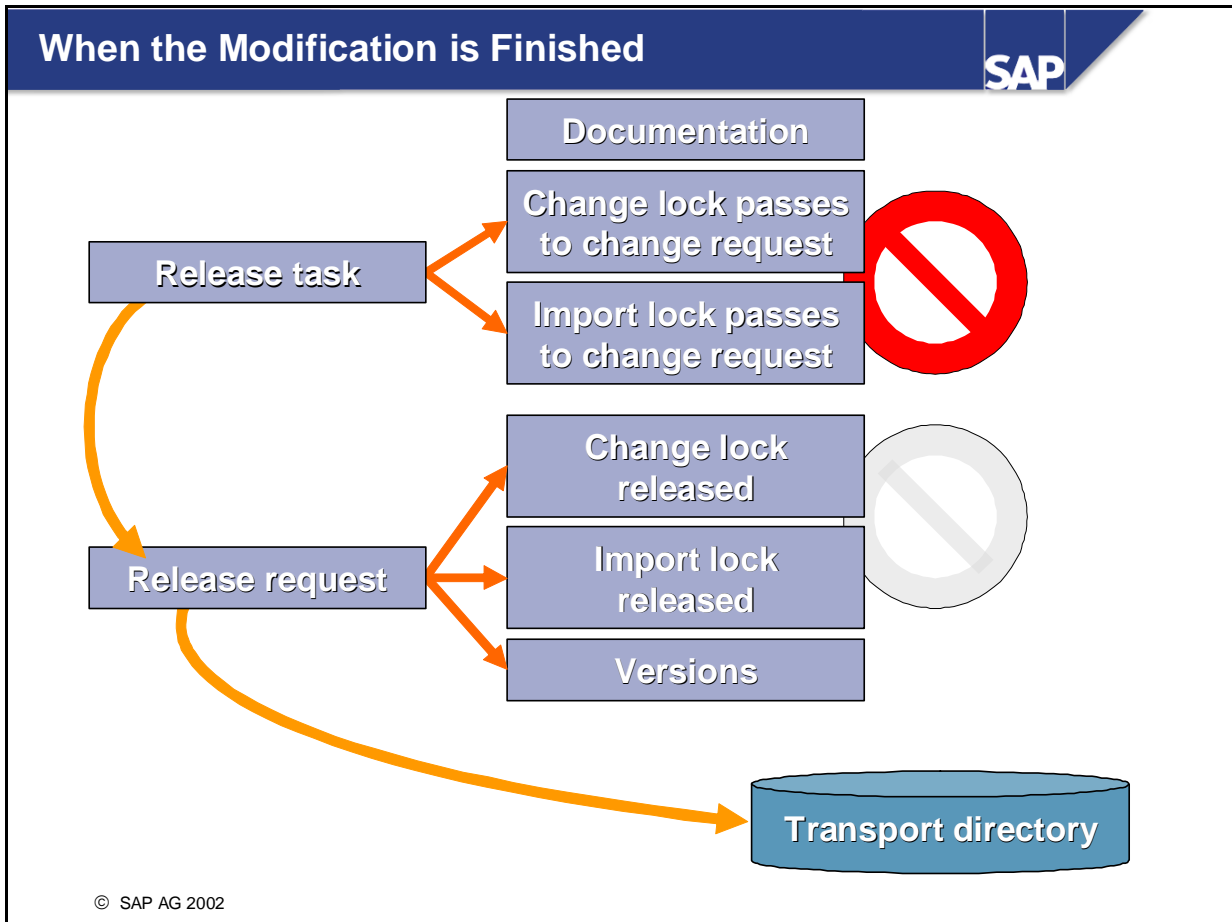


© SAP AG 2009

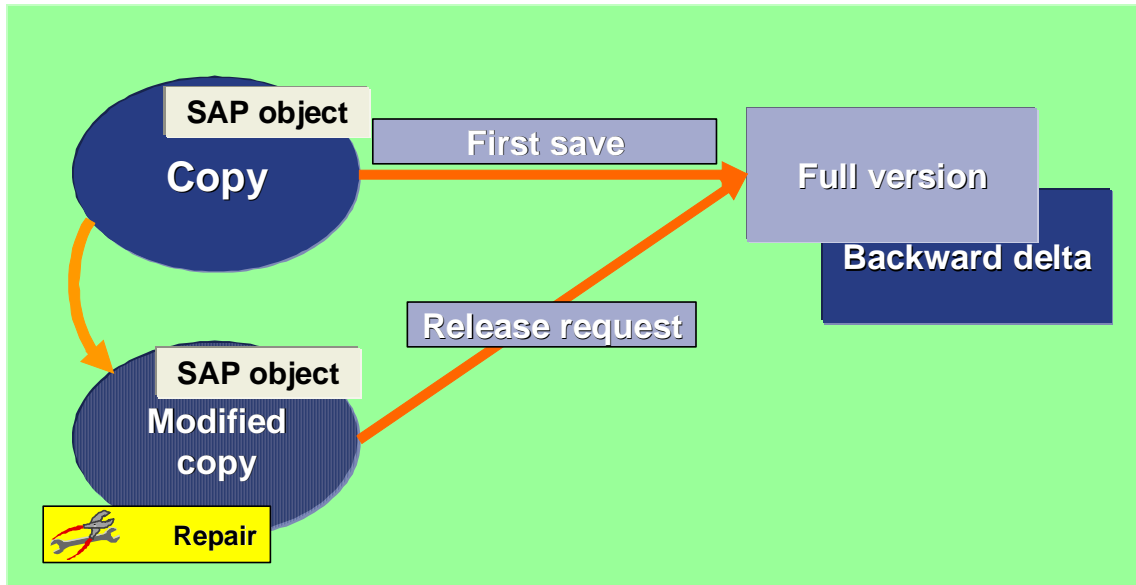
- A registered development user registers changes in SAP sources and manual changes of ABAP Dictionary objects. Exceptions to such registration are matchcodes, database indexes, buffer settings, customer objects, patches, and objects whose changes are based on automatic generation (for example, in Customizing). If the object is changed again at a later time, no new query is made for the registration key. Once an object is registered, the related key is stored locally and automatically copied for later changes, regardless of which registered developer is making the change. For the time being, these keys remain valid even after a release upgrade.
- What are the benefits of SSCR (SAP Software Change Registration)?
- Quick error resolution and high availability of modified systems
All objects that have been changed are logged by SAP. Based on this information, SAP's First Level Customer Service can quickly locate and fix problems. This increases the availability of your ERP system.
- Dependable operation:
Having to register your modifications helps prevent unintended modification. This in turn promotes the dependable operation of your ERP software.
- Simplification of upgrades:
Upgrades and release upgrades become considerably easier due to the smaller number of modifications.



- If you want to change an SAP Repository object, you must provide the Workbench Organizer with the following information:
 - SSCR key
 - Change request
- The above graphic shows you how to obtain an SSCR key. If you now continue to change the object, you must confirm the following warning dialogs: At this point, you can still cancel the action without repairing the object.
- The Workbench Organizer asks you to enter a change request, as it would for your own objects. The object is automatically added to a repair task. The change request has the following functions:
 - Change lock
After the task has been assigned, only its owner can change the object.
 - Import lock
The object cannot be overwritten by an import (upgrade or support package).
 - Version creation
The system generates a new version of the object (see below).



- After development is finished, the programmer releases the task. At this point, the programmer must document the changes made. The objects and object locks valid in the task are transferred to the change request. If the developer confirms the repair, the import lock passes to the change request. If the developer does not confirm the repair when releasing the task, the import lock remains in place. Only the developer can release this lock.
- Once the project is completed, you release the change request. This removes all of the change request's object locks. This applies both to the change locks and the import locks.
- When the change request is released, the objects are copied from the R/3 database and stored in a directory at operating system level. The system administrator then imports them into subsequent systems.
- After the modifications have been imported into the quality system, the developer must test them and check the import log of the request.



© SAP AG 2009

- When you release a change request, a complete version of all objects contained in the change request is written to the versions database.
- If you transport the Repository object again later, the current object becomes a complete copy and the differences between the old and the new object are stored in the versions database as a backwards delta.
- When you assign a Repository object to a task, the system checks whether the current version matches the complete copy in the versions database. If not, a complete copy is created. This process is also initiated the first time you change an object, since SAP does not deliver versions of Repository objects.
- The versions of a Repository object provide the basis for modification adjustment. To support adjustment, information on whether the version was created by SAP or by the customer is also stored.



Encapsulation instead of insertion

```
REPORT sapabap.
IF sy-tabix = 1.
  *#SD_001...#Insertion
  counter = count + 1.
  LOOP AT itab
    WHERE f1 < 10.
    ....
  ENDLOOP.
ENDIF.
```

Insertion

```
REPORT sapabap.
IF sy-tabix = 1.
  *#SD_001...#Insertion
  CALL FUNCTION 'Z_FM'
  CHANGING
    counter = count
  TABLES
    itab = tab.
ENDIF.
```

```
FUNCTION z_fm.
  counter = counter + 1.
  LOOP AT itab
    WHERE f1 < 10.
    ....
  ENDLOOP.
ENDFUNCTION.
```

Encapsulation



Use narrow interfaces during encapsulation

- Encapsulate customer source code in modularization units instead of inserting it directly into SAP source code (with, for example, customer function module calls in program source code, or customer subscreen calls for additional screen fields).
- Be careful to use narrow interfaces when you encapsulate customer-specific functions, in order to ensure good data control



- **Use standardized inline documentation (supported by the Modification Assistant)**
- **Do not delete any SAP source code - comment it out (supported by the Modification Assistant)**
- **Keep a modification logbook (Possibly using the Modification Log from SE95 as a basis)**
- **Do not modify any central Basis Dictionary Objects (Unless directed to by an SAP Note or the SAP Hotline)**
- **Release all requests that contain repairs**

© SAP AG 2009

- Define a company-wide standard for managing the documentation in the source code.
- You should also maintain a list of all modifications to your system (a modification log, see the following slide).
- Any modifications that you make to ABAP Dictionary objects that belong to SAP Basis components (ABAP Workbench and so on) are lost at upgrade - these objects revert to their earlier form and no adjustment help is offered. This can lead to the loss of the contents of certain tables.
- All requests that contain repairs must be released before an upgrade or Support Package import so that all relevant customer versions can be written to the versions database (the system compares versions during adjustment!). .

Modification Logs (Example)



Object type (program, screen, GUI status, ...)	PROG
Object Name	SAPMV45A
Routine	SAVE_DOC
Subject area	SD_001
Request number of repair	DEVK900023
Change date	01.02.2007
Changed by	Smith
Person responsible	Carpenter
Preliminary correction? (yes/no)	No
SAP Note number	—
Valid to release	—
Time required to restore (hours)	4

© SAP AG 2009

- SAP recommends that you keep a **record of all modifications that have been made to your system** (that is, of any changes you have made to Repository objects in the SAP namespace).
- Record the information in a list with the following **columns**:
 - Object type (program, screen, GUI status, ...)
 - Object name
 - Routine (if applicable)
 - Subject area (according to process design blueprint or technical design)
 - Repair number
 - Change date
 - Changed by
 - Preliminary correction? (yes/no)
 - SAP Note number, valid until Release x.y
 - Estimated time to restore the modification during the adjustment



At the conclusion of this topic, you will be able to:

- **Implement modifications using the Modification Assistant**
- **Describe how the Modification Assistant works**

© SAP AG 2002

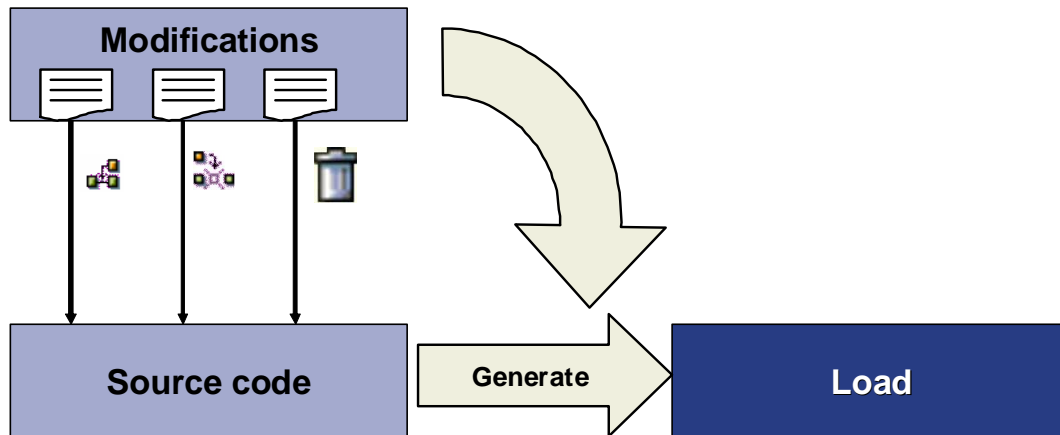
Simplifying modification adjustments

- **Finer granularity**
 - **Modules**
 - **Routines**
 - **Function modules**
- **Changes in a separate software layer using modification exits**

© SAP AG 2002

- The aim of the Modification Assistant is to make modification adjustments easier. In the past, the granularity of modifications was only at include program level. Today, a finer granularity is available. Now, modifications can be recorded at subroutine or module level.
- This is because (among other reasons) the modifications are registered in a different layer. As well as providing finer granularity, this means that you can reset modifications, since the original version is not changed.

How the Modification Assistant Works



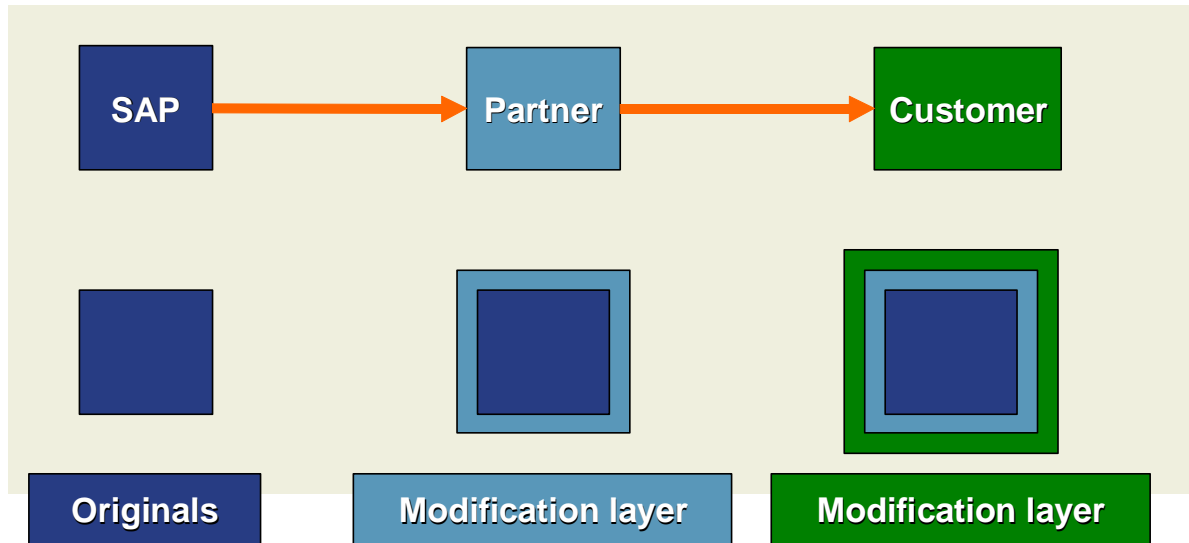
© SAP AG 2002

- The Modification Assistant records changes in a separate layer. This means that the original source code of the object does not change. The modifications are not adopted until the load is generated, so that the executable object is created from both the parts of the original SAP program and the modifications.

- **THEN:**
 - **Granularity: Include program**
 - **Modification adjustment line by line**
 - **Each modification had to be included in the new SAP version manually (cut & paste)**
- **NOW (using Modification Assistant):**
 - **Granularity: Module (such as subroutine)**
 - **Modification adjustment at module level ("modification exit")**

© SAP AG 2006

- In the past, if you modified an include for which SAP provided a new version in an upgrade, a modification adjustment was necessary. The modification adjustment had to be performed line by line. The system provided little support.
- The Modification Assistant changes that situation: The granularity of the change recording has been refined. For example, if you modify a subroutine, the rest of the include remains unchanged. If SAP delivers a new version of the include, the system looks to see if there is also a new version of that subroutine. If this is not the case, your changes can be incorporated into the new version automatically.



© SAP AG 2006

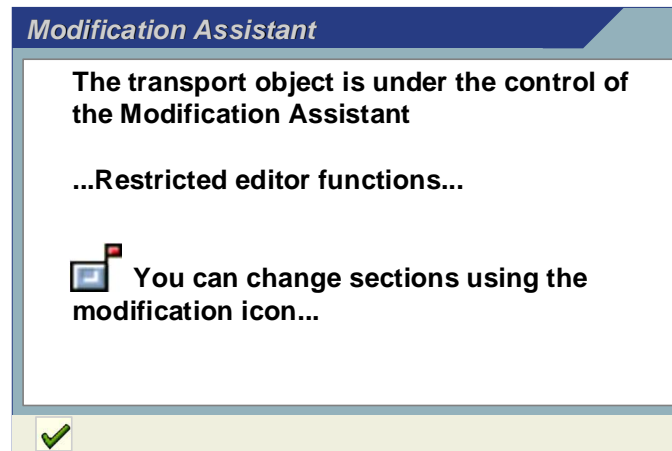
- The original version of each software layer consists of the originals from the previous layer plus current modifications.

- **ABAP Editor**
 - **Modification mode**
- **Screen Painter**
 - **Layout**
 - **Flow logic**
- **Menu Painter**
- **Text Elements**
- **Class Builder**
- **Function Builder**
 - **Adding function modules**
 - **Compatible interface enhancements**
- **ABAP Dictionary**
 - **Append structures are registered**
 - **Data elements: Field labels can be changed**
- **Documentation**
 - **Substituting documentation objects**

© SAP AG 2009

- Above is a list of the tools supported by the Modification Assistant.
- In the ABAP Editor, you can use modification mode to change source code. Only a restricted range of functions is available in this mode. You can add, replace, or comment out source code, all under the control of the Modification Assistant.
- Changes to layout and flow logic in the Screen Painter are also recorded.
- The Modification Assistant also records changes in the Menu Painter and to text elements, as well as the addition of new function modules to an existing function group.
- To avoid conflicts in the upgrade, table appends are also logged by the Modification Assistant.

- R/3 Profile Parameter **eu/controlled_modification**



© SAP AG 2006

- If you want to change an SAP object, you must provide the following information:
 - SSCR key
 - Change request
- The system informs you that the object is under the control of the Modification Assistant. Only restricted functions are available in the editor.
- You can switch the Modification Assistant on or off for the entire system using the profile parameter **eu/controlled_modification**. SAP recommends that you always work with the Modification Assistant.
- You can switch off the Modification Assistant for single Repository Objects. Once you have done so, the system no longer uses the fine granularity that is used in the Modification Assistant.

