TAW12_2

ABAP Objects and Application Areas

SAP NetWeaver

Date _____

Training Center _____

Instructors _____

_____

Education Website _____

Instructor Handbook

Course Version: 63
Course Duration: 5 Day(s)
Material Number: 50090661
Owner: Christian Braun (D035329)

An SAP Compass course - use it to learn, reference it for work

Copyright

Trademarks

Disclaimer

About This Handbook

This handbook is intended to complement the instructor-led presentation of this course, and serve as a source of reference. It is not suitable for self-study.

Typographic Conventions

American English is the standard used in this handbook. The following typographic conventions are also used.

Type Style	Description
Example text	Words or characters that appear on the screen. These include field names, screen titles, pushbuttons as well as menu names, paths, and options. Also used for cross-references to other documentation both internal (in this documentation) and external (in other locations, such as SAPNet).
Example text	Emphasized words or phrases in body text, titles of graphics, and tables
EXAMPLE TEXT	Names of elements in the system. These include report names, program names, transaction codes, table names, and individual key words of a programming language, when surrounded by body text, for example SELECT and INCLUDE.
`Example text`	Screen output. This includes file and directory names and their paths, messages, names of variables and parameters, and passages of the source text of a program.
`Example text`	Exact user entry. These are words and characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Pointed brackets indicate that you replace these words and characters with appropriate entries.

Icons in Body Text

The following icons are used in this handbook.

Icon	Meaning
	For more information, tips, or background
	Note or further explanation of previous point
	Exception or caution
	Procedures
	Indicates that the item is displayed in the instructor's presentation.

Contents

Course Overview

This two-week training course provides a comprehensive and detailed introduction to the basic principles of programming ABAP objects. You will also learn how to make specialized changes to the SAP standard system. Furthermore, you will find out how to evaluate the different methods for modification and choose the right one for the given situation. Web Dynpro is SAP's state-of-the-art technology for creating application user interfaces (UIs). This course explains in detail how to develop ABAP Web-Dynpro-based applications.

Target Audience

This course is intended for the following audiences:

- Development consultants who are responsible for adapting and developing ABAP/ABAP Objects programs

Course Prerequisites

Required Knowledge

- TAW10 (ABAP Workbench Fundamentals)
- Included in booking for TAW12: TAW11 E-Learning (ABAP Details)

Recommended Knowledge

- BC410 - Programming User Dialogs with Dynpro

Course Duration Details

Unit 1: Changing the SAP Standard System	
Changing the SAP Standard System	30 Minutes
Unit 2: Enhancing Dictionary Elements	
Table Enhancements	20 Minutes
Exercise 1: Table Enhancements	20 Minutes
Text Enhancements	30 Minutes
Unit 3: Enhancements Using Customer Exits	
Customer Exits: Overview	10 Minutes
Enhancement Management	20 Minutes
Program Exit	40 Minutes
Exercise 2: Program Exit	30 Minutes

Exercise 17: User Interface: Displaying Tables	35 Minutes
Unit 11: Controller and Context Programming	
Controller and Context Programming	130 Minutes
Exercise 18: Accessing the Context at Runtime	15 Minutes
Exercise 19: Context at Runtime: Binding Internal Tables to Context Nodes	30 Minutes
Exercise 20: Context at Runtime: Lead Selection and Supply Functions	30 Minutes
Unit 12: Internationalization and Messages	
Internationalization and Messages	70 Minutes
Exercise 21: Internationalization: Translatable Text in the UI	10 Minutes

Course Goals

This course will prepare you to:

- Use fundamental elements of object-oriented modeling in UML
- Create ABAP Objects programs that contain all useful object-oriented programming techniques
- Use the relevant tools to create object-oriented Repository objects
- Describe and use the application areas of ABAP Objects
- Define, raise, and handle class-based exceptions
- Query type and class attributes at runtime
- Make specialized changes to the SAP standard system
- Evaluate the different methods for modification and choose the right one for any given situation
- Explain the architecture of a Web Dynpro component
- Describe the parts of a Web Dynpro controller
- Create context elements in the Web Dynpro controller
- Explain how navigation and data transfer in and between Web Dynpro components can be implemented
- Define the UI of a Web Dynpro component
- Internationalize a Web Dynpro application.
- Define and send messages in a Web Dynpro component
- Define input help and semantic help for UI elements in a Web Dynpro component

Course Objectives

After completing this course, you will be able to:

- Use fundamental elements of object-oriented modeling in UML
- Create ABAP Objects programs that contain all useful object-oriented programming techniques
- Use the relevant tools to create object-oriented Repository objects
- Describe and exploit the range of applications of ABAP Objects
- Define, raise, and handle class-based exceptions
- Query type and class attributes at runtime
- Make specialized changes to the SAP standard system
- Evaluate the different methods for modification and choose the right one for any given situation
- Explain the architecture of a Web Dynpro component
- Describe the parts of a Web Dynpro controller
- Create context elements in the Web Dynpro controller
- Explain how navigation and data transfer in and between Web Dynpro components can be implemented
- Define the UI of a Web Dynpro component
- Internationalize a Web Dynpro application
- Define and send messages in a Web Dynpro component
- Define input help and semantic help for UI elements in a Web Dynpro component

It is **imperative** that you read these notes, even if you have already completed an earlier version of this course. If available, read the list of the known errors for this course as well. If it exists, this list has deliberately been saved separately from this printed course on SAP Service Marketplace (http://service.sap.com/curr-abap), to keep the course as up-to-date as possible and minimize technical inputs.

Training courses recommended as preparation:

- SAPTEC
- BC400
- BC401
- BC402
- BC410
- BC425
- BC427
- BC430
- NET310

1. Learn the contents of this course including all items of the *Online documentation* for the respective subject areas. Solve all exercises on your own at least once (without the help of model solutions, including all details and optional sections).
2. Obtain answers to any questions you have about the course materials and the exercises.
3. Develop your own presentation methods and run through each demonstration at least once yourself.
4. Clear up any questions you have on these methods.
5. Attend this course at least once when it is held by an experienced instructor and possibly supervise participants during an exercise yourself.

Training system:

- SAP NetWeaver 7.0 (Support Package 13 or later version) or higher (ABAP Engine is sufficient).

  The training administration team should provide you with a system, including a client. If problems arise with the server assignment, create a message under the component SR-KPS-ISM.

- Either a Windows Terminal Server (WTS) or local PCs with SAP GUI 6.40 (at least patch level 21) or higher installed.

  At the time this document was put together, the WTS address was http://wts:1080/portal/corp/training.html → Common_Training.

Training courses at a customer site or in a third-party training center

You can only establish a connection using the SAP Citrix Secure Gateway (SAP CSG) for training courses at the customer site. Therefore, you requirea CSG user ID. The training department should have created the user ID for the course date. The training department sends you the data (user ID and password). Instructors and participants

use the same CSG user ID for the training course. Use the link http://mywts.sap.com to establish a connection to the training WTS farms. Enter the CSG user ID and the password. Choose your region (US, EMEA or APJ). Choose *Training - Zone*. Connect to *Common Training* if no other WTS farm option is executed for the training course.

Make all the preparations with regard to the following two points **in accordance with the "General instructor guide for all BC4XX courses"**. It is located in the same area of the *SAP Corporate Portal* as this handbook. Refer to the general handbook **first**.

The required system preparations are described below.

Data for exercises and demonstrations: As a rule, the necessary data has already been generated for your training client. If this is not the case, (for example, if your training course takes place at a customer site) you will have to run the data generation program (report SAPBC_DATA_GENERATOR). This will query you on the type and quantity of the data. Select the standard setting and set the indicator for *Postings also cancelled*.

User IDs and initial passwords for participants: Unlike the application data, you must always generate the user IDs. Use the (locked) user ID *TAW12_USER* as a copy template.

Change requests for participants' exercises: In the program used to generate user IDs (transaction ZUSR), you also have the option of creating a **common** change request that contains a task for each user generated in this way. Use it.

Presentation: Make sure you can open the offline presentation for this training course (version 63).

Example objects: All repository objects for this training course belong to the following packages **BC401, BC425, DNW7AW, and NET310**. The following naming conventions were used for the course objects:

- ...BC401_*CCC***D**_...: Demonstration and example objects
- ...BC401_*CCC***S**_...: Model solutions
- ...BC401_*CCC***T**_...: Templates

In each case, *CCC* is a three-character unit code.

Caution: Never modify these standard objects. **Always** create **copies**, if you want to change something for demonstration purposes. Otherwise, you may cause errors in other objects, or prevent other courses from being held in this system.

The same applies for your course participants. If, however, you use the correct copy template for their user IDs, they will probably not have the authorization to do so.

General notes for the course: The time specifications are **recommendations** only. They contain sufficient time “intervals” for more detailed questions. However, you should bear in mind that the more participants learn, the more questions they tend to ask. The group may not be particularly homogenous. You must always guarantee that the entire content can be presented. We recommend keeping within the time limits from the very start.

The course materials should be suitable for participants to work through again after the course. This is why we provide extensive explanations and additional information. It is your task to pick out the most important statements in the material. The decisive factor should be the graphics that appear in the offline presentation and the aspects they represent.

Similar principles apply to the objects of all packages. These packages contain more objects than you will need to present the course successfully. Much of the code is also available as source code extracts in the presentation or in participants’ material (or both). It is up to you whether you want to discuss these in detail, use the demonstration objects themselves, or create similar objects during the presentation and use them. We recommend direct demonstration in the system.

Additional notes for instructors: In addition to these general notes, you will find additional, more detailed notes in each unit and lesson. You will also find other short notes in the material wherever we believe they are necessary to ensure that the content is explained in sufficient detail.

Introductory phase at the start of the course:

- Introduce yourself as the course instructor
- Outline organizational issues
- Introduction of participants (optional, depends on time available.)
- Overview of the course material
- References to more in-depth information
- Overview of the course content

Additional Information about Certification

- Open the browser and enter the following link: http://www.sap.com/services/education/certification/index.epx. Select the following criteria: SAP NetWeaver, Development, Developer/Development Consultant. For more information, select the following option on the next page: SAP Certified Development Associate - ABAP - SAP NetWeaver 7.0.

Other Training Courses

- TAW10
- TAW11

Unit 1

Changing the SAP Standard System

Unit Overview

- Overview of adjustment options and decision diagram for selecting an adjustment technique
- Enhancement types

Unit Objectives

After completing this unit, you will be able to:

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

Unit Contents

Lesson: Changing the SAP Standard System

Lesson Duration: 30 Minutes

Lesson Overview

This lesson provides an overview of the options available for modifying your SAP system, and outlines guidelines that will help you decide which option to use. The principles of the different enhancement options will also be explained here.

Lesson Objectives

After completing this lesson, you will be able to:

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

-

Business Example

You would like to learn about the options available for modifying and enhancing your SAP system.

Options for Modifying an SAP System

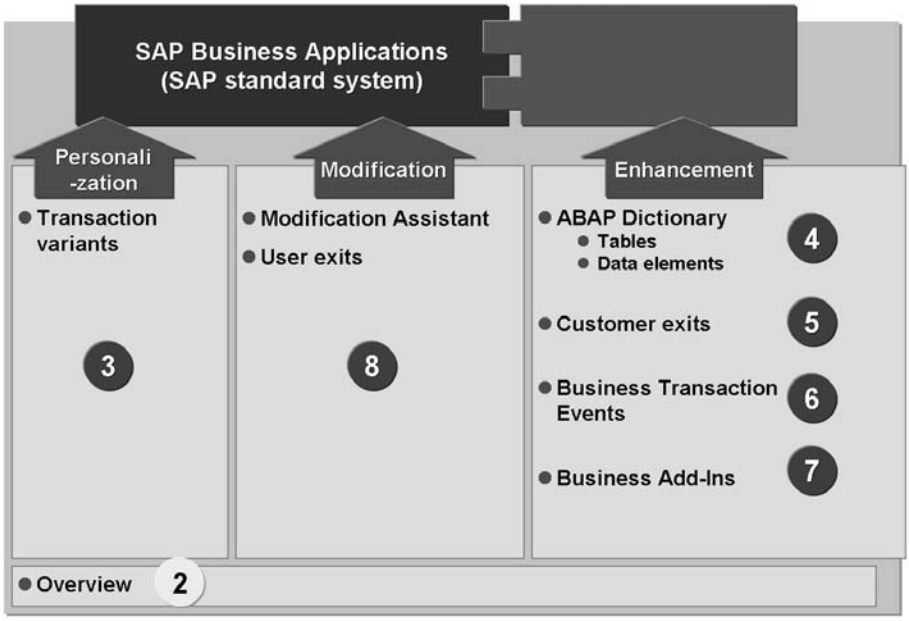


Figure 1: Overview Diagram: Changing the SAP Standard System

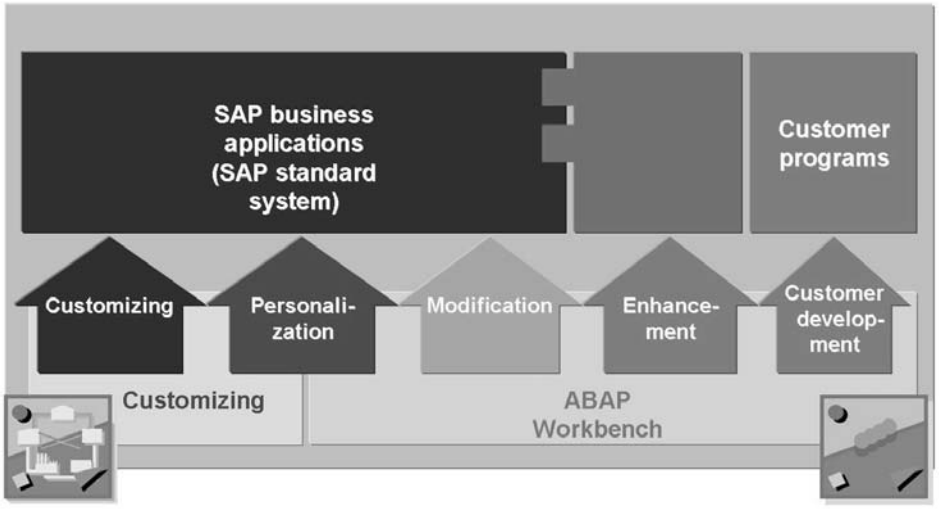


Figure 2: Change Levels

You can tailor the R/3 System to suit your needs in the following ways:

- **Customizing**: This means setting up specific business processes and functions for your system in line with the Implementation Guide. Therefore, all possible changes have been thought through and organized.
- **Personalization**: This means 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. From Release 4.5A, this procedure has been automated thanks to the Modification Assistant.
- **Enhancement**: This means creating Repository objects for individual customers, which refer to objects that already exist in the SAP Repository.
- **Customer developments**: This refers to the creation of Repository objects that are unique to individual customers in a specific namespace that is reserved for new customer objects.

Customizing and most personalization is done using tools found in the SAP Business Engineer; customer developments, enhancements, and modifications are all carried out using the tools available in the ABAP Workbench.

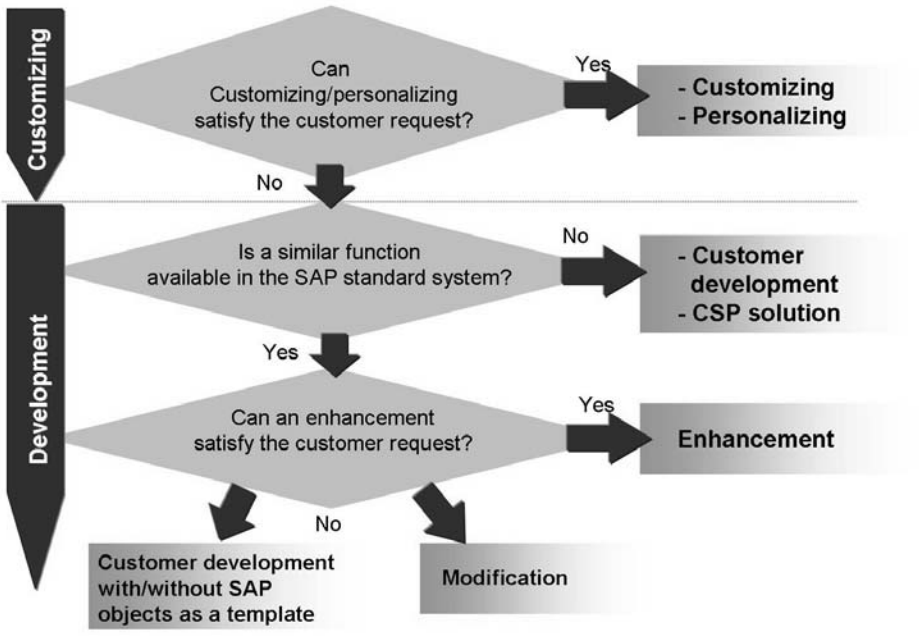


Figure 3: Procedure for Changing Functions

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 that are certified by SAP is available in the SAP Service Marketplace under the alias /softwarepartner.

A development project falls into the customer development category if the SAP standard system does not already contain functions similar to the one you are trying to develop. If, however, a similar SAP function exists, try to assimilate it into 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 modified 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 thanks to the Modification Assistant.

Thus, 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

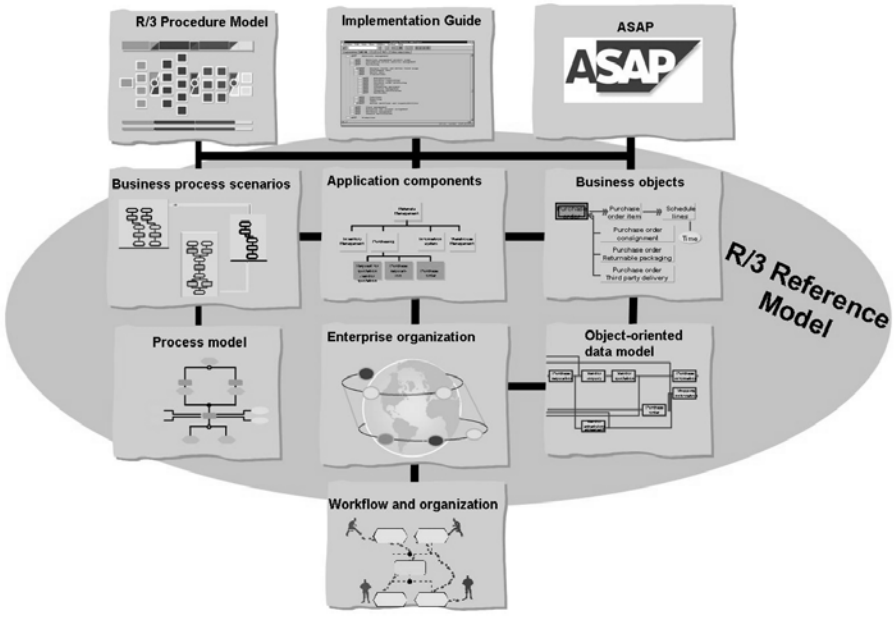


Figure 4: Customizing

Show your students the Implementation Guide. (Select the menu path Tools -> Customizing -> IMG -> Execute Project. Choose the Display SAP Reference IMG button). Perform a simple Customizing transaction, for example: SAP Customizing Implementation Guide -> Enterprise structure -> Definition -> Logistics - General -> Define, copy, delete, check division. Enter PB for plastic bags. Use the Customizing Organizer to show them the table and view currently being edited.

The Business Engineer is made up of all SAP implementation tools. These include:

- **The R/3 Reference Model**Contains all models that describe the R/3 System (the process model, the data model, and the organization model)
- **The Implementation Guide (IMG)** Displays a list of all Customizing activities

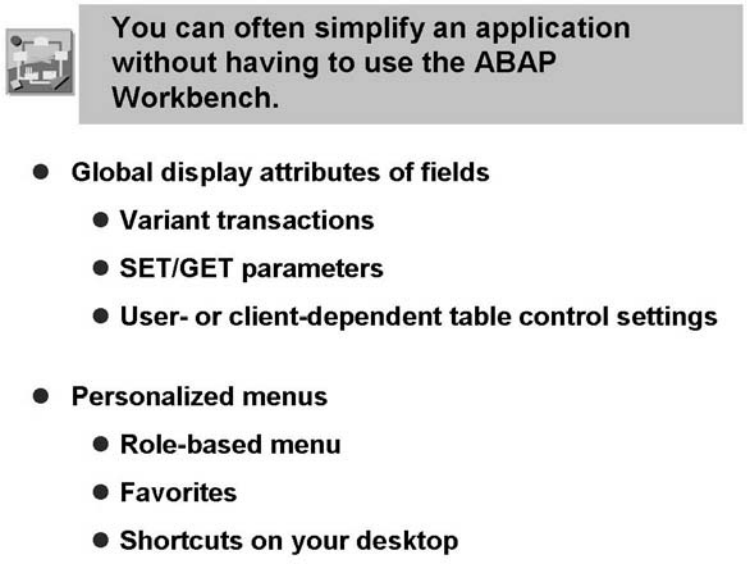


Figure 5: Personalization

Show how to make Global Field Settings Guide (*Tools -> Accelerated SAP -> Personalization -> Global Field Values*, which starts Transaction SHDG). Here you choose *Application Components -> Financial Accounting*. Choose "Change Global Fields" for Financial Accounting. Set Company Code 1000. Mark the sub-tree and activate. However, in most of the training systems your settings won't take effect because the profile parameter "dynpro/global_fields" is not set to "Yes" but has its default value "No".

Show how the administrator can make client-wide ("global") table control settings. Starting your demo with transaction VA01 you can also show SET-/GET-Parameter: then choose:

- Order Type "CD" (Delivery free of Charge)
- Sales organization 1000
- Distribution Channel 10
- Division 00

Now press green check. Sold-to-party and Ship-to-party should be 1000. In the sales items you can insert material M-05, a quantity and a price. Save your work. You can change the table control now: Move some columns. Now you can press the settings button. Choose the administrator functionality. Set the column "Customer material no." invisible. Activate your settings. Now leave transaction VA01 and start TA VA21 (Create Quotation) or VA03 (Display Sales Order). Explain why the order number is already in the field. Show the table control "All items" where you changed the settings.

Personalization accelerates and simplifies how business cases are processed by the R/3 System. 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**to reflect the needs of different user groups within your company.

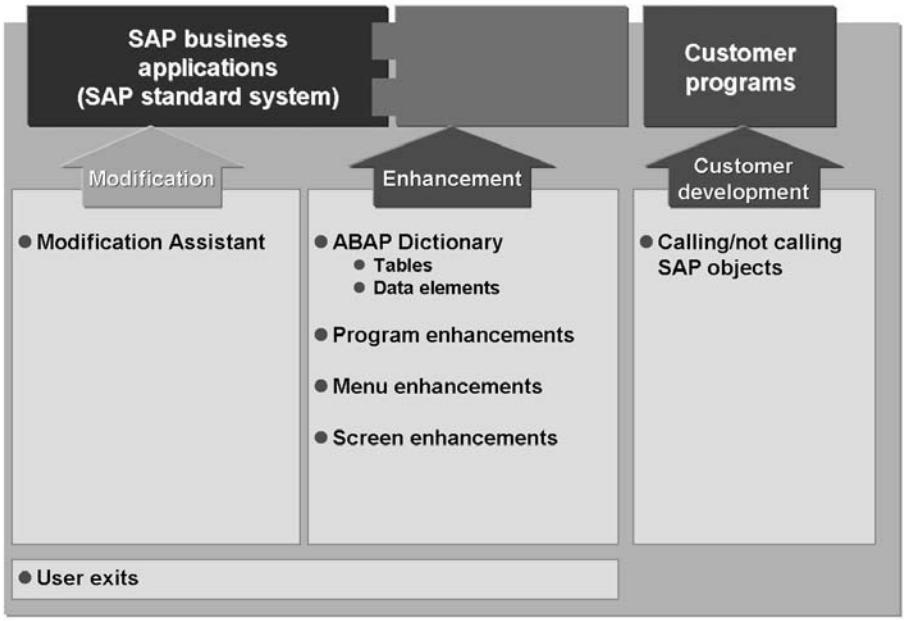


Figure 6: Change Levels Using the ABAP Workbench

Modifications are changes to SAP objects in customer systems. They are:

- Executed with the help of user exits (these are subroutines reserved for customers in 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. Example: A customer creates a program that calls an SAP function module.

The enhancement concepts embody the reverse of this principle: SAP programs call Repository objects that you, as a customer, have created or changed. 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 exit**)
- On screens by processing customer code that refers to a specific field on the screen (**field exit**)
- In ABAP Dictionary tables or structures (**table enhancements**)

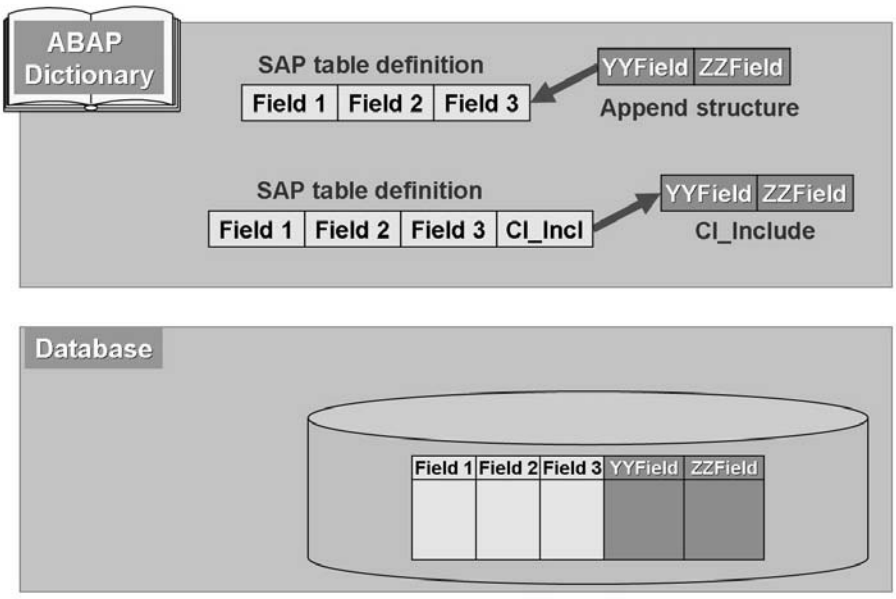


Figure 7: Table Enhancements

SAP provides two ways to add tables and structures to fields.

- Append structures
- Customizing includes (“CI includes”)

Both techniques allow you to attach fields to a table without actually having to modify the table itself.

An append structure is a structure that 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.

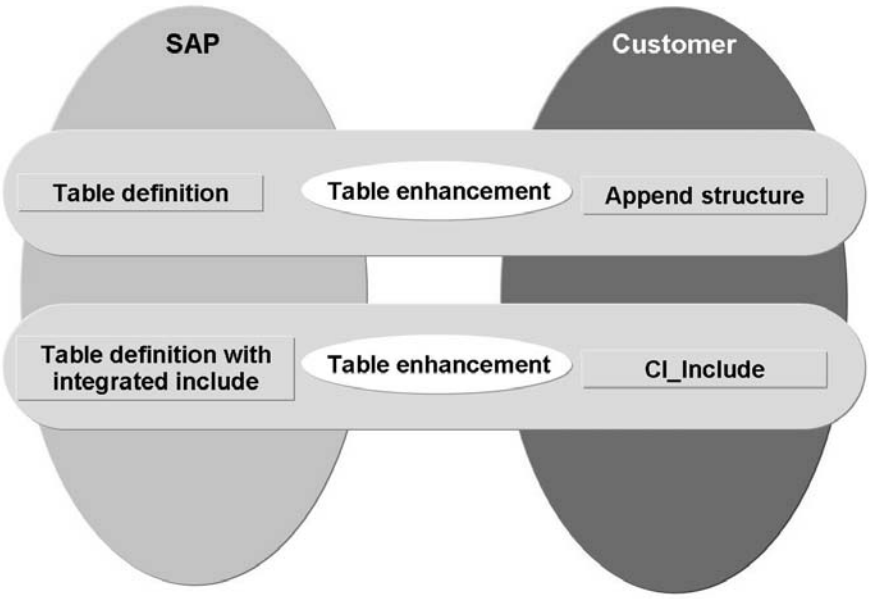


Figure 8: Table Enhancements: SAP and the Customer

Append structures allow you to attach fields to a table without actually having to modify the table itself. Table enhancements that use append structures therefore do not have to be planned by SAP developers. An append structure can only belong to exactly 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. However, table enhancements that use CI_includes have to be planned by SAP developers.

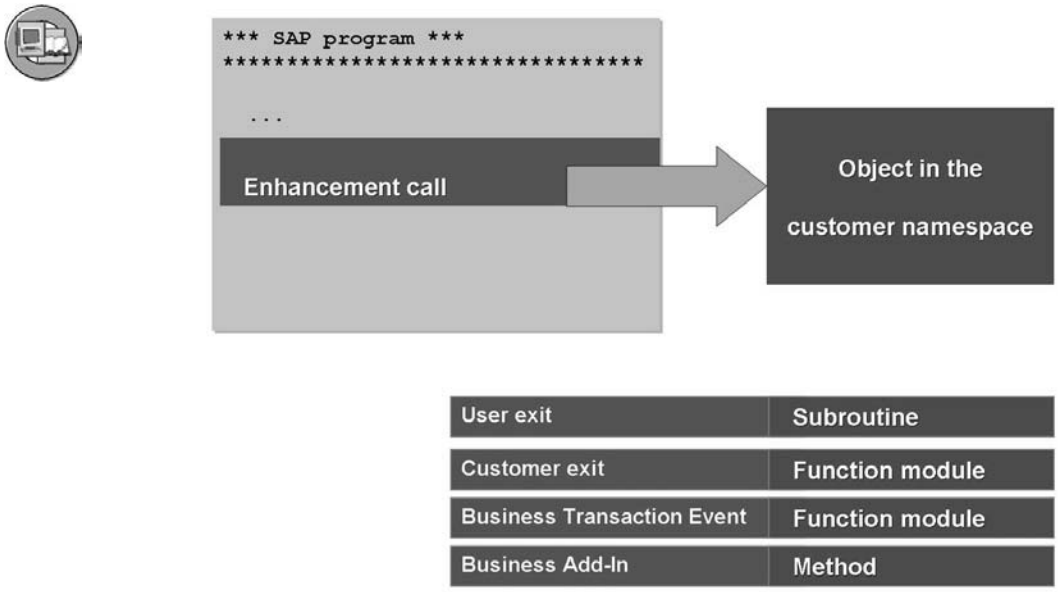


Figure 9: Program Enhancements: How they Work

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. 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.

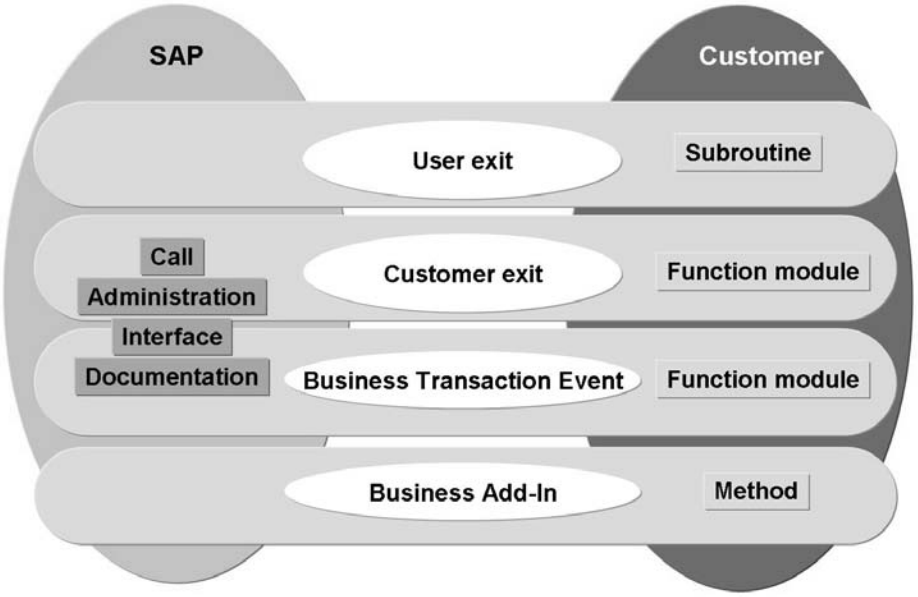


Figure 10: Program Enhancements: SAP and the Customer

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.

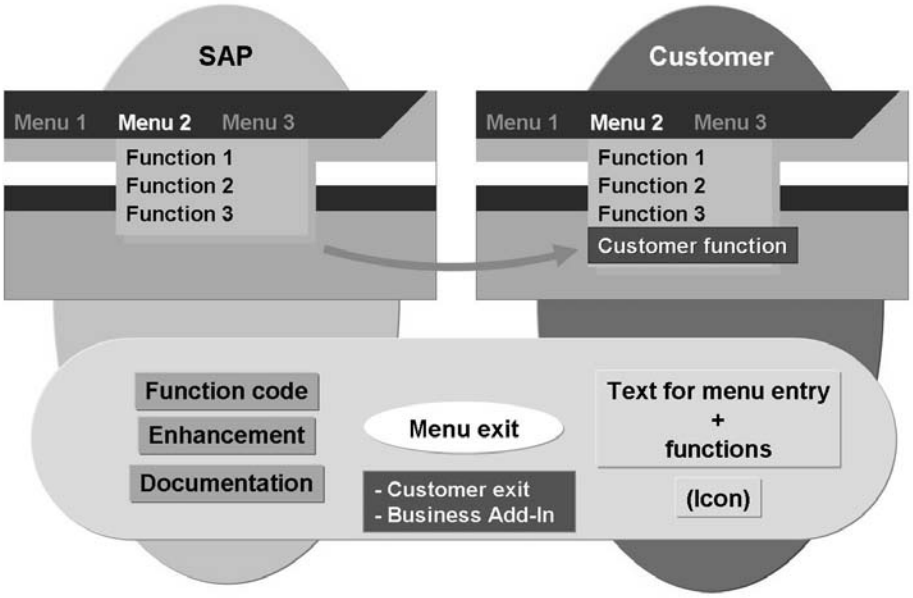


Figure 11: Menu Enhancements: SAP and the Customer

Menu enhancements permit you to add additional menu entries to an SAP standard menu. The system provides two options here:

- 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 this was specified by the SAP developer.

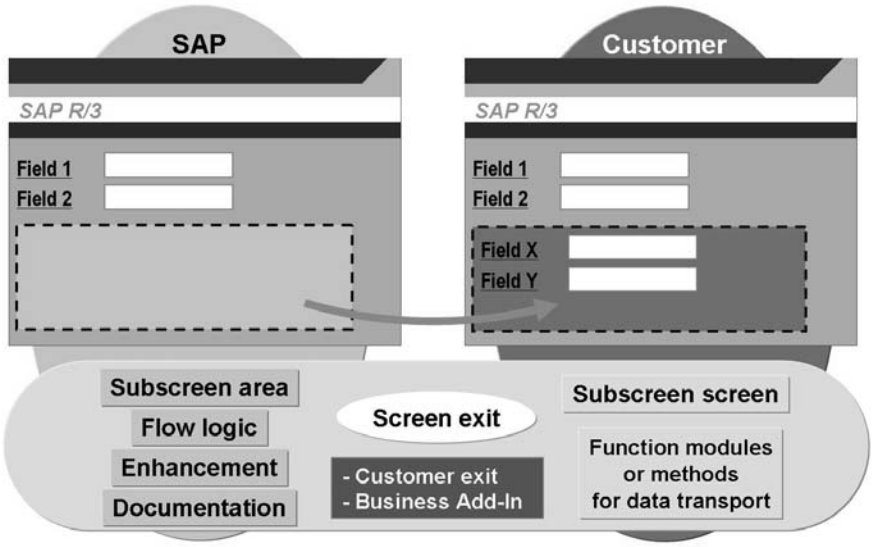


Figure 12: Screen Enhancements

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, using flow logic, for example. You also have to implement the data transport.