Modification Assistant Icons



Insert



Replace



Delete



Undo modification



Modification Overview

© SAP AG 2006

- In modification mode, you cannot use all of the normal functions of the tool with which you are working. You can access these using the appropriate pushbuttons. For example, in the ABAP Editor, you can:
 - Insert
 - The system generates a framework of comment lines between which you can enter your source code.
 - Replace
 - Position the cursor on a line and choose "Replace". The corresponding line is commented out, and another line appears in which you can enter coding. If you want to replace several lines, mark them as a block first.
 - Delete
 - Select a line or a block of source code. The lines are commented out.
 - Undo modifications
 - This undoes all of the modifications you have made to this object.
 - Display modification overview
 - Choose this function to display an overview of all modifications belonging to this object.

Modification Assistant: ABAP Editor Example

SAP

ABAP Editor

```
PROGRAM <SAP-Program>.
```

```
...
```

```
*{INSERT   DEVK900023                                1
```

```
*}
```

```
*{REPLACE   DEVK900023                                2
```

```
*\WRITE: / sy-uname COLOR COL_KEY.
```

```
WRITE: / sy-uname COLOR COL_NORMAL.
```

```
*}
```

```
*{DELETE   DEVK900023                                3
```

```
*\WRITE: / sy-uname COLOR COL_KEY.
```

```
*\
```

```
*}
```

```
...
```


 **Insert**

 **Replace**

 **Delete**



© SAP AG 2006

- The graphic shows the results of changes that are made using the Modification Assistant.
- The Modification Assistant automatically generates a framework of comment lines describing the action. The comment also contains the number of the change request to which the change is assigned, and a number used for internal administration.

Modification Overview 

ABAP Editor


```
PROGRAM <SAP-Program>.  
...  
*{INSERT  DEVK900023  
*}  
*{REPLACE  DEVK  
*\WRITE: / sy-u  
WRITE: / sy-un  
*}  
...  
© SAP AG 2006
```

Modification Overview

SAP program

- without modularization unit
 - SAP program SMITH 01.02.2007 DEVK900023
- Subroutines
 - DO_SOMETHING SMITH 01.02.2007 DEVK900023



- The "modification overview" icon provides you with an overview of the modifications you have made in the current program.
- The display is divided up according to the various modularization units. This corresponds to the structure used by the Modification Assistant to record the modifications.

Restoring the Original

SAP

ABAP Editor

```
PROGRAM <SAP-Program>.  
...  
*{INSERT   DEVK900023           1  
*}  
*{REPLACE  DEVK900023           2  
*\WRITE: / sy-uname COLOR COL_KEY.  
WRITE: / sy-uname COLOR COL_NORMA  
*}  
...
```

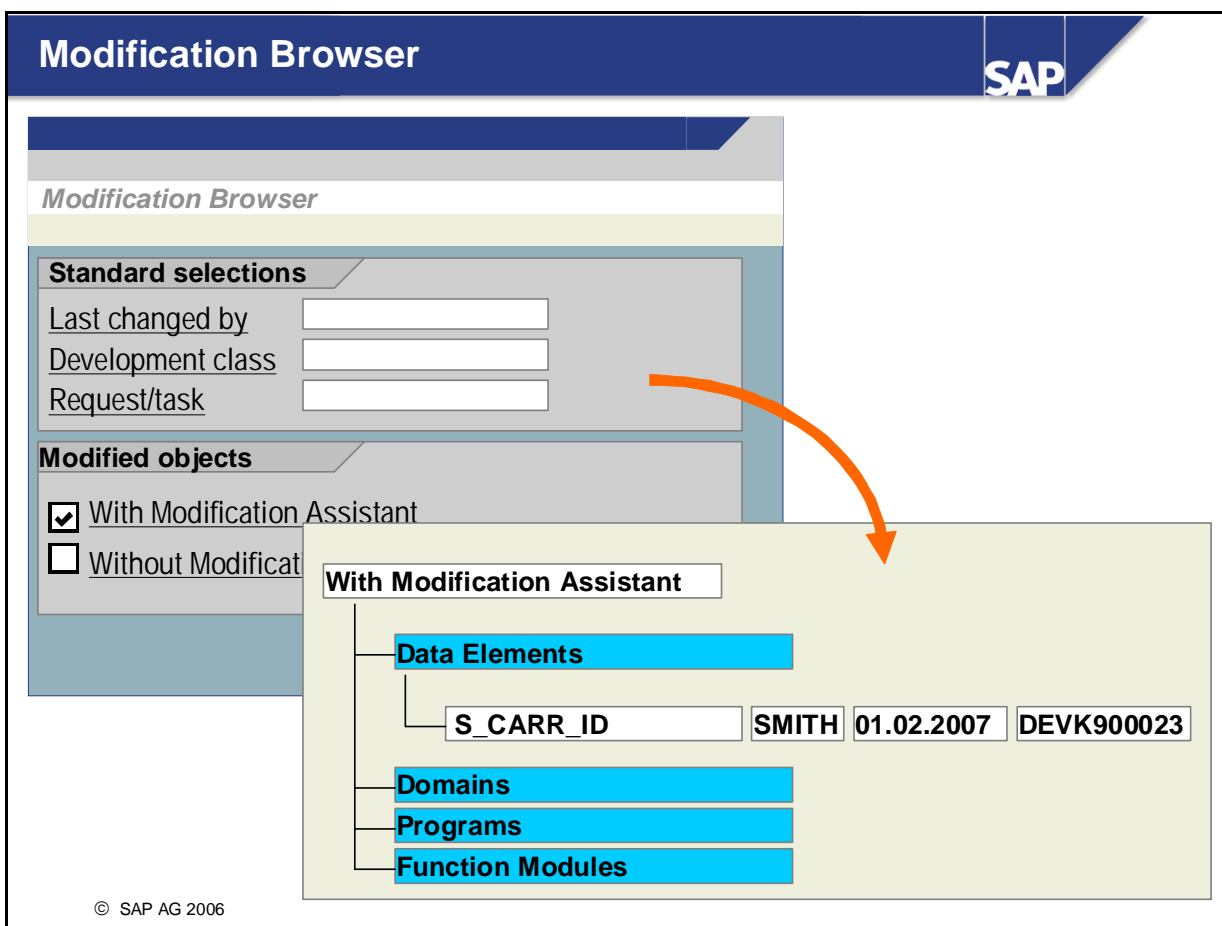


ABAP Editor

```
PROGRAM <SAP-Program>.  
...  
WRITE: / sy-uname COLOR COL_KEY.  
...
```

© SAP AG 2002

- To undo a modification, place the cursor on it and choose *Undo*.
- The record of the modifications is deleted. You cannot restore this record.

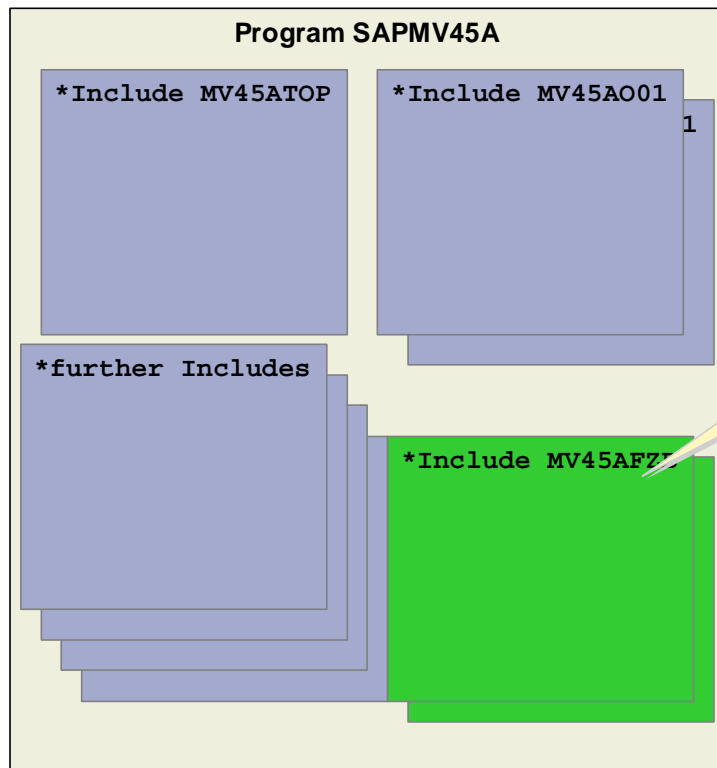


- The Modification Browser provides an overview of all of the modified objects in the system. The Modification Browser differentiates between modifications made using the Modification Browser and those made without it.
- On the initial screen of the Modification Browser, you can restrict the selection according to various criteria. This allows you to find modifications in a particular area.
- The Modification Browser displays the hit list in tree form. Objects are arranged by:
 - Modification type (with/without Assistant)
 - Object type (PROG, DOMA, DTEL, TABL, ...)
- In addition to the simple display functions, you can also use the Modification Browser to undo entire groups of modifications. To do this, select the desired subtree, and choose the button "Reset to Original".



At the conclusion of this topic, you will be able to:

- **Explain what user exits are**
- **Describe how to find user exits in the system and use them to enhance SAP software**

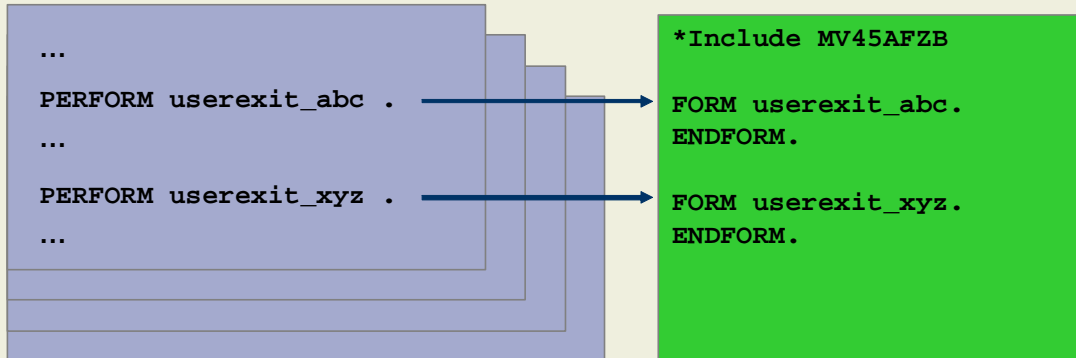


© SAP AG 2006

- A module pool is organized as a collection of include programs. This is particularly good for making the program easier to understand. The organization is similar to that of function groups. In particular, the naming convention by which the last three letters of the name of the include program identify its contents, is identical.
- The main program, as a rule, contains the include statements for all of the include programs that belong to the module pool.
- The includes described as "special" includes in the program are themselves only include programs - technically, they are not different. These programs are only delivered once.

Program SAPMV45A

Includes



- Blank Subprograms
- Includes delivered only once
- Mainly in SD
- Technically: Modification

© SAP AG 2006

- User exits are a type of system enhancement that were originally developed for the R/3 Sales and Distribution Module (SD). The original purpose of user exits was to allow the user to avoid modification adjustment.
- Using a user exit is a modification, since it requires you to change objects in the SAP namespace.
- SAP developers create a special include in a module pool. These includes contain one or more subroutines routines that satisfy the naming convention `userexit_<name>`. The calls for these subroutines have already been implemented in your program. Usually global variables are used.
- After delivering them, SAP never alters includes created in this manner; if new user exits must be delivered in a new release, they are placed in a new include program.

```

***INCLUDE MV45AFZB .

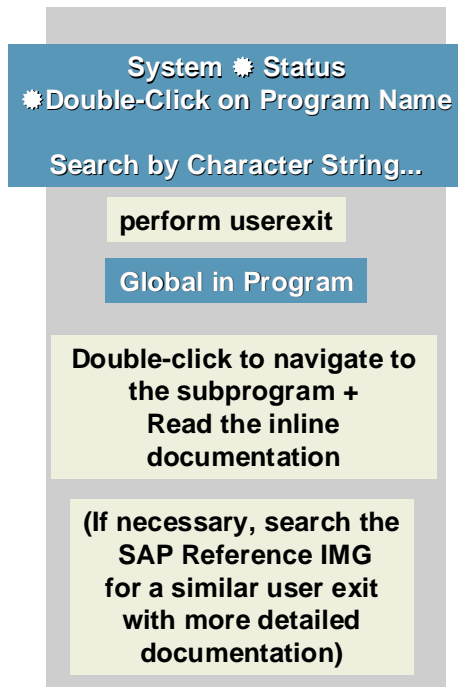
*****
* This include is reserved for user modifications      *
* Forms for sales document processing                  *
* The name of modification modules should begin with 'ZZ'. *
*****
*
*&-----*
*& Form  USEREXIT_FILL_VBAP_FROM_HVBAP
*&-----*
*      This User exit can be used to fill additional data into VBAP*
*      from the main item (HVBAP), i.e. this User exit is called  *
*      when an item is entered with reference to a main item.    *
*      This form is called from form VBAP_FUELLEN_HVBAP.         *
*&-----*
FORM userexit_fill_vbap_from_hvbap.
* VBAP-zzfield = HVBAP-zzfield2.
ENDFORM.

```

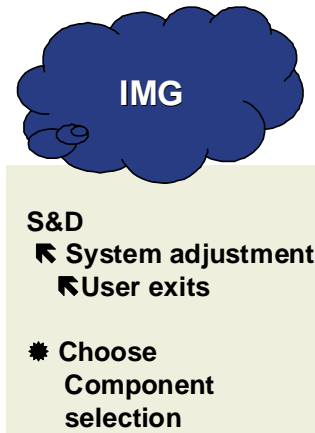
© SAP AG 2009

- User exits are actually empty subroutines that SAP developers provide you with. You can fill them with your own source code.
- The purpose behind this type of system is to keep all changes well away from program source code and store them in include programs instead. To this end, SAP developers create various includes that fulfill the naming conventions for programs and function groups. The last two characters of the Include name indicate the Include to be used by the customer: Normally, there is a "Z" here.
- Example: Program SAPMV45A
 Include MV45AFZB
- This naming convention guarantees that SAP developers will not touch this include in the future. For this reason, includes of this nature are not adjusted during modification upgrade.
- If any new user exits are delivered by SAP with a new release, then they are bundled into new includes that adhere to the same naming convention.

- **Search by Program**



- **Searching Using tools**



© SAP AG 2009

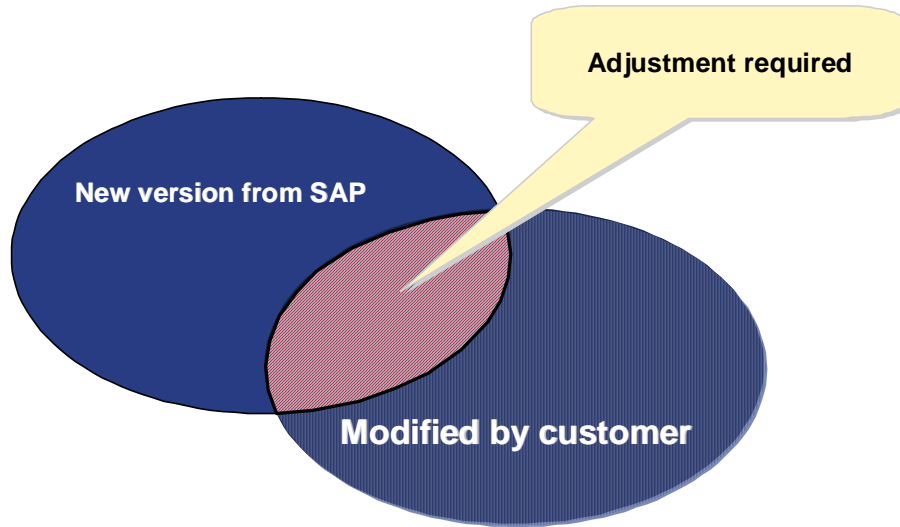
- You can find a list of all user exits in the SAP Reference Implementation Guide.
- There, you will also find documentation explaining why SAP developers have created a particular user exit.
- Follow the steps described in the Implementation Guide.



At the conclusion of this topic, you will be able to:

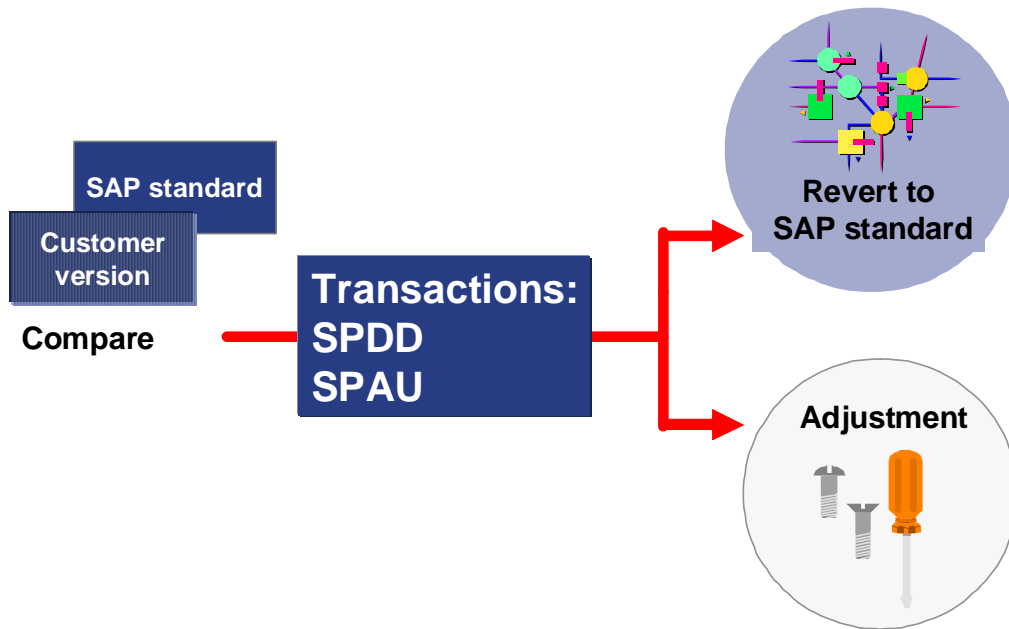
- **Name the steps taken during a modification adjustment**
- **List which objects must be adjusted and when**
- **Describe how the modification adjustment is performed in subsequent systems**

© SAP AG 2002



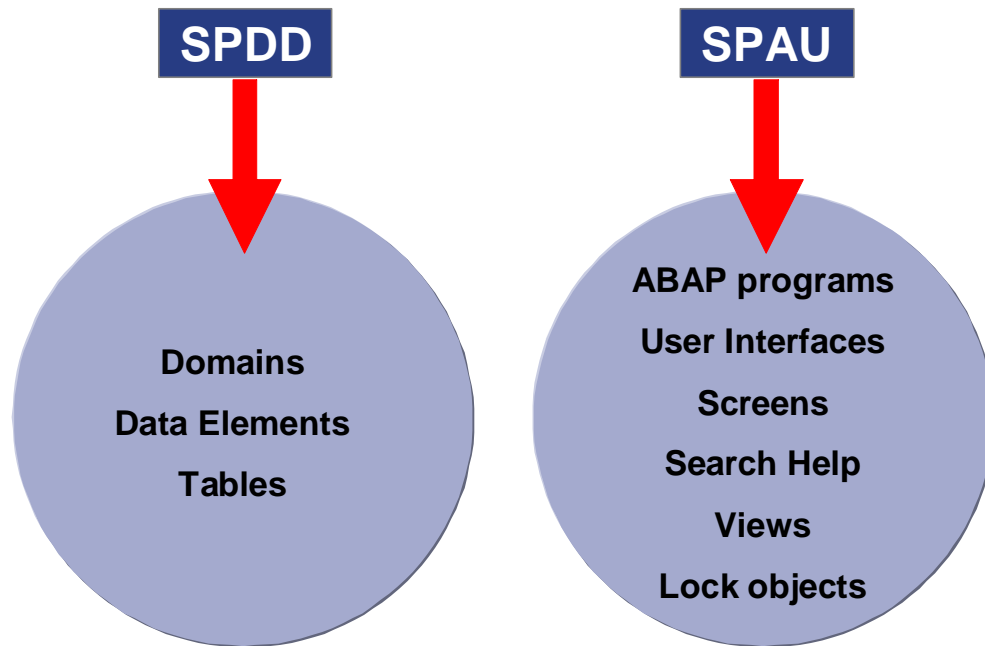
© SAP AG 2002

- The set of objects for adjustment is derived from the set of new objects delivered by SAP in a new release. This is compared with the set of modified objects.
- The intersection of these two sets is the set of objects that must be adjusted when you import an upgrade or support package.



© SAP AG 2009

- During modification adjustment, old and new versions of ABAP Repository objects are compared using transactions SPDD and SPAU.
- You **do not** have to call transaction SPDD to adjust Dictionary objects if:
 - No changes have been made to SAP standard objects in the Dictionary
 - You have only added customer objects to your system. Only SAP objects that have been changed must be adjusted using this transaction.
- All other ABAP Repository objects are adjusted using transaction SPAU. Upgrade program R3up tells you to start the transaction after upgrade has finished. You have 30 days to use transaction SPAU after an upgrade. After 30 days, you must apply for a SSCR key for each object that you want to adjust.
- Transaction SPAU first identifies which objects have been modified and then ascertains for which of the modified objects a new version has been imported during the current upgrade. Modification adjustment allows you to transfer the modifications you have made in your system to your new R/3 Release.

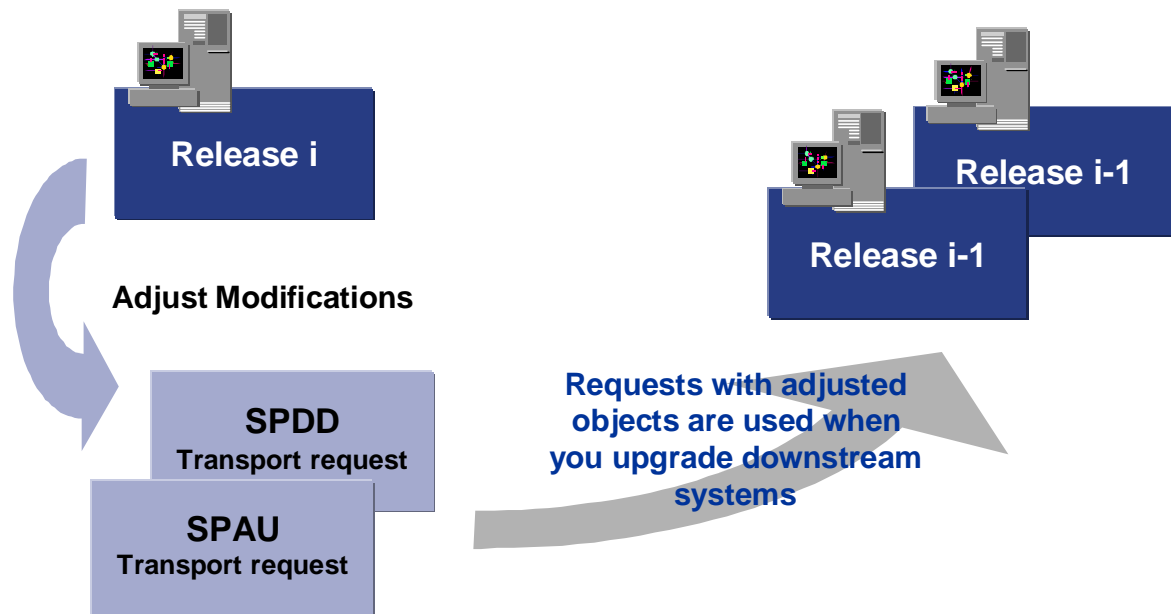


© SAP AG 2006

- During the modification adjustment, use transaction SPDD to adjust the following ABAP Dictionary objects:
 - Domains
 - Data Elements
 - Tables (structures, transparent tables, pool, and cluster tables, together with their technical settings)
- These three object types are adjusted directly after the Dictionary object import (before the main import). At this point in time, no ABAP Dictionary objects have yet been generated. To ensure that no data is lost, it is important that any customer modifications to domains, data elements, or tables are made before you generate them.
- Changes to other ABAP Dictionary objects, such as lock objects, matchcodes, or views, cannot result in loss of data. Therefore, these ABAP Dictionary objects are adjusted using transaction SPAU after both main import and object generation have been completed. You can use transaction SPAU to adjust the following object types:
 - ABAP programs, interfaces (menus), screens, matchcode objects, views, and lock objects.

Transporting Adjustments between Systems

SAP



© SAP AG 2009

- During the modification adjustment, you should carry out your object processing using two separate change requests: one for the SPDD adjustment, one for the SPAU adjustment. These change requests are then transported into other SAP systems you want to adjust. This guarantees that all actual adjustment work takes place solely in your development system.
- When upgrading additional SAP systems, all adjustments exported from the first system upgrade are displayed during the ADJUSTCHK phase. You decide which adjustments you want to accept into your additional systems and these are then integrated into the current upgrade. Afterwards, the system checks to see if all modifications in the current SAP system are covered by the change requests created during the first system upgrade. If this is the case, no adjustments are made during the current upgrade.
- Note: For this process to be effective, it is important that all systems involved have identical system landscapes. This can be guaranteed by first making modifications in your development system and then transporting them to later systems before you upgrade the development system. You can also guarantee that all of your systems have an identical system landscape by creating your development system before upgrade as a copy of your production system and then refraining from modifying the production system again.

Modification Adjustment: Introduction

SAP

Modification Adjustment: Object Selection

Standard selections

Last changed by

Development class

Request/task

Modified objects

☒ With Modification Assistant

☐ Without Modification Assistant

Upgrading and Patch

☒ To be edited

☐ All objects

With Modification Assistant

- Data Elements
 - S_CARR_ID SMITH 01.02.2007
- Domains
- Programs
- Function Modules

© SAP AG 2009

- When you start the modification adjustment (transaction SPAU) you can restrict the hit list on a selection screen. You can decide whether you want to display all objects that are to be adjusted, or only those that have yet to be processed. You can use the following criteria to restrict the selection:
 - Last changed by
 - Development class
 - Request numbers/Task numbers
- The system displays a list of the objects to be adjusted. The list is sorted by:
 - With/without Modification Assistant
 - Object type



Automatic adjustment



Semi-automatic adjustment



Manual adjustment



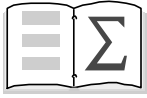
Object adjusted



Original restored

© SAP AG 2002

- The icons in front of the individual objects that need adjustment show how they can be adjusted. The possible methods are:
 - Automatically
The system could not find any conflicts. The changes can be adopted automatically by choosing the appropriate icon or menu entry.
 - Semi-automatically
The individual tools support you in adjusting the objects. When you adjust programs the Split Screen Editor is displayed, so that you can make the changes there.
 - Manually
You must process your modifications with no special support from the system. In this case, the modification adjustment does allow you to jump directly into the relevant tool.
- Adjusted objects are identified by a green tick.
- If you want to use the new SAP standard version, use *Reset to Original*. If you do this, you will have no further adjustment work in future.



You are now able to:

- **Describe what a modification is**
- **List the most important rules when making modifications**
- **Implement modifications using the Modification Assistant**
- **Deploy user exits to implement enhancements**
- **Adjust modifications**

© SAP AG 2008

Exercises



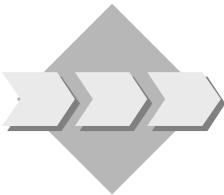
Unit: Modifications

Topic: Implementing Modifications



At the conclusion of these exercises you will be able to:

- Implement modifications using the Modification Assistant.
- Implement non-registered modifications



In addition to the large range of functions in R/3, you also want to implement further functions.

Functions containing errors are very occasionally delivered. This requires inserting corrections before the corresponding support package can be imported.

The Modification Assistant does not allow certain modifications. If you want to implement them regardless, you can deactivate the Modification Assistant.

- 1-1 Modify R/3 objects. Use the Modification Assistant where possible. The objects to be changed are specified below:
- 1-2 Modify the program **SAPBC425_BOOKING_##**.
 - 1-2-1 Enhance the header so that the column with the customer's name also has a header.
 - 1-2-2 Create a new variable for counting the data records. Output the counter in the last column of the list.
 - 1-2-3 Read the fields **LUGGWEIGHT** and **WUNIT** in the table **SBOOK** too, and output them in the list.
- 1-3 Modify the program **SAPBC425_FLIGHT##**.
 - 1-3-1 Change the layout of screen 0100: Insert a frame around the three input fields. Create a pushbutton and assign the function code **MORE** to it.

- 1-4 Modify the data element **S_CARRID##**.
 - 1-4-1 Change the field labels to:
 - Short text: "Airl"
 - Medium: "Airline".
 - 1-4-2 Modify the documentation for this data element. Create a meaningful text.
- 1-5 Check your modifications in the Modification Browser.

Exercises



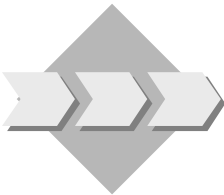
Unit: Modifications

Topic: Modification Adjustment



At the conclusion of these exercises you will be able to:

- Adjust modifications



You need to adjust the modifications made in the system after applying a support package or an upgrade.

2-1 Adjust the modifications you made to the objects you imported into the system.



Unit: Modifications

Topic: Implementing Modifications

- 1-1 Modify R/3 objects. Use the Modification Assistant.
- 1-2 Modify the program **SAPBC425_BOOKING_##**.
 - 1-2-1 You can change the header either directly from the list (*System* → *List* → *List Header*) or in the Editor.
 - 1-2-2 You can create a new variable directly in the SAP program. Use the insert function in the Modification Assistant. Ideally you keep the changes locally in the **data_output** subroutine. Output the counter too. Alternatively you can implement this function in the enhancement, which would not require a modification.
 - 1-2-3 Read the fields **LUGGWEIGHT** and **WUNIT** in the table **SBOOK** too, and output them in the list:
Extend the **SELECT** statement to include these two fields. Output the fields in the **data_output** subroutine.
- 1-3 Modify the program **SAPBC425_FLIGHT##**.
 - 1-3-1 Use the Screen Painter to change the layout of screen 0100.
- 1-4 Modify the data element **S_CARRID##**.
 - 1-4-1 Call the maintenance transaction for data elements. Place the cursor on the corresponding object and choose the modification icon. You can enter new text in the next dialog box.
 - 1-4-2 Choose the "Documentation" pushbutton and enter new text.
- 1-5 To check the modification choose the Modification Browser (transaction **SE95**). Limit the selection with the user name or change request/task.

Solutions



Unit: Modifications

Topic: Modification Adjustment

- 2-1 Start the Patch Manager (transaction **SPAM**). Call transaction **SPAU** from the menu path *Extras* → *Adjust Modifications*. You can adjust the modifications here.

Contents:

- Summary
- Evaluation of the different enhancement techniques

© SAP AG 2006



At the conclusion of this unit, you will be able to:

- **Describe the course contents**
- **Describe how to proceed when changing the SAP standard**
- **List the advantages and disadvantages of modifications**
- **Name the alternatives to modifications**

© SAP AG 2006



- **SAP enhancement concept:**

- SAP objects are not changed
- There is no adjustment during the upgrade

- **Support Packages are imported from the Online Correction Services (OCS)**

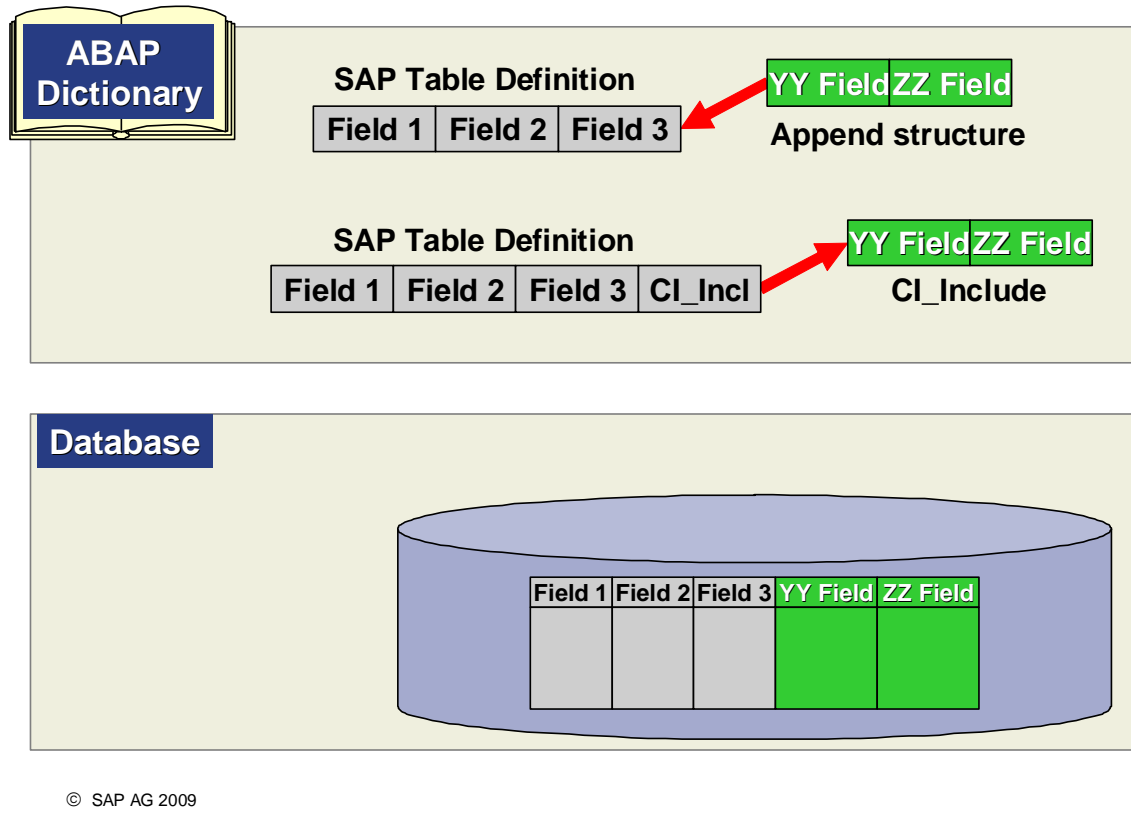


- **Return to the SAP standard**

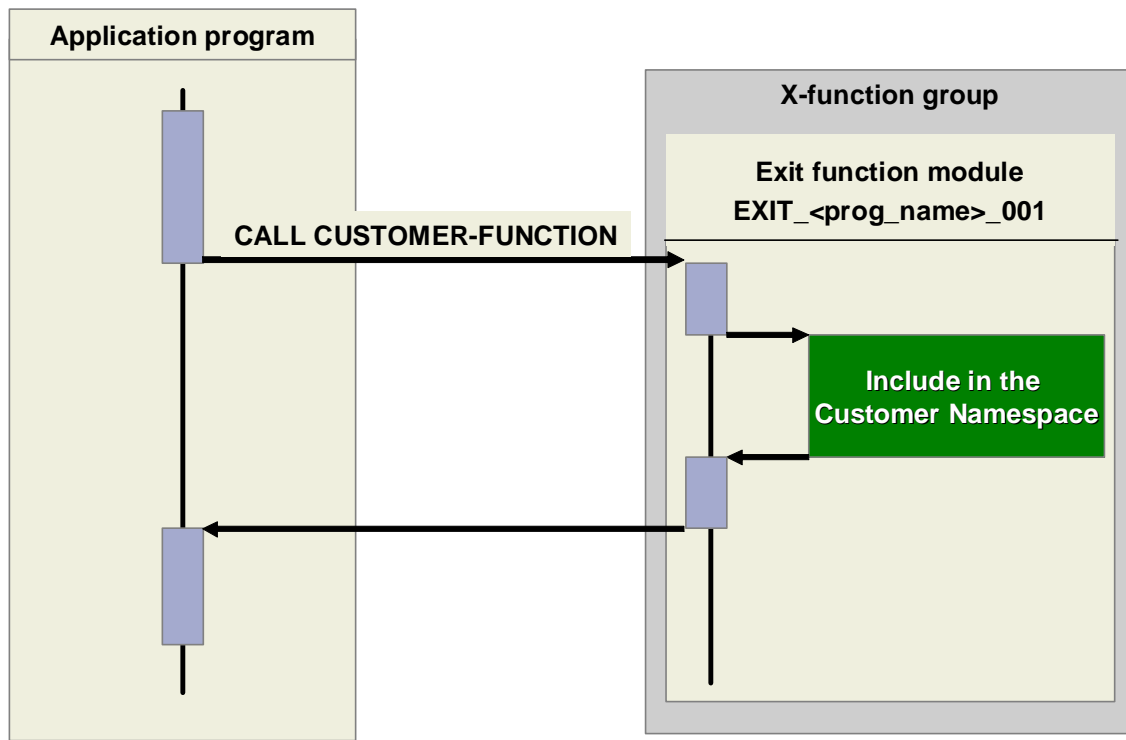
Version database and entries in the WBO and transport system are not changed

© SAP AG 2006

- Modification adjustment is not necessary if you avoid making changes to SAP objects.
- Use program enhancements and appends with SAP tables to enhance SAP objects in such a way that they your changes cannot be overwritten by SAP at upgrade.
- As of Release 3.0, you can use Online Correction Services to automatically import and cancel hot packages and patches (instead of having to insert preliminary corrections manually).