The unit “Enhancements Using Customer Exits” describes how to implement screen exits for “traditional” screen exits. The unit “Business Add-Ins” explains how to implement these exits for new screen exits.

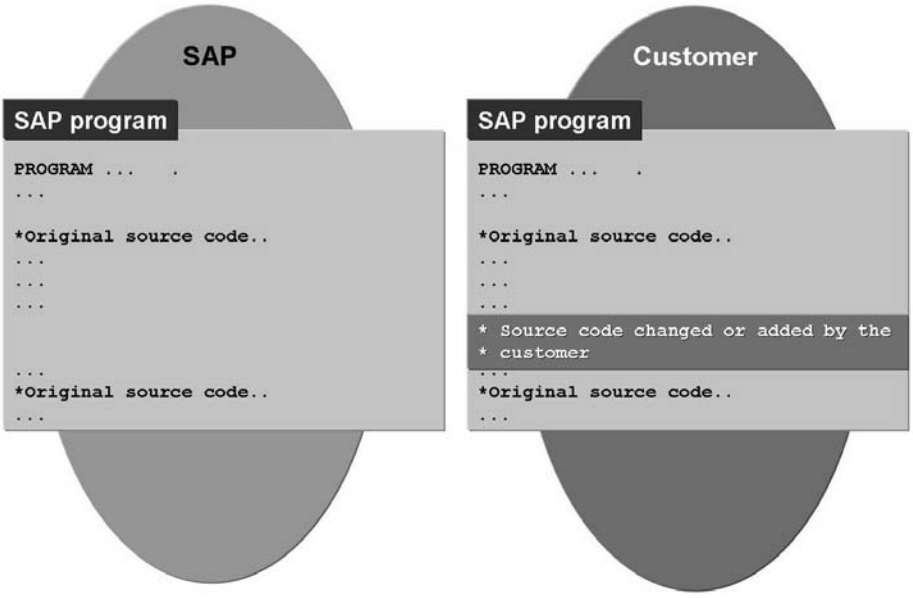


Figure 13: Modifications

Any change that you make in your system to an object that has been delivered by SAP is known as a modification.

Modifications may lead to complications during the next system 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, a finer granularity has been used to record modifications. This has been made possible by the Modification Assistant, as we will see in the “Modifications” unit.

The modification adjustment process has also been revised. The "Modifications" unit also explains how to reconcile modifications.

Facilitated Discussion

-

Discussion Questions

Use the following questions to engage the participants in the discussion. Feel free to use your own additional questions.

-

Lesson Summary

You should now be able to:

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

Unit Summary

You should now be able to:

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

Unit 2

17

Enhancing Dictionary Elements

Unit Overview

- Append structures
- Customizing includes
- Text enhancements

Unit Objectives

After completing this unit, you will be able to:

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

Unit Contents

Lesson: Table Enhancements

18 Lesson Duration: 20 Minutes

Lesson Overview

This lesson will demonstrate how you can use append structures and Customizing includes prepared by SAP to insert additional customer fields in SAP table structures.

Lesson Objectives

After completing this lesson, you will be able to:

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

-

Business Example

You would like additional fields to SAP tables without implementing modifications.

Table Enhancements

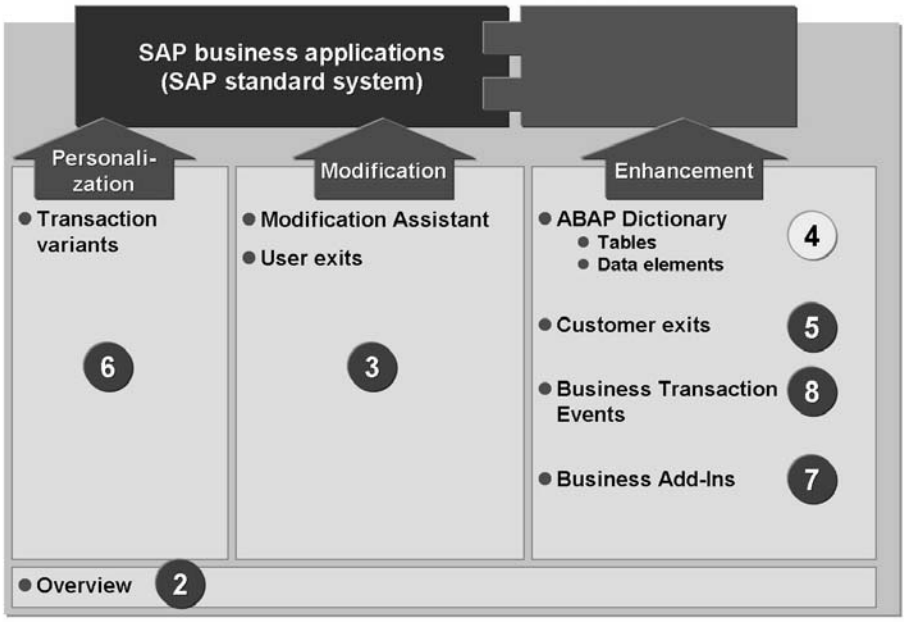


Figure 14: Enhancements to the ABAP Dictionary: Overview Diagram

- 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 in the same way as normal structures in programs
- Customizing include
  - Is already integrated into SAP tables by SAP
  - The customer fills it with the desired additional fields
  - May contain source code or screen exits provided by SAP for processing or displaying the fields

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 system. With append structures, customers can add their own fields to any table or structure they 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 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.

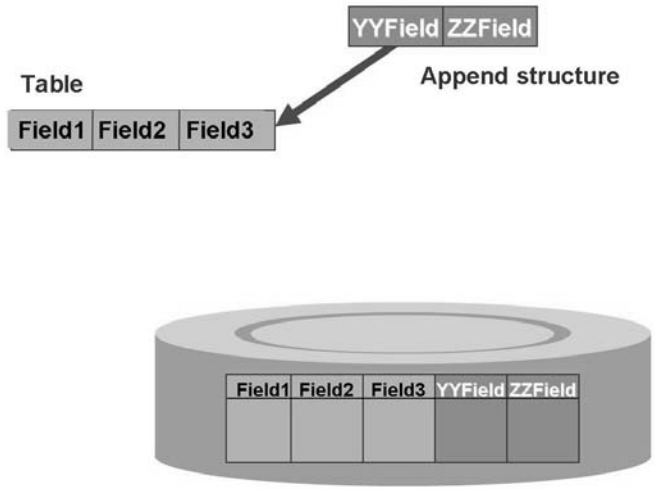


Figure 15: Append Structures

Append structures allow you to attach fields to a table without actually having to modify the table itself.

An append structure is a structure that is assigned to exactly one table. There can be several append structures for a table. 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, the table to which it is assigned is also activated and the changes also take effect there when it is activated.

You can use the fields in append structures in ABAP programs just as you would any other field in the table.

Hint: If you copy a table to which an append structure has been appended, the fields of the append structure become normal fields of the target table.

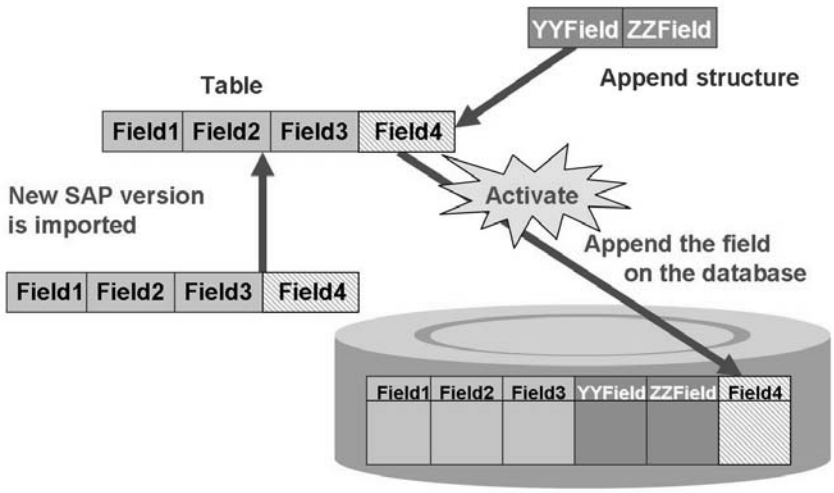


Figure 16: Append Structures at Upgrade

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.

Since the order of the fields in the ABAP Dictionary since Release 3.0 can differ from the order on the database, a conversion is not necessary when you add an append structure or insert fields in an existing append structure. The structure is adjusted when the database catalog is adjusted (ALTER TABLE). When activating in the ABAP Dictionary, the definition of the table is changed and the new field is appended to the database table.

Hint: Please note the following points about append structures:

- No append structures may be created for pooled and cluster tables.
- If a long field (data type LCHR or LRAW) occurs in a table, it cannot be extended with appends structures. This is because long fields must always be in the last position of the field list, that is, they must be the last field of the table. 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 should be in the customer namespace, that is, they must begin with either YY or ZZ. This prevents name collisions with new fields inserted in the standard table by SAP.

Create an append structure for table SFLIGHT00 that will be filled with data later. Create a field named ZZMEAL that refers to data element S_MEAL. Activate your append structure. Create another field with a name not starting with ZZ or YY. Execute a syntax check. Emphasize that the database table does not recognize the append structure: For the database it is just one table.

Show your students table AUFK. Scroll down and show them the Customizing include CI_AUFK. When you double-click CI_AUFK, you will see that the include structure does not exist yet. Create it and activate it. Show the where-used list for CI_AUFK to your students.

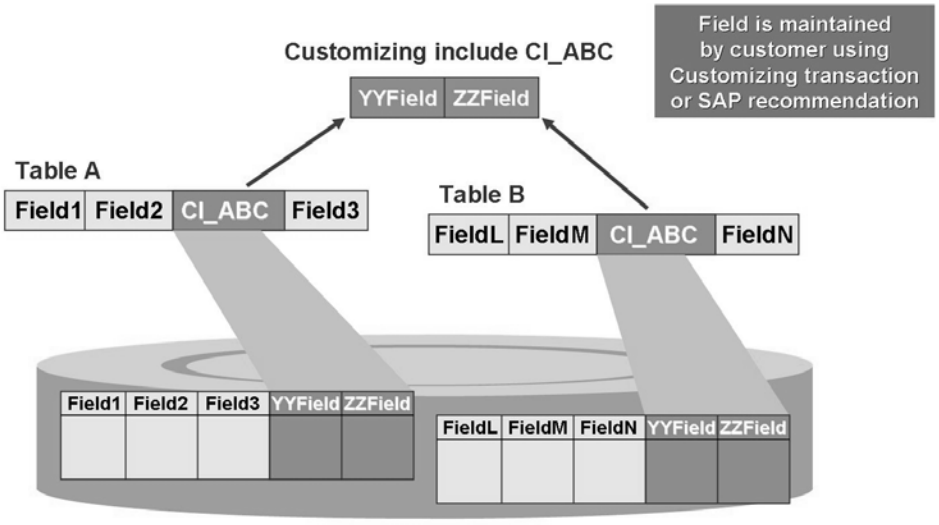


Figure 17: Customizing Includes

Some of the tables and structures delivered with the R/3 standard contain special include statements: These are known as Customizing includes. These are often inserted in 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 includes are part of the customer namespace, and their names start with “CI_”. This naming convention guarantees that nonexistent Customizing includes do not lead to errors. No source code for Customizing includes is delivered with the R/3 standard system.

You create Customizing includes using special Customizing transactions. Some are already part of SAP enhancements and can be created using project management (see the unit “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 that apply to append structures.


23

Exercise 1: Table Enhancements

Exercise Duration: 20 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Enhance tables with append structures.

Business Example

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

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

Task 1:

How can you add these two fields to the table **SFLIGHT##** without modifying it?

1. How do you go about enhancing the table **SFLIGHT##**?
2. Enhance the table **SFLIGHT##** with a technique that does not require modifications.

Task 2:

Create an append structure for table **SFLIGHT##**.

1. Include the following in the structure:

   Pilot name (Type assignment using data element S_PILNAME)

   Meal (Type assignment using data element S_MEAL)

Solution 1: Table Enhancements

Task 1:

How can you add these two fields to the table **SFLIGHT##** without modifying it?

1. How do you go about enhancing the table **SFLIGHT##**?

   a) The only method available to enhance the transparent table **SFLIGHT##** is an append structure, since it contains no Customizing includes.

2. Enhance the table **SFLIGHT##** with a technique that does not require modifications.

   a) The procedure is described in detail in solution for the next task.

Task 2:

Create an append structure for table **SFLIGHT##**.

1. Include the following in the structure:

   Pilot name (Type assignment using data element S_PILNAME)

   Meal (Type assignment using data element S_MEAL)

   a) Create your append structure using either the menu option *Goto → Append Structures …* or the corresponding button. Accept the name that the system proposes. Enter a short description for the append structure and save it under the package you created.

   b) Include two fields in the structure:

      The field names must begin with YY or ZZ. For example **YYPILOT** and **YYMEAL**.

      Activate your append structure. If an error occurs, details are displayed in the activation log.

Lesson Summary

You should now be able to:

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

Lesson: Text Enhancements

Lesson Duration: 30 Minutes

Lesson Overview

In this lesson, you will learn how to overwrite texts delivered with SAP data elements with customer-specific texts. These customer-specific texts are then displayed on screens and selection screens instead of the text delivered by SAP.

Lesson Objectives

After completing this lesson, you will be able to:

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

-

Business Example

You would like to overwrite field labels and documentation for SAP data elements with customer-specific texts.

Text Enhancements

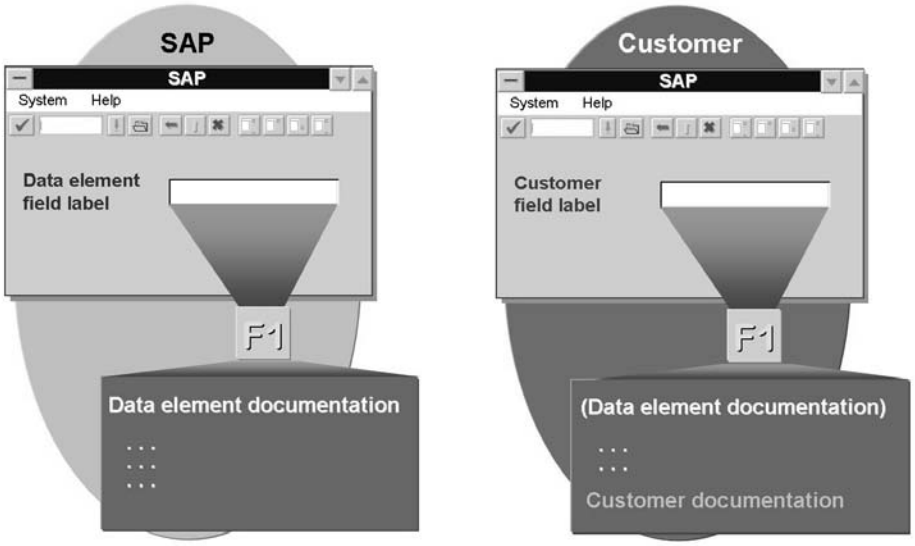


Figure 18: Text Enhancements: Overview

Text enhancements comprise customer-specific field labels and documentation for SAP data elements.

They function in all SAP applications that use the affected data element (that is, they are global enhancements)

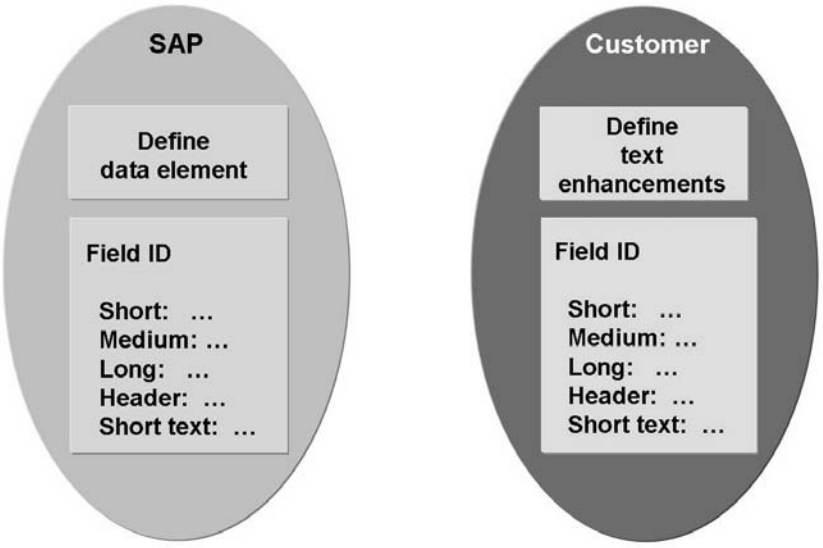


Figure 19: Overwriting SAP Field Labels (1)

SAP application programmers define field labels with different lengths and enter a short description for each data element. Customers can overwrite these texts with their own texts.

New field labels can be provided for all screen fields in this way.

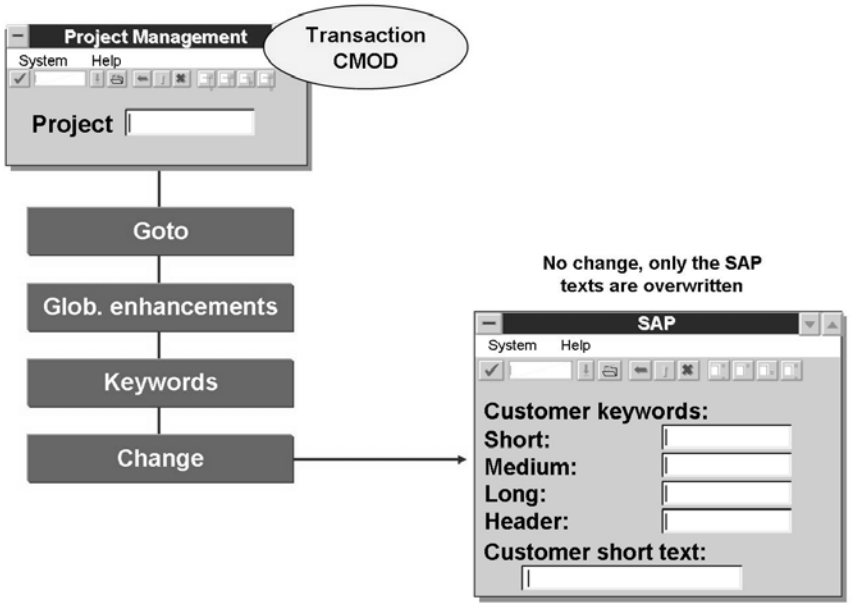


Figure 20: Overwriting SAP Field Labels (2)

You edit text enhancements using project management. (Choose *ABAP Workbench → Utilities → Enhancements → Project Management*) (or enter the transaction CMOD).

Select "*Goto → Global Enhancements*" to open field label and documentation enhancement (overwriting) for SAP data elements

Change the field labels (keywords) for one data element that is used in transaction VA01, for example VKORG. Show your students the results in transactions VA01. You can also change the field label of a text with attribute "F", that is, no changes will be accepted. An example for that is the data element PROGRAMM, which is used in transaction SE38 and SA38.

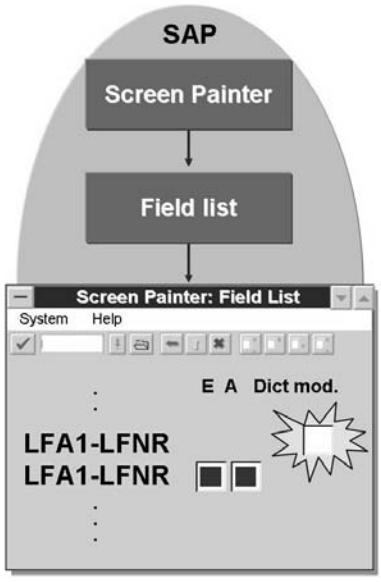


Figure 21: Prerequisites for Overwriting an SAP Field Label

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 heading
- V: Variable text transfer from the Dictionary (as SPACE):
- F: Fixed - no text transfer from the Dictionary.

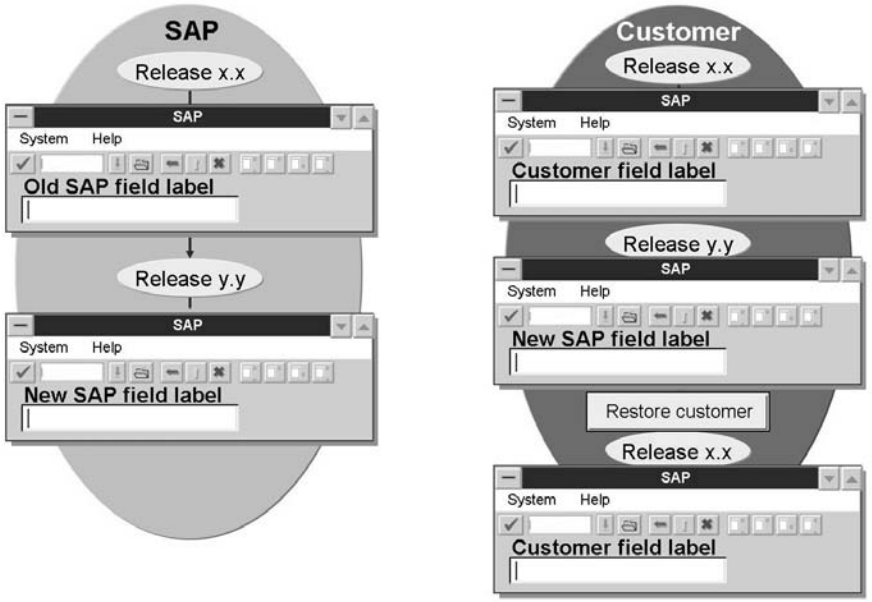


Figure 22: Overwritten Texts in Upgrades

If SAP has redelivered existing field labels, you need to restore your customer field labels after a release upgrade or after you import new corrections. 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 contents of 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.

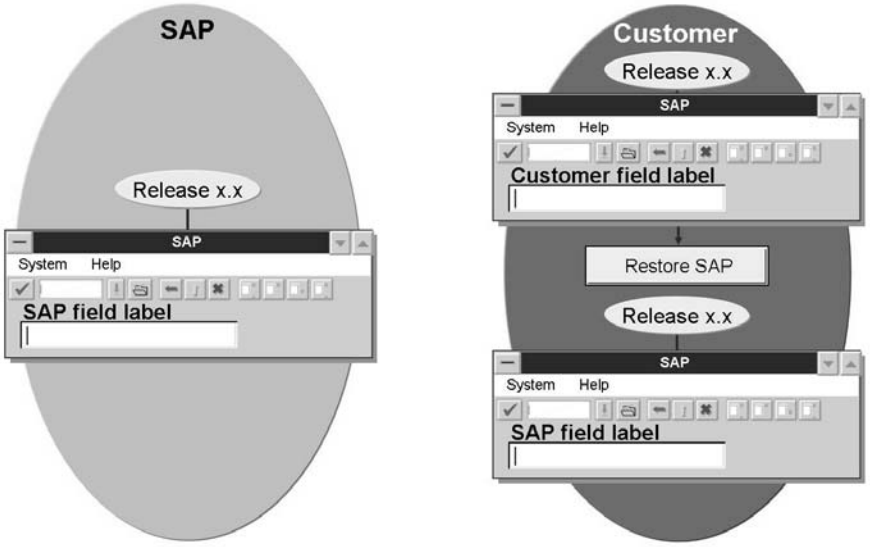


Figure 23: Restoring SAP Field Labels

To undo field label changes, choose the menu option to restore SAP field labels.

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.

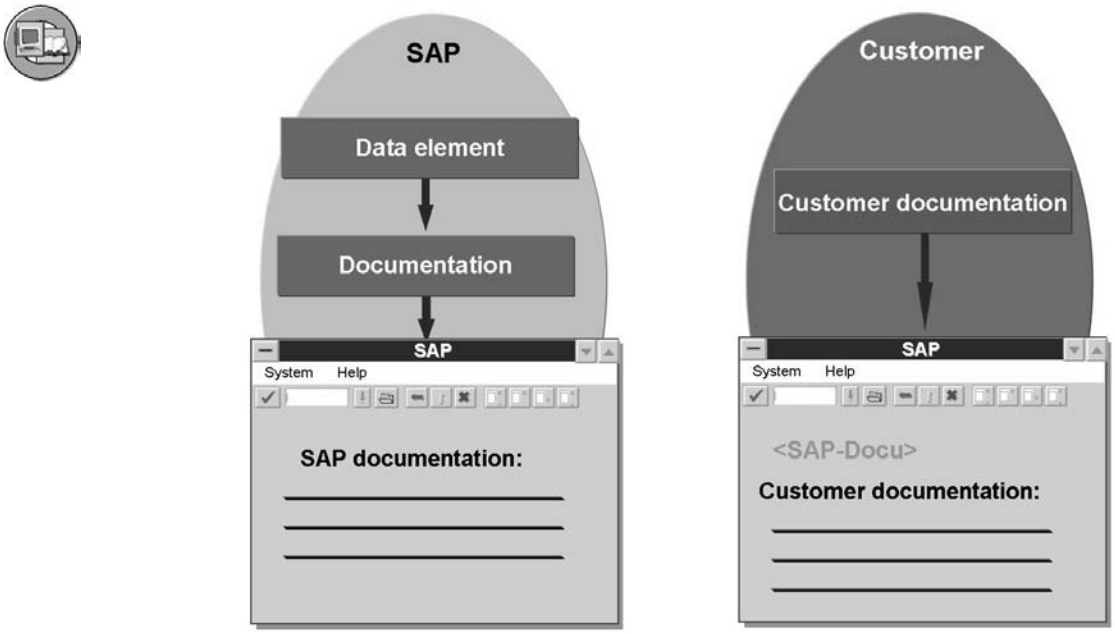


Figure 24: Enhanced Documentation for Data Elements

When you enhance the data element documentation, you can copy the SAP documentation as well as your own. In this case, when you select F1 for the corresponding screen field, the system displays the SAP documentation and the customer-specific documentation.

You can generate a list of data elements that have been changed and edit the relevant customer documentation by selecting the corresponding lines in the list.

Simply delete your own documentation if you the want original SAP documentation to be displayed.

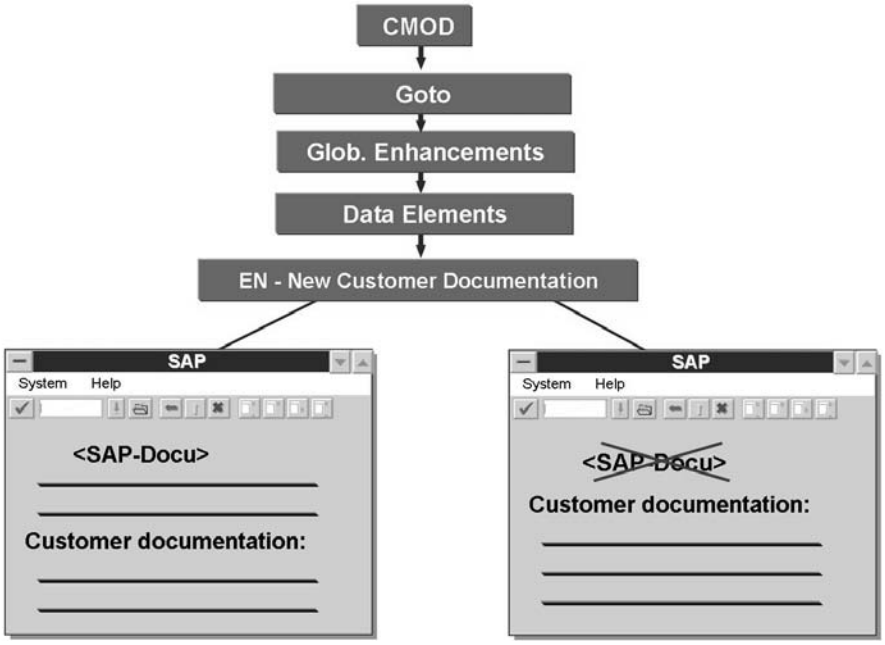


Figure 25: Creating Customer Documentation

The menu path described in the graphic above opens a dialog window in which you must choose either “Original text” or “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 original SAP documentation.

Facilitated Discussion

-

Discussion Questions

Use the following questions to engage the participants in the discussion. Feel free to use your own additional questions.

-

Lesson Summary

You should now be able to:

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

Unit Summary

You should now be able to:

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

Unit 3

37

Enhancements Using Customer Exits

Unit Overview

- Introduction
- Enhancement management
- Function module exits
- Menu exits
- Screen exits
- 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
- Explain how to attach enhancement projects to change requests
- Transport enhancement projects

Unit Objectives

After completing this unit, you will be able to:

- explain how customer exits are organized and basic principles of how they are used
- This lesson explains fundamental aspects of an enhancement project, which you must use to implement existing customer exits.
- Find program exits that are implemented using customer exits, and use them to enhance functions

- Explain how menu exits that are implemented using customer exits work
- Find and use these menu exits
- You would like to find and use screen exits that are implemented using customer exits.

Unit Contents

Lesson: Customer Exits: Overview

39 Lesson Duration: 10 Minutes

Lesson Overview

This lesson provides an overview of customer exits.

Lesson Objectives

After completing this lesson, you will be able to:

- explain how customer exits are organized and basic principles of how they are used

-

Business Example

You would like to obtain an overview of how customer exits are used.

Customer Exits (Overview)

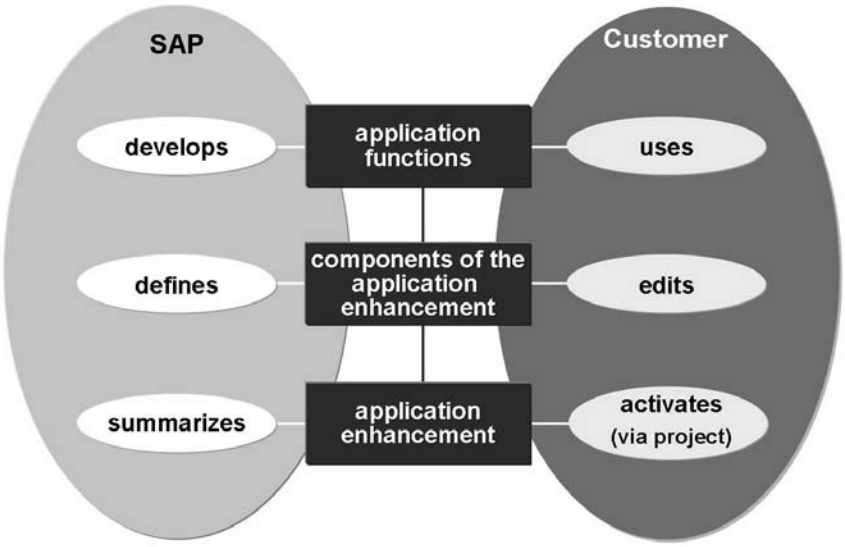


Figure 26: SAP Application Enhancements

Application enhancements allow customers to add 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 have the following characteristics:

- Each enhancement provides you with a set of preplanned, precisely defined functions.
- Each 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.

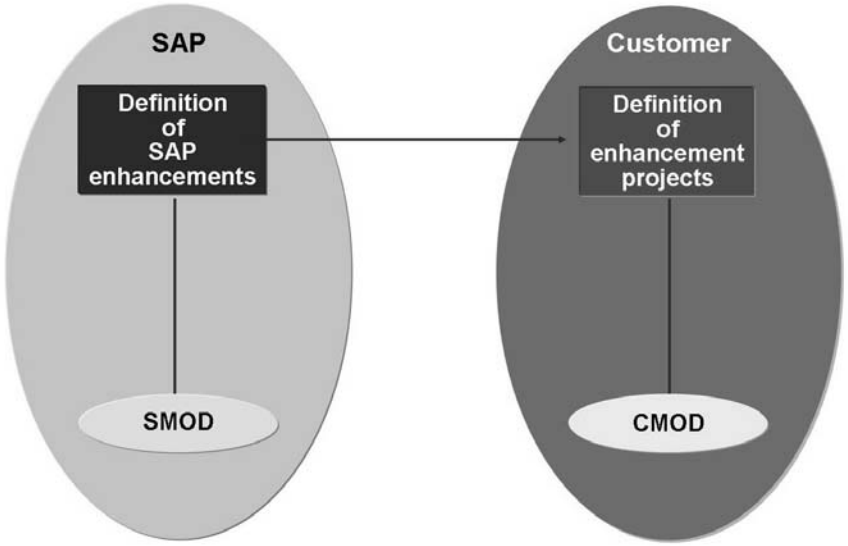


Figure 27: Customer Enhancement Project

SAP programmers create SAP enhancements from program exits, menu exits, and screen exits. A management function is provided for this purpose (transaction code SMOD).

Customers are provided with a list of existing SAP enhancements, and can group desired enhancements into an enhancement project using a separate management function (transaction CMOD).

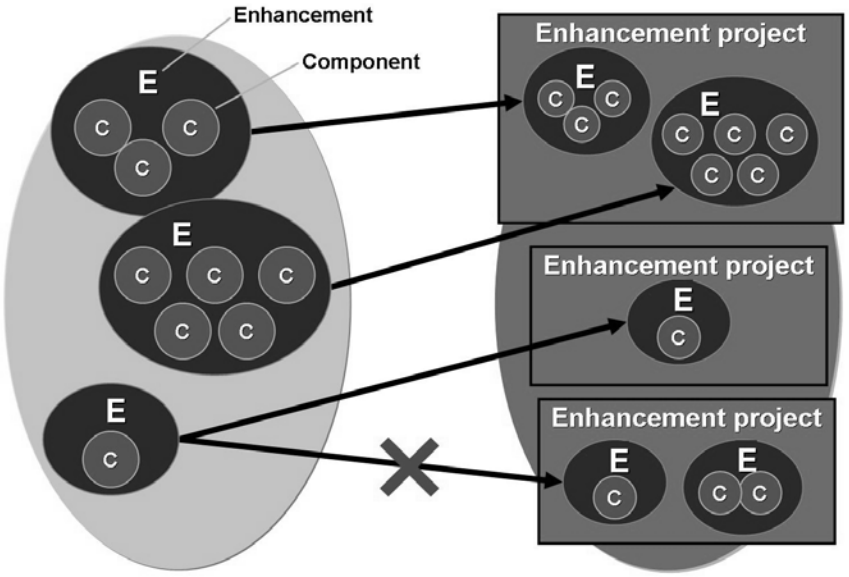


Figure 28: Enhancements and Enhancement Projects

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 all SAP enhancements (this ensures that **SAP enhancements are unique**).

Customer enhancement projects consist of SAP enhancements. Each individual SAP enhancement may be used only once in all customer enhancement projects (this guarantees that **a customer project is unique**).

Create an enhancement project and record your actions using the Workbench Organizer.

Facilitated Discussion

-

Discussion Questions

Use the following questions to engage the participants in the discussion. Feel free to use your own additional questions.

-

Lesson Summary

You should now be able to:

- explain how customer exits are organized and basic principles of how they are used

Lesson: Enhancement Management

43 Lesson Duration: 20 Minutes

Lesson Overview

This lesson explains fundamental aspects of an enhancement project, which you must use to implement existing customer exits.