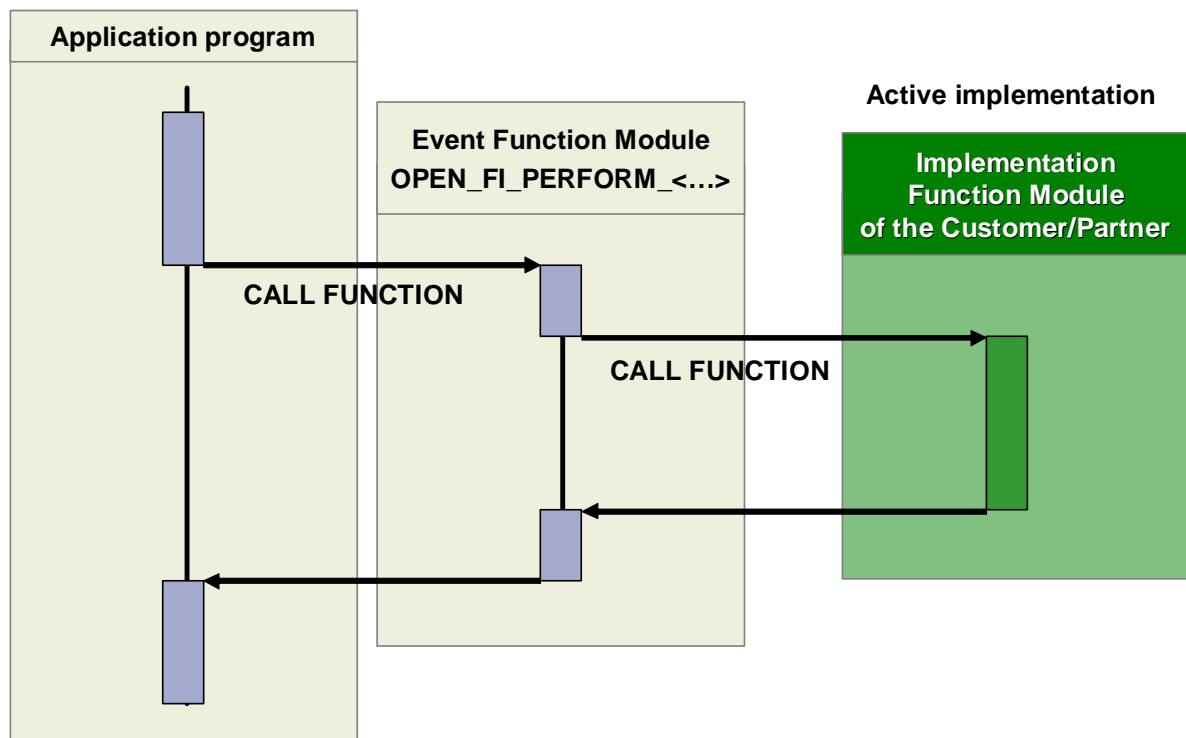
- SAP provides two ways to enhance tables and structures with fields.
 - Append structures
 - Customizing includes ("CI includes")
- Both techniques allow you to attach fields to a table without having to actually modify the table itself.
- An append structure is a structure which is assigned to exactly one table. There can be several append structures for a table. During activation, the system searches for all active append structures for that table and attaches them to the table.
- Append structures differ from include structures in how they refer to their tables. To include fields from an include structure in a table, you must add an '. INCLUDE...' line to the table. In this case, the table refers to the substructure. Append structures, on the other hand, refer to their tables. In this case, the tables themselves are not altered in any way by the reference.



© SAP AG 2006

- This graphic shows the flow of a program providing an enhancement in the form of a program exit.
- The exit function module is called at a point in the source text defined by the SAP application developer. Within the function module, users can add a function to the customer namespace using an include.

BTE Functions: How They Work / Flow Diagram

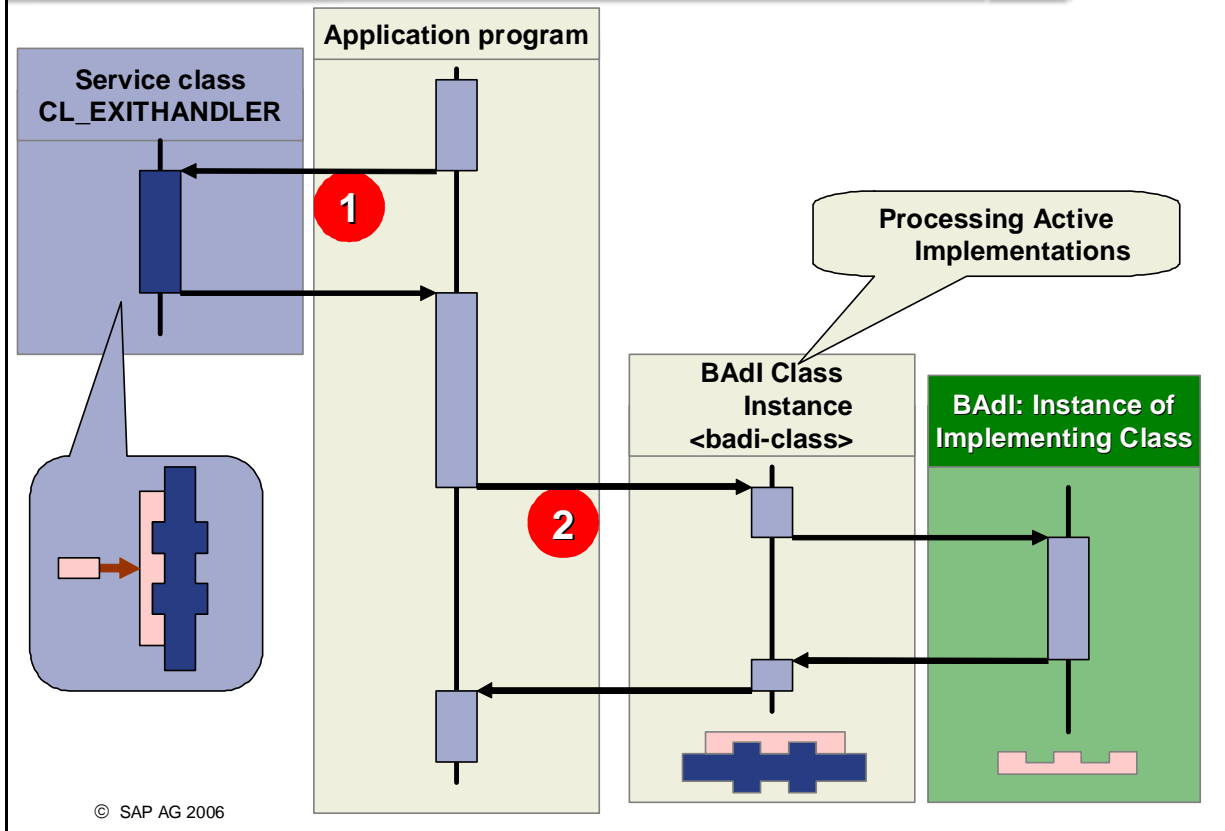


© SAP AG 2006

- The graphic shows the process flow of an SAP program. The program contains an enhancement in the form of a Business Transaction Event. A function module is called in the SAP program, which determines and processes the active implementations. The names of the event function modules begin with "OPEN_FI_PERFORM_" or "OUTBOUND_CALL_".
- The event function module OPEN_FI_PERFORM_<...> or OUTBOUND_CALL_<...> determines the active implementations for each enhancement and stores them in an internal table. Function modules are implemented in the sequence defined by the internal table. At this point the system also considers the conditions under which the function module will be processed in the customer namespace. For example, a country or an application can be entered as a condition. These conditions are also referred to as filter values.

Business Add-Ins: Flow of a Program Exit

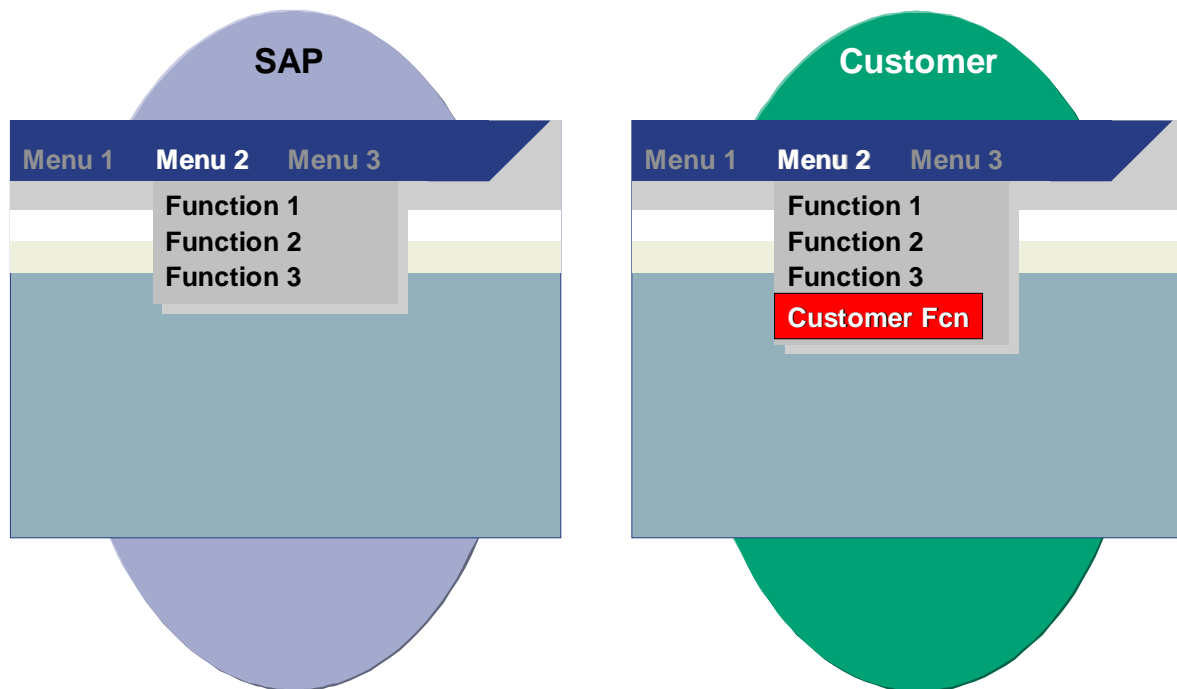
SAP



- This graphic shows the process flow of a program that contains a BAdI call. It enables us to see the possibilities and limitations inherent in Business Add-Ins.
- Not displayed: You must declare a reference variable referring to the BAdI interface in the declaration section.
- An object reference is generated in the first step. This replaces the service class CL_EXITHANDLER provided by SAP. We will discuss the precise syntax later on. This generates the conditions for calling methods of program enhancements.
- When you define a Business Add-In, the system generates a BAdI class, which implements the interface. In call (2), the interface method of the BAdI class is called. The BAdI class searches for all of the active implementations of the Business Add-In and calls the implemented methods.

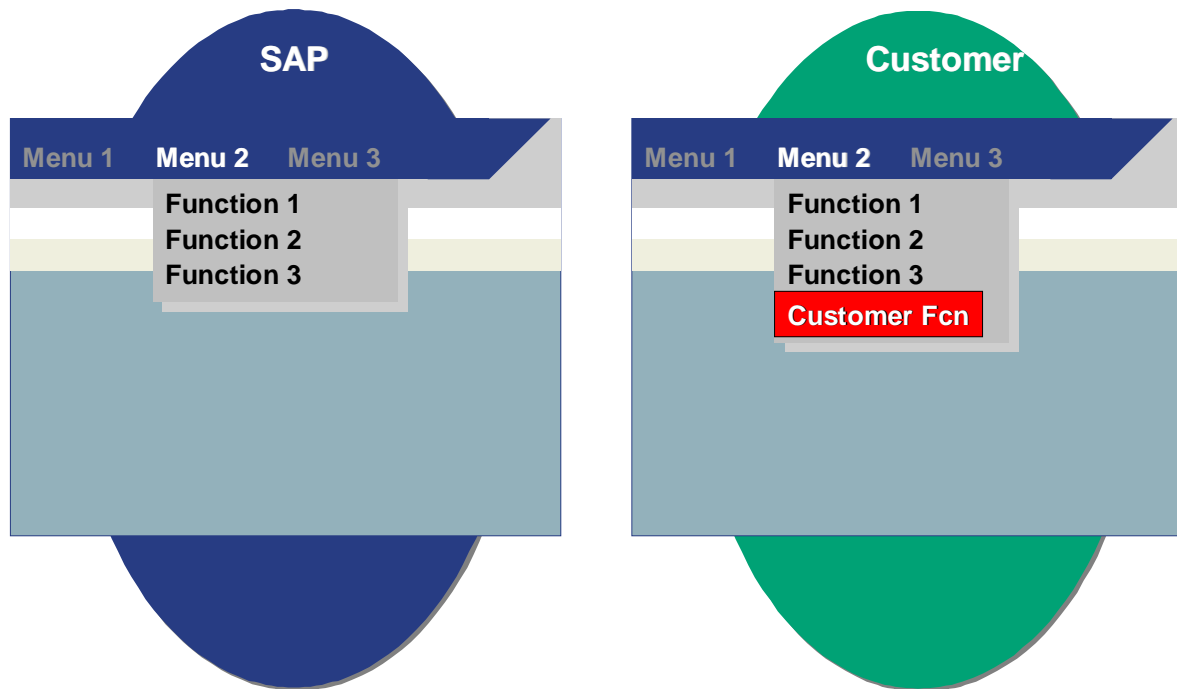
Menu Exits Overview

SAP



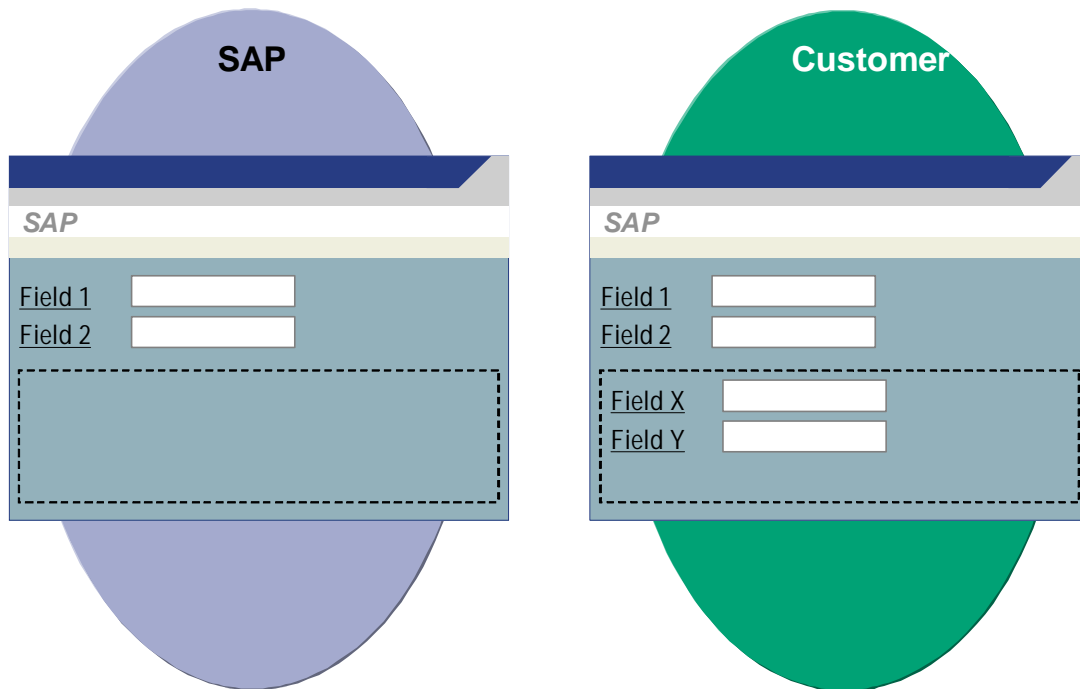
© SAP AG 2009

- Menu exits allow you to attach your own functions to menu options in SAP menus. SAP application programmers reserve certain menu entries in your GUI interface for this. You can specify the entry text yourself.
- Once you activate menu exits, they become visible in the SAP menu. When you choose the corresponding menu option, the system changes to a program exit that contains your customer-specific functions.



© SAP AG 2006

- As with customer exits, you can use menu enhancements with Business Add-Ins. However, the following conditions must be met:
 - The developer of the program you want to enhance must have planned for the enhancement.
 - The menu enhancement must be implemented in a BAdI implementation.

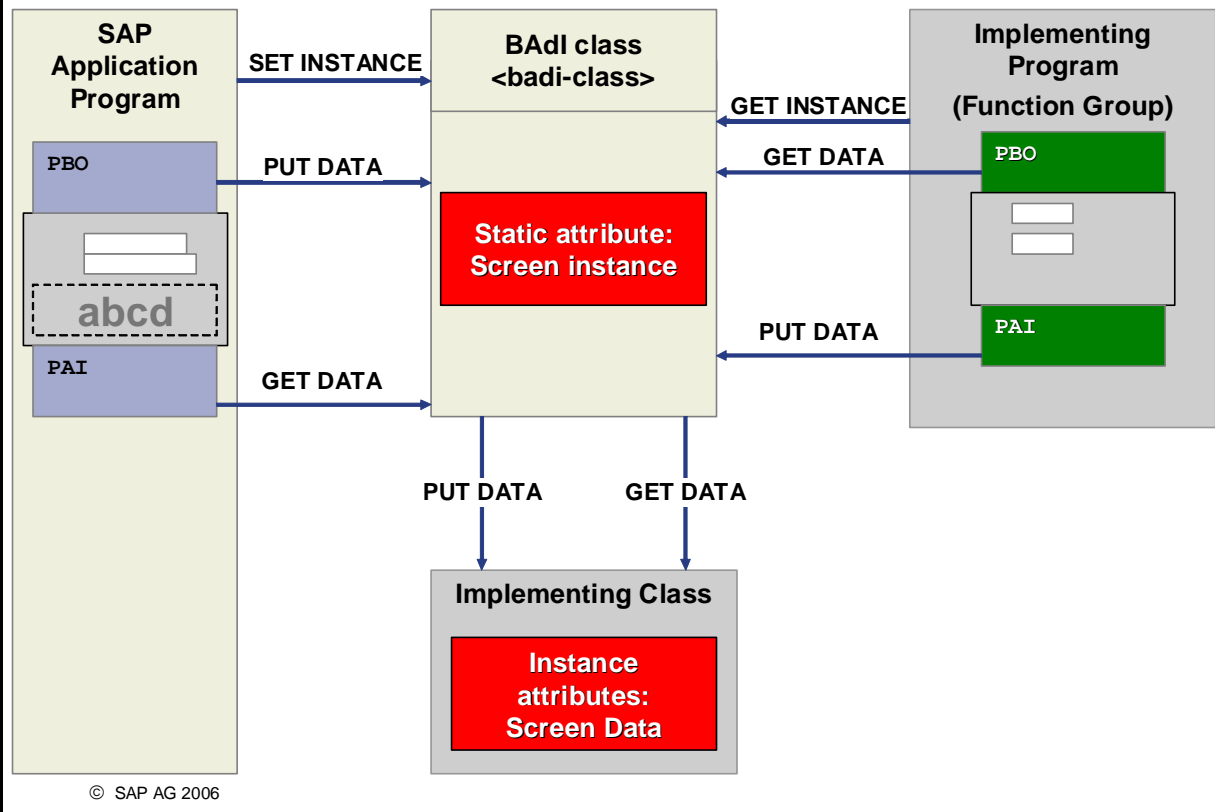


© SAP AG 2009

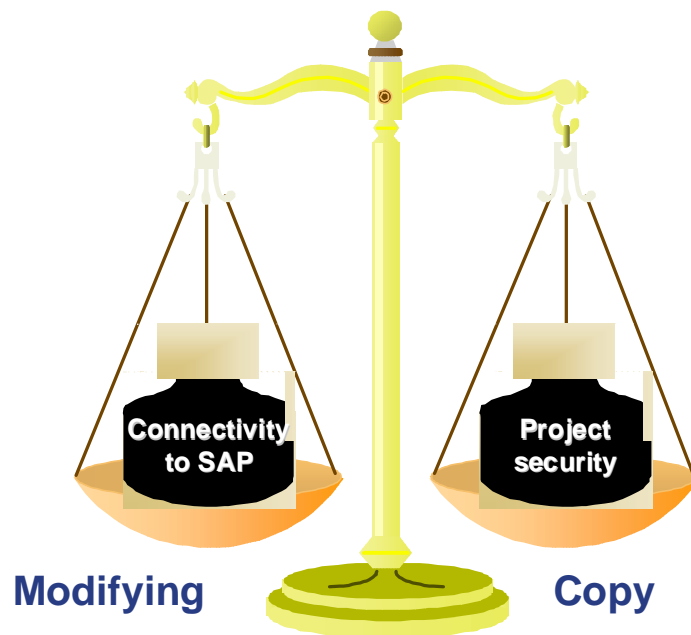
- Screen exits allow you to make use of reserved sections of a main screen (subscreen areas). You can either display additional information in these areas or input data. You define the necessary input and output fields on a customer screen (subscreen).

BAdI Screen Exits: Basic Principles

SAP



- The ABAP virtual machine does not recognize screens bound to classes. Thus, only "classical" programs (of types 1, F, or M) can be used as containers for screens. Screen enhancements must take this into consideration.
- When you create a Business Add-In screen enhancement, the provider reserves a subscreen area on the application program screen, which is then filled with the subscreen of the implementing class (similar to customer exits). However, the application program and the subscreen container program do not communicate directly. They communicate using the generated BAdI class.
- The following slides show this communication process step by step.



© SAP AG 2006

- Modifying has the advantage that your productive Repository objects do not lose their connection to the SAP standard. Copying, on the other hand, has the advantage that no modification adjustment will be necessary for your productive Repository objects during subsequent upgrades.
- Choose copying instead of modifying if
 - You have to make numerous changes to an SAP program
 - Your requirements will not be met by the standard in future R/3 releases.
- During copying, pay attention to a Repository object's environment as well. You should only decide whether to modify or copy after having informed yourself of the consequences for the main program, as well as for all of the includes attached to the main program. The same holds true for function groups and function modules.

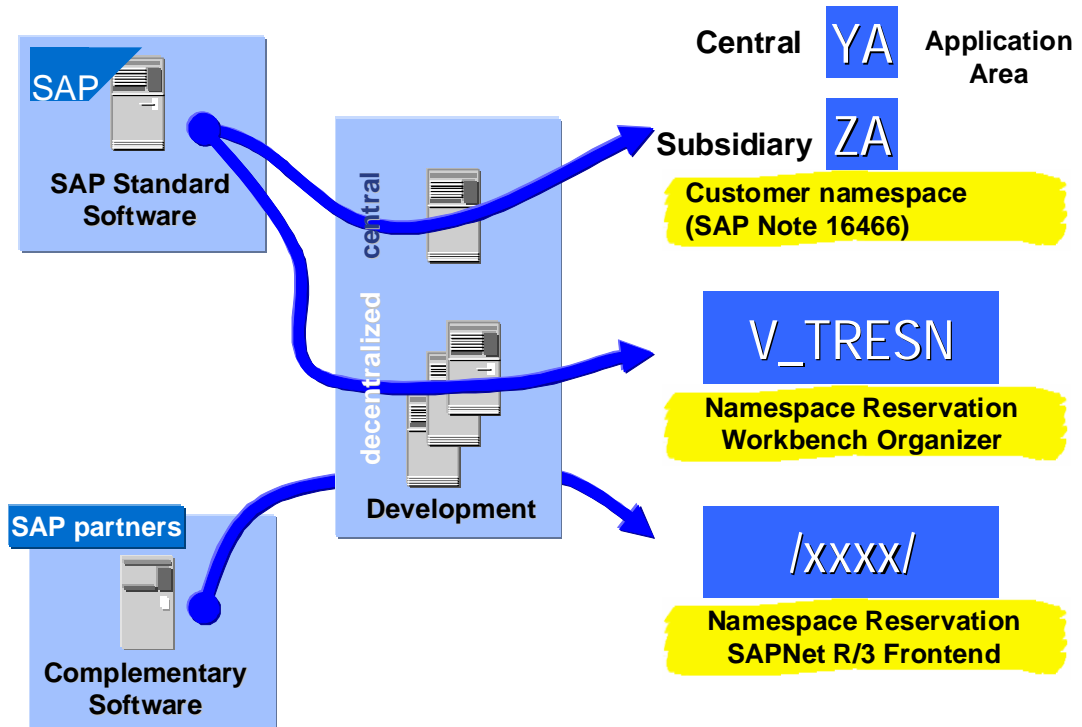
- Questions prior to the development project:
 - Implementation costs?
 - Consequences:
 - Performance
 - Costs during an upgrade
- Alternatives:
 - Customizing
 - Personalization
 - Enhancement techniques:
 - Customer Exits
 - Business Transaction Events
 - Business Add-Ins

© SAP AG 2006

- ABAP development projects can be evaluated according to the following criteria:
 - What will **implementation cost**, measured in manpower (creating the concept, implementation, testing)?
 - What **effect** will the ABAP development project have for
 - Production operation performance?
 - The amount of adjustment at upgrade?
- By calling SAP objects in your own Repository object, you can drastically reduce the amount of effort needed to implement your object. However, any changes that SAP makes to the Repository object you choose to call may make extra adjustment necessary after an upgrade. SAP could conceivably change the user interface of a screen for which you have written a batch input program.

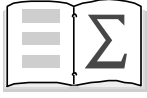
Naming Conventions for Repository Objects

SAP



© SAP AG 2006

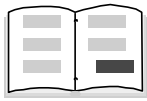
- Naming conventions allow you to avoid naming conflicts and give your Repository objects meaningful names (that can be understood by others).
- The following naming conflicts can occur:
- An SAP Repository object and a customer Repository object conflict
SAP Repository objects and customer Repository objects should be separated from each other by strict adherence to SAP naming conventions. SAP Note 16466 gives you an overview of the current naming conventions for customer Repository objects (usually names that begin with either Y or Z).
- Two customer Repository objects conflict
Naming conflicts can also crop up between customer Repository objects in decentralized development scenarios where more than one development system is being used. You can avoid naming conflicts in this area by reserving a special namespace for development areas within the customer namespace. The Workbench Organizer checks to make sure that you adhere to these conventions by making entries in view V_TRESN.
- Complementary software and customer Repository objects conflict
You can avoid naming conflicts when importing complementary software from SAP partners by reserving special namespaces in SAPNet. In addition, as of Release 4.0, SAP partners can apply in SAPNet for prefixes that they can add to the beginning of their Repository objects' names (For additional information, refer to SAP Notes 84282 and 91032, or the white paper 'Development Namespaces in the R/3 System', order number E:50021723 [English] and D:50021751 [German]).



You are now able to:

- **Describe the course contents**
- **Describe how to proceed when changing the SAP standard**
- **List the advantages and disadvantages of modifications**
- **Name the alternatives to modifications**

© SAP AG 2006



Contents:

- **New Developments in the Enhancement Area as of SAP NetWeaver Application Server 7.0**
- **Field Exits**
- **Append Search Helps**

© SAP AG 2006



At the conclusion of this topic, you will be able to:

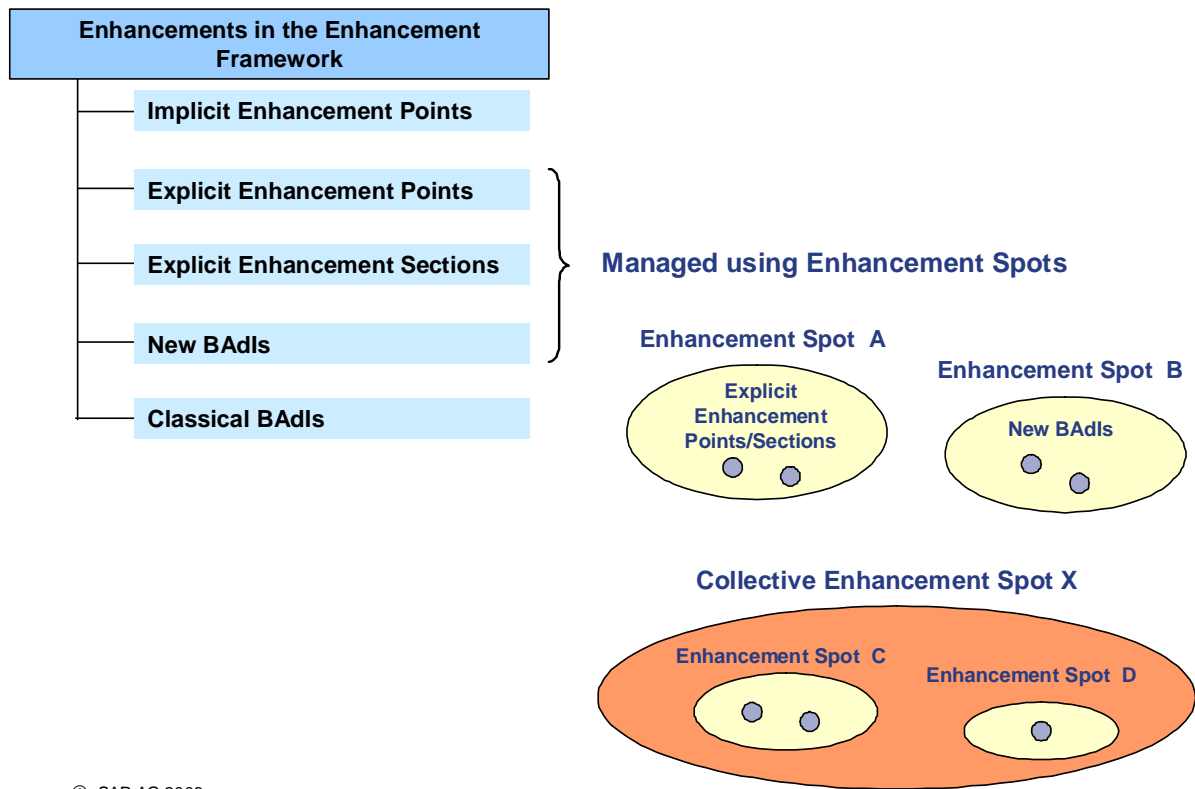
- **List the new developments in the enhancement area as of SAP NetWeaver Application Server 7.0**
- **Explain where implicit enhancement points are found and how you can use them to enhance SAP software**
- **Explain how you can use explicit enhancement points and enhancement sections to enhance SAP software**
- **Explain why SAP introduced the new BAdI technology in Release 7.0 and how new BAdIs are used to enhance SAP software**

© SAP AG 2006

- This section describes the advantages and use of the new enhancement options as well as the new BAdI technology, which have been introduced as of SAP NetWeaver Application Server 7.0.

New Enhancement Concept (Overview)

SAP

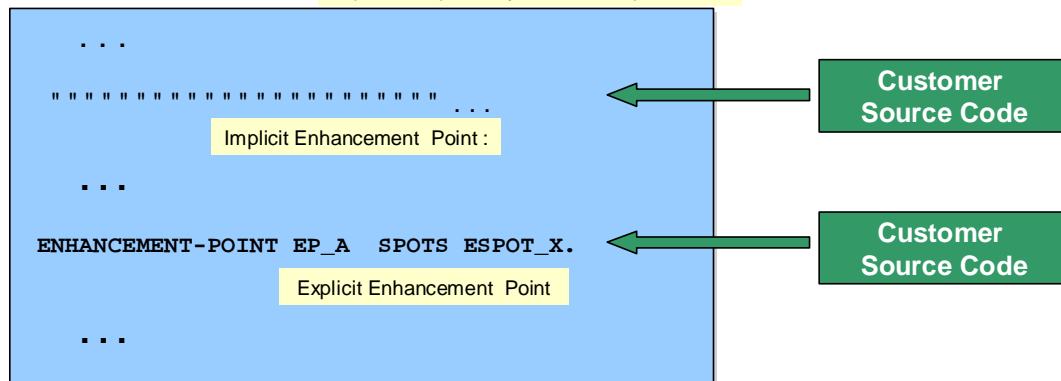


- As of SAP NetWeaver Application Server 7.0, two new enhancement options have been added: Enhancement points and enhancement sections.
- The fact that enhancements using implicit enhancement points - unlike enhancements implemented using previous enhancement technology - require no preparation from SAP is particularly interesting.
- SAP has also introduced a new BAdI technology as of SAP NetWeaver Application Server 7.0, for performance reasons, as well as for other reasons that we will explore later.
- The graphic above displays how enhancement points and sections, as well as BAdIs that have been created using the new technology, are grouped together and managed using enhancement spots. Collected enhancement spots comprise both simple enhancement spots and/or other collected enhancement spots. They serve to semantically bundle enhancement spots.
- Previous BAdIs (classical BAdIs) exist in the system as before. However, SAP will only implement BAdIs using the new technology in future.
- Note that elements of the central SAP Basis cannot be enhanced.
- Documentation about the new enhancement concept is available as follows:
Choose the Information button in the ABAP Editor -> Enter "Enhancement concept" as a search term
-> Glossary Entry: "Enhancement Concept" -> "More"

- Source Code
- Variable or Parameter Declaration

SAP Object

Explicit: Prepared by SAP Developers



- An enhancement point is an option that allows you to add source code, variable declarations and parameter declarations to SAP programs, function modules and classes without having to make a modification.
- Explicit enhancement points are insertion options that are prepared by SAP, while implicit enhancement points are present at particular points in SAP objects by default - that is, without any particular preparation by SAP.

- **At the end of a structure (type) declaration before "END OF ..."**
(to include additional fields)
- **At the beginning and end of**
 - Subprograms
 - Function modules
 - Methods of Local Classes / Global Classes
 (to insert additional functions)
- **At the end of the IMPORTING/EXPORTING/CHANGING declaration block of methods of local classes**
(to include additional interface parameters)
- **In interface definitions of**
 - Function Modules
 - Methods of Global Classes
 (to include additional interface parameters)

...

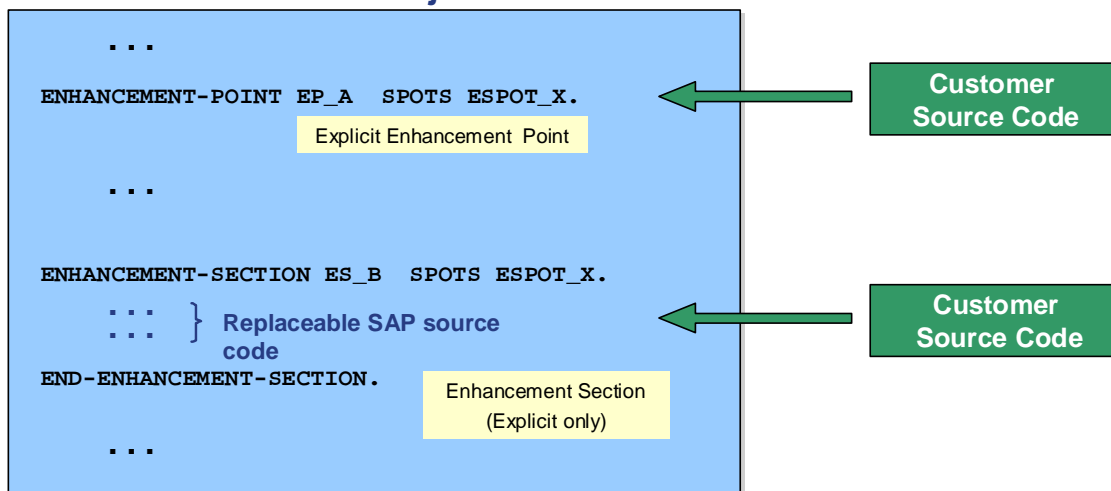
- The mapping above (and following) shows at what points implicit enhancement points are present. To use an implicit enhancement point, you implement an enhancement implementation. Below we describe the procedure for using the various implicit enhancement points.
- How to use implicit enhancement points, which you can use to insert source code (implicit source code plug-ins):
 - Display the SAP object (program, function module, method)
 - Choose the "Enhancement" button in the GUI status
 - Choose the menu option "Edit -> Enhancement Operations -> Show Implicit Enhancement Options"
- This displays the implicit enhancement options
 - Use the editor's context menu to create an enhancement implementation
 - Enter the source code
 - Choose the "Activate" button in the GUI status
 - Note: When you enhance a structure (or structure type) declaration immediately before "END OF ...", you must use the syntax "DATA <additional field> TYPE <type>." !
- Interface enhancement of SAP function modules and methods of global classes:
Open the Function Builder or Class Builder and choose the menu option "Function Module -> Enhance Interface" or "Class -> Enhance" to add a new interface parameter as well as type assignment.
(Interface parameters added in this way are usually optional, and can be addressed in source code enhancements to the corresponding function module or method.

...

- At the end of the **Public/Protected/Private Section** of a local class
(define additional attributes and methods)
- You can define any of the following for global classes
 - Additional attributes
 - Additional methods
- You can define either a
 - Pre method and / or
 - Post method
 for a method of a global class. (Automatic execution when method starts/ends)
 Alternative: Define an overwrite method (replaces the SAP method)
- At the end of the **IMPLEMENTATION** block of a local class
(to implement additional declared methods)
- At the end of Includes
(To implement additional functions)

- Defining additional attributes / methods of global classes:
 - In the Class Builder, choose the menu option "Class->Enhance" to define additional attributes and methods.
 - Double-click on an additional method to open the method editor in order to implement it.
 - Such additional attributes or methods can be addressed in source code enhancements of methods of the global class.
- Defining a pre/post/overwrite method for the method of a global class:
 - Open change mode in the Class Builder by choosing the menu option "Class -> Enhance"
 - Select the desired SAP method
 - Choose the menu option "Edit -> Enhancement Operations -> Insert (Pre/Post) Method" or "-> Add Overwrite Method"
 - Click the new button in the column ("Pre (Post/Overwrite) Exit" to implement the corresponding method.
 - You can define a pre and/or post method for each SAP method.
 - As an alternative, you can create an overwrite method, which replaces the SAP method. Such methods are called automatically at the described points in the SAP method. They are instance methods of an automatically generated local class, and possess an attribute called `CORE_OBJECT`, which is a reference to the current instance of the SAP application.

SAP Object



Enhancement Section :

Option to replace SAP source code without making a modification
In SAP programs, SAP function modules, and SAP methods

- An explicit enhancement point is an option provided in advance by SAP to allow you to enhance the SAP source code without making a modification. An explicit enhancement section is an option provided in advance by SAP that allows you to replace the SAP source code without making a modification. Implicit enhancement sections do not exist.
- Explicit enhancement points and sections are always embedded in enhancement spots.
- Explicit enhancement points and sections that enable source code changes or replacement are called "dynamic". Explicit enhancement points and sections that enable declaration enhancement or replacement are called "static".
- To use explicit enhancement points and sections, you implement an enhancement implementation (an implementation of the higher-level enhancement spot). The following step sequence describes the procedure for using explicit enhancement points and enhancement sections:
 - Display the SAP object (program, function module, method)
 - Find the desired enhancement point(s)/sections(s)
 - Choose the "Enhancement" button in the GUI status
 - Create the enhancement implementation for the enhancement point/enhancement section using the context menu
 - Specify the enhancement implementation name
 - Enter the source code
 - Choose the "Activate Enhancements" button in the GUI status.

Why new BAdI Technology?