Lesson Objectives

After completing this lesson, you will be able to:

- This lesson explains fundamental aspects of an enhancement project, which you must use to implement existing customer exits.

-

Business Example

You would like to learn about the principle of an enhancement project in the customer exit environment.

Enhancement Project - Fundamental Principles

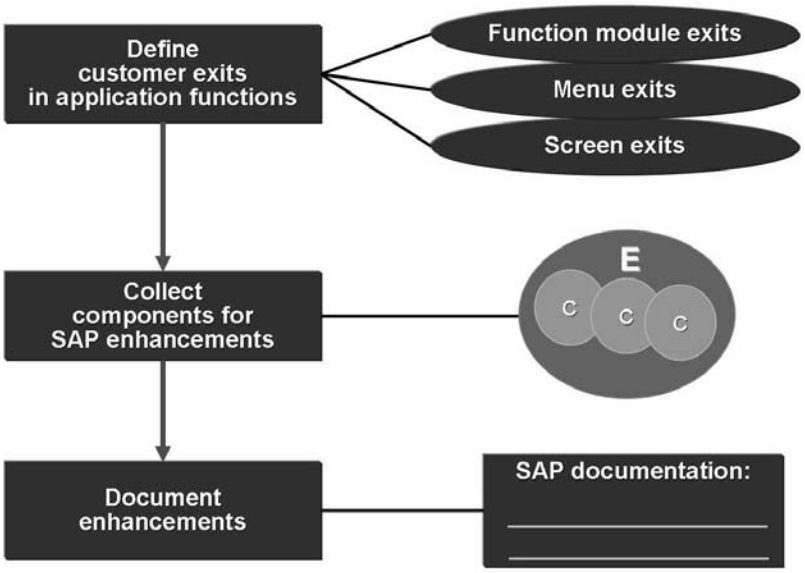


Figure 29: The SAP Enhancement Creation Procedure

The SAP programmer plans possible application enhancements in an application and defines the necessary components. These components are grouped together into SAP enhancements.

The programmer documents enhancements, so that customers can implement the enhancements without having to analyze program or screen source code.

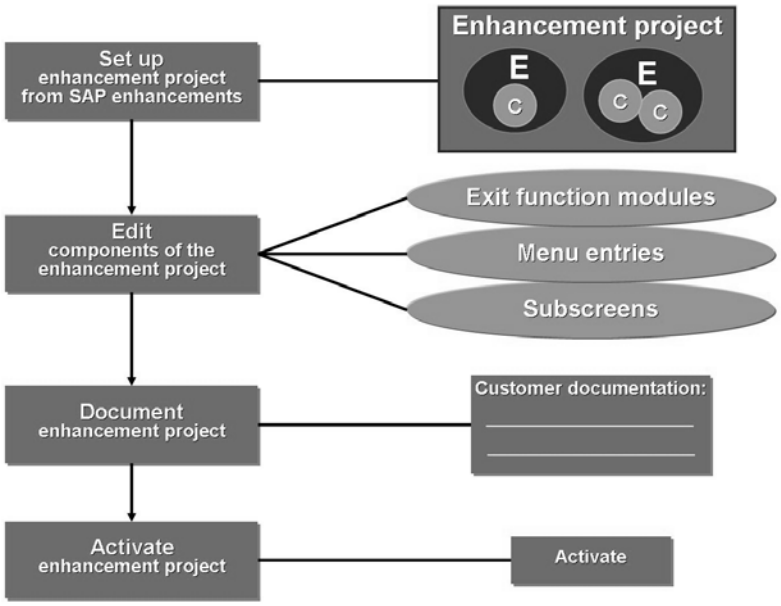


Figure 30: Procedure at the Customer Site

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).

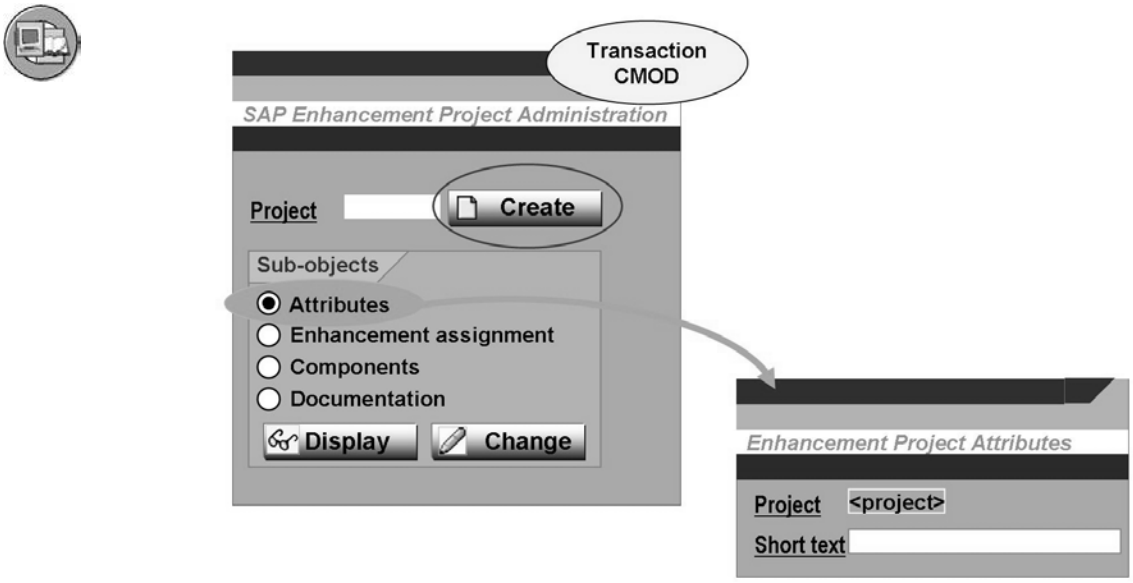


Figure 31: Creating a Customer Enhancement Project

Transaction CMOD starts the project management function. Enter a name for your enhancement project. SAP recommends that you use a naming convention your projects. You can, for example, include the name of the transaction or module pool in the project name. The project name uniquely identifies the enhancement in the system.

Next, go to the project's attributes and enter a short text for the enhancement project. The system enters the remaining attributes (name stamp and time stamp for creating and changing, and status).

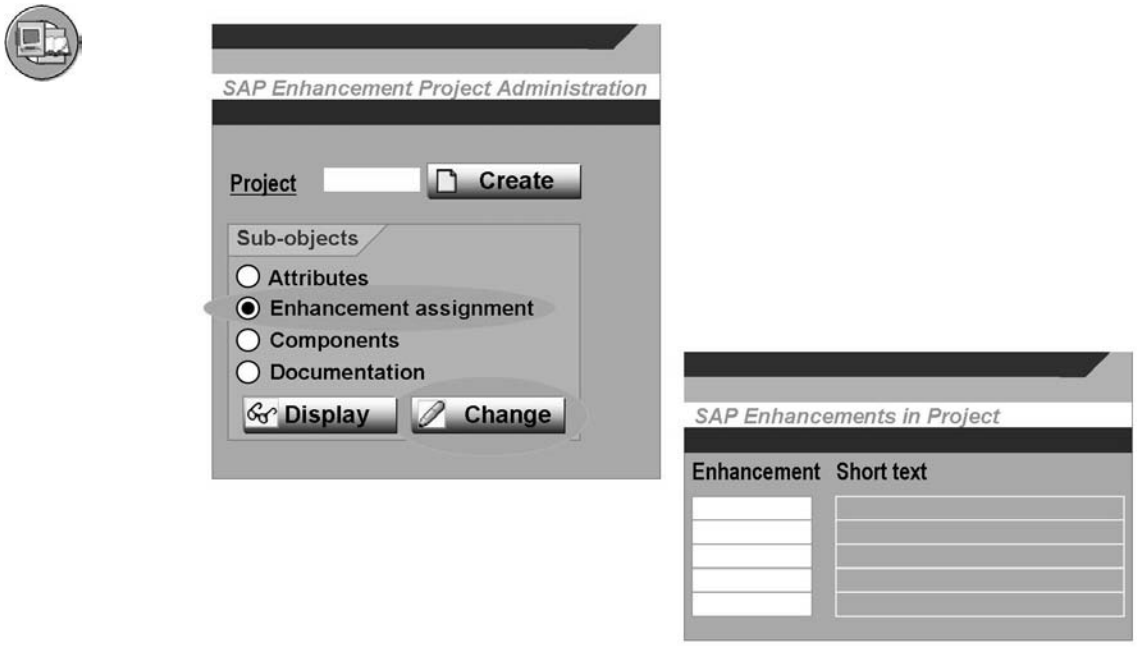


Figure 32: Assigning SAP Enhancements to a Customer Project

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 displays a list of existing SAP enhancements. From here, you can select the enhancements that are of interest to you.

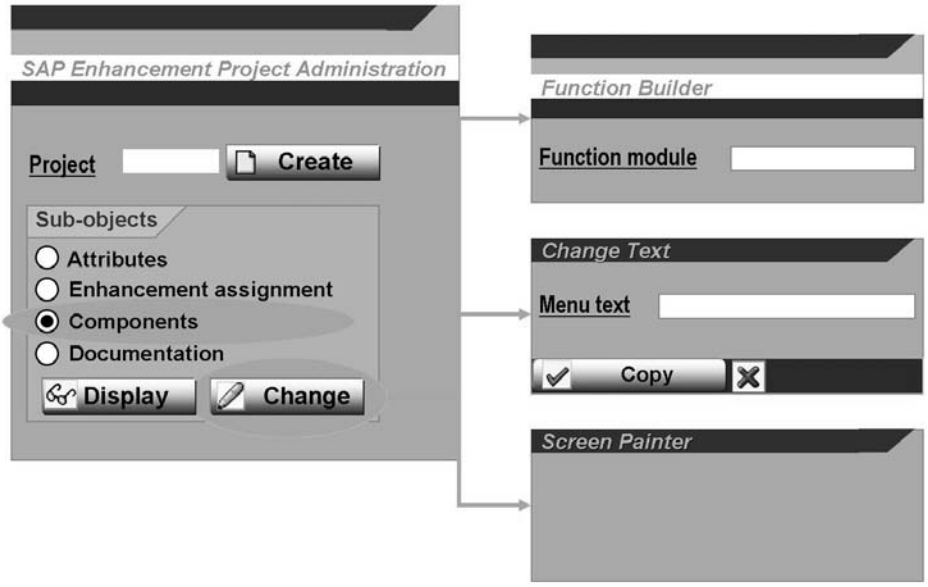


Figure 33: Editing Components

Use the project 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.

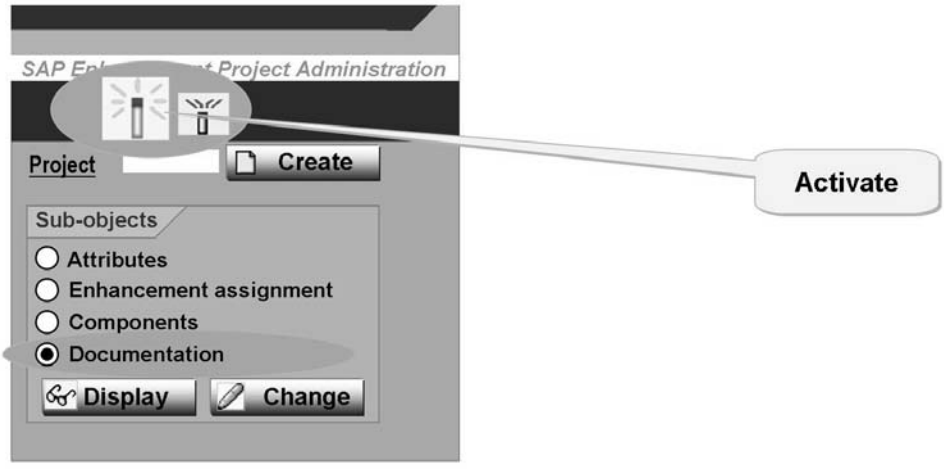


Figure 34: Activating Enhancement Projects

Activating an enhancement project affects all components. Once the project has been successfully activated, it has the status **active**.

When the project is activated, all programs, screens, and menu interfaces that contain components belonging to the project are also regenerated (programs are only regenerated when they are started). After activation, the enhancements are visible in the application functions.

You can use the **Deactivate** function to reset an active enhancement project's status to **inactive**.

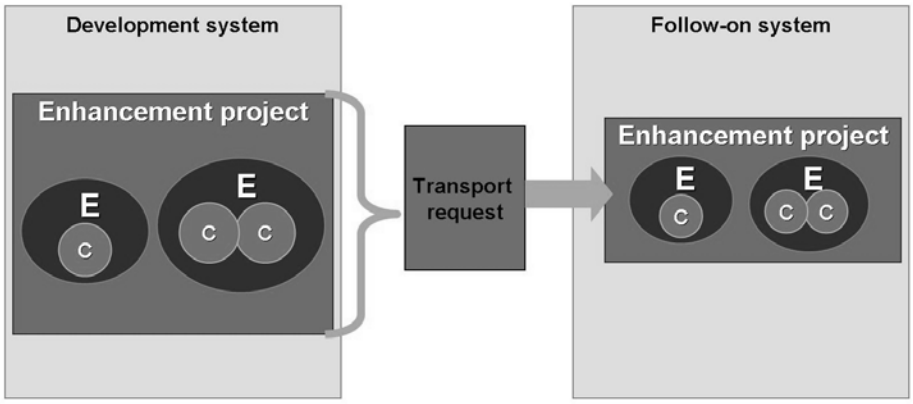


Figure 35: Transporting Projects

To ensure that your enhancement project is transported consistently, assign both the enhancement project itself and all its component parts (include programs, subscreens and includes with subscreen module pools) to one or more tasks within a single change request.

Facilitated Discussion

-

Discussion Questions

Use the following questions to engage the participants in the discussion. Feel free to use your own additional questions.

-

Lesson Summary

You should now be able to:

- This lesson explains fundamental aspects of an enhancement project, which you must use to implement existing customer exits.

Lesson: Program Exit

50

Lesson Duration: 40 Minutes

Lesson Overview

This lesson shows you how a program exit that is implemented using a customer exit works, and outlines possible uses.

Lesson Objectives

After completing this lesson, you will be able to:

- Find program exits that are implemented using customer exits, and use them to enhance functions

-

Business Example

You would like to find program exits that are implemented using customer exits, and use them to enhance functions.

Customer Exits: Program Exit

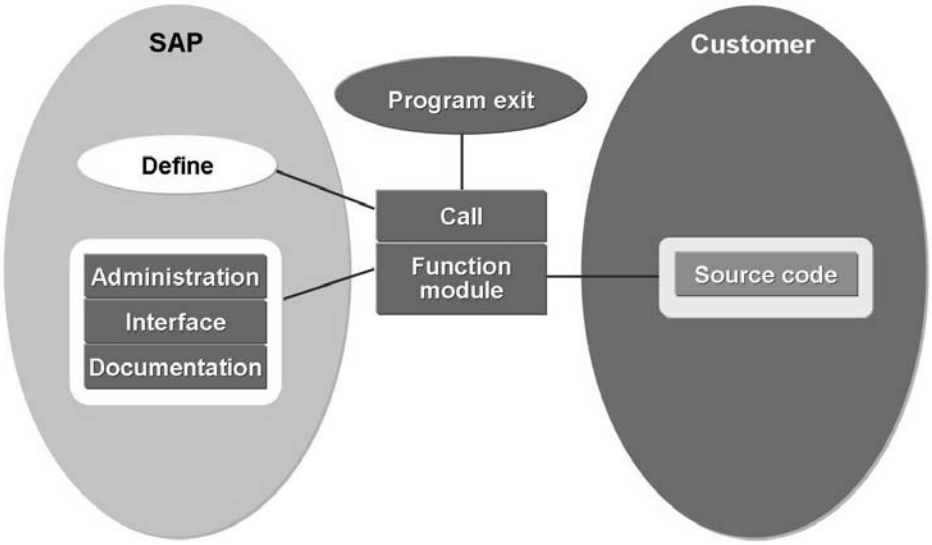


Figure 36: Overview of Program Exit

Program exits allow customers to implement additional logic in application functions. SAP programmers define where program module exits are inserted and what kind of data they transfer. SAP programmers also create the corresponding function modules together with a short text, an interface, and documentation, and describe the functions assigned to a program exit 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 only processes your ABAP code when you activate the enhancement project (of which your function module is a component). Program exits have no effect prior to this.

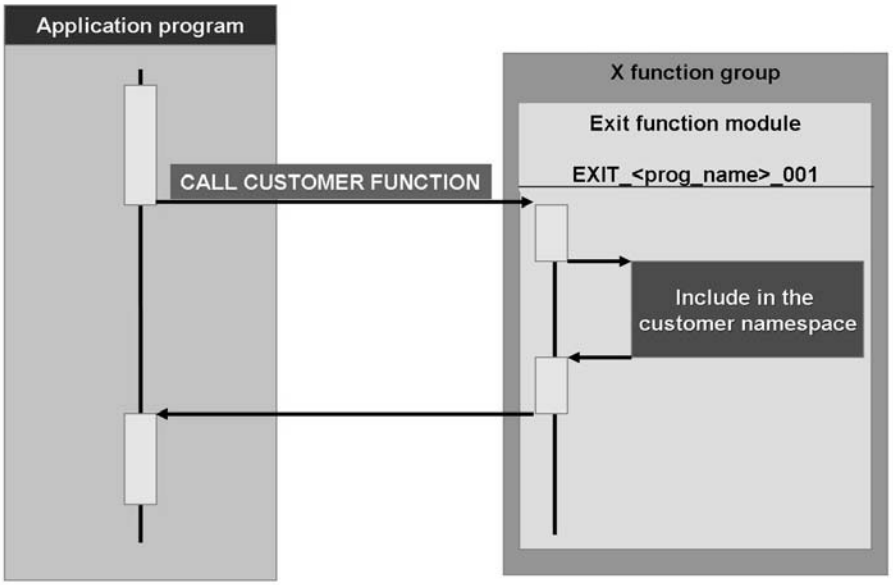


Figure 37: Program Exit: Architecture

This graphic shows the flow of a program that provides 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 developer. Within the function module, users can add a function to the customer namespace using an include.

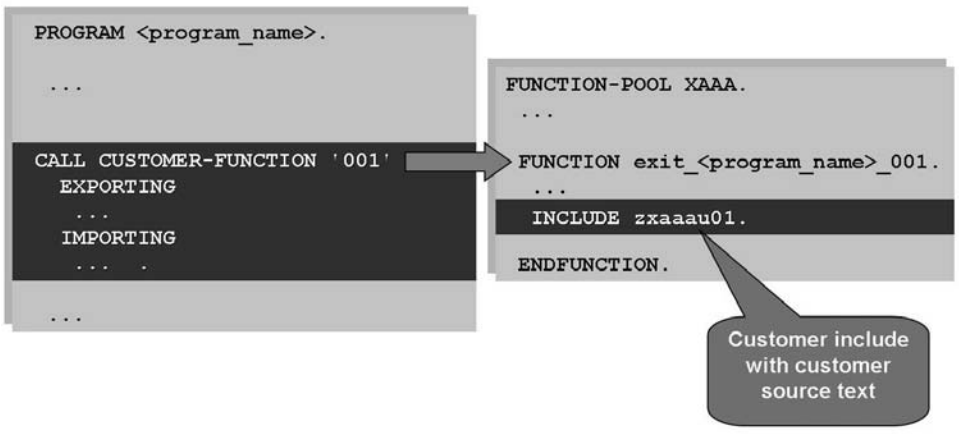


Figure 38: Program Exit: Syntax

The SAP programmer defines the function module call using the ABAP statement CALL CUSTOMER-FUNCTION “nnn”. “nnn” represents a three-digit number. The SAP programmer also creates 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: 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.

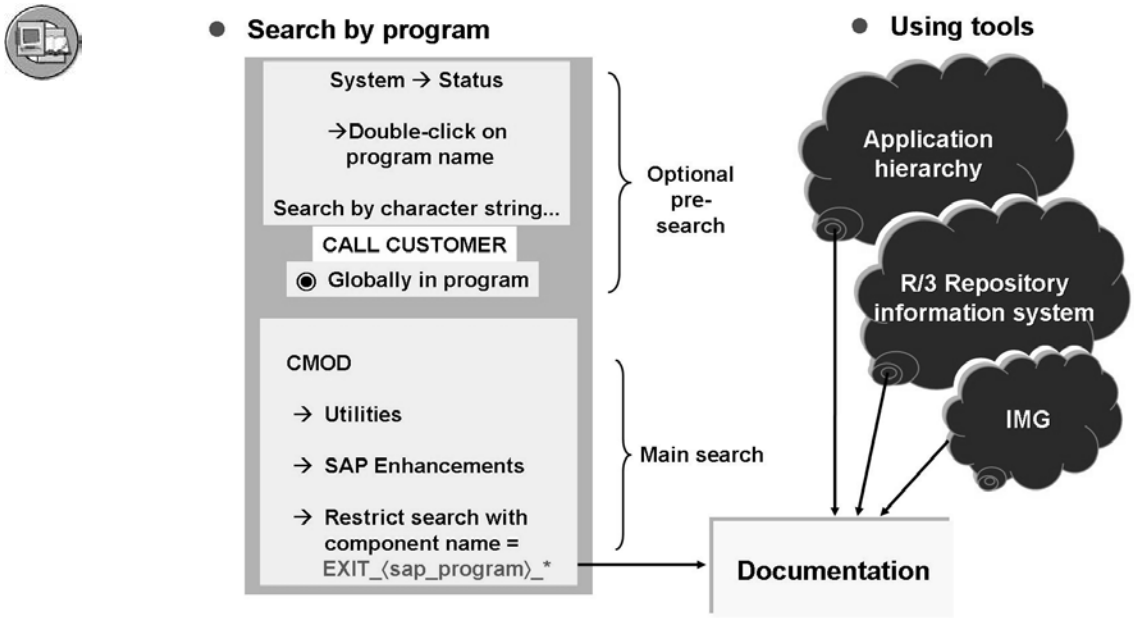


Figure 39: Finding Program Exits

How can you see if an application program contains a program exit? This is a key question for all kinds of enhancement. There are a number of ways of establishing this.

To quickly identify if an application program offers a function module exit, you can use the option displayed on the left-hand side of the graphic: Navigate through the program as described above, to find a certain character string. Select *System* →

Status, to display the the name of the application program. In our example, a suitable character string would be “CALL CUSTOMER”. Conduct the search globally in the program. If your search does not provide any results, you can define a larger search area: Determine the environment for the corresponding program and look for the specific character string in the program environment.

The right side of the graphic shows the search tools you can use to find the name of the required enhancement. You can restrict the search in the Repository Information System using different criteria: These are:

- Package (try generic entries as well)
- Technical name of the enhancement

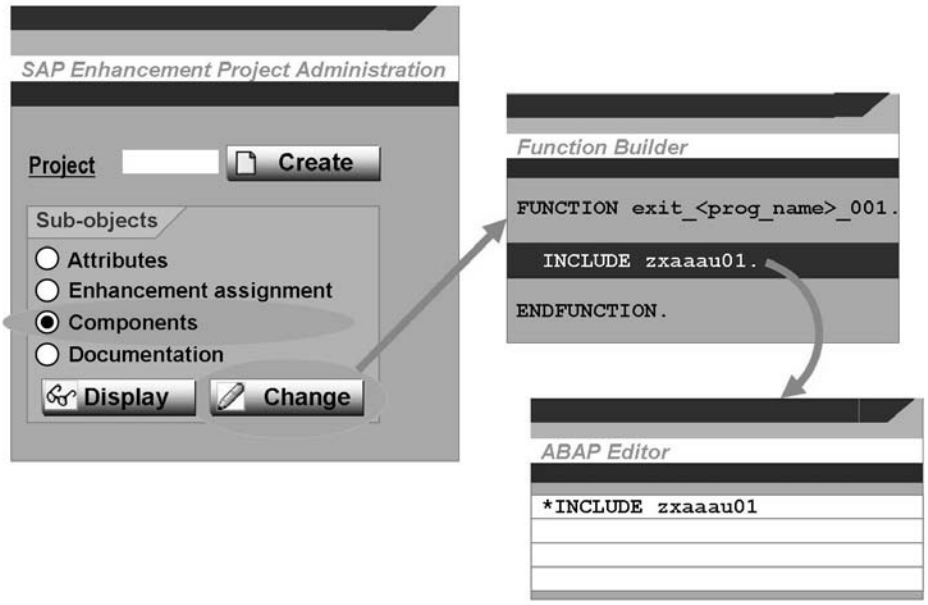


Figure 40: Editing Program Exits

Show your students a simple transaction, for example, BC425_00. Identify the need for a customer enhancement.

Show how to find the corresponding enhancement in the Repository Information System.

Go back to the enhancement project, which you created before. Explain in detail the concept of the function module exit. Again, emphasize that the interface of the function module is defined by the SAP developer and cannot be changed. Outline the motives behind your demonstration: You want to check whether the user has chosen

"AA" and "0017". In this case, you want to send an information message. Assign the enhancement SBC00E01 to your project. Edit your component. Activate your project. Run transaction BC425_00 to show how the enhancement works.

Use the project management function (transaction CMOD), to edit a function module for a program exit.

Select 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 the include name. This automatically takes you to the editor of the include program, where you can enter your source code.

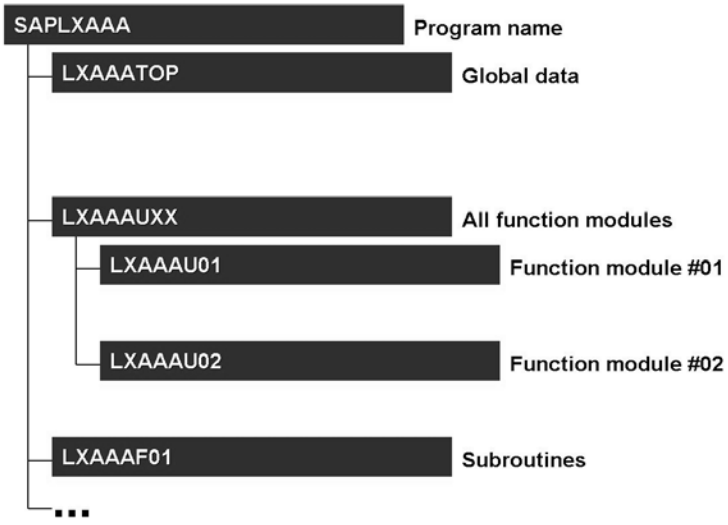


Figure 41: Structure of a Function Group

To understand how an X function group works, you need to understand how a normal function group is structured:

- 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.
- You can freely choose the names of the includes for all other objects (for example, subroutines, modules, and events). However, we advise you to accept the proposed names.

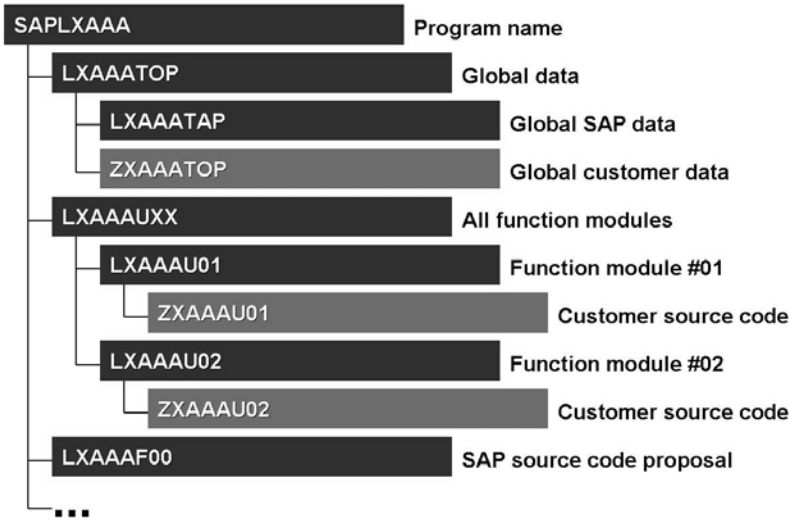


Figure 42: Structure of an Exit Function Group

Exit function groups created by SAP programmers contain includes that begin with either “LX” or “ZX”. You can only edit includes that start with 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 module.

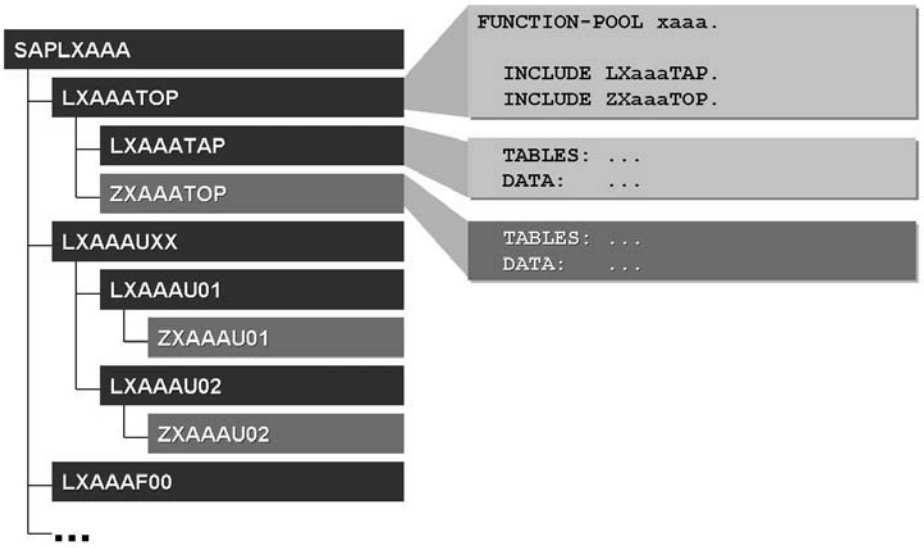


Figure 43: Global Data of an Exit Function Group

SAP programmers can declare global data in the include LXaaaTAP.

You can declare your global data in the include ZXaaaTOP.

The include LXaaaTOP 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).

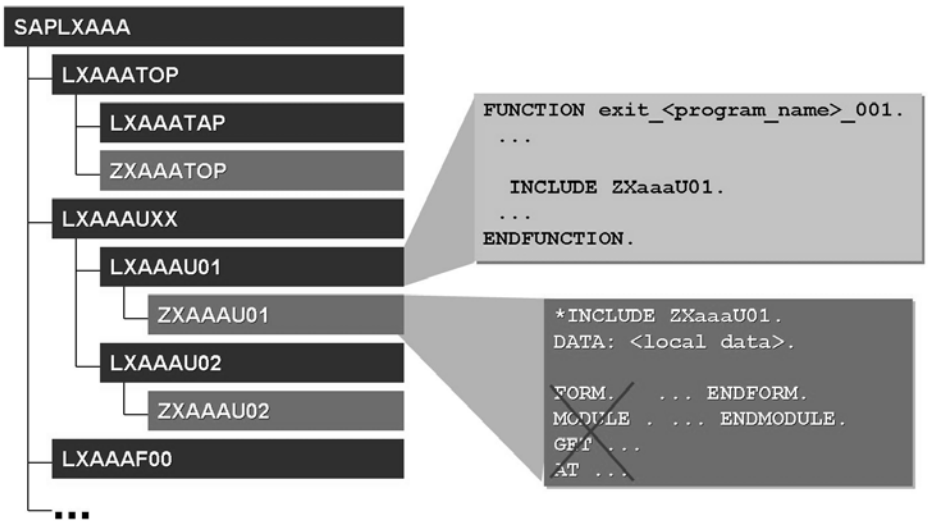


Figure 44: Customer-Specific Processing Blocks

The INCLUDE statement for program ZXaaaUnn is contained in a FUNCTION - ENDFUNCTION block. Because of this, neither events, nor subroutines (FORM) or modules (MODULE) are allowed here. They need to be created in separate includes, a process that will be discussed at a later stage. Data declarations made here with DATA are valid locally in this function module.

The SAP 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). The developer documents this include in 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.

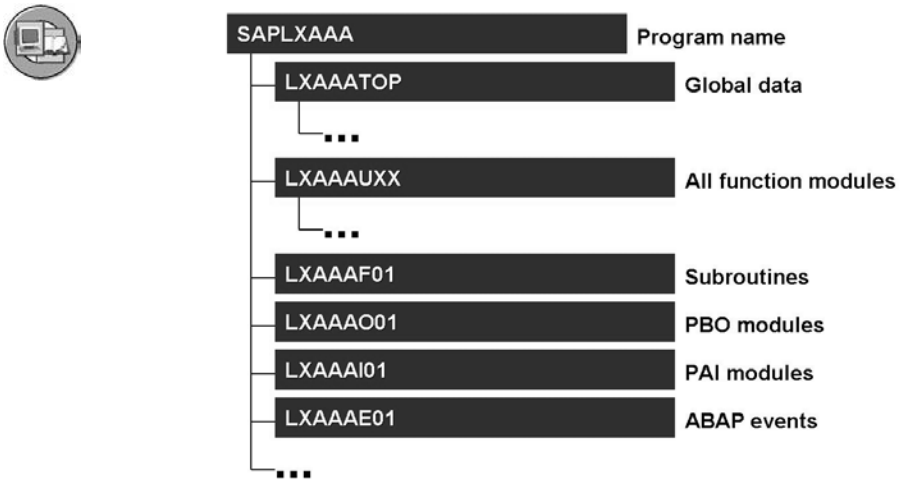


Figure 45: Other SAP Objects in an Exit Function Group

SAP programmers can supply you with default subroutines in the include LXaaaF01.

There could be further includes that contain 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.

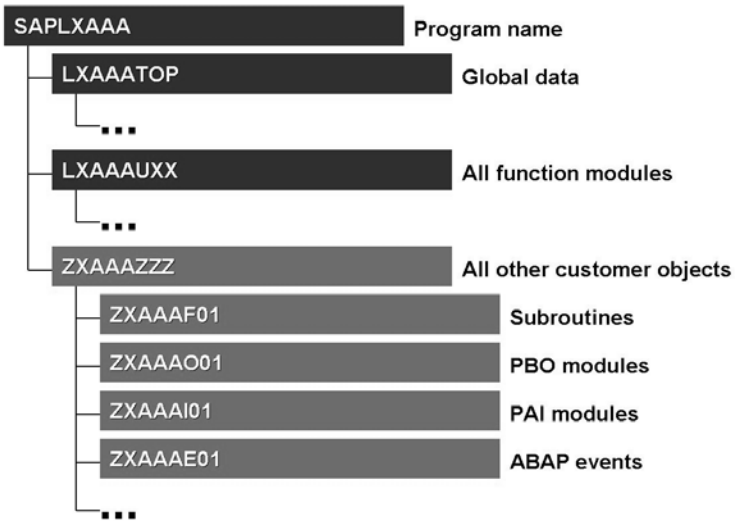


Figure 46: Customer Objects in an Exit Function Group

You create subroutines, modules, and interactive events (AT) using your own includes that are incorporated using the include ZXaaaZZZ.