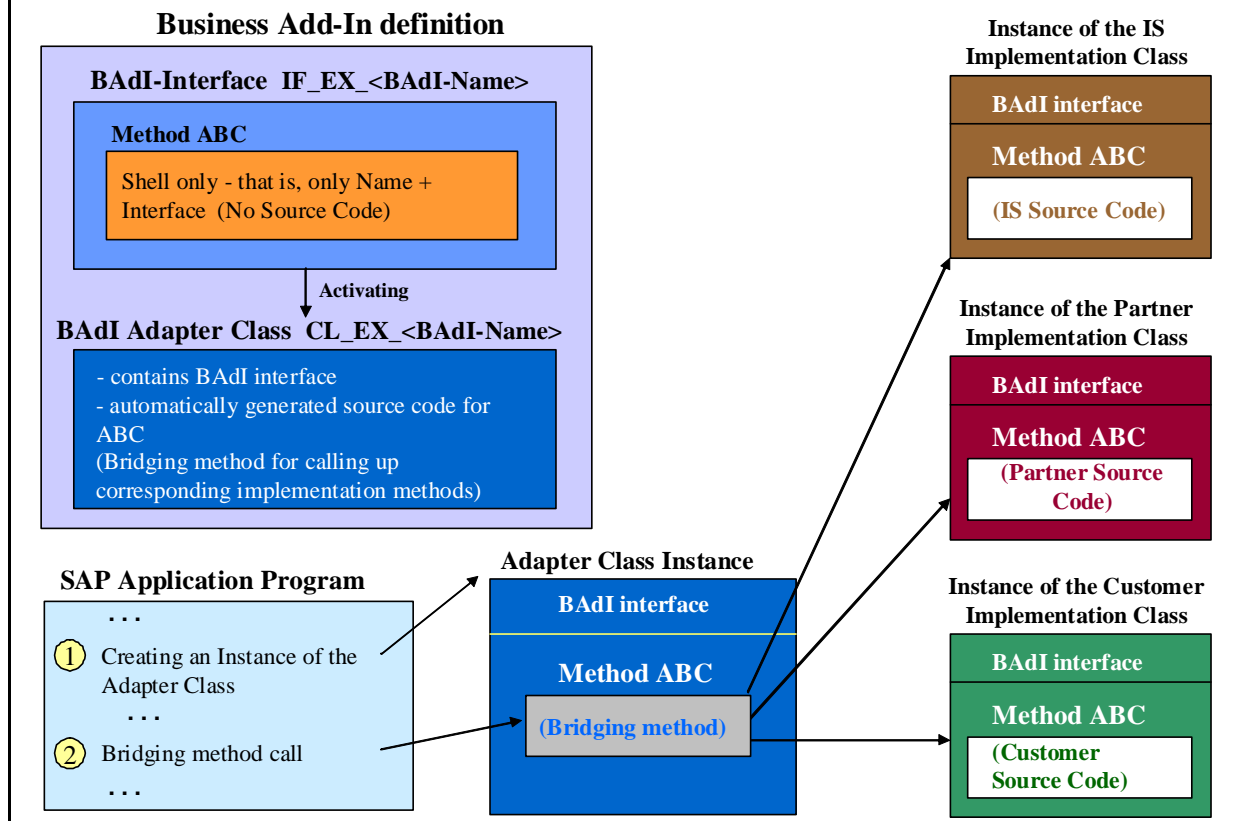
- **Performance Improvement**
- **Implementation of Additional Functions**
 - **Enhanced Filter Concept**
 - **Option to Inherit Attributes from Sample Implementation Classes (selective method redefinition!)**
 - ...
- **Integration into the New Enhancement Framework (together with Enhancement Points and Enhancement Sections)**
- **Integration into the Switch Framework**

© SAP AG 2006

- SAP introduced the new BAdI technology in SAP NetWeaver Application Server 7.0 primarily to improve performance and enhance functions. The individual reasons are listed above.
- The integration of the new BAdIs as well as enhancement points and enhancement sections into the enhancement framework enables you to use the new enhancement options within a single, unified tool.
- By integrating the new enhancement technology into the Switch Framework, SAP enables you to use switches activate and deactivate the BAdI implementations implemented by Industry Solutions (see below).
- Older BAdIs, which were created using the classical technology, remain in the system. In future, new enhancements will only be implemented using the new BAdI technology or using explicit enhancement points and enhancement sections.

Classical BAdIs (Architecture)

SAP



- In the classical BAdI technology, the BAdI adapter class is automatically generated when you define the BAdI or the BAdI interface.
- At runtime, an instance of the adapter class is created in the SAP application program, and the interface method(s) are called from the adapter class instance. The interface methods then call the (identically named) methods of active implementations sequentially.

SAP Application Program

```
...  
DATA r_exit TYPE REF TO <badi-interface> .  
...
```

```
CALL METHOD cl_exithandler=>get_instance
```

```
CHANGING
```

```
instance = r_exit .
```

Create the instance of the adapter class

```
...
```

```
CALL METHOD r_exit->abc
```

```
EXPORTING
```

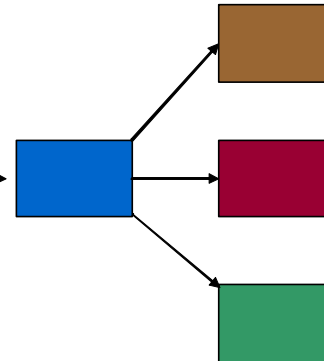
```
...
```

```
IMPORTING
```

```
...
```

```
...
```

Calling the bridging method (BAdI call)

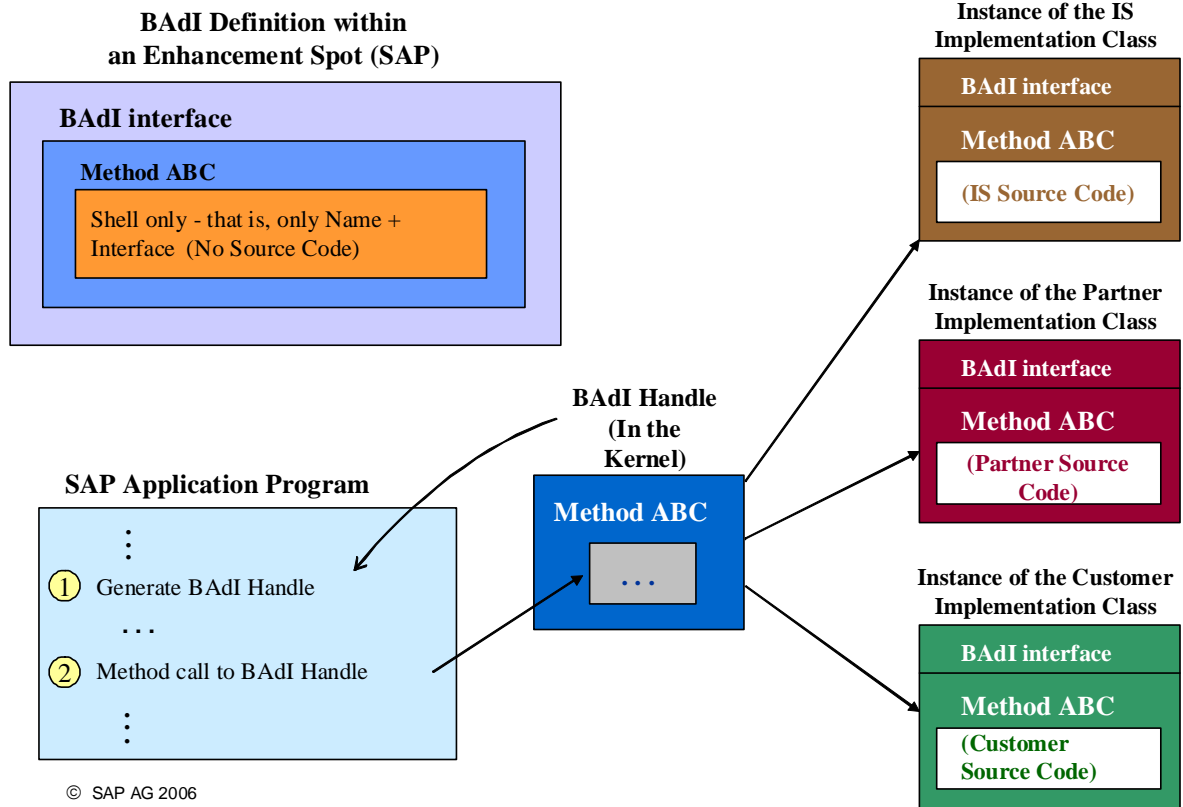


© SAP AG 2009

- The graphic above displays the call syntax of classical BAdIs in SAP Programs.

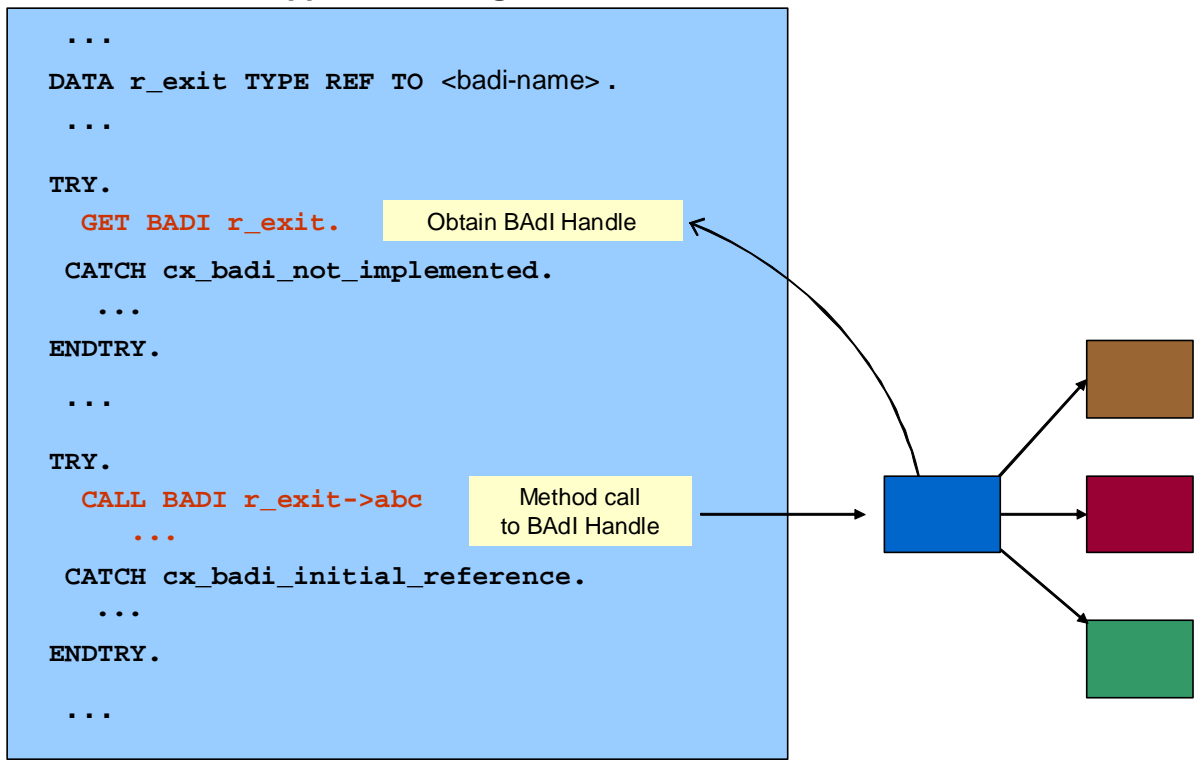
New BAdIs (Architecture)

SAP



- The new BAdI technology works in the same way as the old technology. However, adapter classes are no longer required, which saves the SAP application program from having to instantiate them. Instead, at the application program's runtime, the system generates a BAdI handle in the kernel, which performs the same function as the adapter class, but calls the available implementation methods much more efficiently.

SAP Application Program



- The graphic above displays the call syntax of new BAdIs in SAP Programs.
- If GET BADI finds no active implementation of the BAdI, it triggers the exception `cx_badi_not_implemented`.
- If the handle reference is initial for CALL BADI (for example, because GET BADI failed), the exception `cx_badi_initial_reference` is triggered.

Free Search: SE84

⇒ List of freely selected BAdIs or enhancement spots

Application-related search: SE81 -> SE84

⇒ List of application-related BAdIs or Enhancement Spots

Program-related search:

1. Global search for 'GET BADI'
(you may also search in called function modules and methods)
2. Double-click on the reference variable to navigate to the variable definition
3. Double-click on the BAdI name to navigate to the display for the corresponding enhancement spot

© SAP AG 2006

- The description above displays the procedure for searching for BAdIs.

Steps to Create a BAdI Implementation (BAdI Use):

- 1: Display the corresponding enhancement spot**
- 2. Choose "Implement Enhancement Spot" (F6) and create the enhancement implementation**
- 3. Enter a name for the enhancement implementation**
- 4. Enter a name for the BAdI Implementation(s)**
- 5. Maintain the attributes for the BAdI Implementation(s)**
- 6. In the BAdI's navigation area, click on the corresponding component "Implementing class"**
- 7. Enter the name of the implementing class and choose the "Change" button**
(This can be inherited from sample classes, or you can copy the example classes)
- 8. Double-click on the method(s) to implement/adjust them**
- 9. Activate the method and enhancement implementation**

- To use a BAdI that you have found, you must implement an enhancement implementation (an implementation of the higher-level enhancement spot). This implements a BAdI implementation for each BAdI in the enhancement spot.
- The directions for searching for and using BAdIs relate to program exits (the most common exit type). You can search for and implement menu and screen exits in exactly the same way as the classical BAdI technique, or using the method described above. To obtain the BAdI handle for data transport, you merely need to use the statement GET BADI instead of the method GET_INSTANCE_FOR_SUBSCREENS in the PBO of the customer subscreen screen for screen exits.

Business Add-In

Interface IF_... with Method ABC

Filter:

COUNTRY

(with filter values from value table T005
of the data element / domain LAND1
-> DE, GB, US, ...)

Example

You must specify Filter
values
in implementations!

Implementations
for the filter value. . .

'DE'

ABC-Implementation
for 'DE'

'GB'

ABC-Implementation
for 'GB'

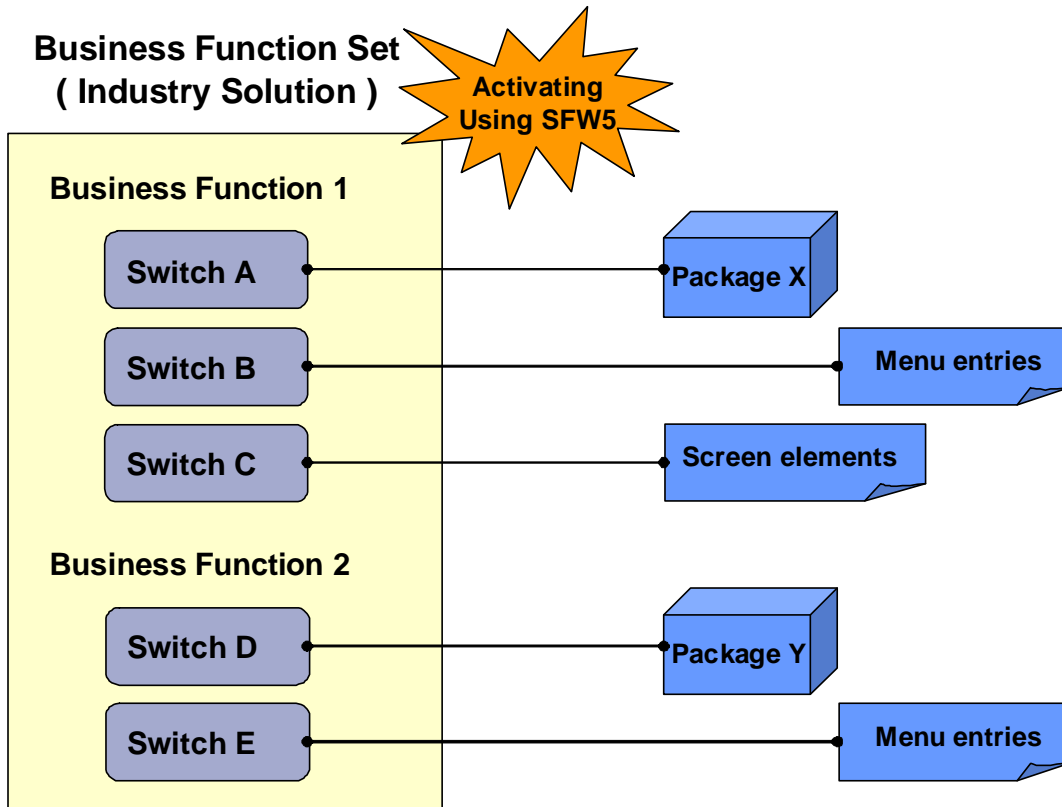
'US'

ABC-Implementation
for 'US'

SAP Application Program

```
...
DATA r_exit TYPE REF TO <badi-name>.
...
GET BADI r_exit.
  FILTERS
    COUNTRY = <current country>. (e.g. 'GB')
  ...
CALL BADI r_exit->abc
...
```

- The graphic above displays the concept of filter-dependent BAdIs, which corresponds to the classical BAdI concept.
- However, the functions have been enhanced. Numerical filters can now also be used.
- Also, you can now specify for implementations not only individual filter values, but also filter conditions, by using the operators <>, >, <, >=, <=, CP, NP.
- You can also define many filters for a given BAdI.



- The Switch Framework was conceived to give SAP customers the option to activate Industry Solutions or Enterprise Add-Ons in order to use them, rather than having to install them.
- The graphic above displays the relationship between individual repository objects, switches, business functions, and business function sets (Industry Solutions). You can use Transaction SFW5 to activate Industry Solutions or Enterprise Add-Ons. Only then are the corresponding repository objects available.
- This has the following effect in conjunction with the new enhancement options:
If a switchable package contains enhancement implementations, they only become active in the system when you activate the corresponding business function.

Using implicit enhancement points to enhance SAP source code

1. Analyze the source code of the SAP Program `SAPD70AW_##_ISP`.
(where ## is your two-digit group number.)
2. Use the corresponding implicit enhancement point to add the fields *cityfrom* and *cityto* to the definition and the structural variable *mystr* (Use the data element `S_CITY` for field type assignment in each case).
Enter `ZBC425_##_ISP` as the name of your enhancement implementation.
3. Assign the values of the formal parameter *f_str* to the additional fields *cityfrom* and *cityto* using the implicit enhancement point in the subprogram before the output of the fields *carrid* and *connid*.
4. Use the corresponding implicit enhancement point in the subprogram after the output of the fields *carrid* and *connid* to also output *fldate*, *cityfrom* and *cityto*.

Solution: Refer to the procedure in the course materials.

Exercise: Explicit Enhancement Points and Enhancement Sections

SAP

Using explicit enhancement points and enhancement sections to enhance or replace SAP source code

1. Analyze the source code of the SAP Program SAPD70AW_##_ESP.
(where ## is your two-digit group number.)
2. Use the explicit enhancement point D70AW_##_EP1 to also output the fields *distance* and *distid*.
Enter ZBC425_##_ESP as the name of your enhancement implementation.
3. Use the explicit enhancement section D70AW_##_ES1 to output a different text than the text provided by SAP.

Solution: Refer to the procedure in the course materials.

Searching for New BAdIs and Using them to Enhance Functions

1. Search for a BAdI program exit in the SAP program SAPD70AW_##_BADI.
(where ## is your two-digit group number.)
2. Display the higher level enhancement spot and the BAdI definition.
3. Create an enhancement implementation (for the spot) with a BAdI implementation (for the BAdI).

Name of the enhancement implementation: ZBC425_##_ESPOT_BADI

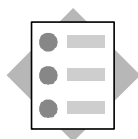
Name of the BAdI implementation: ZBC425_##_BADI

4. Implement the BAdI implementation so that additional fields of the internal table are output.

Note: You can either

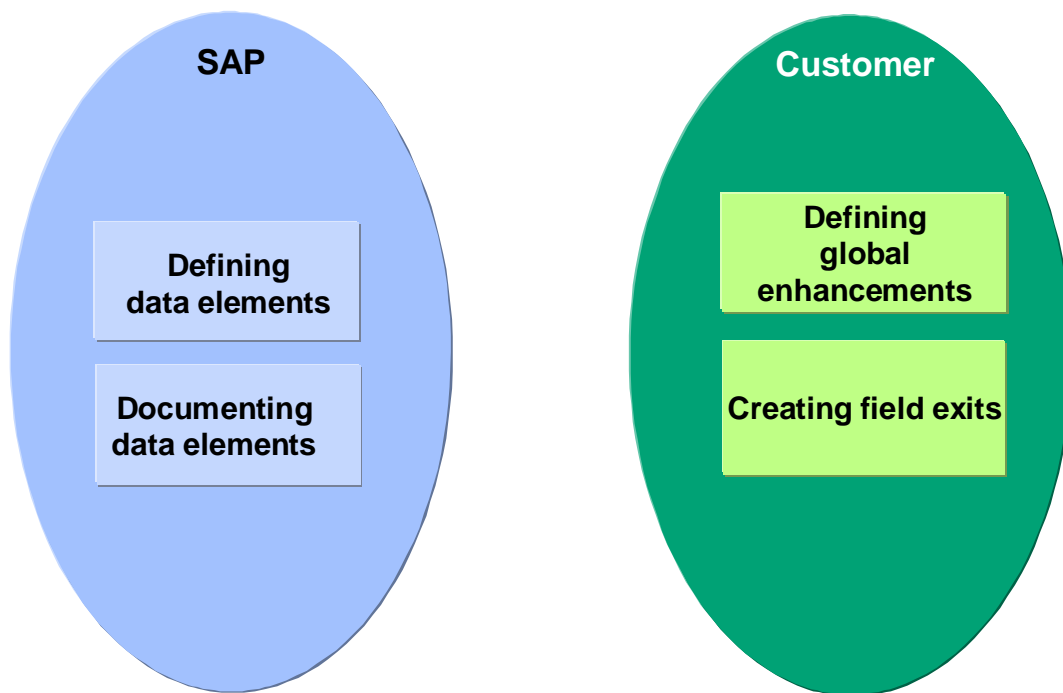
- Copy the sample implementation class to your implementation class and adjust the copied methods, or
- Create an empty implementation class and implement the method yourself.

Solution: Refer to the procedure in the course materials.



At the conclusion of this topic, you will be able to:

- **Use field exits to apply checks to a screen field**

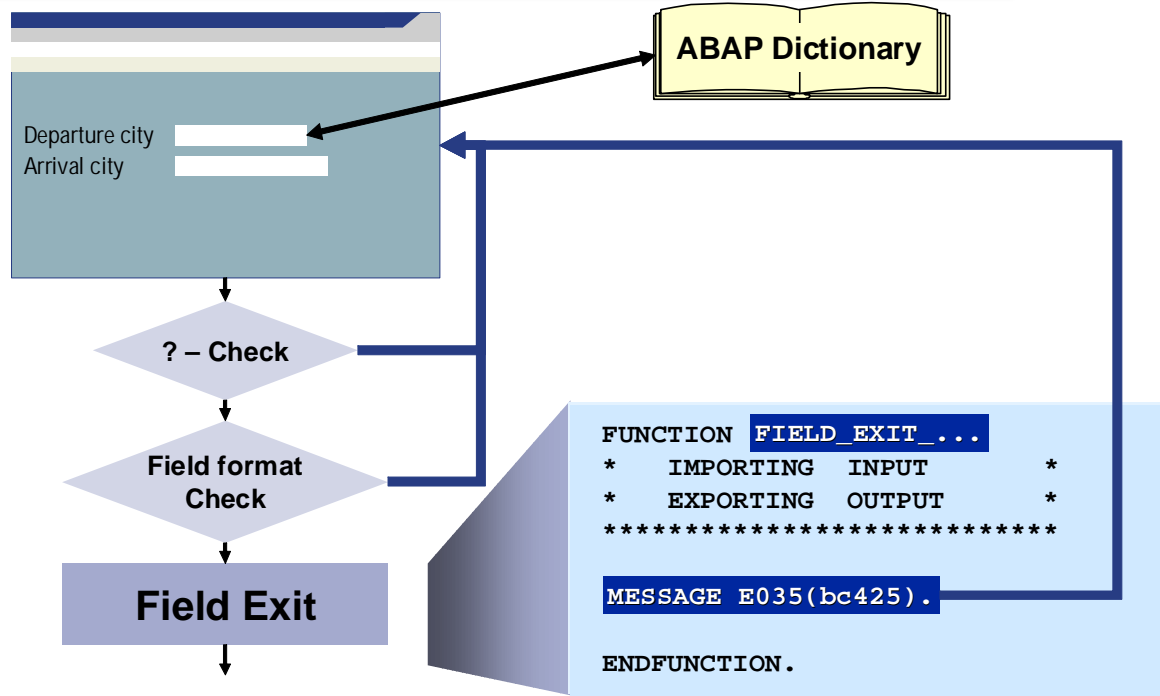


© SAP AG 2009

- SAP application programmers define keywords in different lengths and a short description for each data element.
- You create field exits in *Project Management*. Field exits are processed when the user leaves a screen that contains a field that refers to a data element containing a field exit.

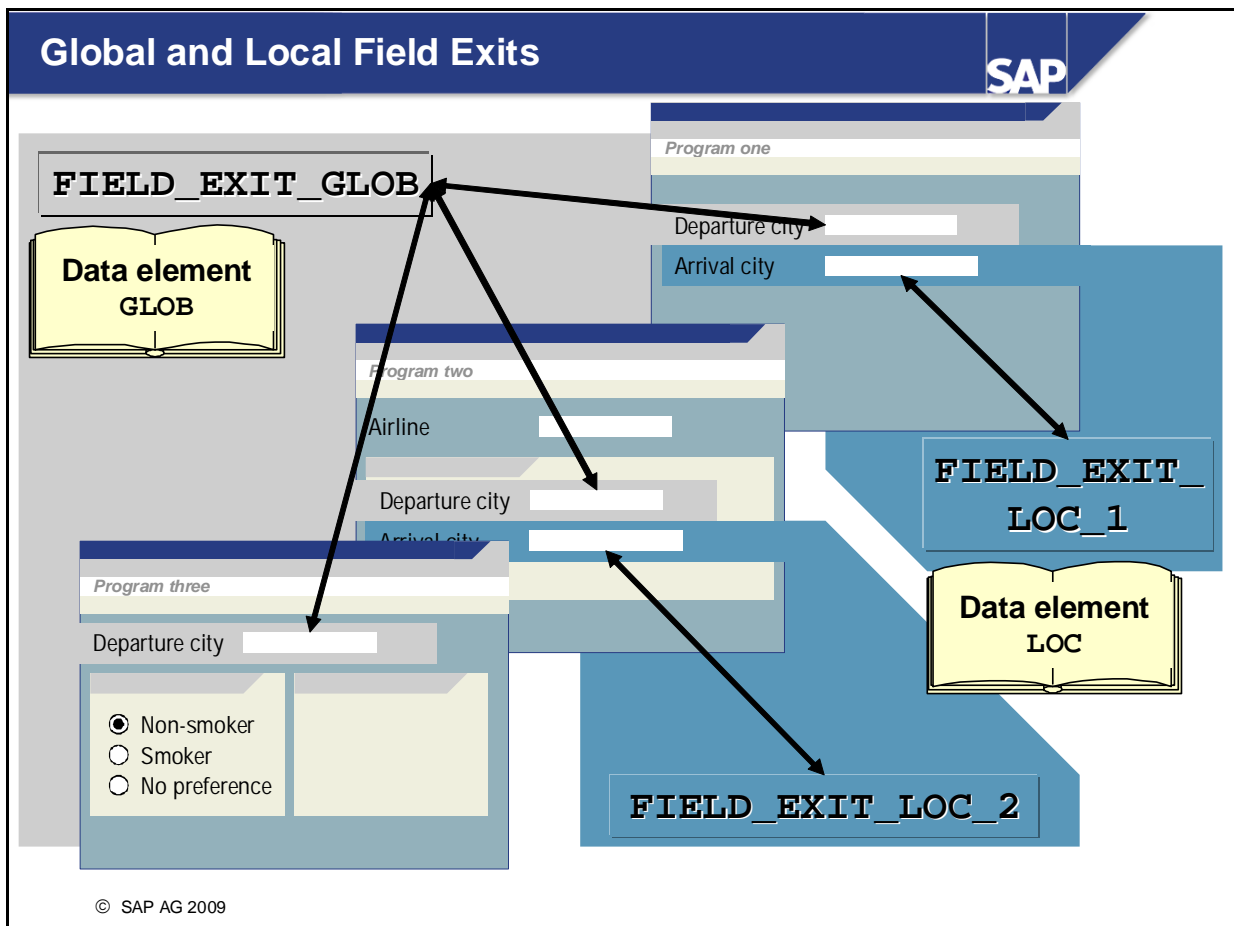
Field exits

SAP



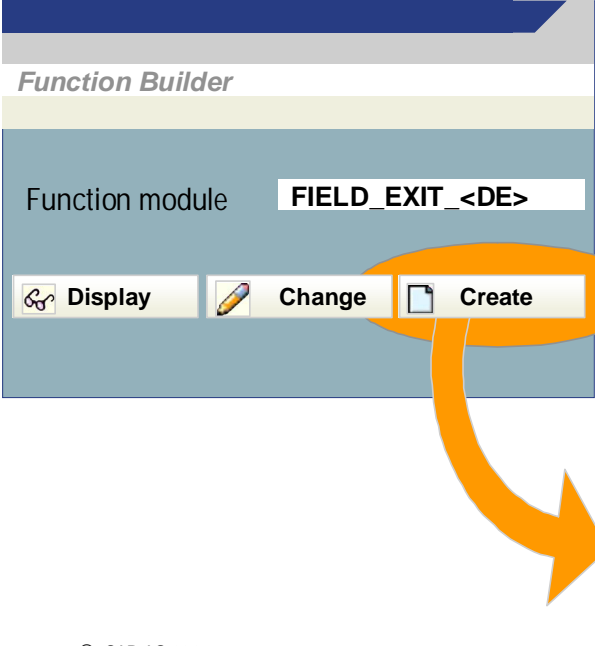
© SAP AG 2009

- SAP makes it possible for you to create a field exit for every input-ready screen field that has been created with reference to the ABAP Dictionary. The additional program logic is stored in a function module and is executed at a specific point at the PAI event.
- The graphic clarifies the sequence of processing: Before the modules defined in the screen's PAI are executed, the system executes checks - first, the system checks if all obligatory fields have been filled. If a required field is empty, the screen is shown again.
- The system then checks that data has been entered in the correct format.
- Any defined field exits are executed next. For example, by sending an error message you can have the screen sent again.
- Once all the field exits have been checked, the screen is processed as normal.
 - Field transport
 - Foreign key check
 - Processing the screen's PAI module.



- Field exits take you from a screen field with a data element reference to a function module. Field exits can be either global or local.
- **Global field exits** are not limited to a particular screen. If a global exit's data element is used on several screens, the system goes to the function module for all these screens after activating the field exit. Here you can, for example, edit the contents, force a new entry to be made by outputting an error message, or prohibit certain users from proceeding further.
- **Local field exits** are valid for one screen only. If you assign a screen from a specific program to a field exit, then the system will go to the appropriate function module from this screen once the exit has been activated.
- You can either create one global field exit or up to 36 local field exits for a data element, but not both.
- Each exit number refers to a different function module. Field exit function modules adhere to the following naming convention:
 - Prefix: FIELD_EXIT_
 - Name: <Data element>
 - Suffix (for local field exit): _0 to _9, _A to _Z

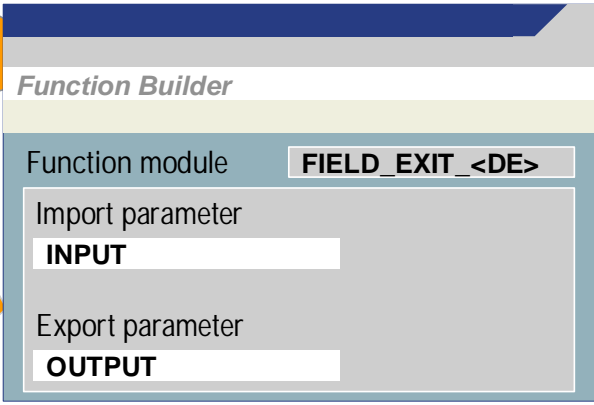
- Start program RSMODPRF



- Field exit -> Create

Function module **FIELD_EXIT_<DE>**

Display Change Create



© SAP AG 2009

- To create field exits, choose **Utilities** in the ABAP Workbench. Choose **Enhancements** and then **Project Management** to edit field exits and to implement customer exits.
- With *Goto -> Global enhancements -> Field exits* you start the transaction for maintaining field exits. To create a new enhancement, choose the menu path, *Text Enhancements -> Create*.
- Enter the name of the data element to which your screen field refers in the modal dialog box. The Function Builder is started. The system specifies the name of the field exit. Do not change this name. Create the function module in a customer function group.
- The function module interface is fixed and cannot be changed. The function module has an import parameter INPUT and export parameter OUTPUT. The contents of the screen field are stored in parameter INPUT. The contents of OUTPUT are returned in the screen field when you leave the function module.

Source code

```
FUNCTION field_exit_<DE>.  
...  
output = input.  
ENDFUNCTION.
```

● These statements are not allowed in field exits:

- BREAK-POINT
- CALL ..., SUBMIT
- COMMIT WORK / ROLLBACK WORK
- STOP, REJECT
- Message I, Message W



© SAP AG 2002

- In the source text, export parameter OUTPUT of the function module must be assigned a value that is transported back to the screen field. Otherwise the screen field would be initial after executing the field exit.
- The following ABAP statements are not allowed in field exit function modules:
 - CALL SCREEN, CALL DIALOG, CALL TRANSACTION, SUBMIT
 - COMMIT WORK, ROLLBACK WORK
 - COMMUNICATION RECEIVE
 - EXIT FROM STEP-LOOP
 - MESSAGE I, MESSAGE W
 - STOP, REJECT
- When you debug a screen that is referenced by a field exit, the field exit code is ignored by the debugger. As with any normal function module, you can, however, debug the field exit code in the Function Builder's test environment.

Field exits for data elements

Field exit	Program name	Screen
1	<program_name>	0100

Function Builder

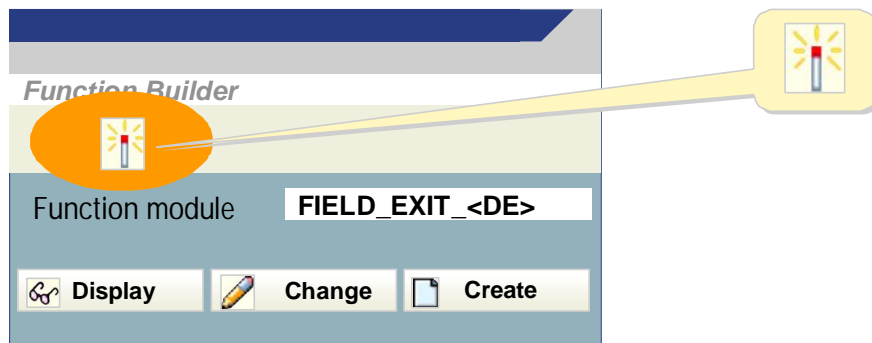
Function module: **FIELD_EXIT_<DE>_1**

Buttons: Display, Change, Create

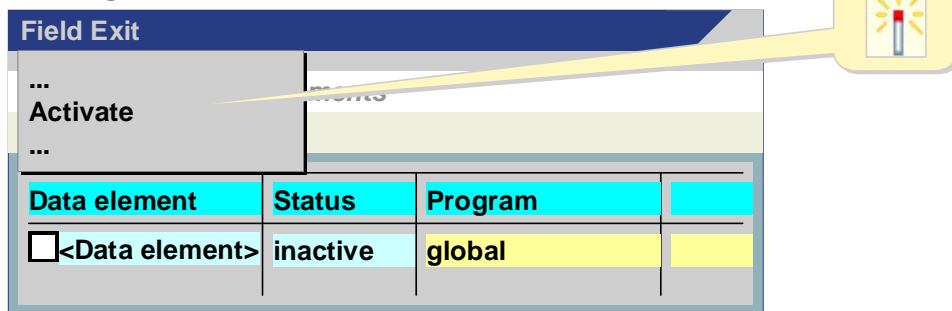
© SAP AG 2002

- You can create local field exits that relate to a specific screen. Create a local field exit. A global field exit must already exist. Edit the local field exit based on the global field exit.
- You can create up to 36 local field exits, each of which carries a unique suffix. The system proposes a name for the function module; you should use this name.
- Defining local field exits means that the function module of the global field exits initially created are no longer used. For technical reasons, however, you must not delete the global function module. The field exits in the system would be deleted if you deleted the global function module of the field exit from the list.

- **Activating function modules**

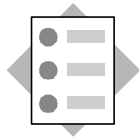


- **Activating field exits**



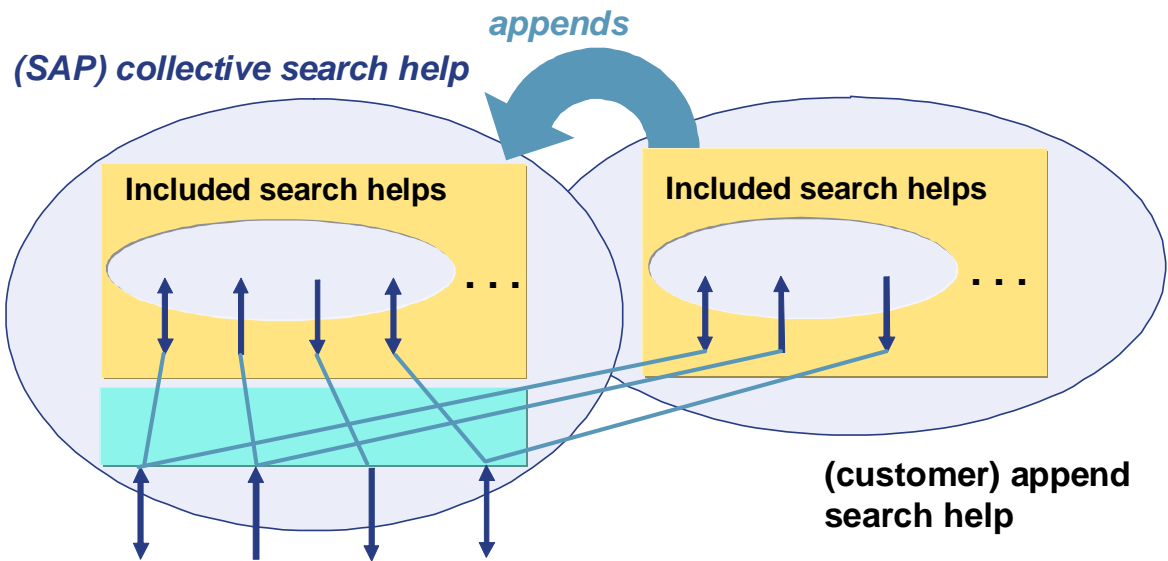
© SAP AG 2002

- You must activate the field exit as well as the function module. Also note that field exits are only taken into account during screen execution if the R/3 profile parameter **abap/fieldexit** = **YES** has been set for all application servers. (This profile parameter is set to **NO** by default).
- If you declare field exits for multiple screen fields, you have **no control** over the order in which they are processed. In particular, you cannot access the contents of other screen fields in a field exit.
- Also read note 29377 about field exits.