Additional includes must adhere to the following naming convention:

- ZXaaaFnn for subroutines
- ZXaaaOnn for PBO modules
- ZXaaaInn for PAI modules
- ZXaaaEnn for events

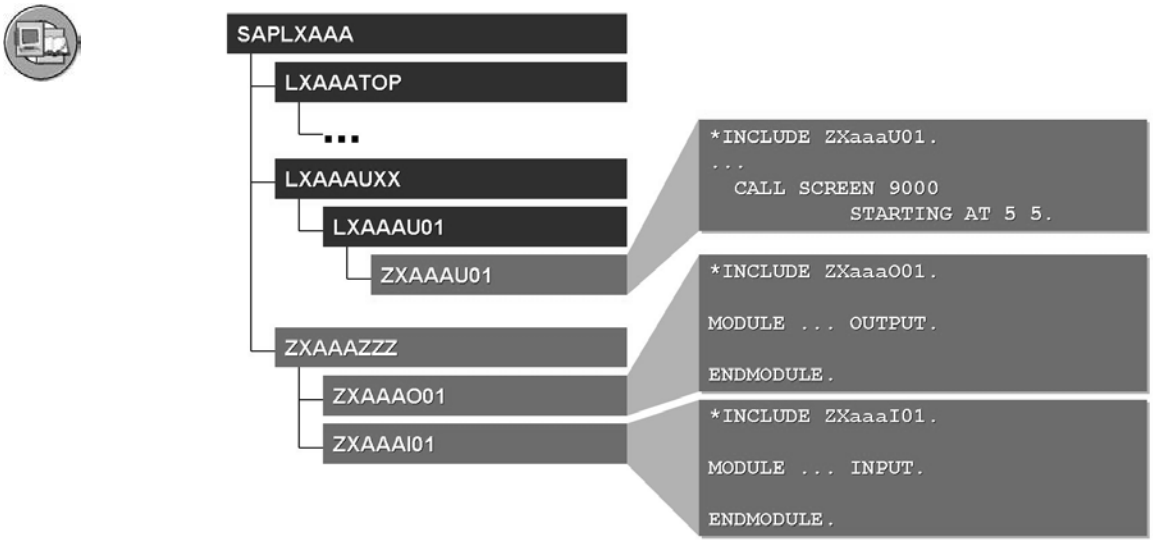


Figure 47: Customer Screens

Work on your demo in parallel to the slides: Insert a PERFORM ROUTINE statement in the customer's include. Double-click the name of the subroutine to create it. Insert a CALL SCREEN 9000 statement into the enhancement. Create the screen by doubleclicking and inserting some text. Create some modules for the screen. All these objects can be empty: You only demonstrate that the objects you create are created in includes in the customer name range.

You can use CALL SCREEN, to call your own screens. Create the related include programs for the PBO and PAI modules in the include program ZXaaaZZZ.

Use forward navigation (select an object and then double-click 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 the include ZXaaaO01; the PAI modules are located in the include ZXaaaI01.

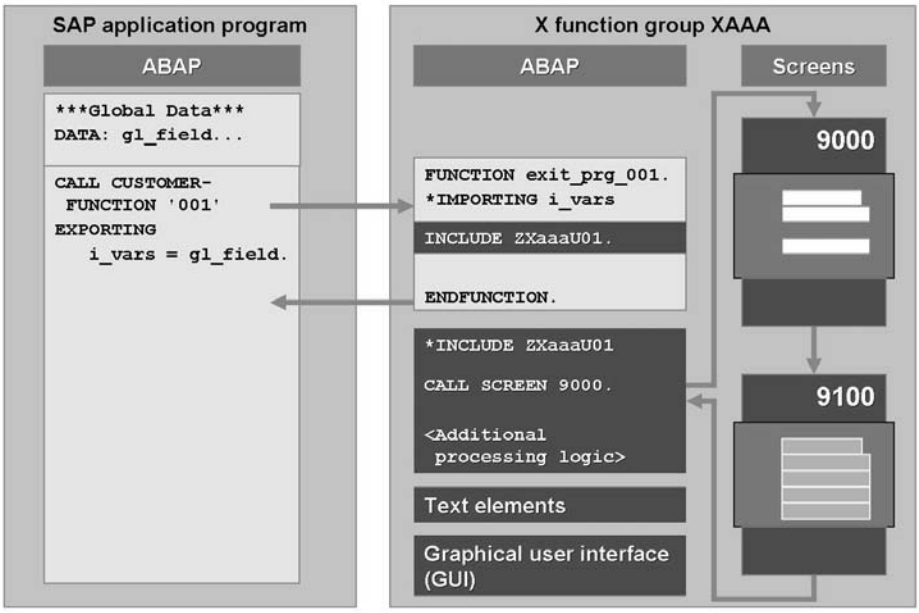


Figure 48: Summary: Program Exits

You can enhance SAP applications by adding your own processing logic at predefined points.

You can include your own screens with the corresponding processing logic and graphical user interface in these enhancements, and create your own text elements.

61

Exercise 2: Program Exit

Exercise Duration: 30 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Implement an enhancement using a program exit

Business Example

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.

Modify 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.

Task 1:

Check if it is possible to enhance the transaction.

1. Did the SAP developer implement a customer exit for the given transaction that you can use to add the required functions?
2. What is the name of their corresponding enhancement? Choose the enhancement with which you can implement a supplementary check when you leave the first screen of the transaction.

Task 2:

Implement the enhancement.

1. Name the enhancement project **TG##CUS1**.
2. Program the following check: Check if the date that was entered is before today's date (that is, if it lies in the past). If this is the case, issue a warning with an appropriate text. To do this, use the message **011** from the message class **BC425**.
3. Check your results.

Solution 2: Program Exit

Task 1:

Check if it is possible to enhance the transaction.

1. Did the SAP developer implement a customer exit for the given transaction that you can use to add the required functions?

   a) Select *System → Status*, to display the name of the corresponding program (**SAPBC425_FLIGHT##**).

2. What is the name of their corresponding enhancement? Choose the enhancement with which you can implement a supplementary check when you leave the first screen of the transaction.

   a) You now have several ways to look for customer exits: You can either search for the character string **CALL CUSTOMER-FUNCTION** globally in the program or you can use the Repository Information System, to search for enhancements that contain the program name in the technical name of the component (use **EXIT_SAPBC425_FLIGHT##_*** in the component name, to restrict the search. The enhancement you were looking is called **SBC##E01**. The documentation for the enhancement shows that it is intended for supplementary checks of the first screen of the transaction.

Task 2:

Implement the enhancement.

1. Name the enhancement project **TG##CUS1**.

   a) Choose transaction CMOD, to implement the enhancement.

   To go to the transaction CMOD, choose *Tools → ABAP Workbench → Utilities → Enhancements → Project Management* in the menu. Create a project called **TG##CUS1** and save it.

2. Program the following check: Check if the date that was entered is before today's date (that is, if it lies in the past). If this is the case, issue a warning with an appropriate text. To do this, use the message **011** from the message class **BC425**.

   a) Include the enhancement **SBC##E01**, which you find, in your project.

Continued on next page

3. Check your results.

   a) Edit the components. Double-click to open the source code for the exit function module. 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.

Lesson Summary

You should now be able to:

- Find program exits that are implemented using customer exits, and use them to enhance functions

Lesson: Menu Exit

65 Lesson Duration: 30 Minutes

Lesson Overview

This lesson demonstrates how menu exits that are implemented using customer exits work. You will also learn how to find and use them.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain how menu exits that are implemented using customer exits work
- Find and use these menu exits

-

Business Example

You would like to find and use menu exits that are implemented using customer exits.

Customer Exits: Menu Exit

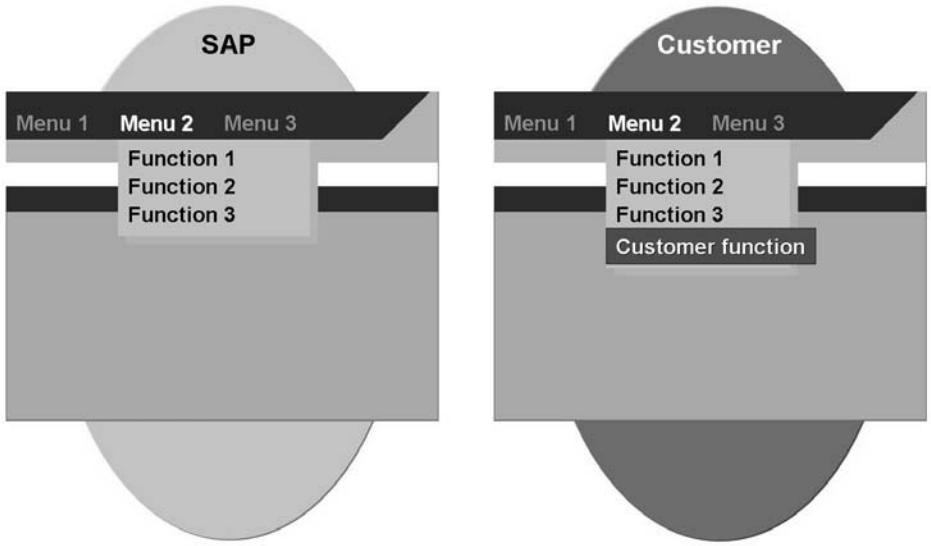


Figure 49: Overview of Menu Exits

Menu exits allow you to attach your own functions to menu options in SAP menus. SAP programmers reserve certain menu entries in your GUI interface for this. You can specify the corresponding entry text yourself.

Once you activate menu exits, they become visible in the SAP menu. When you choose the corresponding menu option, the system switches to a program exit that contains your customer-specific functions.

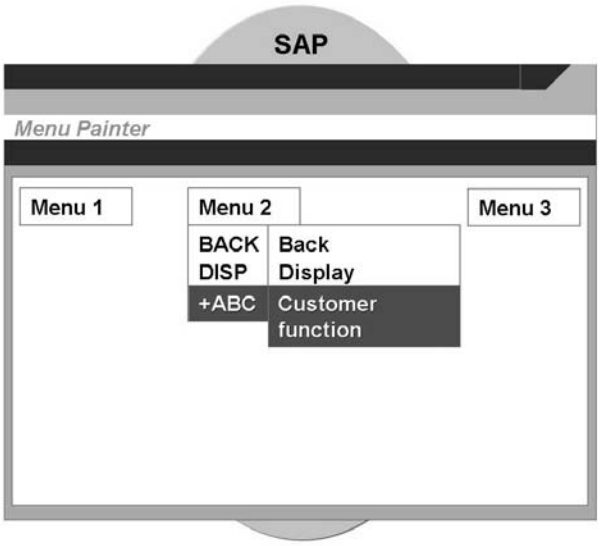


Figure 50: Menu Exit Requirements

To implement menu exits, SAP programmers must equip the GUI interface of your application program with function codes that begin with a “+” sign.

These function codes are inactive at firstand do not appear on the screen. It only becomes visible when it has been activated.

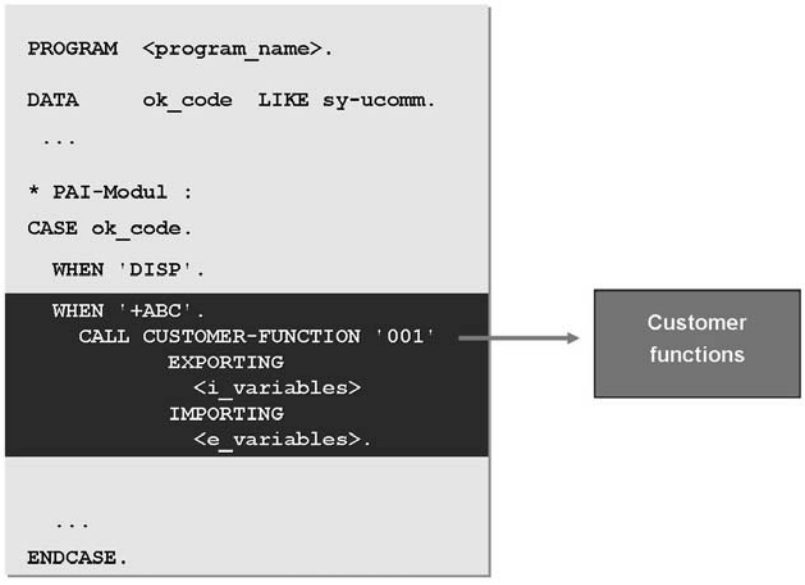


Figure 51: Menu Exits and Program Exits

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

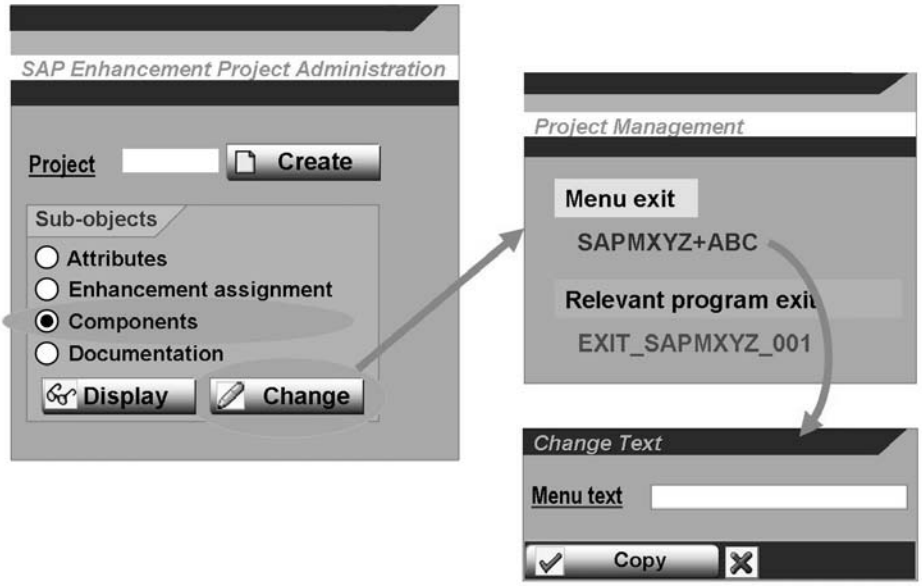


Figure 52: Naming and Editing Menu Exits

First, show your students a menu for transaction BC425_00. Go to the function key overview where you can find all function keys that belong to that program. Now, show your students the enhancement containing both a menu exit and a function module exit (SBC00E02). Implement the enhancement: Change the text of the menu. Create the include in the exit function module. Simply send an appropriate information message. The example demonstrates the high degree to which these two kinds of enhancements are interconnected. Course participants should now understand why an additional hierarchy level for enhancements must exist above the hierarchy level for components.

Menu exits are edited using the project management transaction (CMOD).

The button for editing components calls a dialog box where you can enter short descriptions and choose a language for each additional menu entry.

Hint: You may not make any changes to the GUI interface.

69

Exercise 3: Menu Exit

Exercise Duration: 30 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Implement an enhancement using a menu exit and a program exit.

Business Example

Your co-workers are thrilled with the new functions that you have built into the system. 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 called **SAPBC425_BOOKING_##**. However, users had to call it separately and enter current data into the program selection screen.

Task 1:

Examine transaction **BC425_##**. Is it possible to call the program using a menu entry in the transaction?

1. Did the SAP developer implement a customer exit for the transaction that you can use to add the required functions?
2. What is the name of the corresponding enhancement? Choose the enhancement with which you can implement a menu enhancement.

Task 2:

Implement the enhancement.

1. Name the enhancement project **TG##CUS2**.
2. Edit the components of the enhancement. Start the program specified above by choosing the additional menu entry. Return to transaction **BC425_##** again when you leave the list.
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.

Continued on next page

Check your results.

Solution 3: Menu Exit

Task 1:

Examine transaction **BC425_##**. Is it possible to call the program using a menu entry in the transaction?

1. Did the SAP developer implement a customer exit for the transaction that you can use to add the required functions?

   a) Examine the transaction as in the last exercise. It is useful to search using the Repository Information System or transaction CMOD.

   b) Search for an enhancement that has a component name that contains the program name (restrict your search using the component name **SAPBC425_FLIGHT##***).

2. What is the name of the corresponding enhancement? Choose the enhancement with which you can implement a menu enhancement.

   a) The enhancement you were looking is called **SBC##E02**.

Task 2:

Implement the enhancement.

1. Name the enhancement project **TG##CUS2**.

   a) Choose transaction CMOD, to implement the enhancement.

   b) To go to the transaction CMOD, choose *Tools → ABAP Workbench → Utilities → Enhancements → Project Management* in the menu. Create a project called **TG##CUS2** and save it.

2. Edit the components of the enhancement. Start the program specified above by choosing the additional menu entry. Return to transaction **BC425_##** again when you leave the list.

   a) Include the enhancement **SBC##E02**, which you find, 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.

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.

Continued on next page

Check your results.

a) For group 00, the source text of the include should read as follows:

```
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.

Lesson Summary

You should now be able to:

- Explain how menu exits that are implemented using customer exits work
- Find and use these menu exits

Lesson: Screen Exit

73 Lesson Duration: 30 Minutes

Lesson Overview

This lesson shows you how to find a screen exit that is implemented using a customer exit, and outlines possible uses.

Lesson Objectives

After completing this lesson, you will be able to:

- You would like to find and use screen exits that are implemented using customer exits.

-

Business Example

You would like to find and use screen exits that are available in the SAP standard system.

Customer Exits: Screen Exit

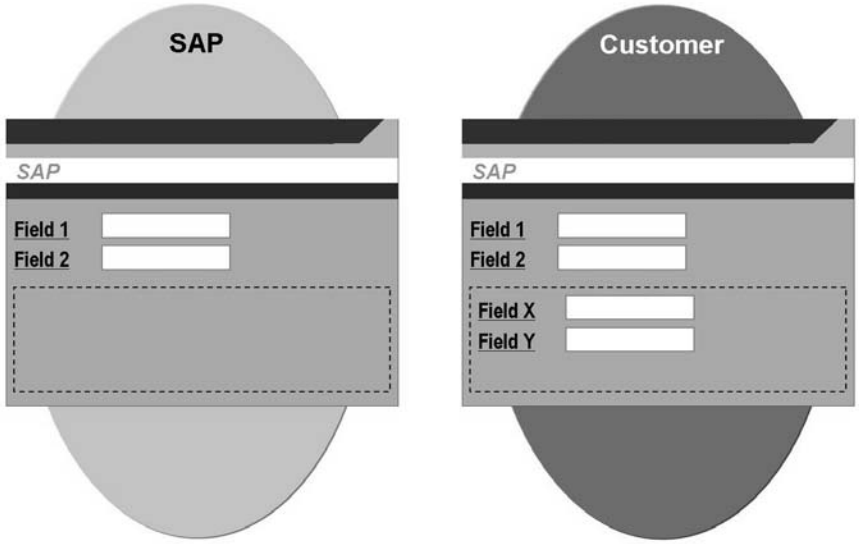


Figure 53: Overview of Screen Exits

Screen exits allow you to make use of sections of a main screen that have been reserved by SAP programmers (subscreen areas). You can either display additional information or input data in these areas. You define the necessary fields on a customer screen (subscreen).

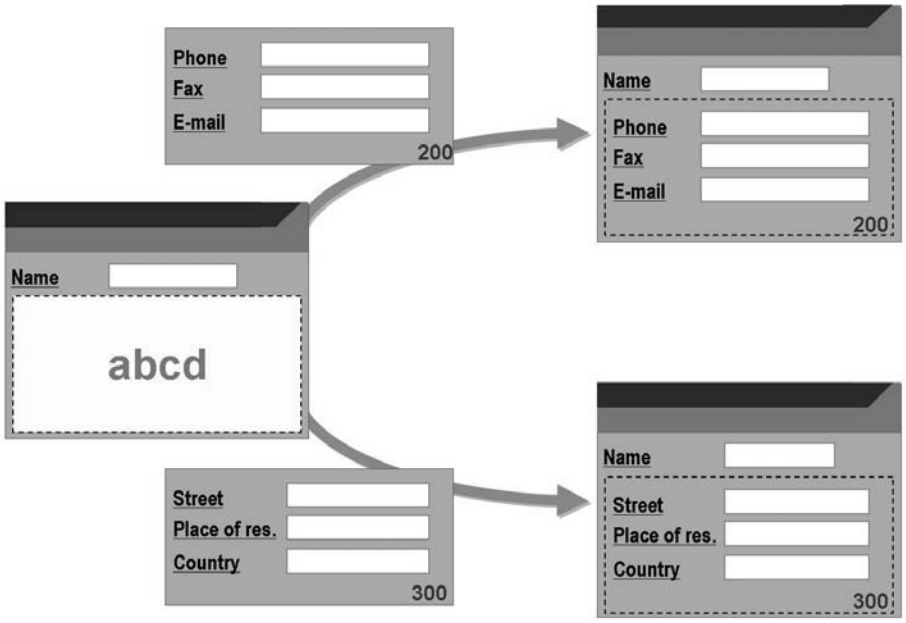


Figure 54: Subscreens (General)

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

- **Flow control on 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.
```

Figure 55: Calling a Normal Subscreen

Your system determines which screen will be displayed in a subscreen area at PBO. 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 the “CALL SUBSCREEN”"calls in the flow logic of the main screen determines the sequence in which the flow logic of the individual subscreen screens is processed.

Caution:

- The function code can only be processed using the main screen
- Subscreens are not allowed assign a name to “their” command field.
- Subscreens cannot define GUI statuses.
- No value for the next screen can be entered in a subscreen’s flow control.

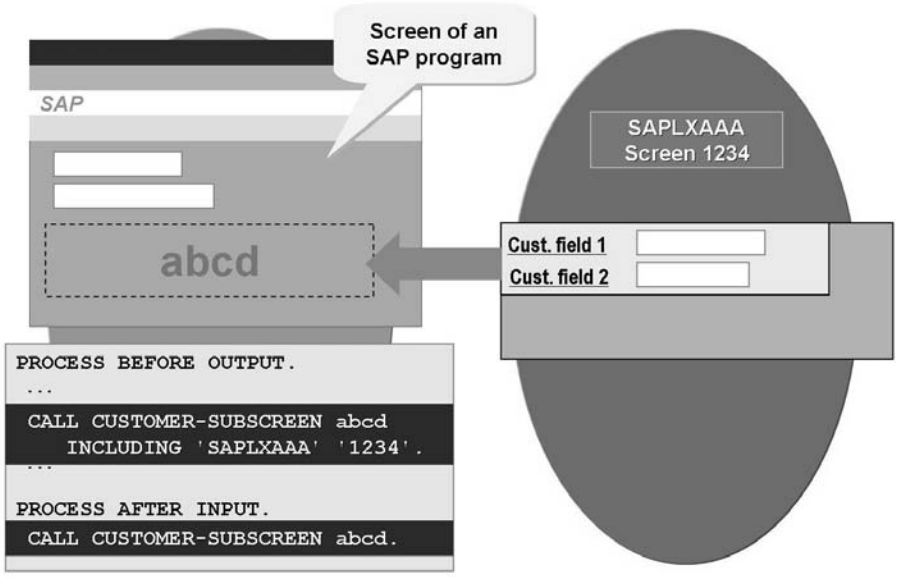


Figure 56: Defining Screen Exits

The SAP programmer can reserve multiple subscreen areas for a screen.

The subscreen is called during flow control of the main screen using 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.

The call only becomes effective after a corresponding subscreen has been created in an enhancement project, and this project has been activated.

You create subscreens in an X function group. Normally, these function groups also contain exit function modules.

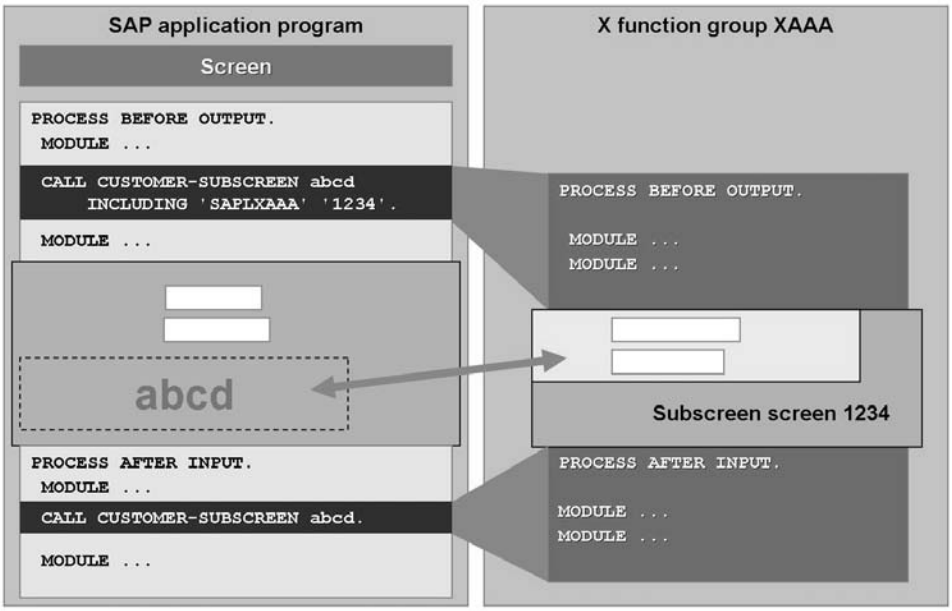


Figure 57: Calling Customer Subscreens

Show your students screen 200 of transaction BC425_00. Navigate to the Screen Painter to show the flow logic and the subscreen area. Now, search for the corresponding SAP enhancement SBC00E03. Create a new project. Assign the SAP enhancement to the project. You can now edit the components. First, create the subscreen. Place fields of table SFLIGHT00 to the subscreen. Go back and activate your project. Show the result: No data is displayed in the new fields. This will be the next step.

Whenever the statement CALL CUSTOMER-SUBSCREEN <subscreen-area> INCLUDING <X-Function-Pool> <dynpro_no> occurs at PBO in the flow control of a screen, a subscreen is included in the subscreen area defined by SAP 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 <subscreen-area>.

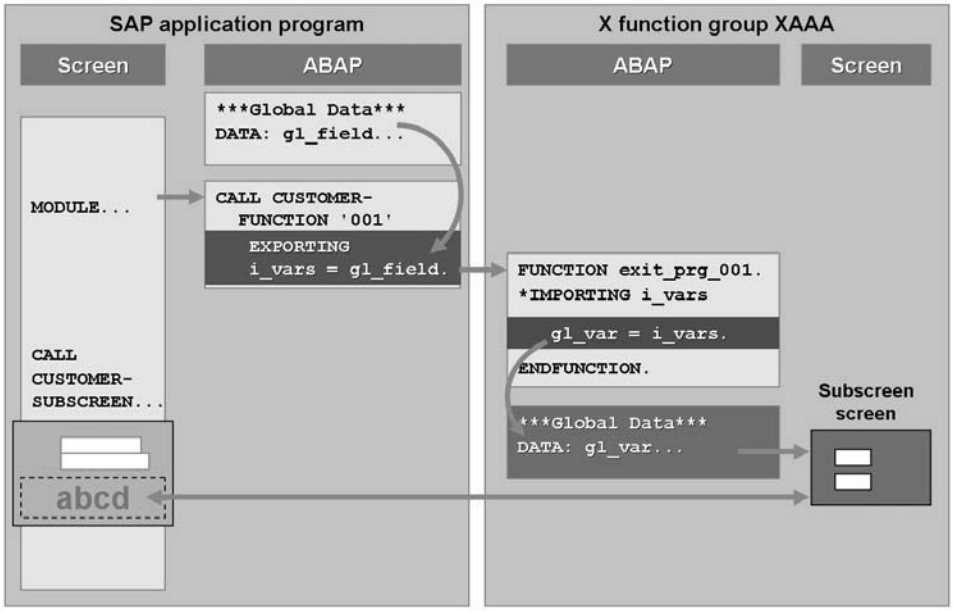


Figure 58: Transporting Data to Subscreens

The global data of the calling program is not known to the X function group that contains your subscreen. SAP programmers use program exits to explicitly provide this data to subscreens.

In order to facilitate data transport, modules are called in the flow control of the calling program that contain program exits for transferring data using interface parameters.

The corresponding exit function modules are in the same X function group in which you must create the customer subscreen.

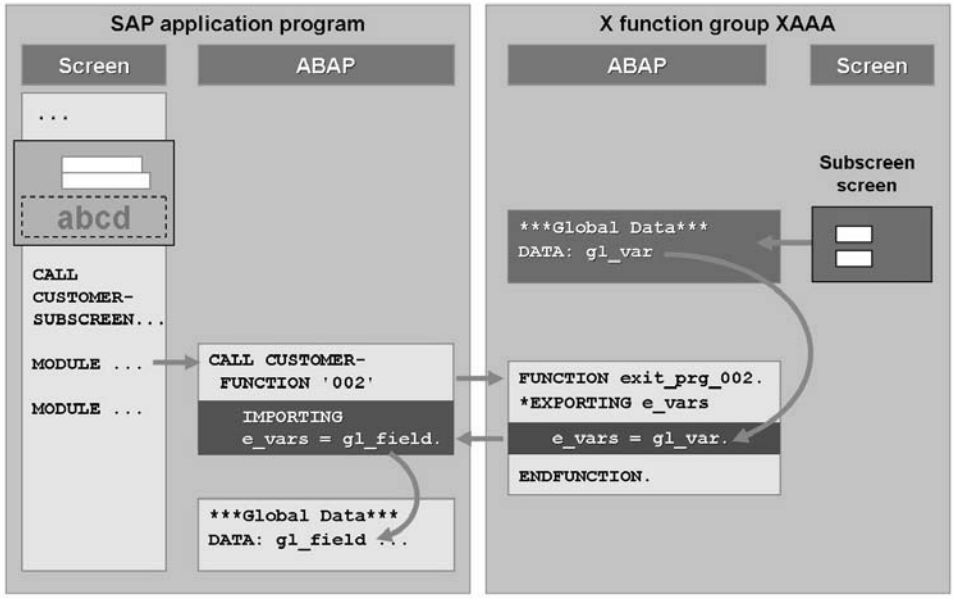


Figure 59: Transporting Data from Subscreens

Continue working on your demo: Implement the function module exit and bring the data to the subscreen.

Data must be transported in the other direction as well, as global data from the X function group that contains the user entries on the subscreen is also not known to the calling program. For this reason, SAP programmers use program exits to return to the calling program any data that was changed in the subscreen.

To do this, the system calls a module at the PAI event of the main screen. This module contains a program exit that can receive the relevant global data for the X function group.

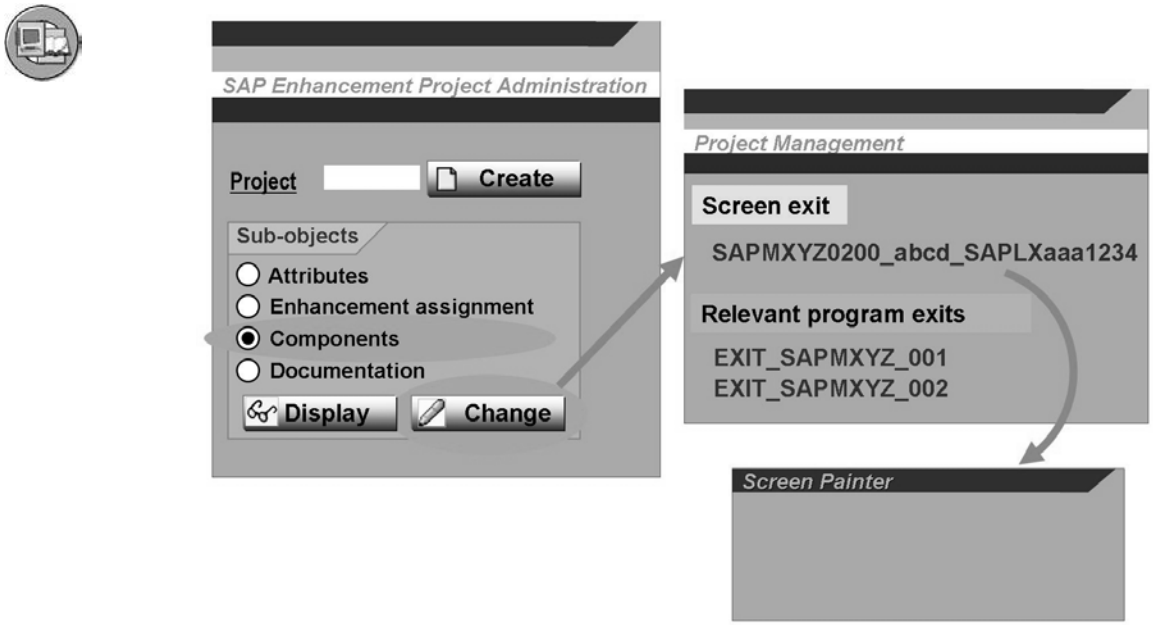


Figure 60: Naming and Editing Screen Exits

Screen exits are edited using the project management transaction (CMOD).

The technical name of a screen exit consists of the name of the calling program, the four-digit 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.

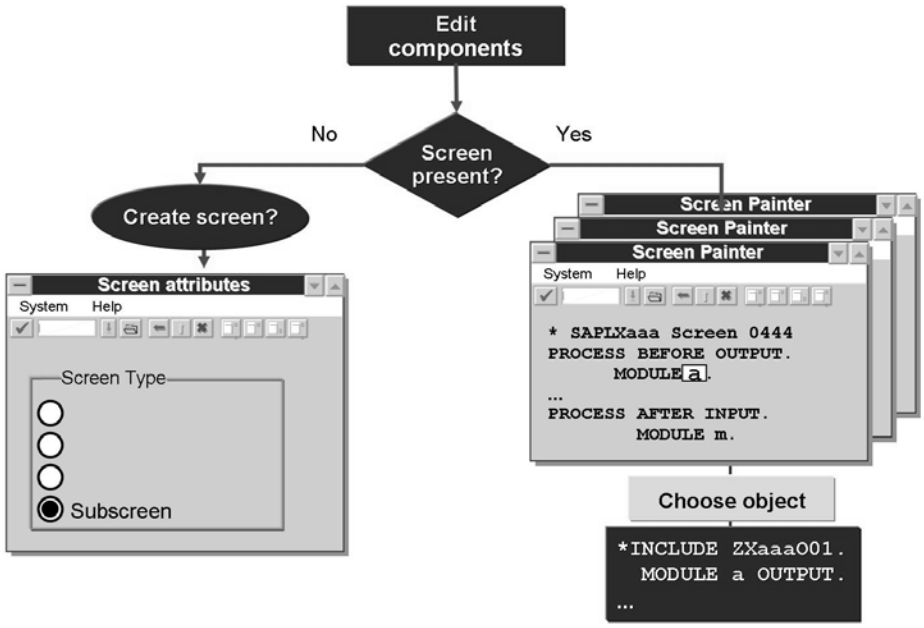


Figure 61: Editing Subscreens

The SAP development environment supports the creation of customer subscreens and the corresponding PBO and PAI modules as a part of forwards navigation.

Hint: When you create the **subscreen**, ensure that it has the subscreen screen type.

You are not allowed to change any of the interfaces in the X function group to which the subscreen and the program exits belong, nor are you allowed to add your own function modules.


83

Exercise 4: Screen Exit

Exercise Duration: 45 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Display further fields on an SAP transaction screen and fill them

Business Example

"It would be really great if the details list of transaction **BC425_##**, which displays 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 new fields to the second screen of this transaction (screen number 200).

Task 1:

What kind of options are there for placing additional fields on a screen? Take a closer look at screen 200 in transaction **BC425_##**, and see if this is possible.

1. Is there a screen exit for enhancing the screen?
2. If this is the case, what is the name of the corresponding enhancement?

Task 2:

Implement the enhancement, to achieve the following (project name: **TG##CUS3**):

1. Add three fields to the screen. The following should appear:

Pilot's name
Meal
The number of free places on the current flight

2. Ensure that the data is correctly transported to the subscreen.

Task 3:

Checking your results

1. To check the results of your work, proceed as follows:

Solution 4: Screen Exit

Task 1:

What kind of options are there for placing additional fields on a screen? Take a closer look at screen 200 in transaction **BC425_##**, and see if this is possible.