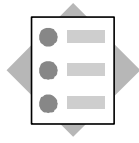
At the conclusion of this topic, you will be able to:

- **Create search helps to define your own search paths**
- **Create search helps to hide the search paths delivered with the standard system**



© SAP AG 2002

- The set of search paths that are meaningful for an object greatly depends on the particular circumstances of the SAP customer. The customer often would like to enhance the standard SAP collective search helps with his own elementary search helps. Release 4.6 provides an append technique that permits the enhancement of collective search helps without modifications.
- An **append search help** is a collective search help that is assigned to another collective search help (its appending object) and that enhances it with the search helps it includes. The append search help uses the interface of its appending objects.
- The append search help lies in the customer namespace. Normally the search helps included in the append search help are also created by the customer and lie in the customer's namespace. However, the required elementary search help might already be provided by SAP, in which case, the customer only has to add it to his own append search help.
- Append search helps are used with SAP to improve component separation. Some SAP collective search helps therefore already have one or more append search helps in the standard search help. Customer enhancements should always be made by creating a separate append search help.
- SAP collective search helps often contain elementary search helps that are not required by all customers. The search helps you do not need can be hidden using an append search help. To do this, the corresponding search help must be included in the append search help and the *hidden* flag must be set.



At the conclusion of this topic, you will be able to:

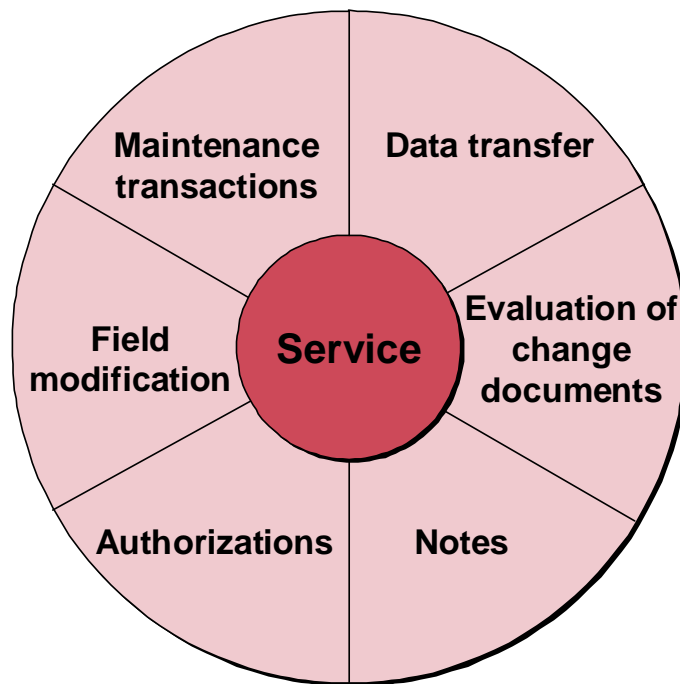
- **Describe the enhancement options offered by the Business Data Toolset**

- **Definition:**
Toolset for master data and simple transaction data
- **Design goals:**
 - Extensibility
 - Configurability
 - Divisibility
 - Alternative user interfaces
 - Usability
 - Faster development
 - Generic object services



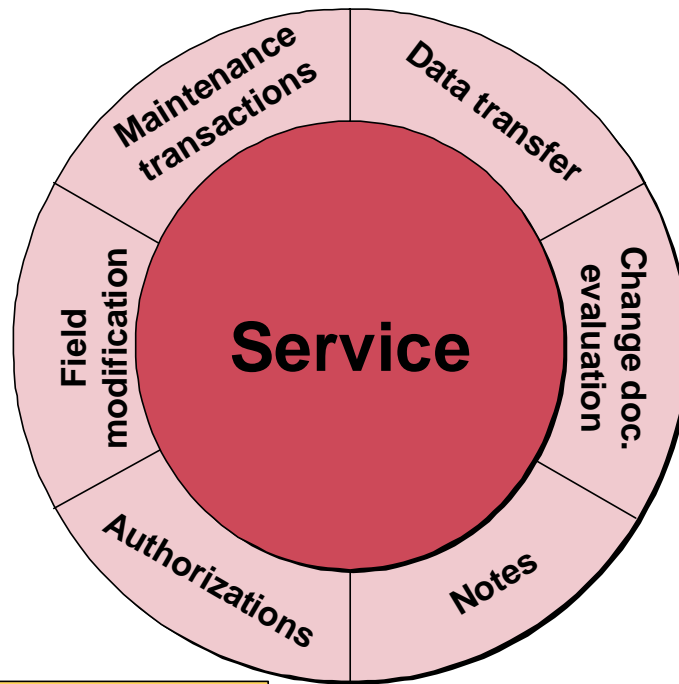
© SAP AG 2002

- The Business Data Toolset (BDT) - formerly known as master data administration or business partner administration - is the main administration tool for maintaining master data and simple transaction data, either using dialogs, direct input, or function modules.
- The BDT also provides generic objects services for consistent recurring requests – such as changing document lists, field modifications, and the deletion program. The BDT manages generic parts of these objects and the objects themselves and calls the applications using predefined interfaces (control tables and events) – such as writing to and reading from the application tables in the database.
- The BDT is used at SAP to maintain different application objects. Development partners and customers use BDT interfaces to enhance these objects **without modifying the source code**.
- In addition, the BDT enables you to use alternative user interfaces by separating the UI from the programming logic. Generic services like direct input and field administration are provided.



© SAP AG 2002

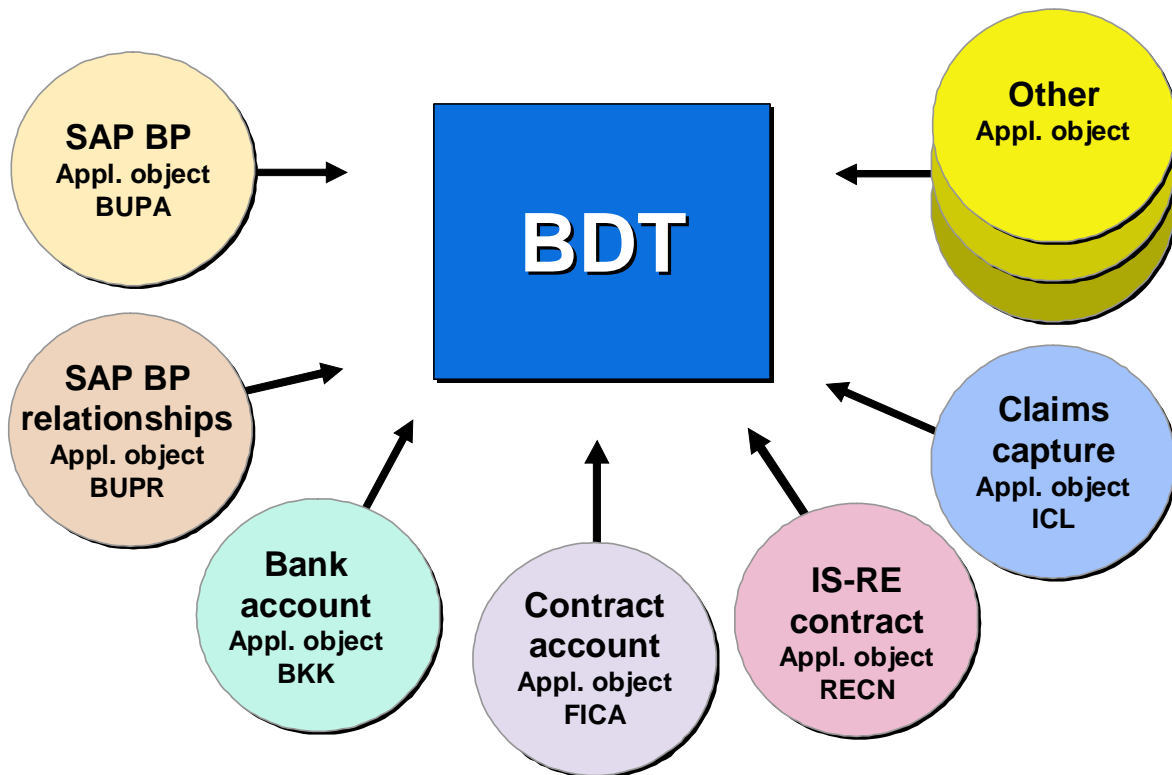
- You need the same functions over and over again in both development and implementation projects. The SAP Web Application Server provides some service functions, but most of these functions still need to be developed.
- Field modification, change document evaluation, and screen checks from the ABAP Dictionary are examples of such services.



**Maintenance performed centrally:
requires fewer development
resources**

© SAP AG 2002

- Since the BDT controls dialog processes, the applications need only implement business functions. The BDT also provides services in which applications can be integrated. These factors considerably reduce the development time.
- The applications lose a little of their individual character but in return gain the benefits of reduced object maintenance, standardized dialogs, generic Object Services, and more rapid development.



© SAP AG 2002

■ The following are just some examples of objects developed using the BDT:

■ **Central Business Partner**

- Partner maintenance
- Relationship maintenance

■ **Contract Accounts Receivable and Payable**

- Contract account

■ **IBU Banking**

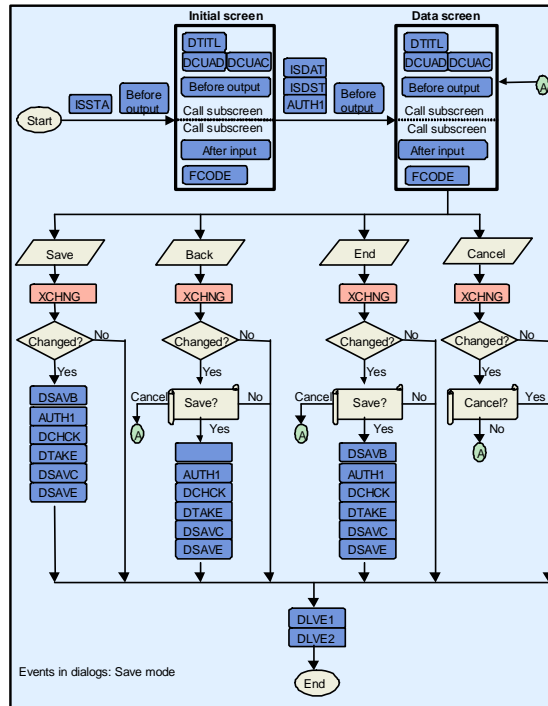
- Bank account
- Standing order
- Financial product
- Financial conditions
- Risk object
- Variable transactions

■ **IBU Insurance**

- Insurance: Claims
- Insurance: Loss event
- Commissions: Remuneration agreement

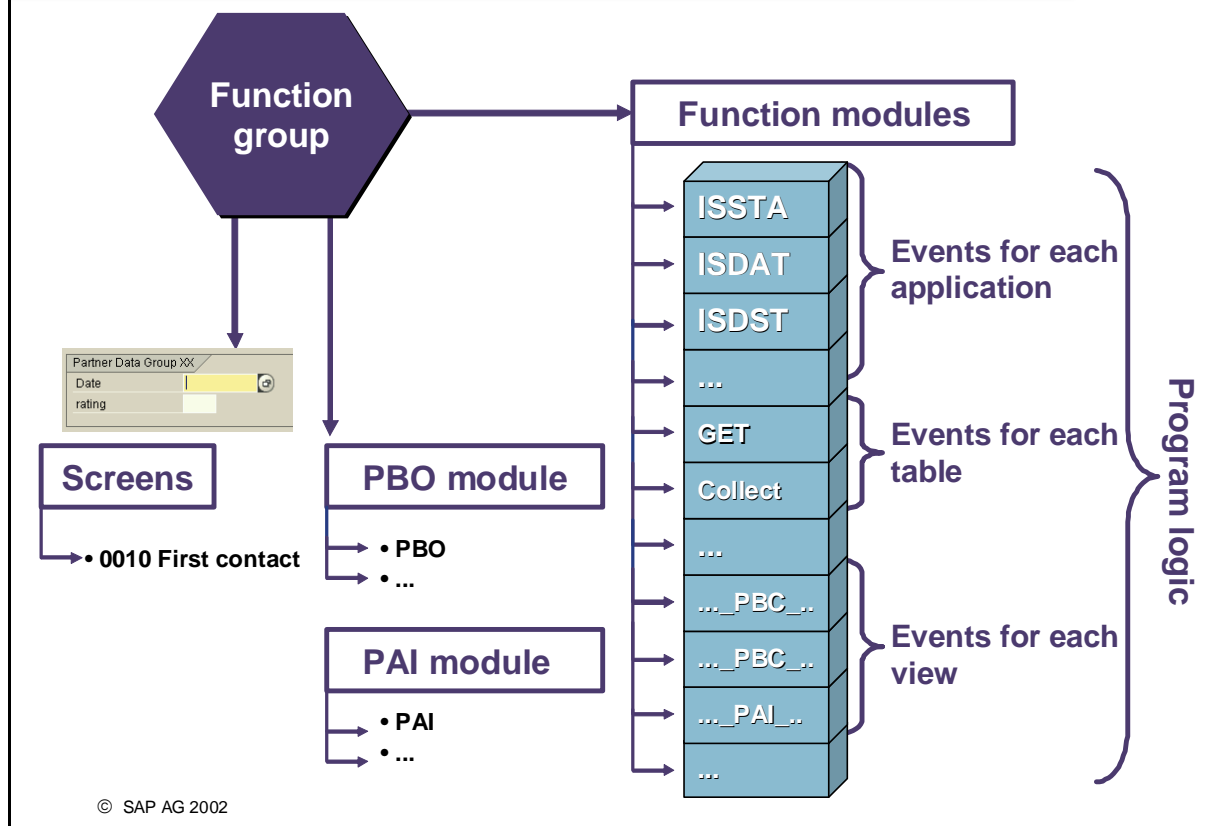
■ **IS-RE**

- Real estate contract
- Cost efficiency analysis



© SAP AG 2002

- Within the program flow, applications can take events defined by the BDT and define their own program logic for them in function modules. These function modules can be defined for any event and are then called dynamically by the BDT.
- The starting point is a normal transaction code. This code always launches the program (BUSSTART in the Business Partner). The BDT reads all the necessary information from the control tables along with the name of the transaction code.
- The program flow is fully specified. All function modules that belong to the part of the object executed are called in the appropriate event.



- Each application develops within a separate function group.
- Screens (that is, subscreens), PBO modules, PAI modules, and function modules are created for the events (for each application, table, and view) in this function group.
- The PBO module calls only one BDT service function module to set the field attributes.
- The PAI module calls only one BDT service function module to get the cursor position.
- Program logic:
 - Events for applications (reading, checking, saving data)
 - Events for tables (communication between applications/function groups)
 - Events for views
 - A) PBC - event for formatting a table (such as sorting)
 - B) PBO - event before output (such as reading text descriptions from Customizing tables)
 - C) PAI - event after input (such as checking input values)
 - Note: The same code is executed in maintenance modes without dialogs (such as direct input)
You need not program this code again.



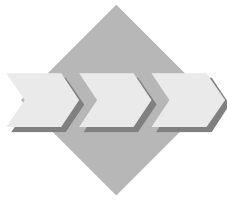
Unit: Appendix

Topic: Field exits



At the conclusion of this exercise, you will be able to:

- Implement a field exit that can be used to make supplementary checks of a screen field.



The transaction that your co-workers use to display flight information (**BC425_##**) allows you to access data for all airline carriers. The customer service personnel, however, should only be able to access the airlines for which it has explicit authorization.

1-1 What is the name of the program for the above transaction?

- 1-1-1 What is the name of the data element referenced by the input field for the airline?
- 1-1-2 Are the requirements met for linking a field exit to this screen field?

1-2 How can you create a field exit?

- 1-2-1 Create a field exit for the screen field found under 1-1. Reference the corresponding data element.
- 1-2-2 The Function Builder is started. Can you change the interface of the Function Builder? You will need to create a function group before you continue. Create one from Object Navigator and name it **ZBC425_##**.
- 1-2-3 What do you have to code in the source text? Program an authorization check. How do you perform an authorization check? Use the authority check object **S_CARRID** with an activity of '03'.
- 1-2-4 If the check is negative, send a message to this effect. Use message **010** in message class **BC425**.
- 1-2-5 Activate the function module and the field exit.

1-3 Check your results.

- 1-3-1 For which airline(s) do you not have authorization?

- 1-4 Create a local field exit for screen 0100 of transaction **BC425_##** based on the global field exit.

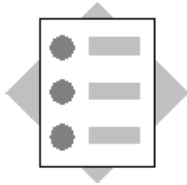


Use a second session. This is logical, especially when creating function groups in parallel.

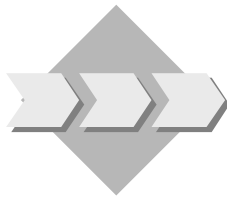


Unit: Appendix

Topic: Field exits



- Implement a field exit that can be used to make supplementary checks of a screen field.



The transaction that your co-workers use to display flight information (**BC425_##**) allows you to access data for all airline carriers. The customer service personnel, however, should only be able to access the airlines for which it has explicit authorization.

- 1-1 The name of the program for transaction **BC425_##** is **SAPBC425_FLIGHT##**. You can get this information by choosing the menu path *System* → *Status*.
 - 1-1-1 The name of the data element to which the input field for the airline refers is **S_CARRID##**.
 - 1-1-2 You can find the screen field and its data element by going to the Screen- Painter for screen 0100 in program **SAPBC425_FLIGHT##**. You can see that attribute "Dictionary" is set for field **SFLIGHT00-CARRID** in the general attributes of the element. The data element you need is **S_CARRID##**.
 - 1-1-3 Create a function group **Z_BC425_##** from the Object Navigator.
- 1-2 Create a field exit using program **RSMODPRF**. Execute the report **RSMODPRF** and enter the name of the data element **S_CARRID##**. Press F8 to run the report.
 - 1-2-1 The function builder is displayed with the correct name of the field exit already supplied. Do **not** change this name and do **not** attempt to build the field exit from the Function Builder!
 - 1-2-2 The importing and exporting parameters are already defined and must **not** be changed. Select the tab for source code. Use the Pattern pushbutton and select the radiobutton for **AUTHORITY-CHECK**. Enter the value **S_CARRID** and press the Enter key. Complete the coding as shown.

The source text should be as follows:

```
output = input.
AUTHORITY-CHECK OBJECT 'S_CARRID'
                  ID 'CARRID' FIELD input
                  ID 'ACTVT' FIELD '03'.
IF sy-subrc <> 0.
  MESSAGE e045(bctrain) WITH input.
ENDIF.
```

- 1-3 Activate the function module and all of the related function group objects. Use the menu function BACK to return to the selection screen for the report RSMODPRF. Leave both fields on the selection screen blank and press F8 to execute the report. A list of all field exits is displayed. Select your field exit and activate it with ***Field exit → Activate***.