1. Is there a screen exit for enhancing the screen?

   a) Look at the transaction screens in the Screen Painter. You will see that screen 200 of transaction **BC425_##** offers a screen exit.

   b) 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.

2. If this is the case, what is the name of the corresponding enhancement?

   a) You can display the name of the enhancement by searching in the Repository Information System, for example (see previous exercises). The name of the enhancement is **SBC##E03**.

Task 2:

Implement the enhancement, to achieve the following (project name: **TG##CUS3**):

1. Add three fields to the screen. The following should appear:

Continued on next page

Pilot's name
Meal
The number of free places on the current flight

a) Implement the enhancement in the same way as described in the previous exercises. Using the transaction CMOD, create a project called **TG##CUS3**. Include the enhancement **SBC##E03** in your project. Edit the components.

b) Use the screen exit to enhance the screen. You can create screen 0500 by double-clicking the corresponding enhancement component. Make sure that you select the screen type "Subscreen". Copy the fields from the corresponding structure **SFLIGHT##** in the Dictionary. You have two options for placing a field on the screen for the free places: You can declare a variable in the TOP include of the X function group, and generate the program. You can then place this program field on the screen. Alternatively, you can enhance your append structure. You should not do this in the exercise as the trainer uses a program to fill the fields of the append structure. Enhancing the append structure could result in errors in this program.

2. Ensure that the data is correctly transported to the subscreen.

a) Use the enhancement's program exit to transport the data correctly. 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 activate your enhancement project.

Task 3:

Checking your results

1. To check the results of your work, proceed as follows:

a) Implement transaction **BC425_##** and check the results of your work.

Lesson Summary

You should now be able to:

- You would like to find and use screen exits that are implemented using customer exits.

Unit Summary

You should now be able to:

- explain how customer exits are organized and basic principles of how they are used
- This lesson explains fundamental aspects of an enhancement project, which you must use to implement existing customer exits.
- Find program exits that are implemented using customer exits, and use them to enhance functions
- Explain how menu exits that are implemented using customer exits work
- Find and use these menu exits
- You would like to find and use screen exits that are implemented using customer exits.

Unit 4

89

Business Add-Ins

Unit Overview

- Searching for Business Add-Ins
- Implementing Business Add-Ins

Unit Objectives

After completing this unit, you will be able to:

- Explain why SAP introduced classic BAdIs as an enhancement in Release 4.6.
- Search for available Business Add-Ins in SAP programs
- Use Business Add-Ins to implement program enhancements
- Explain what an extensible filter type is
- Explain the default and sample code of a BAdI

Unit Contents

Lesson: Business Add-Ins: Purpose

90 Lesson Duration: 5 Minutes

Lesson Overview

In this lesson, you will learn why SAP introduced classic BAdIs as an enhancement in Release 4.6.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain why SAP introduced classic BAdIs as an enhancement in Release 4.6.

-

Business Example

You want to add new functions to a flight maintenance transaction in an SAP system. To minimize the inputs 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.

Purpose of Classic BAdIs

Business Add-Ins: Purpose

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

• Motivate the benefits of Business Add-Ins and compare them to older techniques • Emphasize that BAdIs are the first enhancement technique that can be used to provide enhancements even by CSP solution providers or customers themselves.

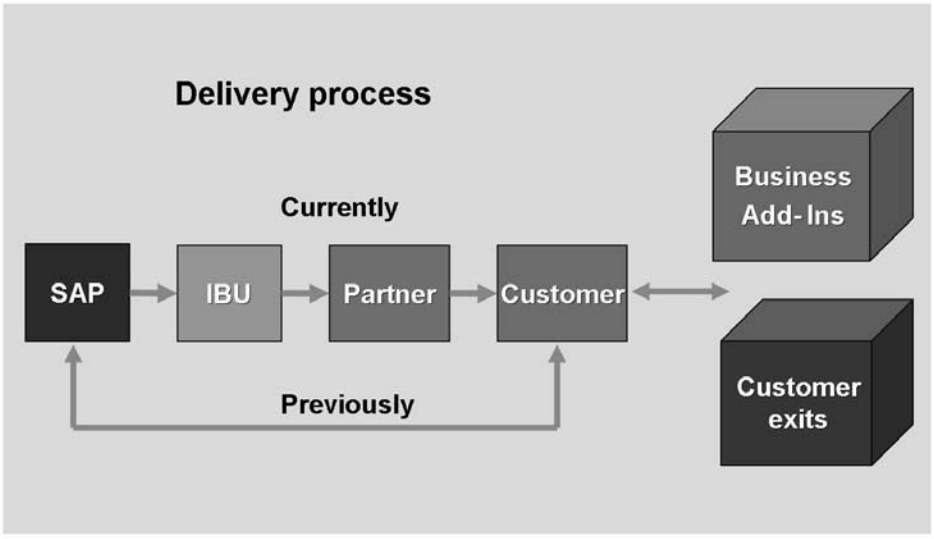


Figure 62: Software Delivery Process

Facilitated Discussion

-

Discussion Questions

Use the following questions to engage the participants in the discussion. Feel free to use your own additional questions.

-

Lesson Summary

You should now be able to:

- Explain why SAP introduced classic BAdIs as an enhancement in Release 4.6.

Lesson: Creating and Implementing a BAdI

93 Lesson Duration: 40 Minutes

Lesson Overview

In this lesson, you will learn how program and menu exits implemented using BAdIs work. You will also find out how to find and use these BAdIs.

Lesson Objectives

After completing this lesson, you will be able to:

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

-

Business Example

You would like to use the classic BAdIs delivered in the standard SAP system, to enhance your SAP software.

Classic BAdIs

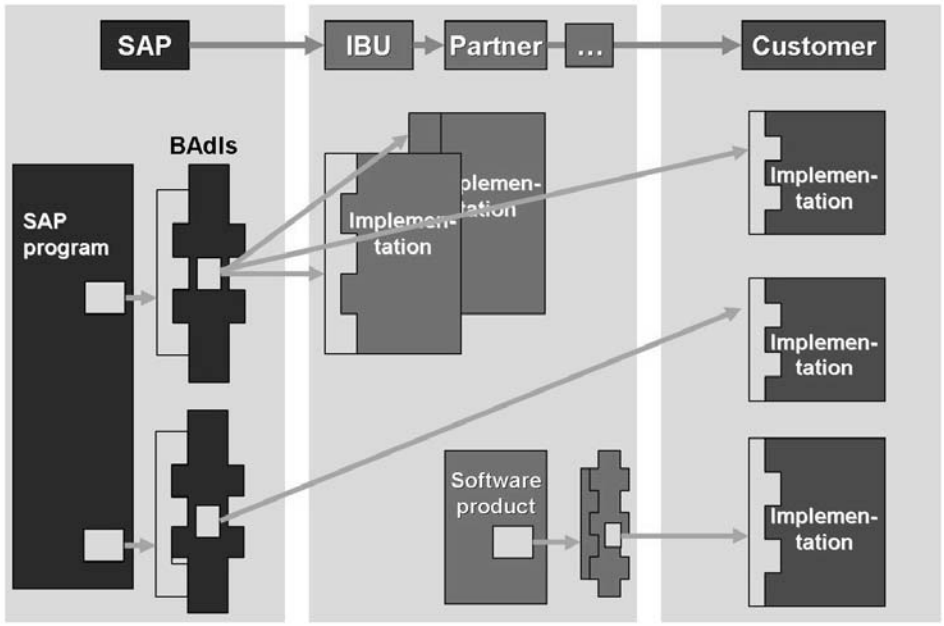


Figure 63: Business Add-Ins: Architecture

Business Add-Ins, unlike customer exits, take into account the changes to the software delivery process. The top part of the graphic illustrates the typical delivery process: It no longer merely consists of provider and user. Instead, it can now contain a whole chain of intermediate providers.

The bottom part of the graphic explains how Business Add-Ins work. Enhancements are made possible by SAP application programs. This requires at least one interface and a BAdI class that implements it. The interface is also implemented by the user.

The main advantage of this concept is its capacity for reuse. A BAdI can be implemented several times, even by the elements on the right of the software delivery chain.

Business Add-Ins also enable each element of the software delivery chain to offer enhancements.

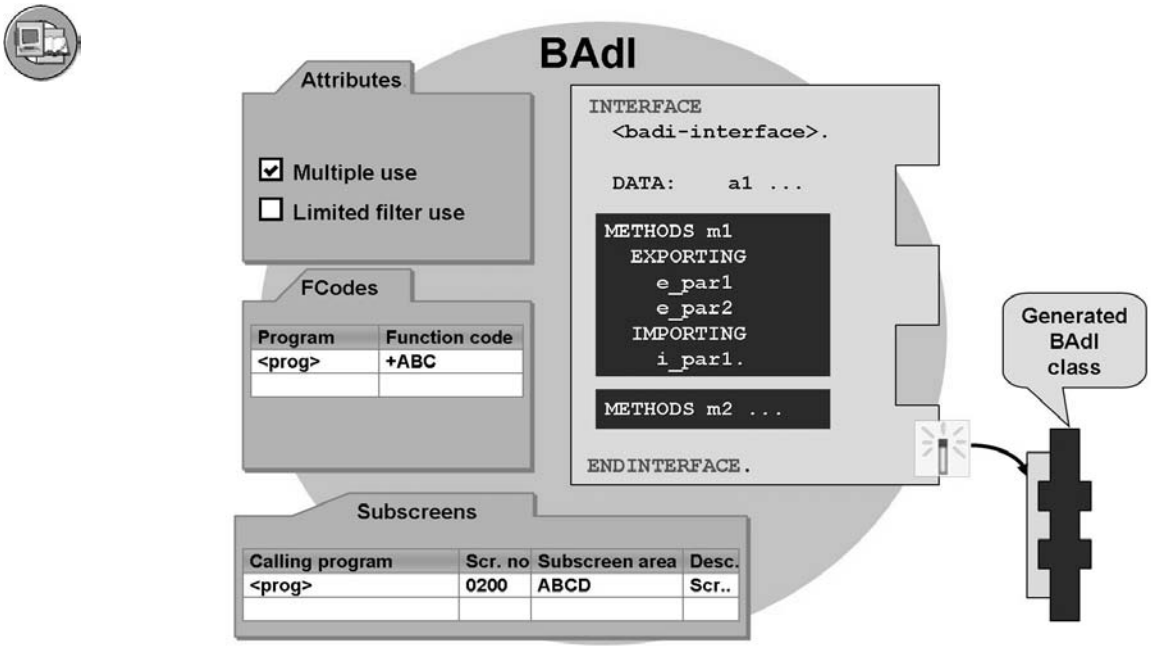


Figure 64: Business Add-Ins: Components

A Business Add-In contains the components of an enhancement. Each Business Add-In can contain the following components:

- Program enhancements: In a 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 method of the generated BAdI class.
- Menu enhancements: Like customer exits, function codes can be entered in a BAdI. The corresponding menu entries are available in the CUA definition and are displayed once the BAdI is implemented.
- Screen enhancements: Like customer exits, screen enhancements that you can implement can be defined in a BAdI.

Several components are created when you define a Business Add-In:

- The interface
- The generated class (BAdI class) that implements the interface

The generated class (BAdI class) performs the following tasks:

- Filtering: If a BAdI should only be implemented under specific conditions, the BAdI class ensures that only the relevant implementations are called.
- Control: The BAdI class calls the active implementations.

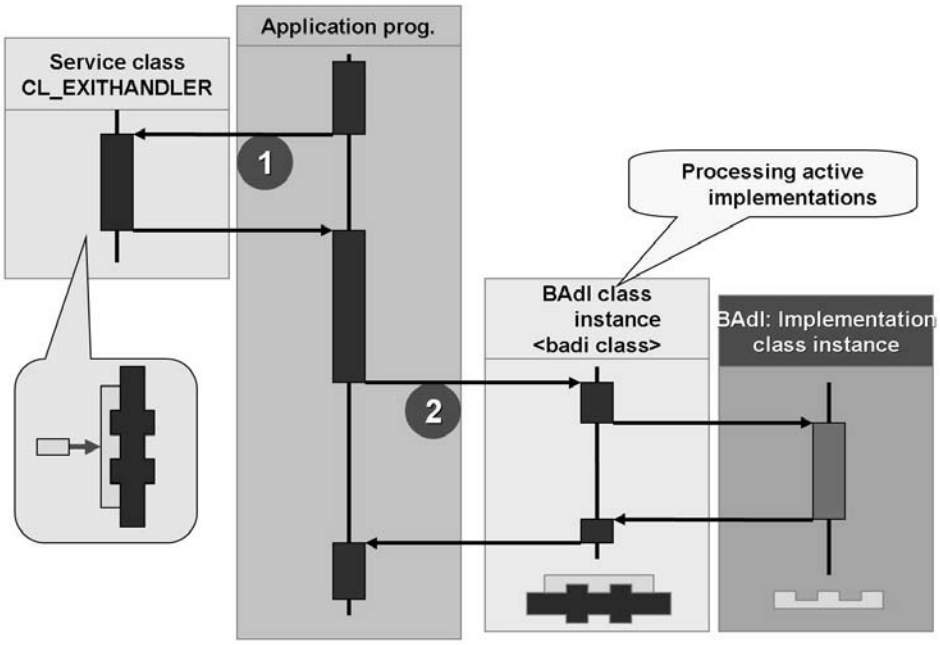


Figure 65: Business Add-Ins: Program Exit Flow

This graphic shows the process flow of a program that contains a Business Add-In call. It enables us to see the options and limitations inherent in Business Add-Ins.

Not displayed: In the declaration section, you must declare a reference variable that refers to the BAdI interface.

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 creates the conditions for calling the methods of the program enhancement.

When you define a BAdI, 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 the active implementations of the BAdI and calls the methods that have been implemented.

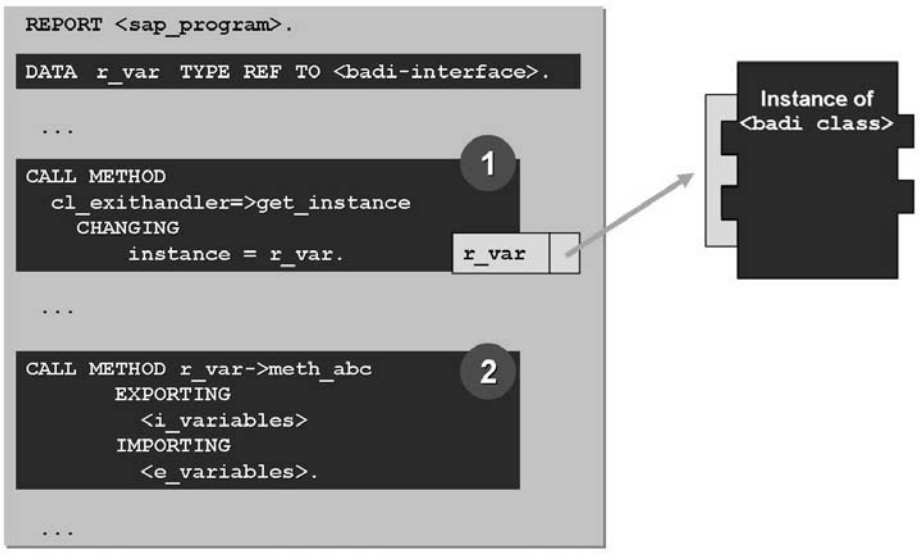


Figure 66: Business Add-Ins: Call Syntax in the SAP Program

This graphic contains the syntax with which you call a Business Add-In. The numbered circles correspond to the calls from the previous graphic.

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).

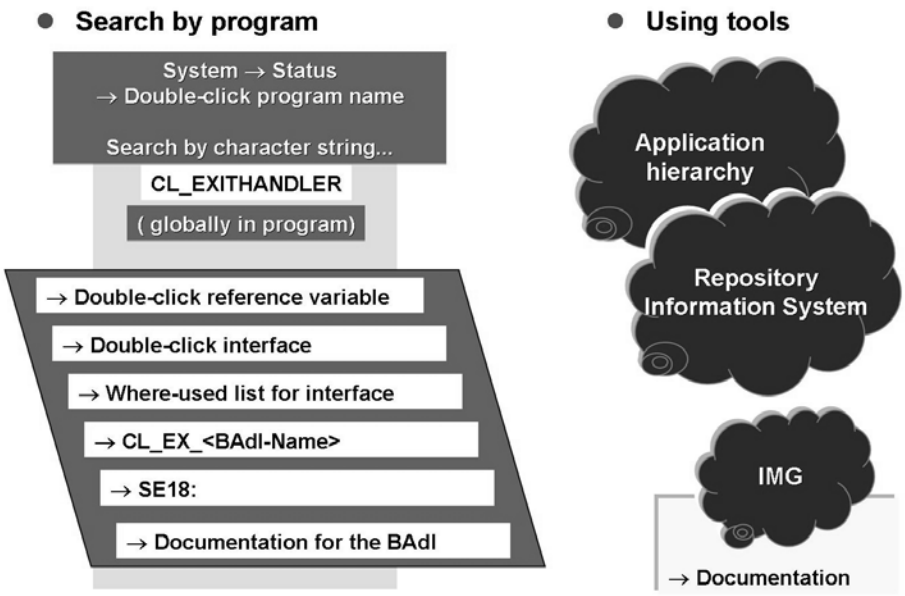


Figure 67: Finding a Business Add-In

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, the “GET_INSTANCE” method of this class must be called.

Using forward navigation, and based on the naming convention, you can then reach the definition of a business add-in, which contains documentation and a guideline on 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.

Create an implementation. Edit the components following the subsequent slides.

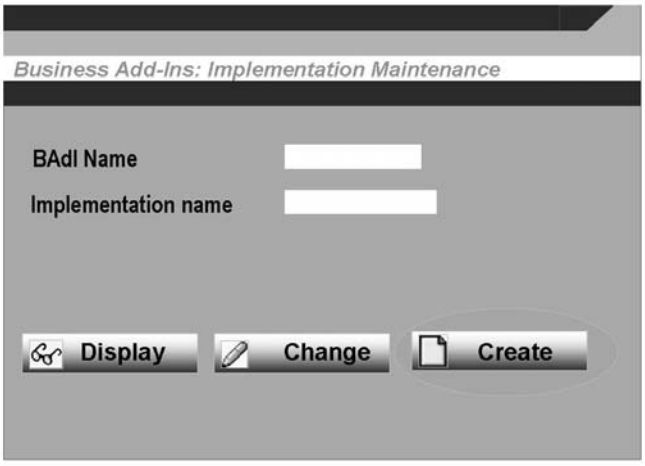


Figure 68: Implementing Business Add-Ins: Initial Screen

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 display an overview of the existing implementations. You can also create new implementations from here.

Note: For *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 accordingly. To create an implementation for a classical BAdI, select “Classical BAdI” on the lower input area of the initial screen. Enter the BAdI name in the corresponding field, and choose “Create Impl.”.

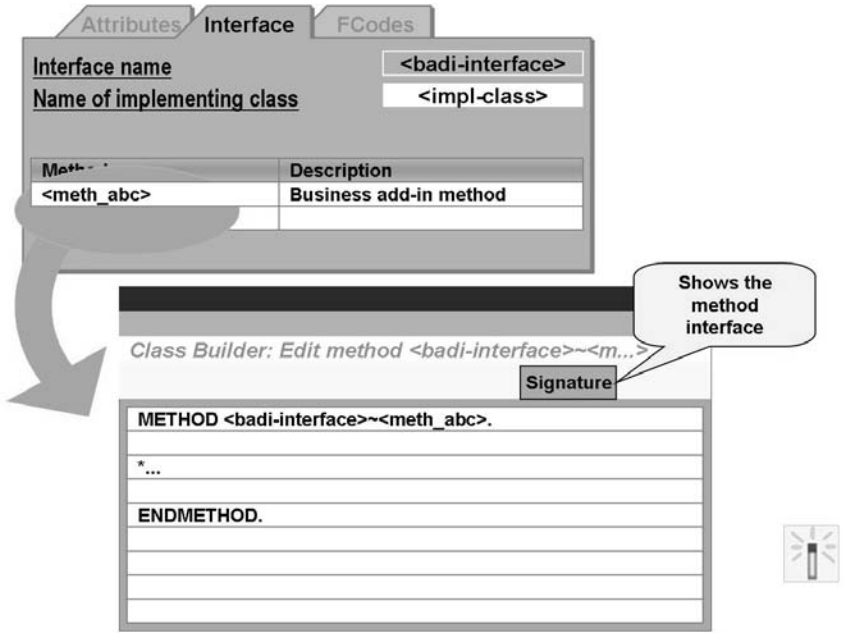


Figure 69: Implementing Business Add-Ins: Methods

You can assign any name to the implementing class. However, it is a good idea to observe the proposed naming convention. The name is derived 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.

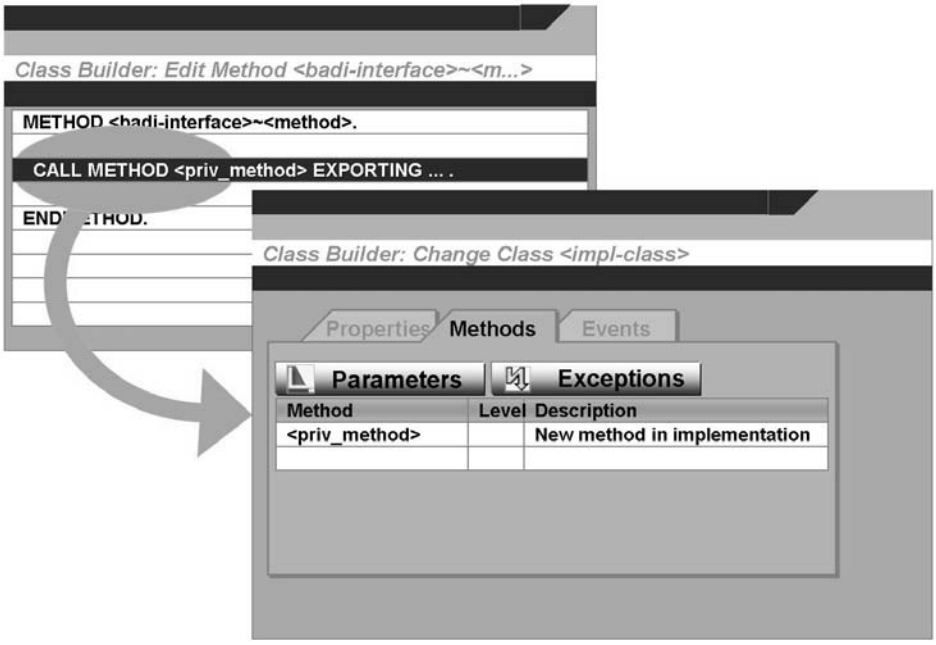


Figure 70: Implementing Business Add-Ins: Private Methods

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 the interface. Specify a visibility for the method, and implement it.

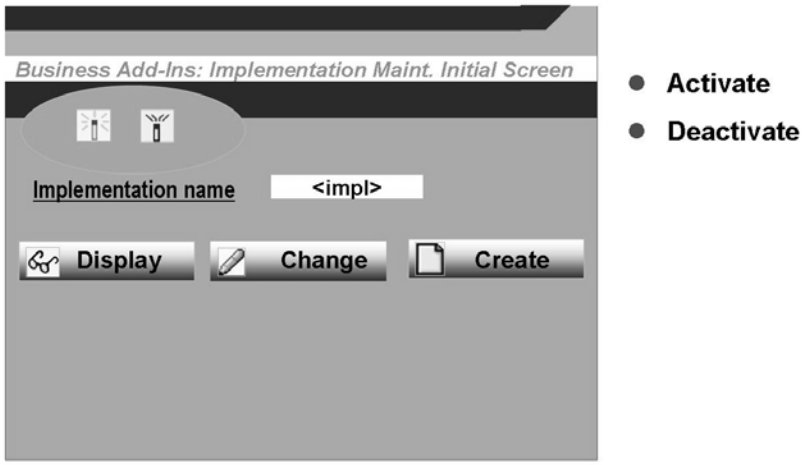


Figure 71: Implementing Business Add-Ins: Activating an Implementation

Use the corresponding 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.

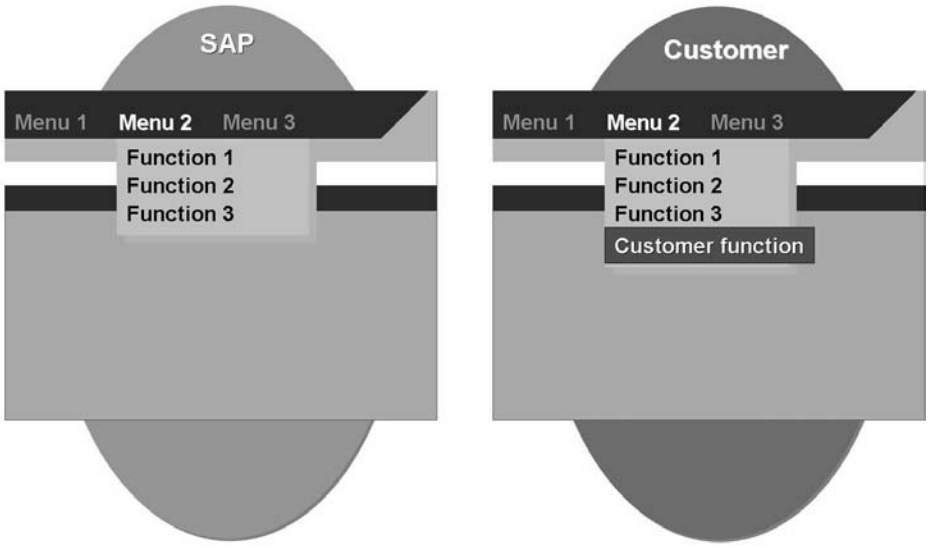


Figure 72: BAdI: Menu Exit (Overview)

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 business add-in implementation.

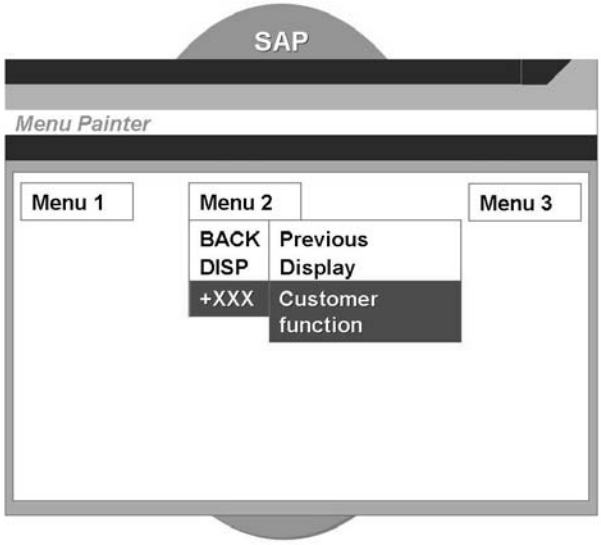


Figure 73: BAdI: Menu Exit (SAP Preparation)

Function codes of menu enhancements begin with a plus “+” sign.

The menu entry will only appear if there is an active business add-in implementation that contains 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.
```

Figure 74: BAdI: Menu Exit (Processing Function Codes)

Arbitrary. There is no standard demo for menu exits. However, you could create one very quickly.

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 directly together and are inseparable. Having the former without the latter would make no sense. For this reason, it is important that the two enhancement components are contained in a single enhancement - the Business Add-In.

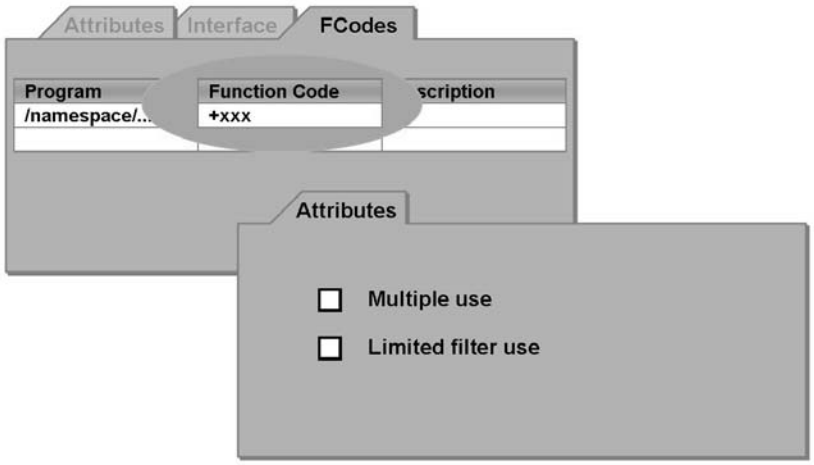


Figure 75: BAdI: Menu Exit (Restriction)

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?")

105

Exercise 5: Creating and Implementing a BAdI

Exercise Duration: 30 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Implement an enhancement using Business Add-Ins

Business Example

The customer service personnel at the travel agency wants 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 the customer number.

Task 1:

Check if the program **SAPBC425_BOOKING_##** (## = group number) can be enhanced.

1. Check the program for enhanced options.
2. Check if an enhancement option is suitable for outputting further information in the list.

Task 2:

Implement the enhancement you found. Name of the implementation: **ZBC425IM##**.

1. What data is passed to the interfaces of the methods? Are there already fields here that should be displayed in the list?
2. The table **SCUSTOM** contains information on the customers. Get the customer's name from the customer number. Output the name.

Task 3:

Format the list.

1. How can you move the vertical line so that the additional fields are displayed within the frame?

Continued on next page

2. Is the **CHANGE_VLINE** method suitable for changing the position of the vertical line? If so, use it.

Task 4:

1. Check your results.

Solution 5: Creating and Implementing a BAdI

Task 1:

Check if the program **SAPBC425_BOOKING_##** (## = group number) can be enhanced.

1. Check the program for enhanced options.

   a) Check if the program **SAPBC425_BOOKING_##** (## = group number) for enhancement options, as follows:

   b) From the list display: Place the cursor on the list and choose *F1* → *Technical Information*. Double-click the program name (You can also fo directly to the ABAP Editor.). Look for the character string **CL_EXITHANDLER** in the program. Double-click the transfer parameter exit_book. Double-click the interface used to define the type of exit_book. The Class Builder starts. Call a where-used list for the interface in classes. The class **CL_EX_BADI_BOOK##** is displayed. The name of the Business Add-In is **BADI_BOOK##**.

2. Check if an enhancement option is suitable for outputting further information in the list.

   a) Start transaction **SE18** (BAdI Builder: Initial Screen for Definitions). Read the documentation about Business Add-Ins.

Task 2:

Implement the enhancement you found. Name of the implementation: **ZBC425IM##**.

1. What data is passed to the interfaces of the methods? Are there already fields here that should be displayed in the list?

   a) Implementing the enhancement: From transaction **SE18**, select *Implementation* → *Create*, to open the transaction to create the implementation of BAdIs. Name of the implementation: **ZBC425IM##**.

   b) You can display the interface parameters by double-clicking the method 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.

Continued on next page

2. The table **SCUSTOM** contains information on the customers. Get the customer's name from the customer number. Output the name.

   a) Double-click 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.
```

Task 3:

Format the list.

1. How can you move the vertical line so that the additional fields are displayed within the frame?

   a) You can use the change_vline method to format the list. You can move the right edge of the list here.

   b) The parameter c_pos defines the position of the right vertical line.

2. Is the **CHANGE_VLINE** method suitable for changing the position of the vertical line? If so, use it.

   a) The method can be implemented as follows:

```
METHOD if_ex_badi_book00~change_vline.
  c_pos = c_pos + 25.
ENDMETHOD.
```

Task 4:

1. Check your results.

   a) –

Lesson Summary

You should now be able to:

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

Lesson: BAdIs – Additional Information

110 Lesson Duration: 15 Minutes

Lesson Overview

This lesson contains additional information on BAdIs.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain what an extensible filter type is
- Explain the default and sample code of a BAdI

-

Business Example

You want to add new functions to a flight maintenance transaction in an SAP system. To minimize the inputs 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.

Additional Information on BAdIs

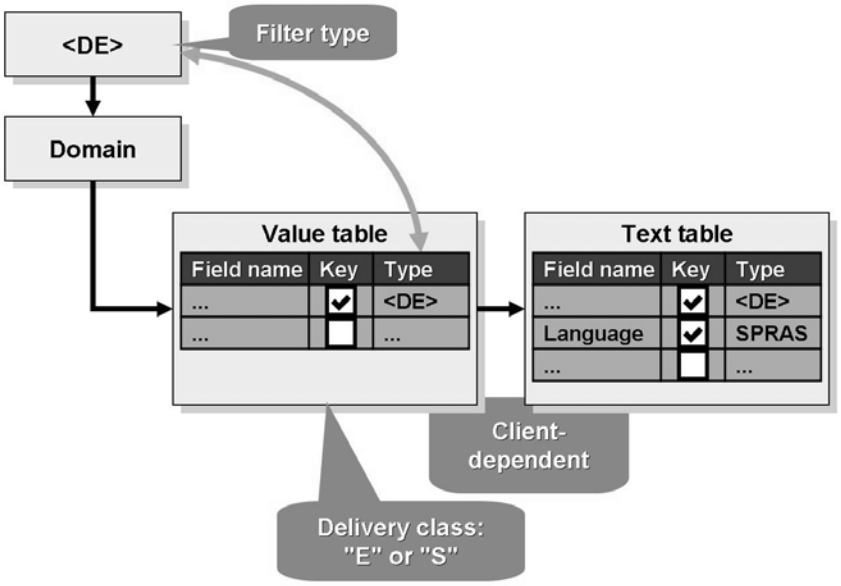


Figure 76: Extendible Filter Types: Prerequisites

The assignment of the *extendible* attribute is subject to the following restrictions:

The domain that has an extendible filter type must have the following characteristics: It must be linked with a *client-independent* value table. The value table has exactly one key field which has the data element of the filter type as its field type. It 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, a field that contains the string ’TEXT’ or ’TXT’ as a partial string must exist in this table. 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.

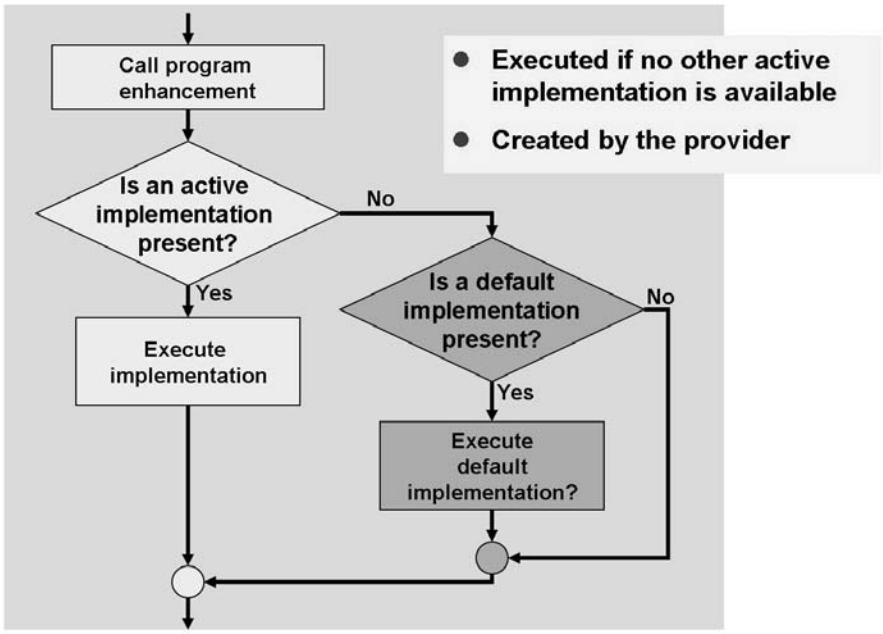


Figure 77: Default Implementation

A *default implementation* is executed if no active implementation exists for a Business Add-In. 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.

	Customer exits	Business Transaction Events	Business Add-Ins
Program exit	+	+	+
Menu exit	+	–	+
Screen exit	+	–	+
Append fields on screens	+	–	(+)
Administration level	+	–	+
Multiple use	–	+	+
Filter-specific	–	+	+

Figure 78: Comparison With Other Enhancement Techniques

Business add-ins are a natural further development of conventional enhancement techniques. 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 this has been implemented using a consistent object-oriented approach.

- **BAdI 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)

Figure 79: Naming Conventions (BAdI Definition)

- **BAdI 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)

Figure 80: Naming Conventions (BAdI Implementation)

Facilitated Discussion

To wrap up this topic, allow participants to briefly discuss the use of BAdIs.

Discussion Questions

Use the following questions to engage the participants in the discussion. Feel free to use your own additional questions.

What experience of BAdIs did you have prior to the training course?

During the training course, did you discover BAdIs that you can use in your application area?

Lesson Summary

You should now be able to:

- Explain what an extensible filter type is
- Explain the default and sample code of a BAdI

Unit Summary

You should now be able to:

- Explain why SAP introduced classic BAdIs as an enhancement in Release 4.6.
- Search for available Business Add-Ins in SAP programs
- Use Business Add-Ins to implement program enhancements
- Explain what an extensible filter type is
- Explain the default and sample code of a BAdI

Unit 5

117

Modifications

Unit Overview

Unit Objectives

After completing this unit, you will be able to:

- Describe what a modification is
- Explain the basic concepts in the modification environment
- Describe what you must bear in mind when implementing modifications
- Explain how the Modification Assistant works
- Conduct specialized modifications using the Modification Assistant
- List your modifications using the Modification Browser
- Describe what user exits are and how they work
- Describe how to find user exits in the system and use them to enhance SAP software
- Use the Note Assistant to implement source code corrections in an SAP system
- List the advantages of the Note Assistant
- Use the Note Assistant to adjust corrections implemented using SAP Notes
- Name the different steps in modification adjustment
- List which objects must be adjusted, and when
- Describe how modification adjustment is carried out in follow-on systems

Unit Contents

Lesson: Modifications

119 Lesson Duration: 20 Minutes

Lesson Overview

In this lesson, you will learn about modifications, and become familiar with basic terms in the modification environment.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe what a modification is
- Explain the basic concepts in the modification environment

-

Business Example

You would like to learn about basic modification principles.

Modifications: Introduction

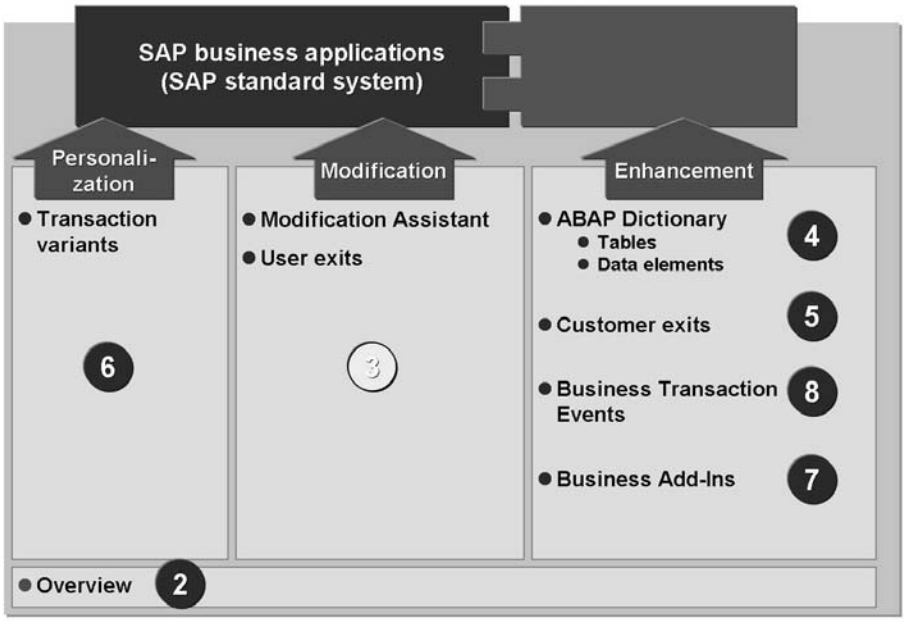


Figure 81: Modifications: Overview Diagram

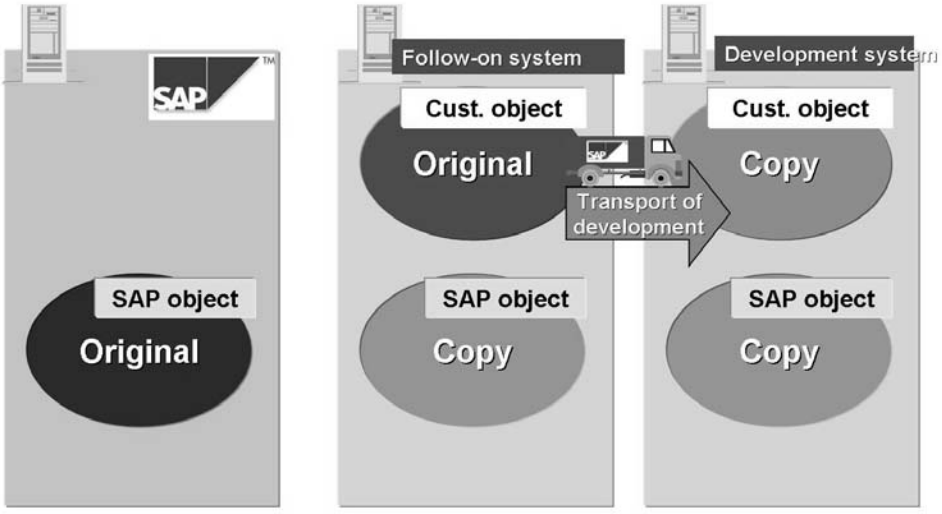


Figure 82: Originals and Copies

An object is original in **only one system**. The “original system” for objects delivered by SAP 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. This has the type “Development/Correction”.

This request ensures that the objects are transported from the development system into follow-on systems.

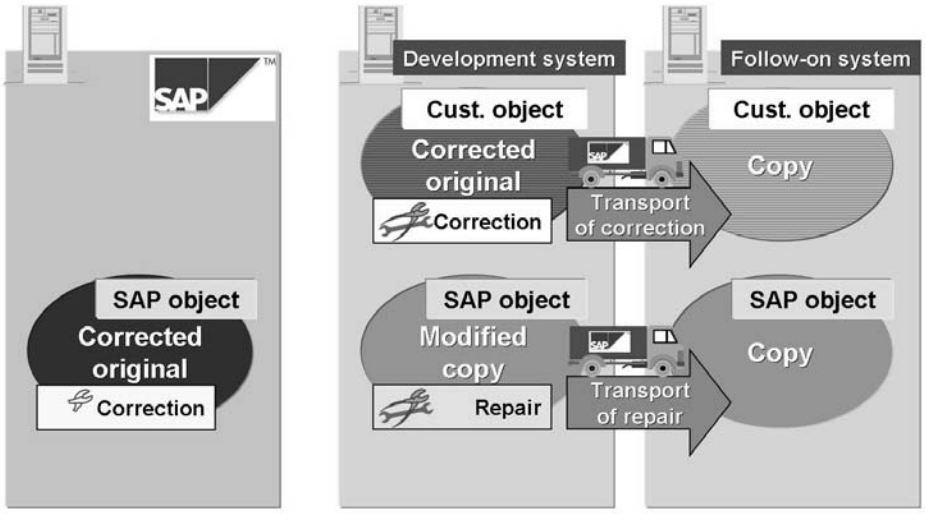


Figure 83: Corrections and Repairs

Changing an original is called a **correction**. The system records these changes in a request that contains tasks of 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**”. A repair of an SAP object is called a **modification**.

The repairs made to your own objects (for example, due to an emergency in the production system) can also be made immediately to the originals in the development system.

Caution: When you change copies, you must correct the original immediately.

However, you cannot do this with SAP objects, because the originals are not in any of your systems.

You should only modify the SAP standard system if the modifications you want to make are absolutely necessary for optimizing workflow in your company. Note also that good background knowledge of the application structure and flow are essential in deciding what kind of modifications to make to the standard system and how these modifications should be designed.

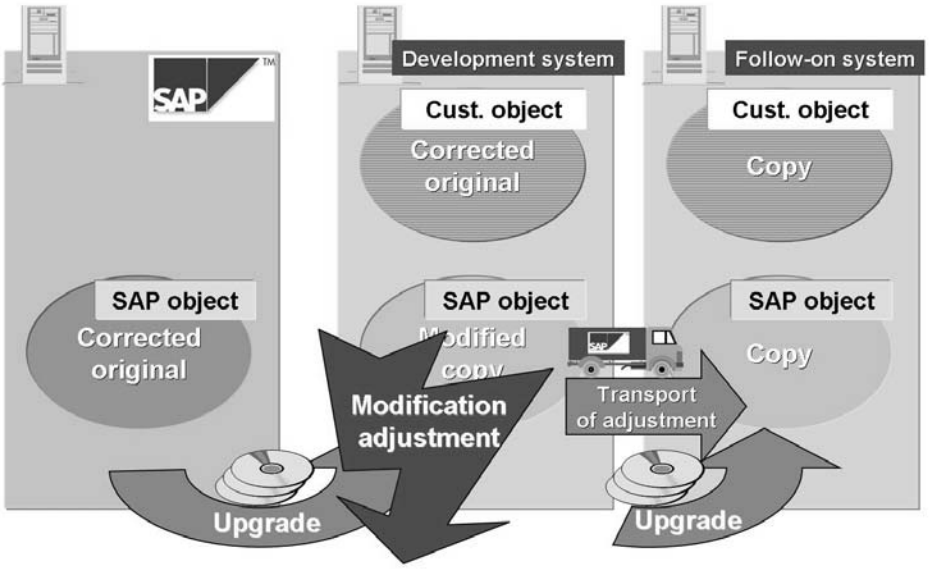


Figure 84: Modifications During Upgrade

Whenever you upgrade your system, apply a hot package, or import a transport request, conflicts can occur.

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 R/3 System.

If you want to save your changes, you must perform a **modification adjustment** for the relevant objects. Many modified SAP objects can cause substantial delays when importing an upgrade.

To ensure that the development system and the next system are consistent, we strongly advise that you only make the modification adjustment in the development system. The objects from the adjustment can then be transported into follow-on systems.

Facilitated Discussion

-

Discussion Questions

Use the following questions to engage the participants in the discussion. Feel free to use your own additional questions.

-

Lesson Summary

You should now be able to:

- Describe what a modification is
- Explain the basic concepts in the modification environment

Lesson: Making Modifications

124 Lesson Duration: 20 Minutes

Lesson Overview

This unit shows you what you should bear in mind when making changes.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe what you must bear in mind when implementing modifications

-

Business Example

You would like to know what you must bear in mind when implementing modifications.

Making Modifications

The following slides summarize important guidelines for modifying your system. Adhering to these rules will drastically reduce the amount of work required at upgrade. These rules apply irrespective of whether you use the Modification Assistant. An additional modification log can also prove very valuable when making modification adjustments. Emphasize that this slide can only serve as an example.

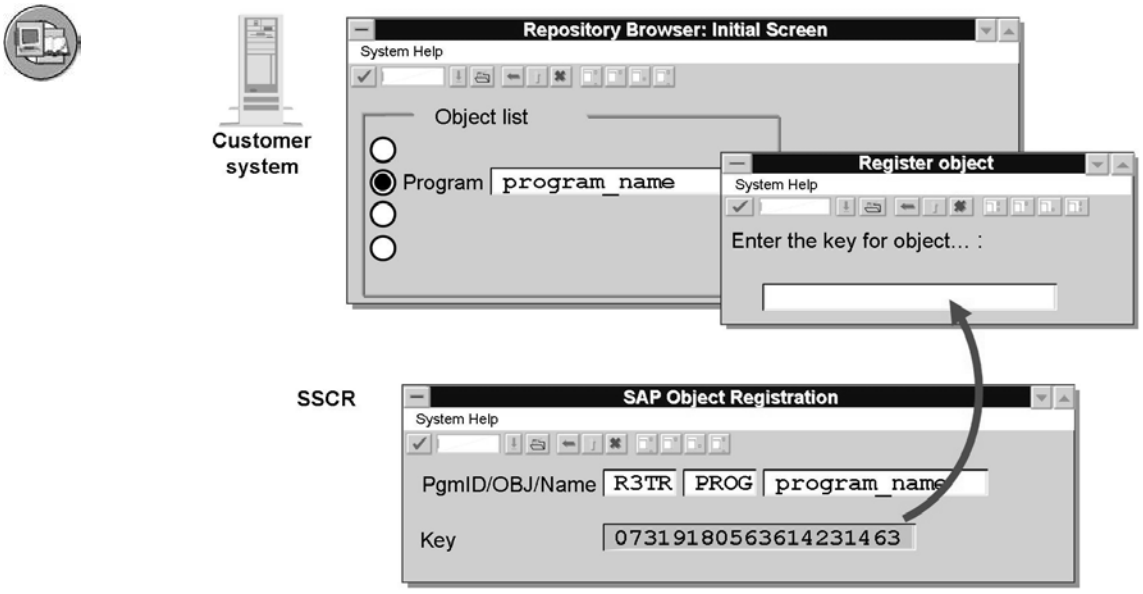


Figure 85: Registering Modifications in SSCR

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, advance corrections, and objects whose changes are based on automatic generation (for example, from Customizing). If the object is changed again at a later stage, 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.

How do you benefit from SSCR (SAP Software Change Registration)?

- Quick resolution of errors and speedy 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 R/3 System.
- Reliable operation Having to register your modifications helps prevent unintended changes. This in turn ensures the reliable operation of your standard R/3 software.
- Simplified upgrades Upgrades and release upgrades are considerably easier due to the smaller number of modifications.

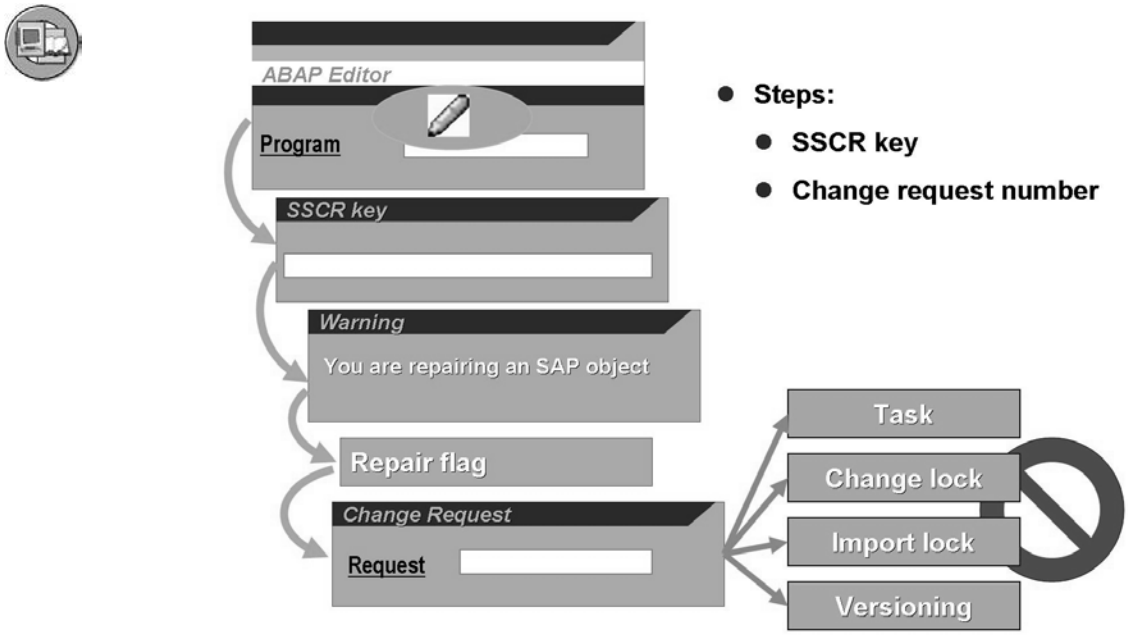


Figure 86: Carrying Out a Registered Modification

If you want to change an SAP Repository object, you must provide the Workbench Organizer with the following information:

- An SSCR key
- A change request

We saw above how to get an SSCR key. If you now continue to change the object, you must confirm the following warning dialogs: At this stage, you can still cancel the action.

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).
- Versions The system generates a new version of the object (see below).

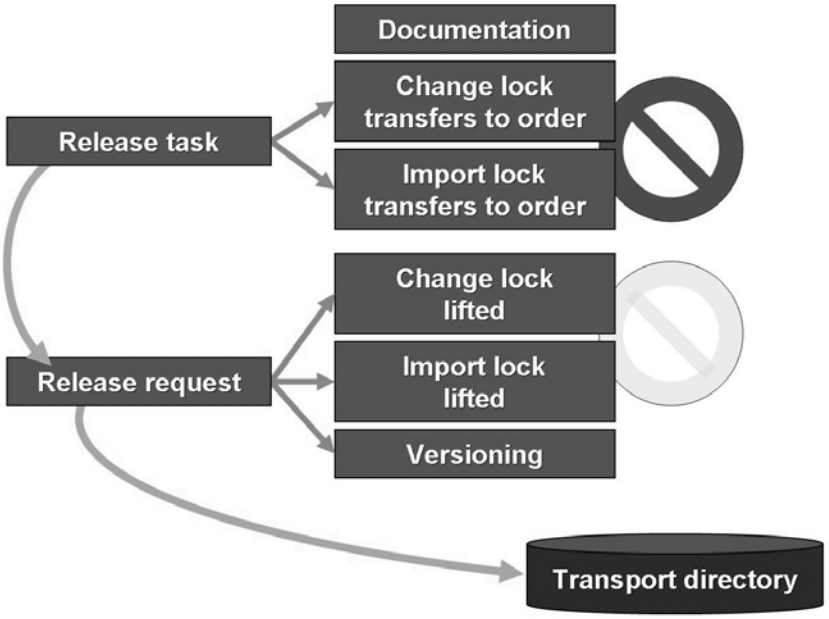


Figure 87: When the Modification is Finished

After development is finished, the programmer releases the task. At this stage, he or she must document the changes made. The objects and object locks valid in the task are transferred from the task 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. The locks on the objects in the change request are released. 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 the operating system level. They can then be imported into follow-on systems by the system administrator.

After the modifications have been imported into the quality system, the developer must test them and check the import log of the request.

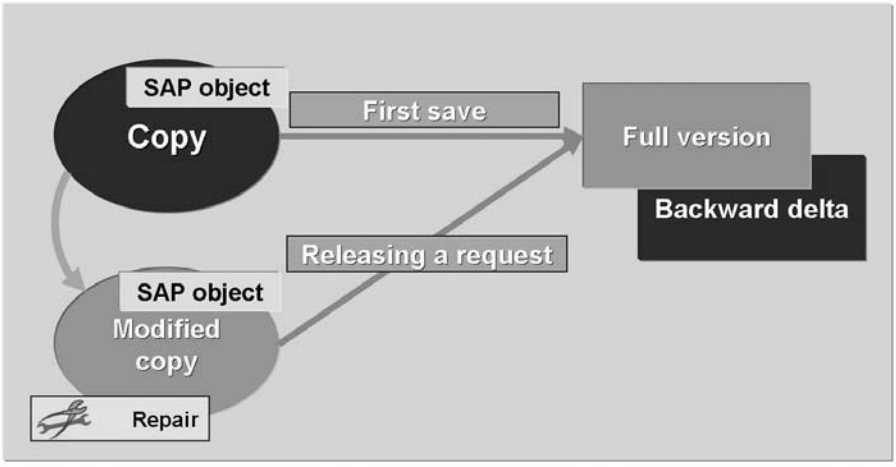


Figure 88: Versions

When you release a change request, the system stores a complete version of all the objects of the change request to the version database.

If you edit 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.

Whenever you assign a Repository object to a task, the system checks whether the current version matches the complete copy in the version database. If not, a complete copy is created. This process is also initiated the first time you change an SAP 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.

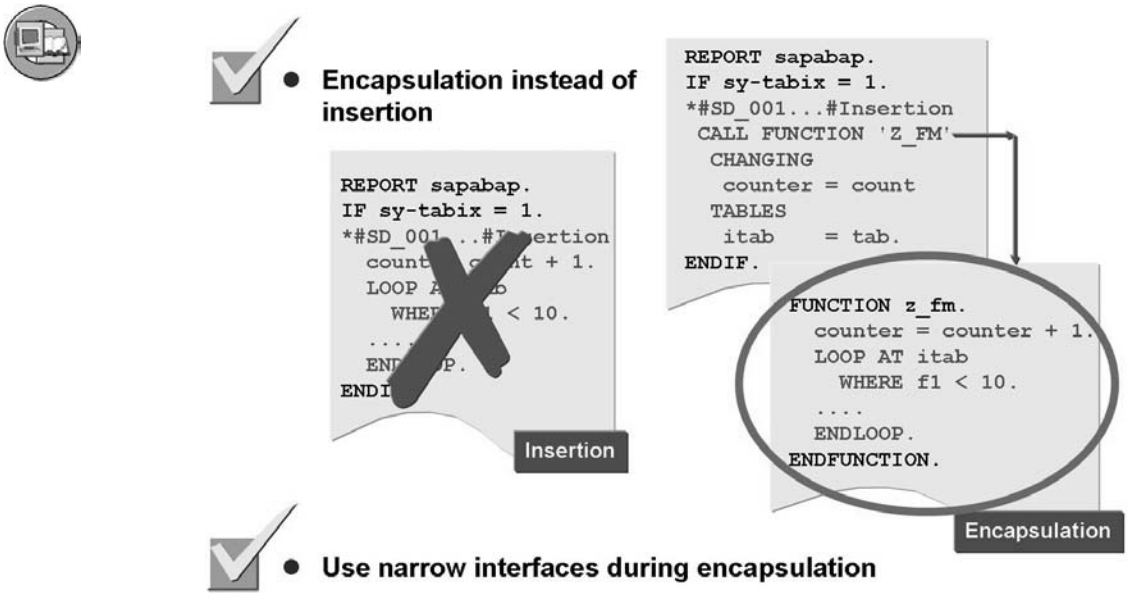


Figure 89: Critical Success Factors (1)

Encapsulate customer source code in modularization units instead of inserting it directly into long passages of SAP source code (with, for example, customer function module calls in program source code, or customer subscreen calls for additional screen fields).

Hint: Be careful to use narrow interfaces when you encapsulate customer-specific functions, to ensure good data control.

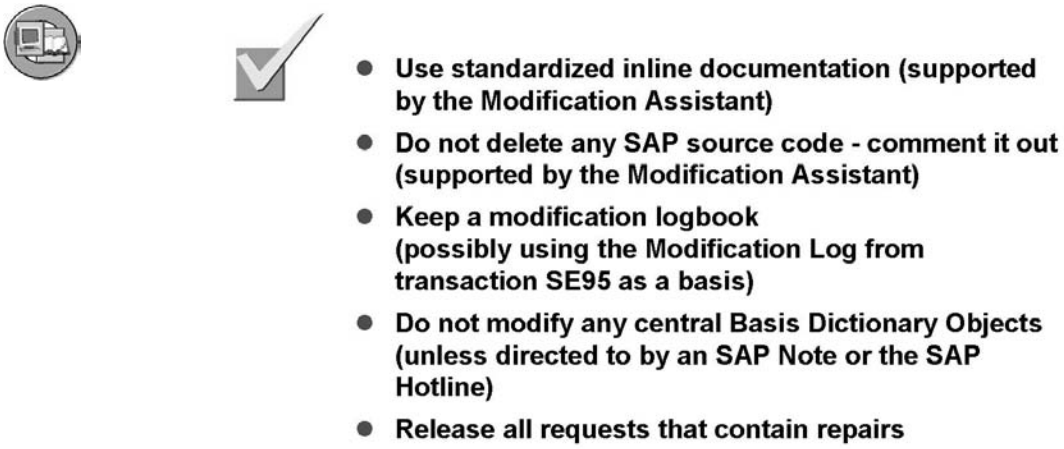


Figure 90: Critical Success Factors (2)

Define a company-wide standard for managing source code documentation.

Keep 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 adjustment is not an option. This can lead to the contents of certain tables being lost.

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 version database (the system compares versions during adjustment).

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

Figure 91: Modification Logs (Example)

SAP recommends that you keep a **list of all modifications** that you have made (that is, any changes to Repository objects in the SAP namespace).

The list can contain 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 inputs to restore the modification during the adjustment

Facilitated Discussion

-

Discussion Questions

Use the following questions to engage the participants in the discussion. Feel free to use your own additional questions.

-

Lesson Summary

You should now be able to:

- Describe what you must bear in mind when implementing modifications

Lesson: Modification Assistant

132 Lesson Duration: 20 Minutes

Lesson Overview

This lesson describes how the Modification Assistant and the Modification Browser work, and explains how to use these tools.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain how the Modification Assistant works
- Conduct specialized modifications using the Modification Assistant
- List your modifications using the Modification Browser

Outline the basic principle of the Modification Assistant: To reduce the amount of work when making modification adjustments. In the past, you had to do a lot manually. You had to adjust a program when the include was changed by SAP and the customer. Now, only a reasonable amount of work is required when the exact part of the include is changed by SAP and by you. Give the example of a changed subroutine: You changed the subroutine; SAP changes the include but not the subroutine. As a result, the object will appear in the list of objects to be adjusted at adjustment time. However, the system determines that the adjustment can be carried out automatically. You can simply click the green traffic light. The Modification Assistant registers the modifications in different tables rather than directly in the source code. When the LOAD is generated, the contents of the modification tables and the source is taken into account. As a result: • Modifications can be reset by deleting the contents of the modification tables • The first version required by the system is created when the modification is saved.

Business Example

You would like to use the Modification Assistant to make changes in an orderly manner, to ensure that you can display an exact list of changes at a later stage, and make it easier to carry out modification adjustment.

Modification Assistant

To make modification adjustments easier

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

Figure 92: Modification Assistant: Goals

The Modification Assistant is designed to make modification adjustments easier. To do this, a finer granularity is used to log modifications. In the past, the granularity of modifications was limited to the include program level. A finer granularity is now available, and modifications can be recorded at the subroutine or module levels.

Amongst other things, this is because the modifications are registered in a different layer. As well as providing finer granularity, this means that you can reset modifications, as the original version has not changed.

Modification Assistant: How it Works

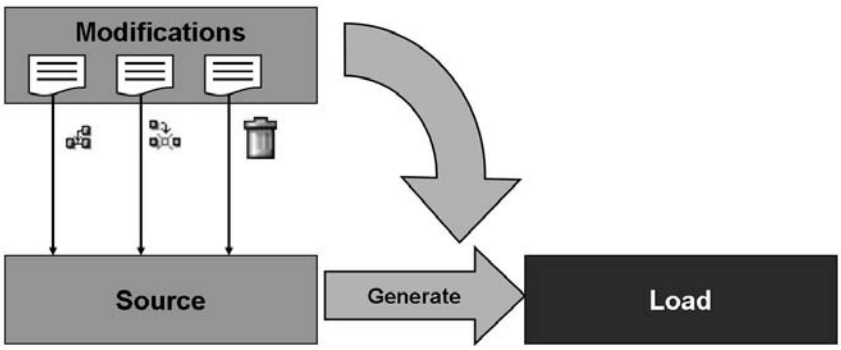


Figure 93: Modification Assistant: How it Works

The Modification Assistant records changes at a separate level, leaving the original source of an object unchanged. The modifications are only transferred when the load is generated. This means that the executable object is generated from components of the original SAP program and the modifications.

Modification Adjustments Then and Now

- THEN:
  - Granularity: Include source
  - Line-by-line modification adjustment
  - Each modification had to be included manually in the new SAP version (using the cut and paste function)
- NOW (using the Modification Assistant):
  - Granularity: Module (for example, subroutine)
  - Modification adjustment at the module level (“modifications exit”)

If, in the past, 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 during adjustment.

This situation has changed considerably thanks to the Modification Assistant. Modifications are now recorded with finer granularity. 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.

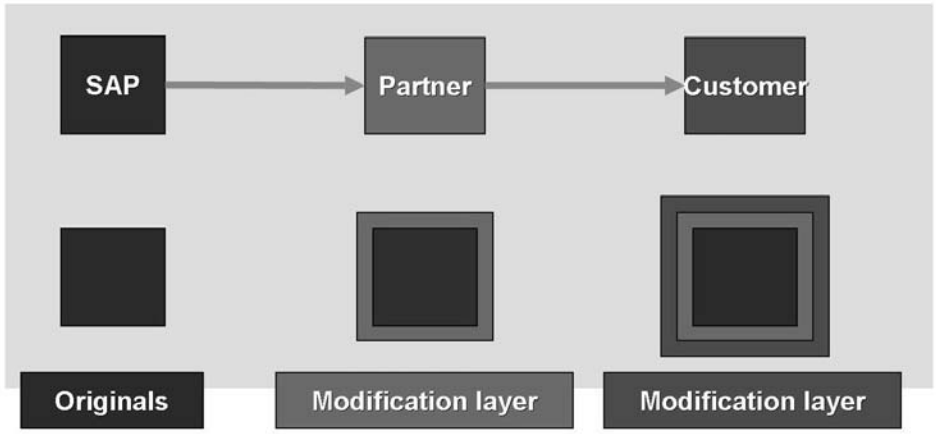


Figure 94: Modification Assistant: Software Layers

The original version of each software layer, plus current modification, comprises the originals for subsequent software users.

Modification Assistant: Tools Supported

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

The Modification Assistant supports the key tools of the ABAP Workbench:

In the ABAP Editor, you can use the 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 in the Screen Painter are also recorded. This applies to both the layout and the flow logic.

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 during the upgrade, the creation of table appends is also logged by the Modification Assistant.

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

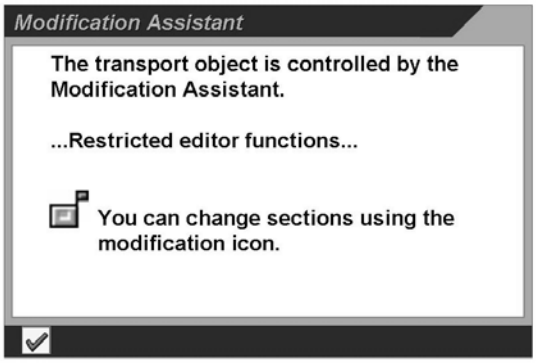


Figure 95: Modification Assistant: Prerequisites

If you want to change an SAP object, you must provide the following information:

- An SSCR key
- A change request

The system informs you that the object is controlled by the Modification Assistant. Only restricted functions are available in the editor.

You can switch the Modification Assistant on or off for the entire R/3 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 individual Repository Objects. Once you have done so, the system no longer uses the fine granularity that is used in the Modification Assistant.

Demonstrate the features of the Modification Assistant: Explain the icons (although they are more or less self explanatory). Modify one of your programs as an example. Use the ABAP Editor to modify the source code. Show the icons: The modification overview and the Restore Original features are important. Explain what happens: Tell your students that a table is used to log all modifications and that the modifications themselves are stored in another table separately from the source code. This enables the system to undo a modification.

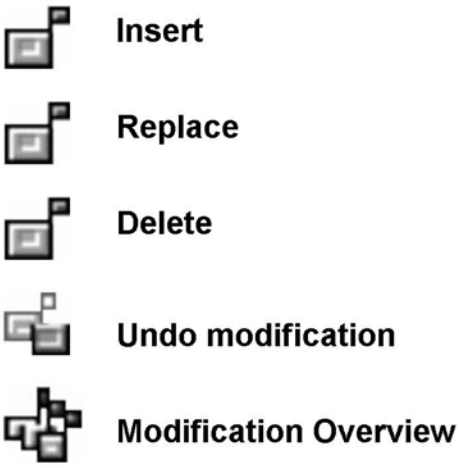


Figure 96: Modification Assistant Icons

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 buttons. For example, in the ABAP Editor, you can:

- Insert The system generates a framework of comment lines in 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.

Modify a program. Use one of your demo programs. Show modification in the editor. Show screen modifications. Show how to create a new text for a data element: There are two alternatives. And so on. Do this in parallel to presenting the slides.

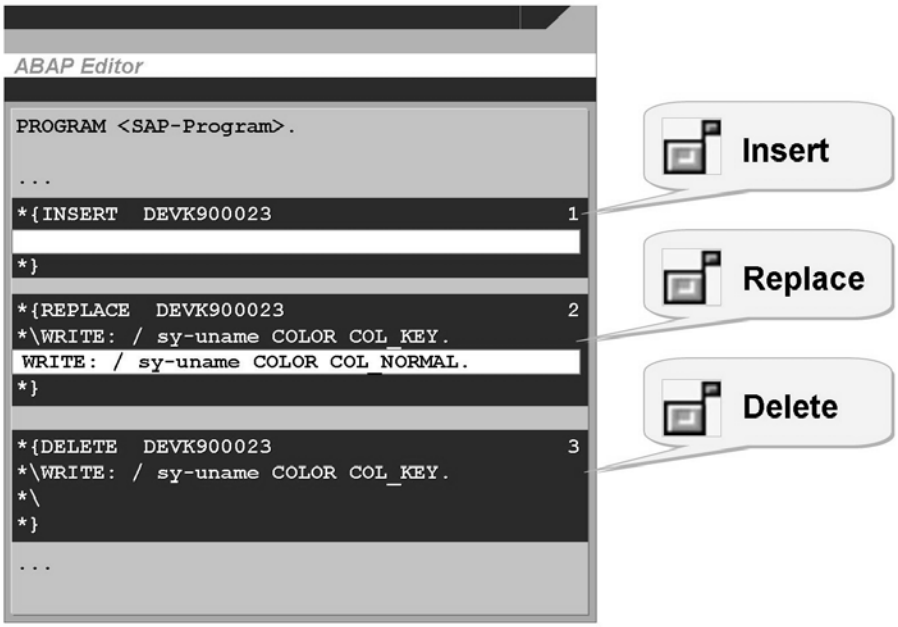


Figure 97: Modification Assistant: Example of ABAP Editor

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.

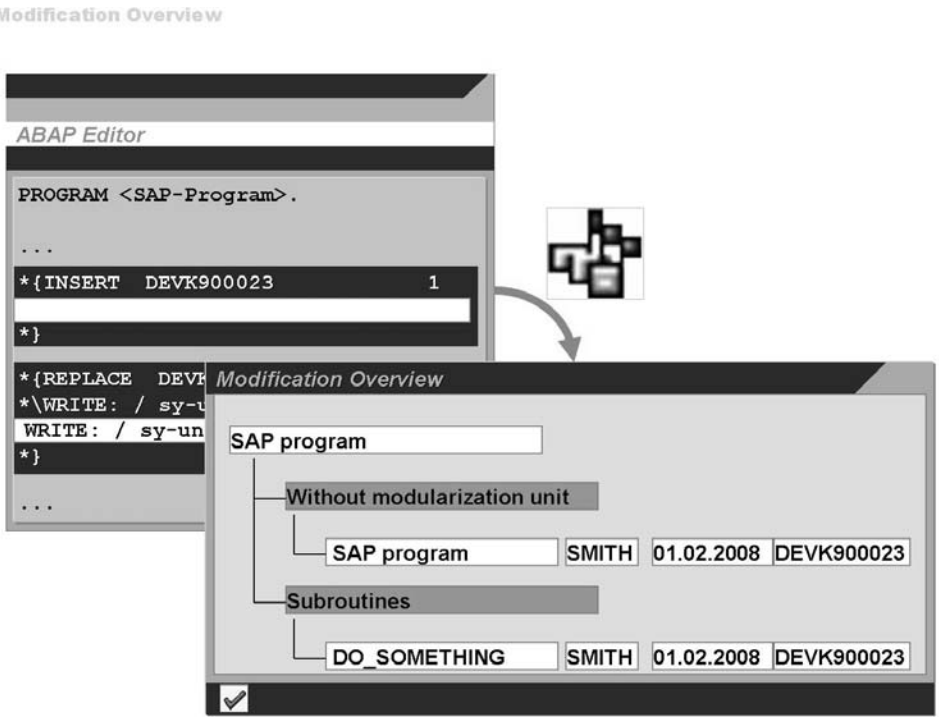


Figure 98: Modification Overview

The “modification overview” icon provides you with an overview of the modifications you have made in the current program.

The display is broken down into the various modularization units. This corresponds to the structure used by the Modification Assistant to record the modifications.

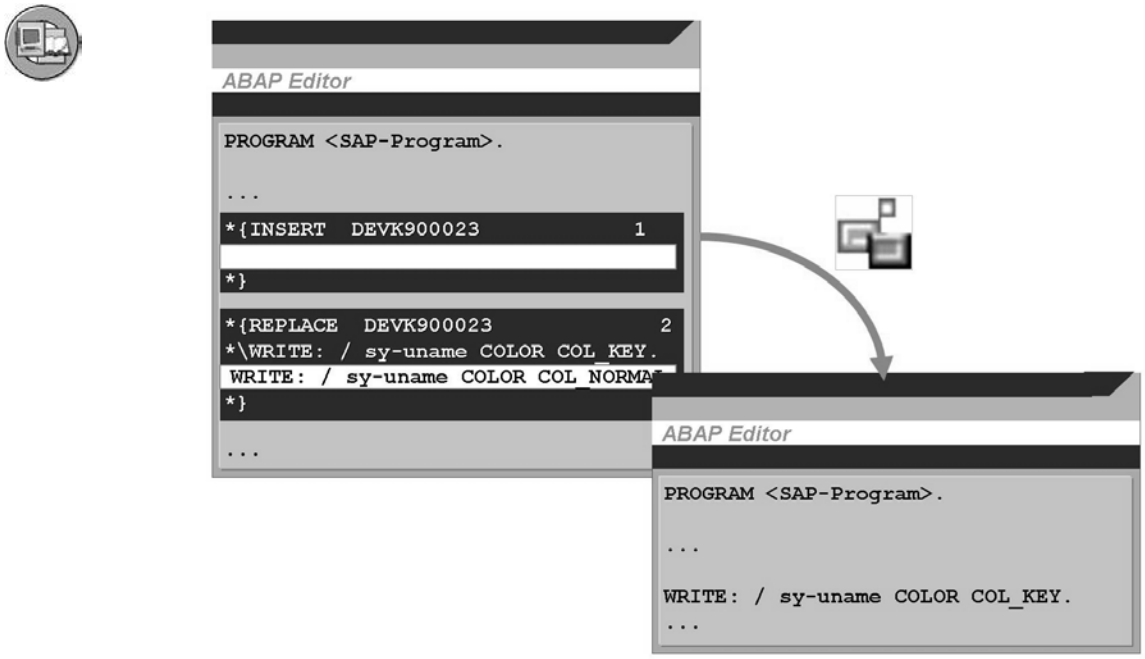


Figure 99: Restoring the Original

To undo a modification, place the cursor on the relevant modification. Select the button to undo the modification.

The record of the modifications is deleted. It is not possible to restore the deleted modification.

Show the most important features of the Modification Browser: Find the objects you modified recently. Undo a modification, and so on.

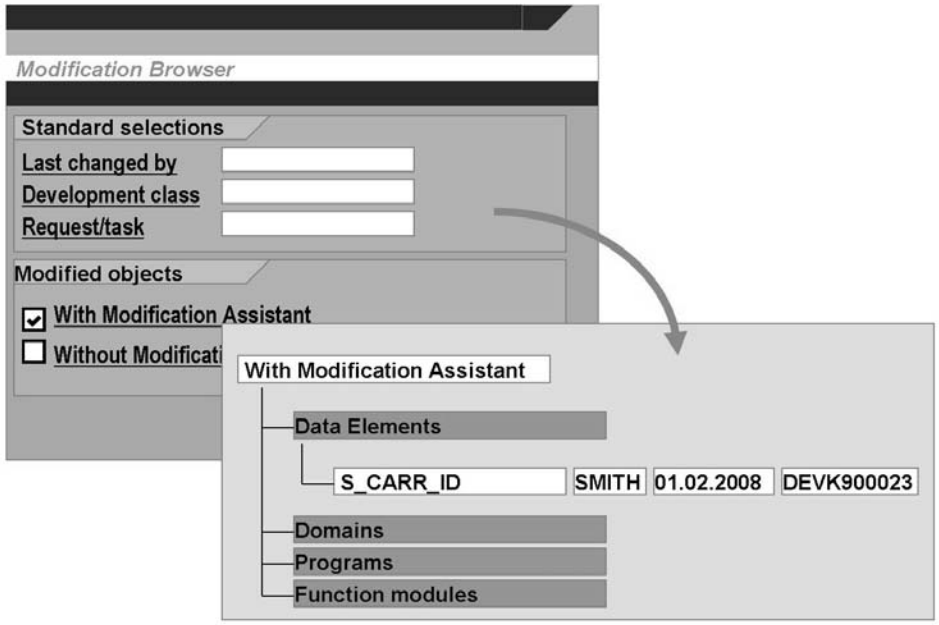


Figure 100: Modification Browser

The Modification Browser provides an overview of all of the modified objects in the system. The Modification Browser differentiates between modifications made with and without the Modification Browser.

On the initial screen of the Modification Browser, you can restrict the selection based on 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 “Reset to Original” button.


143

Exercise 6: Implementing Modifications

Exercise Duration: 30 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Carry out modifications using the Modification Assistant
- Implement non-registered modifications

Business Example

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

Incorrect functions 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.

Task 1:

Modify R/3 objects. Use the Modification Assistant where possible. The objects to be changed are specified below:

Modify the program **SAPBC425_BOOKING_##**.

1. Enhance the header so that the column with the customer's name also has a header.
2. Create a new variable for counting the data records. Output the counter in the last column of the list.
3. Read the fields **LUGGWEIGHT** and **WUNIT** in the table **SBOOK** too, and output them in the list.

Task 2:

Modify the program **SAPBC425_FLIGHT##**.

1. Change the layout of screen 0100: Insert a frame around the three input fields. Create a button and assign the function code **MORE** to it.

Continued on next page

Task 3:

Modify the data element **S_CARRID##**.

1. Change the field labels to: Short: “Airline”, Medium: “Airline company”.
2. Modify the documentation for this data element. Create a meaningful text.

Task 4:

1. Check your modifications in the Modification Browser.

Solution 6: Implementing Modifications

Task 1:

Modify R/3 objects. Use the Modification Assistant where possible. The objects to be changed are specified below:

Modify the program **SAPBC425_BOOKING_##**.

1. Enhance the header so that the column with the customer's name also has a header.

   a) You can change the header either directly from the list (*System → List → List Header*) or in the Editor.

2. Create a new variable for counting the data records. Output the counter in the last column of the list.

   a) You can create a new variable directly in the SAP program. Use the insert function in the Modification Assistant. Ideally, keep the changes locally in the **data_output** subroutine. Output the counter as well. Alternatively, you can implement this functionality in the enhancement, which would not cause a modification.

3. Read the fields **LUGGWEIGHT** and **WUNIT** in the table **SBOOK** too, and output them in the list.

   a) Read the additional fields **LUGGWEIGHT** and **WUNIT** in the table **SBOOK**, and output them in the list. Add two fields to the **SELECT** statement. Output the fields in the **data_output** subroutine.

Task 2:

Modify the program **SAPBC425_FLIGHT##**.

1. Change the layout of screen 0100: Insert a frame around the three input fields. Create a button and assign the function code **MORE** to it.

   a) Use the Screen Painter to change the layout of screen 0100.

Continued on next page

Task 3:

Modify the data element **S_CARRID##**.

1. Change the field labels to: Short: “Airline”, Medium: “Airline company”.

   a) 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.

2. Modify the documentation for this data element. Create a meaningful text.

   a) Choose the “Documentation” button and enter new text.

Task 4:

1. Check your modifications in the Modification Browser.

   a) To check the modification, choose the Modification Browser (transaction **SE95**). Limit the selection with the user name or the change request/task.

Lesson Summary

You should now be able to:

- Explain how the Modification Assistant works
- Conduct specialized modifications using the Modification Assistant
- List your modifications using the Modification Browser

Lesson: User Exit

148 Lesson Duration: 10 Minutes

Lesson Overview

In this lesson, you will learn how user exits work, how you can find them, and use them to enhance functions.

Lesson Objectives

After completing this lesson, you will be able to:

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

User exits are a long-standing method of enhancing an SAP System. Although there should be no new user exits in an R/3 System, a wide range are still available. Briefly explain to participants what user exits are and what they are used for.

Business Example

You would like to find user exits in the system and use them to enhance SAP software

User Exits

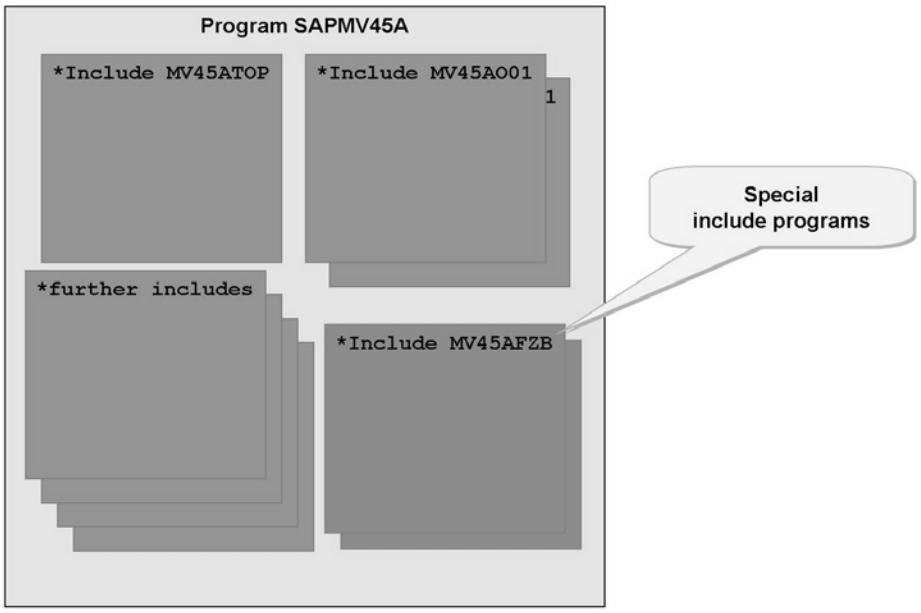


Figure 101: User Exit: Structure of an SAP Module Pool

In the first step, explain how a module pool is built. Describe the distinguishing features of a user exit. Provide your students with an example: Demonstrate a program that contains user exits (for example SAPMV45A). Show your students where the use of user exits is described in the Implementation Guide (Sales and Distribution -> System Modifications).

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 program described as "special" include programs in the above figure are themselves only include programs - technically, they are not different. These programs are only delivered once.

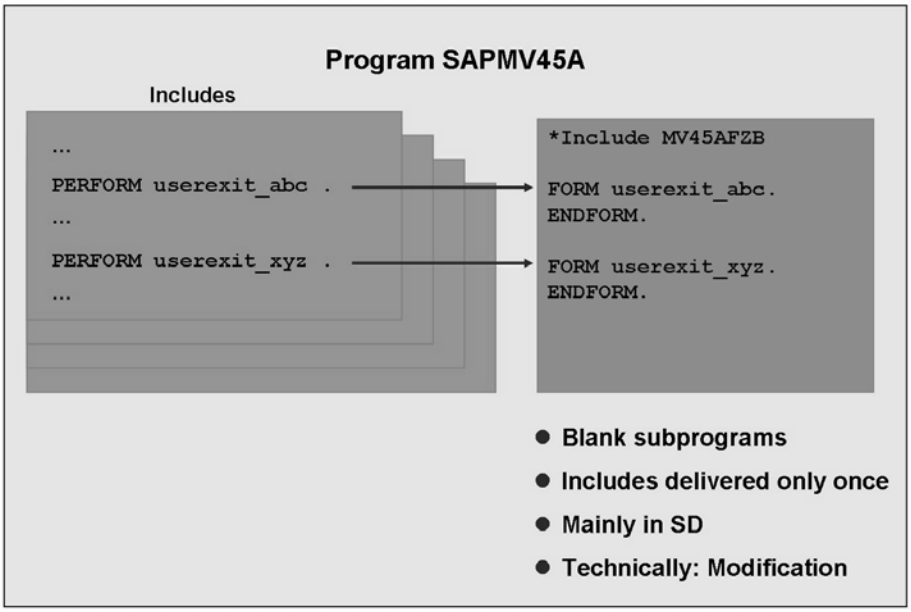


Figure 102: User Exit: Implementation

User exits are a type of system enhancement that were originally developed for the Sales and Distribution (SD) component. The original purpose of user exits was to allow users avoid the difficulties encountered with modification adjustments.

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 internal 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 created 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 Userexit can be used to fill additional data into VBAP*
*       from the main item (HVBAP), i.e. this Userexit 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.
```

Figure 103: User Exit: Example

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 includes of programs and function groups. The last two letters in the name of the include refer to the include that the customer should use: “Z” is usually found 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 adjustments.

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.

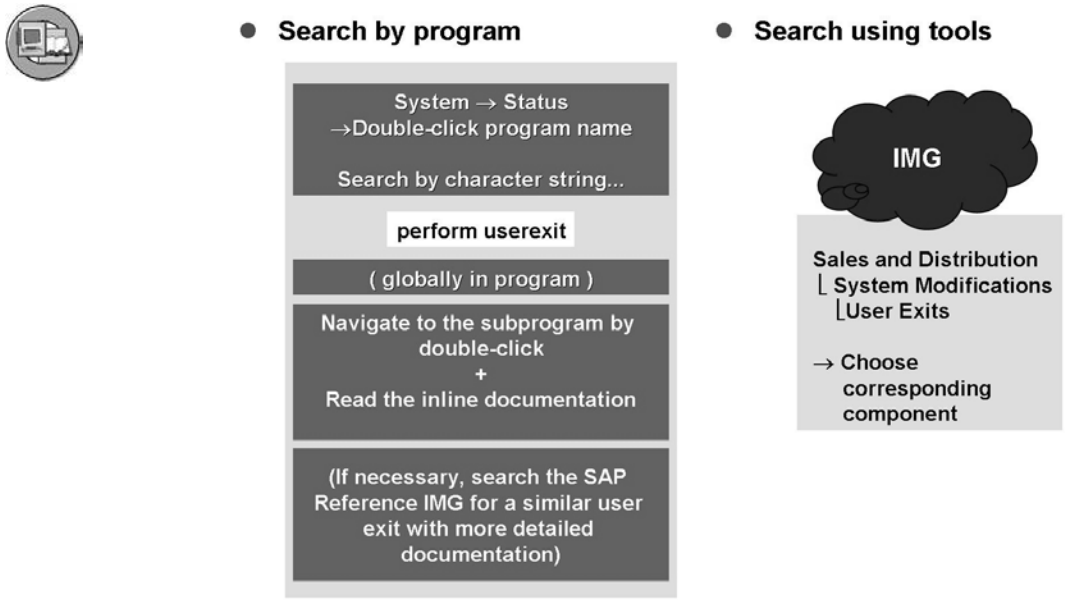


Figure 104: User Exits: Search

You can find a list of some of the available user exits in the SAP Reference Implementation Guide. Here, you will also find documentation explaining why SAP developers have created a particular user exit.

Follow the steps described in the Implementation Guide.

You can also search for user exits for specific programs. The graphic above describes how you can do this.

Facilitated Discussion

-

Discussion Questions

Use the following questions to engage the participants in the discussion. Feel free to use your own additional questions.

-

Lesson Summary

You should now be able to:

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

Lesson: The Note Assistant

153 Lesson Duration: 30 Minutes

Lesson Overview

In this lesson, you will learn how to use the Note Assistant, which allows you to comfortably install SAP Notes. Other advantage of the Note Assistant include the recognition of Note dependencies, and easy adjustment.

Lesson Objectives

After completing this lesson, you will be able to:

- Use the Note Assistant to implement source code corrections in an SAP system
- List the advantages of the Note Assistant
- Use the Note Assistant to adjust corrections implemented using SAP Notes

The Note Assistant helps your customers to implement corrections contained in SAP Notes. Without the Note Assistant, corrections had to be adjusted manually after you implemented a Support Package or upgrade. Customers also had to request an SSCR key before implementing SAP Notes. All this is much easier with the Note Assistant:

- No SSCR key is needed.
- Implementation support due to usage of the Modification Assistant.
- Dependencies between SAP Notes are automatically recognized.
- Validity is automatically tested.
- Modification adjustment is supported.

Business Example

The Note Assistant

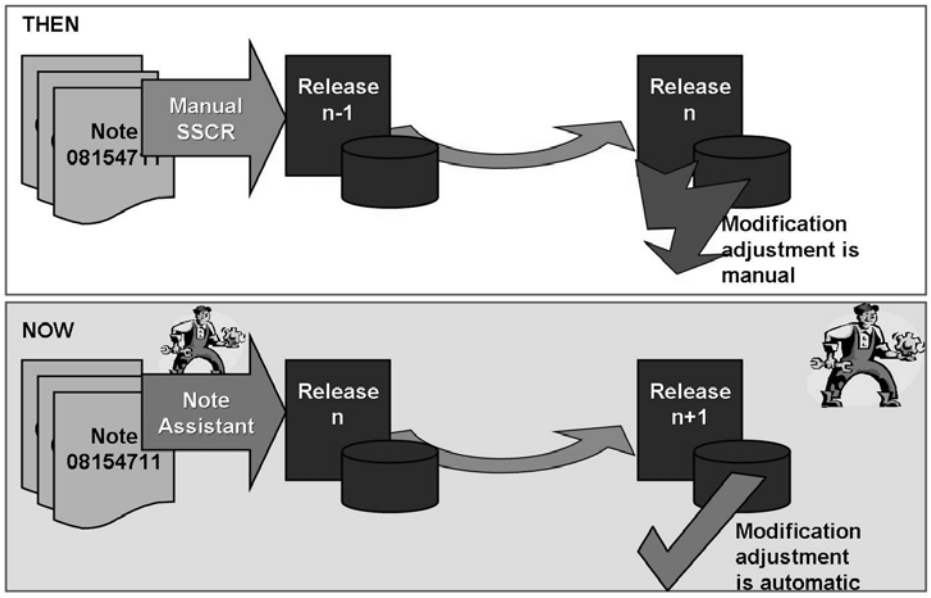


Figure 105: The Note Assistant: Principles

Before the Note Assistant was introduced, you had to manually implement all SAP Notes that required objects in your SAP system to be adjusted. Apart from the risk of error, this procedure also had other disadvantages: Manual adjustment of an SAP object requires a registration key to be specified for the object. Developers had to interpret the Note, to establish whether the correction of errors already in the system had any knock-on dependencies. Finally, the implemented error correction had to be compared manually during the import of Support Packages or upgrades.

The Note Assistant is used to address most of the disadvantages listed above. You do not have to enter a registration key. The Note Assistant uses the Modification Assistant to automatically implement in the system the corrections contained in the Note. It automatically recognizes dependencies to other Notes. If these Notes have not already been implemented in the system, the Note Assistant prompts you to import them. Finally, SAP Notes are only valid for a certain period of time. When you implement Support Packages and upgrades, the system checks whether the corrections contained in the SAP Note are still valid. If not, you simply select a button to reverse the correction.

- Automatically implement SAP Notes
  - Limited to corrections to ABAP source code
- Automatically handle dependencies
  - With other SAP Notes
  - With Support Packages
  - With modifications
- Overview of all Notes implemented in your SAP system
- Support of modification adjustment after upgrade or Support Packages
- Important: Correction of individual errors. Does not replace Support Packages.

Figure 106: The Note Assistant: Advantages

The Note Assistant helps you implement source code corrections in your SAP system. It usually implements these corrections automatically, which means that you do not have to import them manually, which significantly reduces the risk of error. Furthermore, you do not have the register the objects to be corrected in the SAP Software Change Registration (SSCR).

The Note Assistant offers many advantages. In addition to the automatic implementation of source code corrections without an SSCR key, the Assistant recognizes dependencies between different Notes. It also determines if an error described in a Note has already been eliminated in a Support Package, and if it has, marks the Note as obsolete. Modifications implemented in the object to be corrected are registered, and as a result, these objects are identified during modification adjustment.

A useful characteristic of the Note Assistant is the fact that you can display an overview of all Notes that have been implemented in the system.

The Notes are implemented in a separate category at modification adjustment, which takes place after the upgrade or Support Packages have been implemented. The Notes that are now obsolete in the current release are identified, which allows the object to be easily transferred to the standard system.

Hint: The Note Assistant allows you to easily implement individual source code corrections. However, it is not designed to replace the implementation of Support Packages.

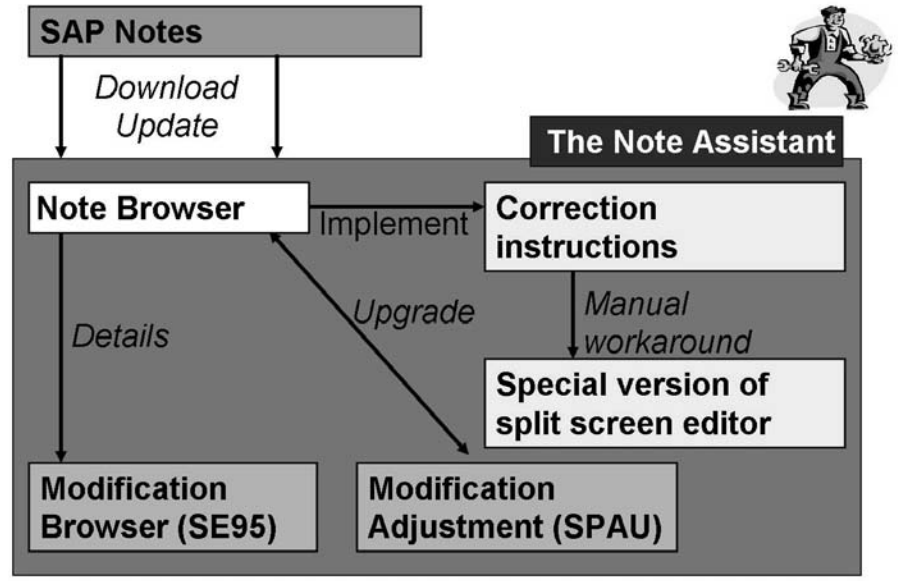


Figure 107: The Note Assistant: Work Steps

Using the Note Assistant, you can download existing SAP Notes from the SAP Service Marketplace. The Note Browser provides an overview of all Notes that have been implemented in your system. It also contains a link to the Modification Browser, which gives you an overview of all objects corrected using an SAP Note. Using the status specification, you can identify which SAP Notes can be implemented, have been implemented, or already obsolete.

Once you have carefully read the Note, you can implement it based on the correction instructions. The Note Assistant determines whether the Note can be implemented automatically, and if this is the case, a green traffic light is displayed beside the object to be implemented. If conflicts are identified during implementation, for example because the Note Assistant does not find the context block, you can implement the correction manually using a special version of the splitscreen editor.

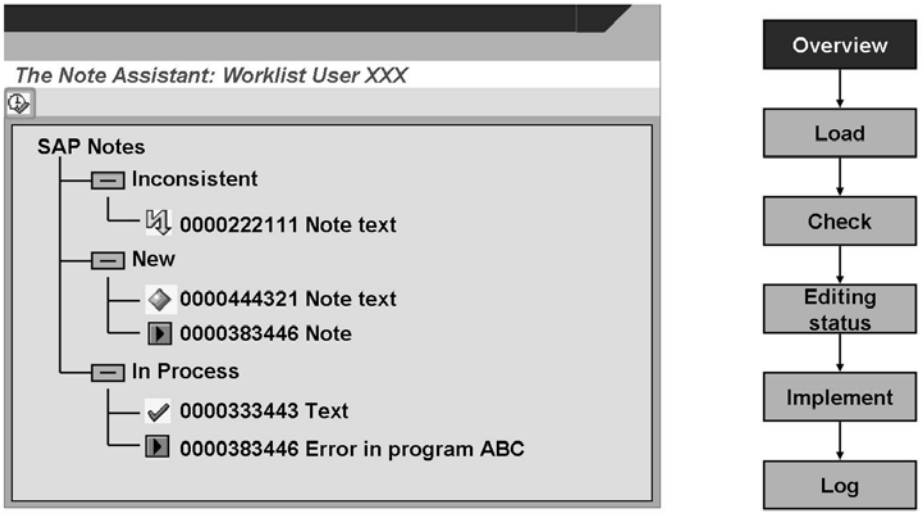


Figure 108: The Note Assistant: Worklist

Enter the transaction SNOTE in the command field. A list of Notes, which are assigned to different nodes, appears. This screen displays the following:

- Notes that you are responsible for editing
- All new Notes
- All Notes that have an inconsistent status

.

Notes that can be implemented are identified by a corresponding indicator (go to the menu path Utilities → Color to view the significance of the different icons). The example provided above also contains a new Note that is identified as obsolete, and cannot be implemented. This is because there are no correction instructions (the Note is for information purposes only), or because the validity period does not correspond to the system release and patch level.

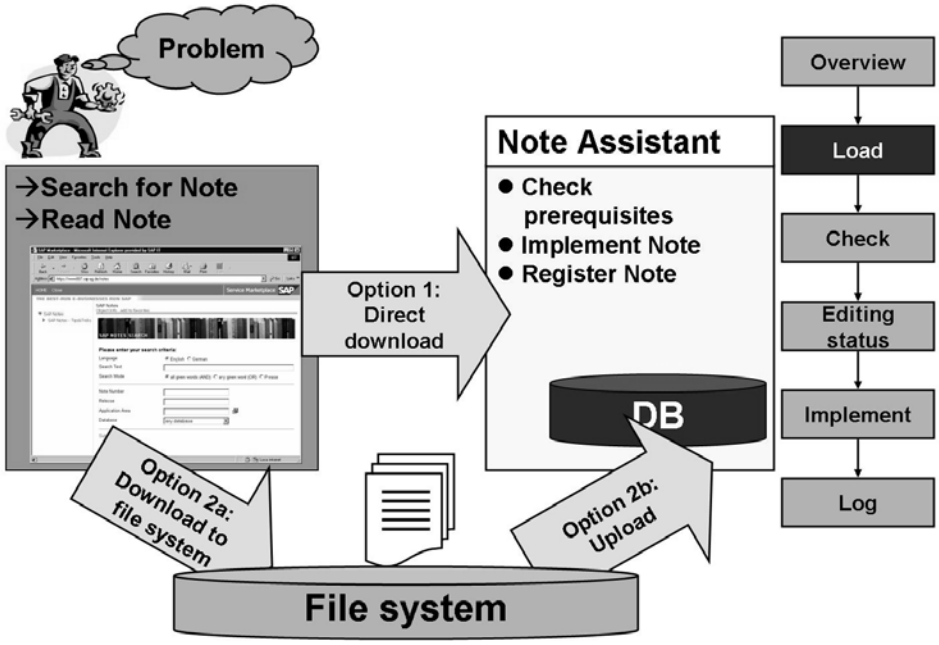


Figure 109: The Note Assistant: Loading an SAP Note

You can load SAP Notes from the SAP Service Marketplace or from the SAPNet – R/3 Frontend. You can use an RFC connection (Note download) or file transfer (Note upload).

When you download a Note, you can load the Note directly from the SAPNet – R/3 Frontend into your system using the Note Assistant.

When you upload a Note, you must first download the Note from the SAP Service Marketplace and save it to your PC's local drive. You must then upload the Note by file transfer using the Note Assistant.

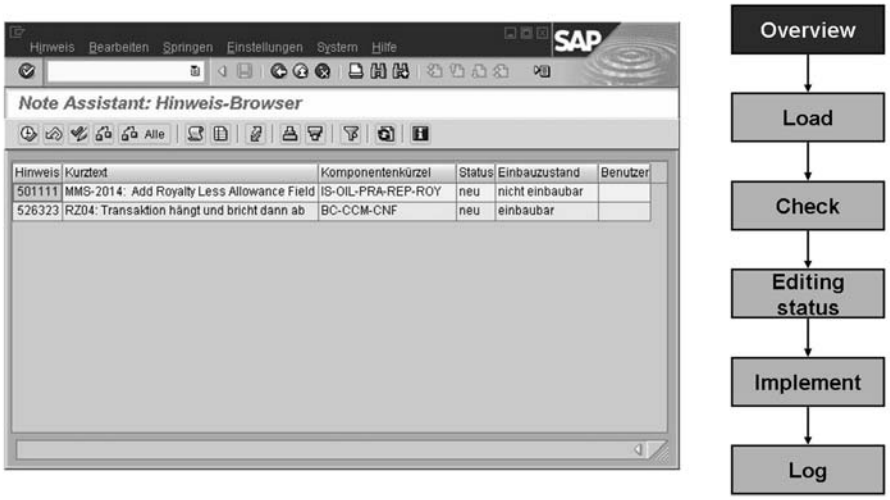


Figure 110: The Note Assistant: Note Browser

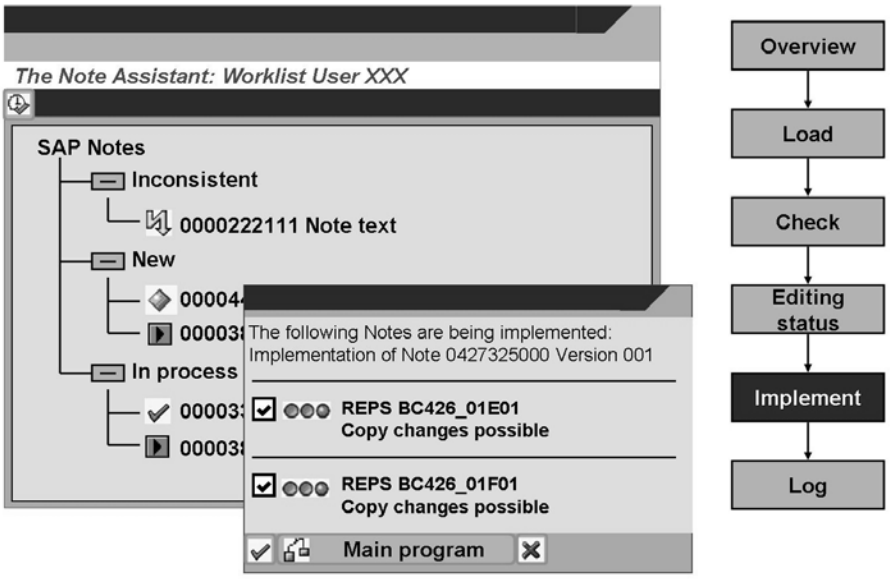


Figure 111: The Note Assistant: Implementing an SAP Note

You can implement the Note once you have read it carefully. The system automatically checks that any prerequisites have been fulfilled and whether there are dependencies on other Notes. If prerequisite Notes have not yet been registered in your system, the Note Assistant sends a list of Notes that need to be implemented first.

Note Assistant: Additional Characteristics

- Splitscreen Editor
- Varied log options:
  - Display the implementation status
  - Action Log
- Reverse Note implementation

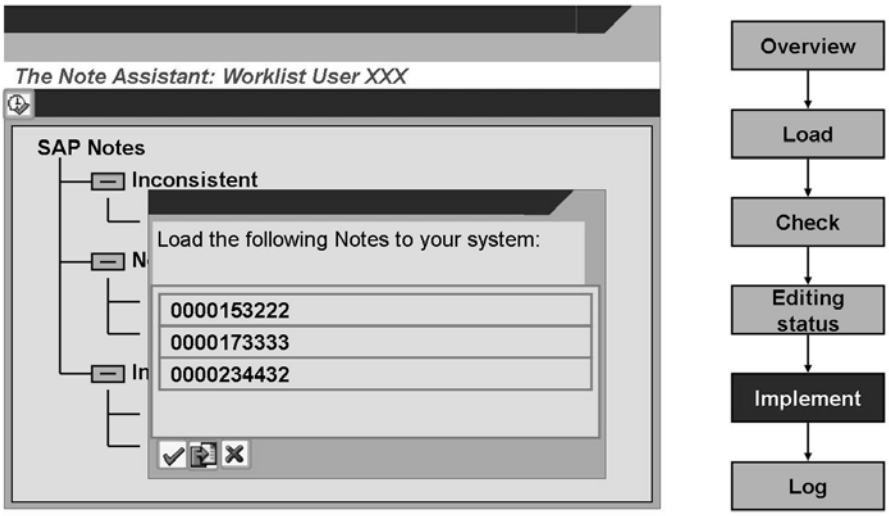


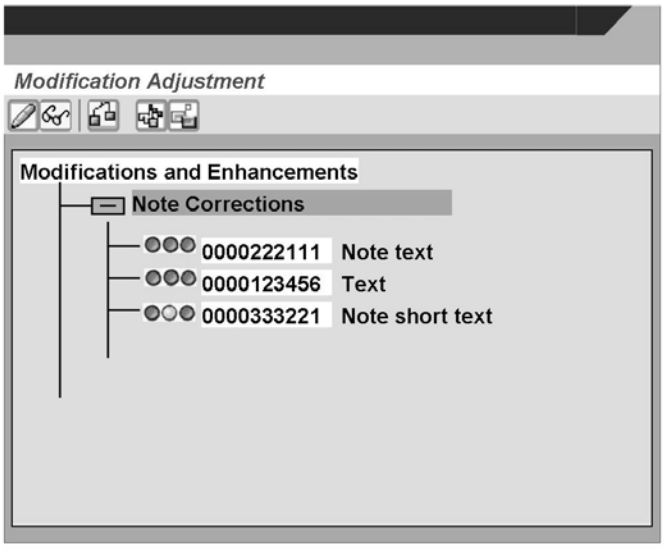
Figure 112: The Note Assistant: Notes with Dependencies

If the implementation of a Note requires that you implement other Notes first, the Note Assistant determines the prerequisite Notes at implementation, and imports them into your system. It displays the Notes you have selected with all additional Notes in a queue. The Notes must be implemented in the specified sequence.

You have the following options:

- Implement several Notes at once If you select this option, the system implements as many of the specified Notes as possible, one after the other. The system can implement several Notes at once if all corrections from the Notes can be transferred, without having to make changes. If the relevant includes contain your own modifications, the system may not be able to implement individual changes. The system implements these Notes individually, to allow you to adjust your modifications.
- Implement each Note individually The system implements the specified Notes individually. You can track in detail which source code change belongs to which of the specified Notes, and modify the source code change, if necessary.
- Cancel Note implementation The system does not implement any source code corrections.

Show how to implement a Note. You can use the Notes in the optional exercises for this topic. First, demonstrate the implementation of Note 427325000. Then disassemble this Note and implement Note 427706000. You can modify program SAPBC426_00 and program SAPMBC426_00 as mentioned in the exercises. Release your change request. Now, you can implement Note 427423000 to show the splitscreen editor.



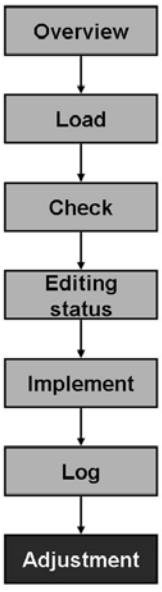


Figure 113: The Note Assistant: Modification Adjustment

A Support Package contains a collection of error corrections. You can import Support Packages into your system only as a complete package and only in a specified sequence. Each correction that is contained in a Support Package is documented in an SAP Note.

Once you implement a Support Package, the following scenarios exist:

- The correction has been implemented using an SAP Note and is contained in a Support Package (traffic light has no color): When you implement a Support Package, the system automatically checks whether you have already implemented a Note that contains individual corrections from the Support Package. During modification adjustment, these Notes are displayed using a traffic light with no color, which indicates that they are obsolete. You have to reset the relevant objects to SAP Original.
- The Note has been implemented, but is not contained in the Support Package (yellow traffic light) The system checks whether corrections that you have already implemented using an SAP Note have been overwritten in a Support Package that does not contain these corrections. You must then implement these source code corrections again. The system displays a yellow traffic light to indicate these Notes at modification adjustment (transaction SPAU). In your worklist, these Notes are displayed with the status *Inconsistent*, which indicates that they must be implemented again.
- The SAP Note is still implemented, although parts of it are contained in the Support Package (green traffic light) A Note contains several correction instructions with different validity periods, for example. When you implement the Support Package, one correction instruction becomes obsolete whereas the other remains valid. In this case, which only occurs rarely, the Note is displayed with a green traffic light.

Manual Creation of SAP Notes

- Program SCWN_REGISTER_NOTES
  - Enter Notes
  - Notes are loaded
  - Validity is checked
  - Implementation is checked
- Advantages:
  - Display in the Modification Browser
  - Dependencies are correctly specified
  - At modification adjustment, corrections are in the *Note Corrections* category.

If you have already implemented SAP Notes before you install the Note Assistant, you can retroactively inform the Note Assistant of this fact. This is important, as the Note Assistant cannot automatically identify if a Note has been implemented manually in your system without the Note Assistant. Registration offers the following benefits:

- All SAP Notes that have been implemented in your system are displayed in the Note Browser.
- If an SAP Note that has been implemented manually without the Note Assistant is a prerequisite for another SAP Note that you would like to implement using the Note Assistant, the Assistant recognizes that the prerequisite Note has already been implemented, and does not request you to implement it again.
- At modification adjustment, the manually implemented SAP Notes will also be displayed in the Note Corrections category for Support Packages or upgrades that will be implemented at a later stage. The system determines whether these Notes must be implemented again, or reset to the original state.

Note: To register Notes manually, start the program SCWN_REGISTER_NOTES. On the selection screen that appears, enter the numbers of the SAP Notes that are to be registered as fully implemented. The system loads the SAP Notes and checks whether they are valid for your release and Support Package level. The program also checks whether the Notes have already been registered as implemented in the Note Assistant.

Facilitated Discussion

Discussion Questions

Use the following questions to engage the participants in the discussion. Feel free to use your own additional questions.

Lesson Summary

You should now be able to:

- Use the Note Assistant to implement source code corrections in an SAP system
- List the advantages of the Note Assistant
- Use the Note Assistant to adjust corrections implemented using SAP Notes

Lesson: Modification Adjustment

164 Lesson Duration: 30 Minutes

Lesson Overview

In this lesson, you will learn how to implement a modification adjustment.

Lesson Objectives

After completing this lesson, you will be able to:

- Name the different steps in modification adjustment
- List which objects must be adjusted, and when
- Describe how modification adjustment is carried out in follow-on systems

Make sure that your students know that only objects for which they can make versions can be adjusted. This does not include logical databases, match codes, number range objects, or table indexes.

Business Example

You would like to correctly carry out modification adjustment after you have implemented an upgrade/Support Package.

Modification Adjustment

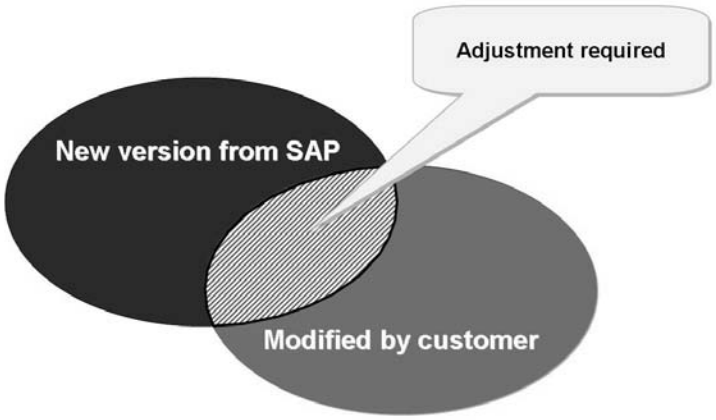


Figure 114: Objects for Adjustment

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.

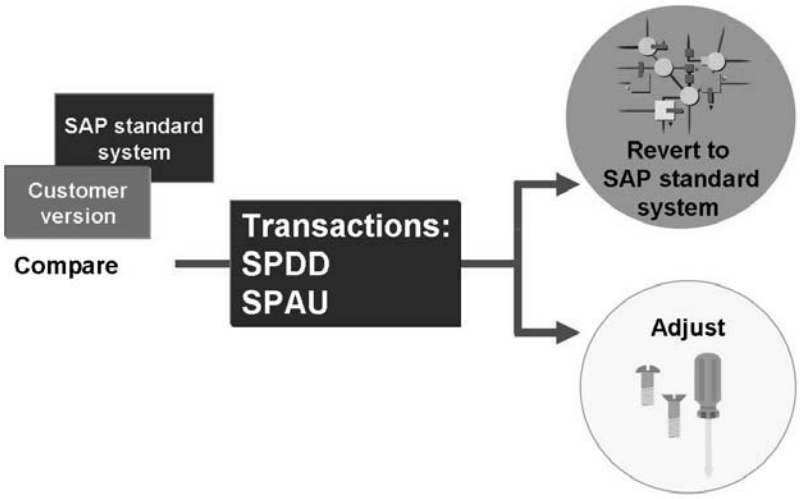


Figure 115: Modification Adjustment: SPDD and SPAU

The modification adjustment compares ABAP Repository objects from before the upgrade (older version) with the corresponding object after the upgrade (new version). During the adjustment, you can use 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

All other ABAP Repository objects are adjusted using transaction SPAU. The upgrade program R3up tells you to start the transaction after upgrade has finished. After an upgrade, you have 30 days to use transaction SPAU. After 30 days, you must reapply for an SSCR key for each object that you want to adjust.

Transaction SPAU determines which objects have been modified and 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.

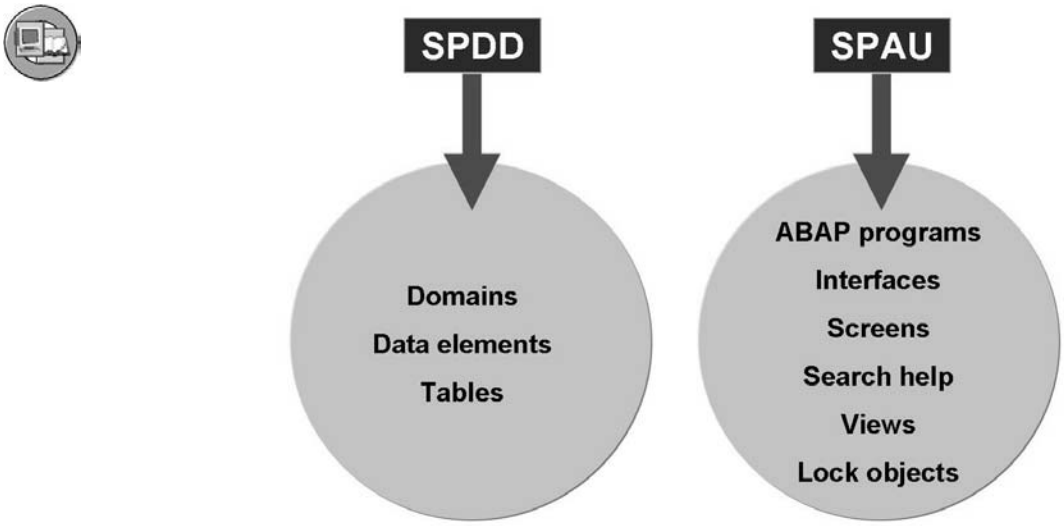


Figure 116: Modification Adjustment: Objects

Introduce transactions SPDD and SPAU. Show the relevant menu paths. Explain the individual buttons available on the interfaces of the adjustment tools.

Make sure that your students know that only objects for which they can make versions can be adjusted. This does not include logical databases, match codes, number range objects, or table indexes.

Explain that the upgrade tool can include one change request for ABAP Dictionary objects and one for all other Repository objects. This means that all objects that can be adjusted using SPAU should be assigned to the same change request.

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 objects have been imported (before the main import). At this point in time, no ABAP Dictionary objects have been generated. To ensure that no data is lost, it is important that any customer modifications to the object types listed above are undertaken prior to their generation, as data may otherwise be lost.

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.

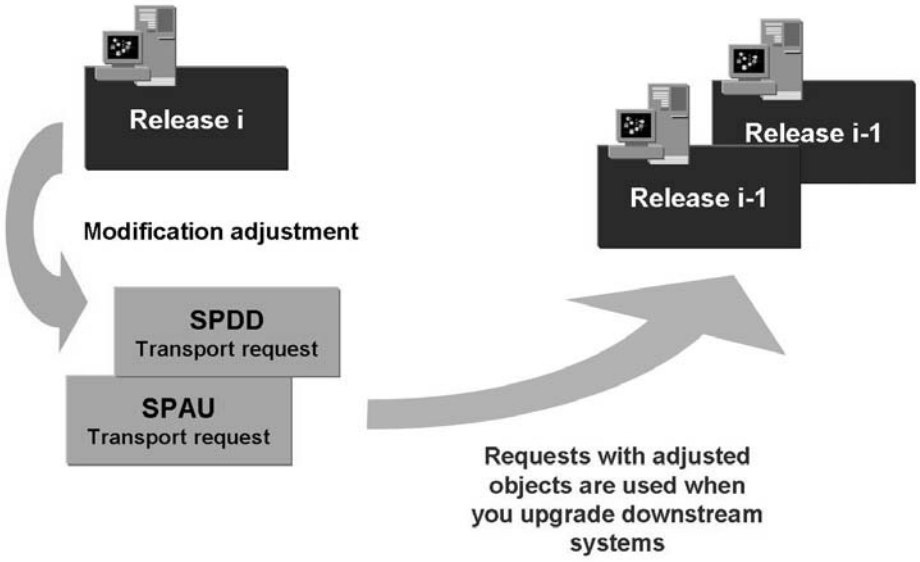


Figure 117: Transporting Adjustments between Systems

During modification adjustment, you should use two different change requests to implement the changes you have made: One for SPDD adjustments and another for SPAU adjustments. These change requests are then transported into the follow-on R/3 systems that you want to adjust. This guarantees that all actual adjustment work only needs to be carried out in your development system.

When upgrading additional R/3 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 R/3 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.

Hint: For this process to be effective, it is important that all systems in your landscape have the same status. This can be guaranteed, for example, by first making modifications in your development system and then transporting them to follow-on systems before you upgrade the development system. You can also guarantee that all of systems in your landscape have the same status by creating your development system before upgrade as a copy of your production system and then refraining from modifying the production system again.

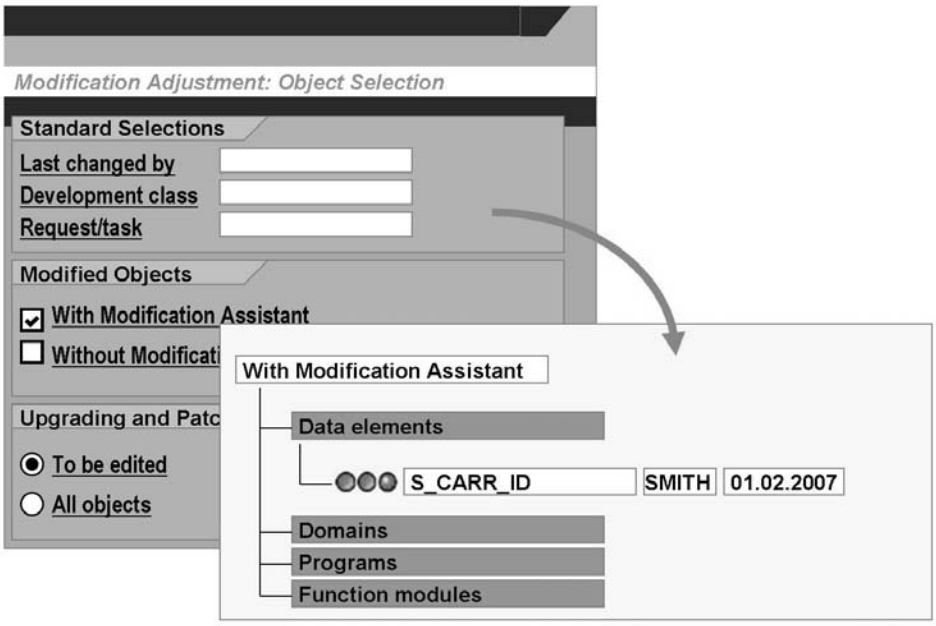


Figure 118: Modification Adjustment: Initial Screen

When you start 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

Show how modification adjustments work. Even if you do not implement your Support Package, there are enough programs to use for demonstration purposes.

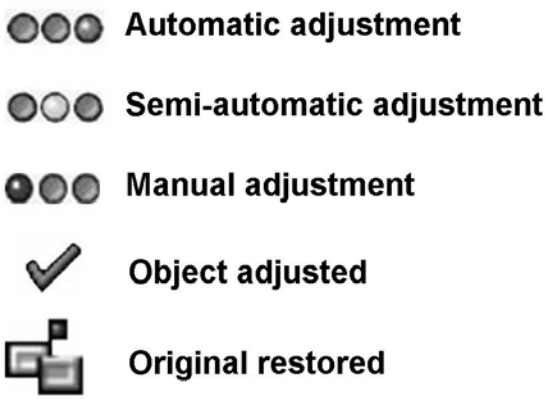


Figure 119: Modification Assistant Icons

The icons in front of the individual objects that need adjustment show how they can be adjusted. The following adjustment methods can be used:

- Automatic The system does not identify any conflicts. The changes can be adopted automatically To use this option, select the relevant icon, or choose the relevant entry from the menu.
- Semi-automatic The individual tools support you during the adjustment. When the programs are adjusted, the splitscreen editor is called. You can use this to transfer your changes.
- Manually You must process your modifications with no specific support from the system. In this case, the modification adjustment allows you to go directly to the relevant tool.

Adjusted objects are denoted by a green tick.

If you want to use the new SAP standard version, use “Restore Original”. If you do this, you will have no further adjustment work in future.

171

Exercise 7: Modification Adjustment

Exercise Duration: 20 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Adjust modifications

Business Example

You need to adjust the modifications made in the system after implementing a Support Package or an upgrade.

Task:

Modification Adjustment

1. Adjust any modifications you made to the system to the new objects that you have imported into the system.

Solution 7: Modification Adjustment

Task:

Modification Adjustment

1. Adjust any modifications you made to the system to the new objects that you have imported into the system.

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

173

Exercise 8: Modification Adjustment – Optional

Exercise Duration: 30 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Adjust modifications after you have implemented a Support Package.

Business Example

You have implemented Notes as advance corrections in your system. You have also made some modifications. Notes are also always contained and delivered as part of Support Packages. When you implement a Support Package, some of the modifications and corrections implemented using Notes may be overwritten. For this reason, you need to implement a modification adjustment.

Task 1:

Start modification adjustment by selecting *SAP Menu → Tools → ABAP Workbench → Utilities → Maintenance → Upgrade Utilities → Program Compare* or call transaction **SPAU**.

1. Enter your user name in the “Last changed by” field. Execute the program.
2. View the objects in the different categories. What significance do the icons have?

Task 2:

Adjust the modifications. Start at the top and work your way down:

1. Start with the objects in the “Note Corrections category”.
2. If adjustment mode has not been defined, restart the function by double-clicking the question mark.
3. If necessary, create a new change request.
4. Continue with the category “With Modification Assistant”.
5. Finally, work on the category “Without Modification Assistant”.

Continued on next page

Task 3:

1. Once all object have been adjusted, release your change request.

Solution 8: Modification Adjustment – Optional

Task 1:

Start modification adjustment by selecting *SAP Menu → Tools → ABAP Workbench → Utilities → Maintenance → Upgrade Utilities → Program Compare* or call transaction **SPAU**.

1. Enter your user name in the “Last changed by” field. Execute the program.

   a) –

2. View the objects in the different categories. What significance do the icons have?

   a) The significance of the icons is explained in the documentation on modification adjustment.

Task 2:

Adjust the modifications. Start at the top and work your way down:

1. Start with the objects in the “Note Corrections category”.

   a) –

2. If adjustment mode has not been defined, restart the function by double-clicking the question mark.

   a) –

3. If necessary, create a new change request.

   a) –

4. Continue with the category “With Modification Assistant”.

   a) –

5. Finally, work on the category “Without Modification Assistant”.

   a) –

Task 3:

1. Once all object have been adjusted, release your change request.

   a) –

Lesson Summary

You should now be able to:

- Name the different steps in modification adjustment
- List which objects must be adjusted, and when
- Describe how modification adjustment is carried out in follow-on systems

Unit Summary

You should now be able to:

- Describe what a modification is
- Explain the basic concepts in the modification environment
- Describe what you must bear in mind when implementing modifications
- Explain how the Modification Assistant works
- Conduct specialized modifications using the Modification Assistant
- List your modifications using the Modification Browser
- Describe what user exits are and how they work
- Describe how to find user exits in the system and use them to enhance SAP software
- Use the Note Assistant to implement source code corrections in an SAP system
- List the advantages of the Note Assistant
- Use the Note Assistant to adjust corrections implemented using SAP Notes
- Name the different steps in modification adjustment
- List which objects must be adjusted, and when
- Describe how modification adjustment is carried out in follow-on systems

Unit 6

177

Enhancements

Caution: As the package DNW7AW is assigned to the software component TRAINING, which has not (yet) been integrated into the training system, you cannot enhance objects in this package. Until the TRAINING component is available in the training system, you must change the software component assignment of the package DNW7AW to SAP_ABA (modification) before the course starts, to run all the demos and exercises. The easiest way to do this is with the report SAPDNW7AW_COURSE_PREPARATION. (This directly manipulates the management table TDVEC – we do not recommend you follow this same procedure in live systems.)

Unit Overview

Enhancements give you the option of adapting SAP software to customer requirements without the need for modifications. Various techniques have been developed for this over the years, such as user exits, customer exits, Business Transaction Events (BTEs) and, as of *R/3 Release 4.6A*, Business Add Ins (BAdIs). In *SAP Web Application Server Release 6.10*, BAdIs were extended to include screen exits. These will be dealt with in the first lesson.

In *SAP NetWeaver Release 7.0*, the existing enhancement options were significantly increased with the introduction of the Enhancement Framework. Implicit enhancements constituted one if the most important innovations here. You can use them to add your own source code at certain points in programs, such as at the beginning of a function module or the end of an include. You can also add your own parameters to function modules or global methods. The new BAdIs also constitute one of the most important components of the Enhancement Framework. These are more efficient and flexible than the classic variant. The Enhancement Framework is dealt with in the second lesson.

Unit Objectives

After completing this unit, you will be able to:

- Explain the new enhancement options that exist and how they are managed
- List where implicit enhancement points exist and use these enhancement points for enhancing SAP software
- Find explicit enhancement points and enhancement sections and use these for enhancing or replacing SAP software
- Explain why SAP introduced the new BAdI technology
- Explain the concept and the runtime architecture of the new BAdI technology
- Search for new BAdIs and use these to enhance SAP software
- Explain how the existing enhancement implementation of industry solutions are integrated in the Switch Framework and can be activated

Unit Contents

Lesson: The New Enhancement Concept

179 Lesson Duration: 180 Minutes

Lesson Overview

In this lesson you will learn the advantages and use of the new enhancement options as well as the new BAdI technology, which were introduced with *SAP NetWeaver 7.0*.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain the new enhancement options that exist and how they are managed
- List where implicit enhancement points exist and use these enhancement points for enhancing SAP software
- Find explicit enhancement points and enhancement sections and use these for enhancing or replacing SAP software
- Explain why SAP introduced the new BAdI technology
- Explain the concept and the runtime architecture of the new BAdI technology
- Search for new BAdIs and use these to enhance SAP software
- Explain how the existing enhancement implementation of industry solutions are integrated in the Switch Framework and can be activated

---

Due to the complexity of the subject, it is important that you first provide the participants with a good overview of the options before going into detail.

You should definitely provide demos to illustrate the respective content. You will find these in the package DNW7AW.

The participants should go through the exercises to extend the knowledge they have acquired.

Caution: Since the package DNW7AW is assigned to the software component TRAINING, which has not (yet) been integrated into the training system, you cannot enhance objects in this package. Until the TRAINING component is available in the training system, you have to change the software component assignment of the package DNW7AW to SAP_ABA (modification) before the course starts, so that all the demos and exercises can run. The easiest way to do this is with the report SAPDNW7AW_COURSE_PREPARATION (which directly manipulates the management table TDVEC – we do not recommend you follow this same procedure in productive systems...)

Business Example

After upgrading to *SAP NetWeaver 7.0*, you want to use the new enhancement options and the new BAdI technology to enhance your SAP software.

Enhancement Points and Enhancement Sections

In this section you will first be provided with an overview of the new enhancement options and how to manage them. After this, you will learn where implicit enhancement points exist and how you can use these for enhancing SAP software. Finally, you will learn how you can use explicit enhancement points and enhancement sections as enhancement options.

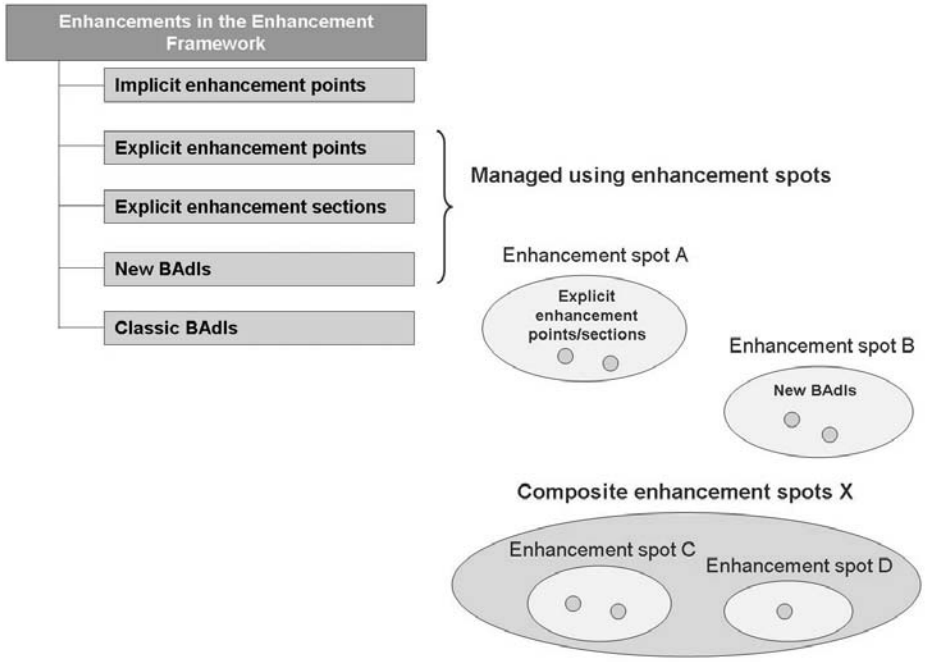


Figure 120: New Enhancement Concept (Overview)

As of *SAP NetWeaver 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. Furthermore, SAP implemented the new BAdI technology for performance reasons, and other reasons that will be described later.

The graphic above illustrates how enhancement points and sections, and BAdIs that have been created using the new technology, are grouped together and managed using enhancement spots. Composite enhancement spots comprise both simple enhancement spots and/or other composite enhancement spots. They serve to semantically bundle enhancement spots.

Previous BAdIs (classic BAdIs) exist in the system as before. In future, however, SAP will offer only BAdIs using the new technology.

The next graphic shows what enhancement points are.

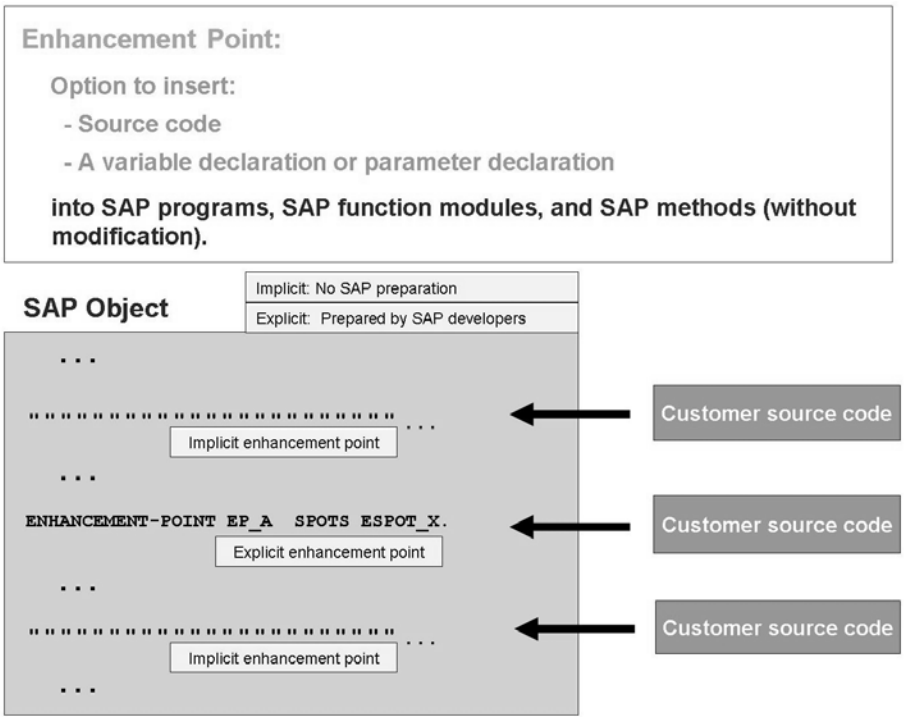


Figure 121: Enhancement points

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. The following graphics show at what points implicit enhancement points are present.

Implicit Enhancement Points

- At the end of a structure (type) declaration before "END OF ..."
  (to include additional fields)
- At the beginning and end of:
  - Subroutines
  - 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)

...

Figure 122: Implicit Enhancement Points (1)

...

- At the end of the public/protected/private section of a local class
  (to define additional attributes and additional methods)
- For global classes, you can define any of the following:
  - Additional attributes
  - Additional methods
- For a method of a global class, you can define:
  - A pre-method and/or
  - A post-method
  (automatic execution when method starts/ends)
  Alternative: Define an overwrite method (this 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)

Figure 123: Implicit Enhancement Points (2)

To use an implicit enhancement point, you implement an **enhancement implementation**. The following describes the **procedure** for using the various implicit enhancement points.

Using implicit enhancement points with which you can insert source code (implicit source code plugins):

1. Display the SAP object (program, function module, method)
2. In the GUI status, choose the enhancement button
3. Choose the menu path *Edit → Enhancement Operations → Show Implicit Enhancement Options* to show the implicit enhancement options.
4. Create an enhancement implementation using the context menu in the editor
5. Insert the source code
6. In the GUI status, choose the "Activate Enhancements" button.

Important: When you enhance a structure (type) declaration directly before `END OF...`, you **must** use the syntax

```
DATA <additional field> TYPE <Type>
```

because otherwise the program would be syntactically incorrect.

At this point the participants could do the exercise on implicit enhancement points.

Enhancing the interfaces of SAP function modules and methods of global classes:

Insert new interface parameters including typing in the Function Builder or Class Builder by following the menu path *Function Module → Enhance Interface* or *Class → Enhance*.

Hint: These newly added interface parameters are generally optional and they can be activated in the source code enhancements of the corresponding function modules or methods.

Enhance a simple function module, for example FI_COMPANYCODE_GETDETAIL. Since you will probably not be the only one teaching on the system, you should add a statement to your source code such as the following:

```
IF sy-uname = <YOUR_NAME>.
```

Defining additional attributes/methods of global classes:

Define additional attributes and methods in the Class Builder by following the menu path *Class → Enhance*.

Hint: Double-click an additional method to branch to the method editor for the implementation.

Such additional attributes and methods can be addressed in the source code enhancements of methods of the global class.

Defining a pre/post/overwrite method for the method of a global class:

1. Switch to enhancement mode in the Class Builder by following the menu path *Class → Enhance*.
2. Use the cursor to select the required SAP method.
3. Choose the menu path *Edit → Enhancement Operations*, then choose one of the menu entries: Insert Pre-Method, Insert Post-Method or Add Overwrite Method
4. Choose the new pushbutton in the column “Pre-(Post-/Overwrite-)Exit” to implement the corresponding method.

Hint: For each SAP method, you can define a pre method and/or a post method. As an alternative, you can create an overwrite method, which replaces the SAP method. These enhancement methods have the same interface as the original 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 class. They cannot access the private components of the class.

Demo: Enhance the class CL_DNW7AW_ENH_IMPLIC_PLANE_D1 with a private additional instance attribute `me_span TYPE i DEFAULT 30.`, for example. Define an appropriate additional getter method. Enhance the constructor with a suitable additional parameter and use the implicit enhancement option at the end of the constructor source code to map the parameter to the attribute. Then create an overwrite method for DISPLAY_ATTRIBUTES that also outputs the additional attribute.

You could have the participants do the exercise on class enhancements here.

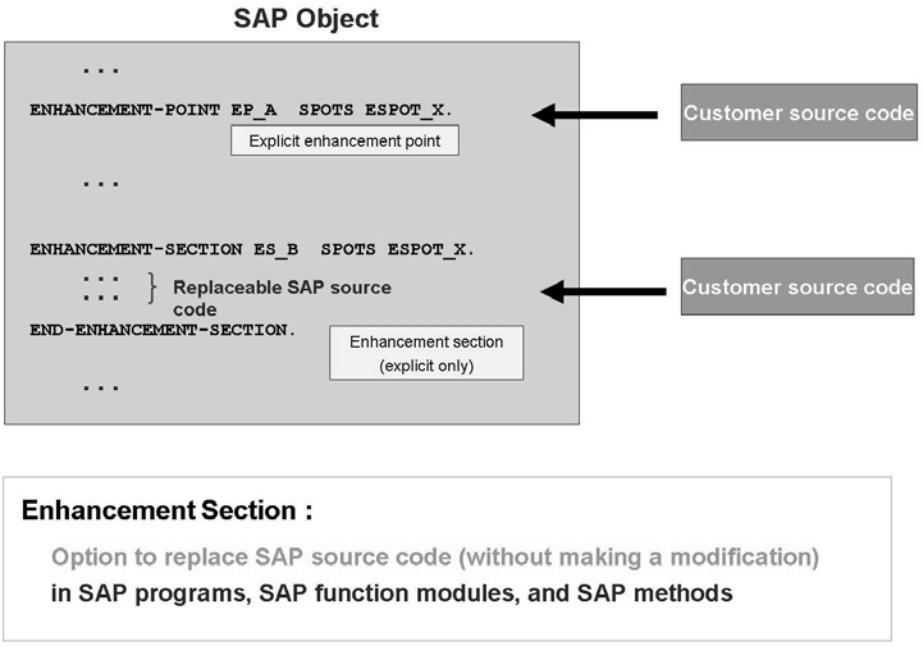


Figure 124: Explicit Enhancement Points and Enhancement Sections

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 to allow you to replace the SAP source code without making a modification. There are no implicit enhancement sections.

Explicit enhancement points and sections are always embedded in enhancement spots.

Explicit enhancement points and sections that enable ***source code enhancement 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, implement an **enhancement implementation** (an implementation of the higher-level enhancement spot). The following steps describe how to use explicit enhancement points and enhancement sections:

1. Display the SAP object (program, function module, method)
2. Search for the required enhancement point/section
3. In the GUI status, choose the enhancement button
4. Create the enhancement implementation using the context menu of the enhancement point/section
5. Specify the name of the enhancement implementation
6. Enter the source code
7. In the GUI status, choose the *Activate Enhancements* button

Hint: Note that elements of the central SAP Basis cannot be enhanced.

Demo: It is best to create a simple program with explicit enhancement options yourself. In the second step, create implementations.

At this point, the participants could do the exercise on explicit enhancement points and enhancement sections.

The New BAdI Technology

SAP introduced the new BAdI technology with *SAP NetWeaver 7.0* primarily to improve performance and enhance functions. The individual reasons are shown in the following graphic.

Why new BAdI Technology?

- **Improved performance**
- **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**

Figure 125: Reasons for the New BAdI Technology

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 to activate and deactivate the BAdI implementations implemented by Industry Solutions (see below).

Older BAdIs that were created using the classic technology remain in the system. In future, SAP will offer only BAdIs using the new technology or using explicit enhancement points and enhancement sections.

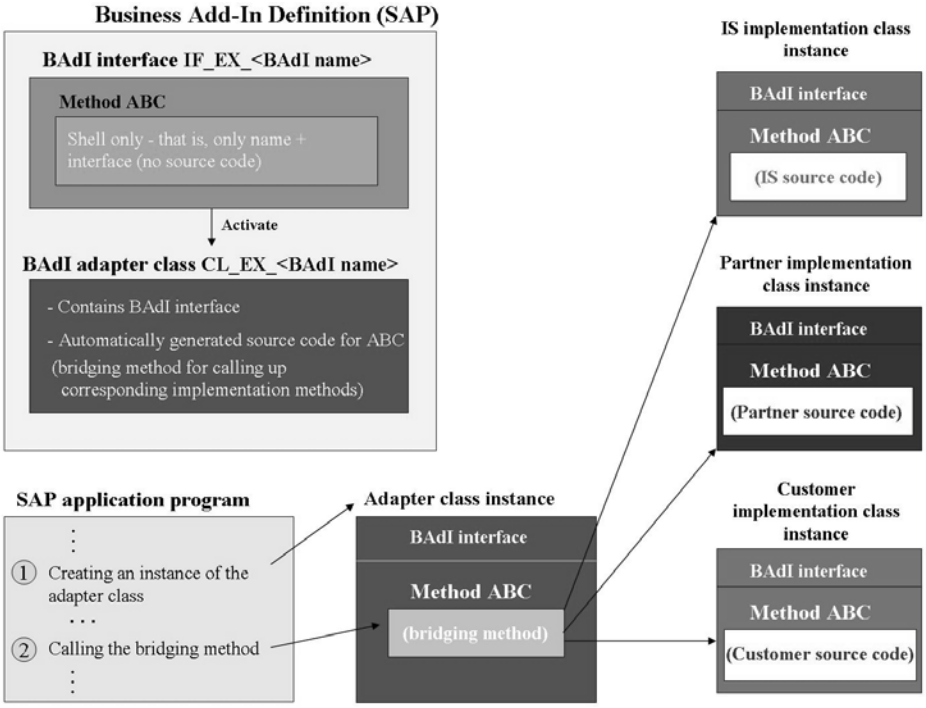


Figure 126: Classic BAdIs (Architecture)

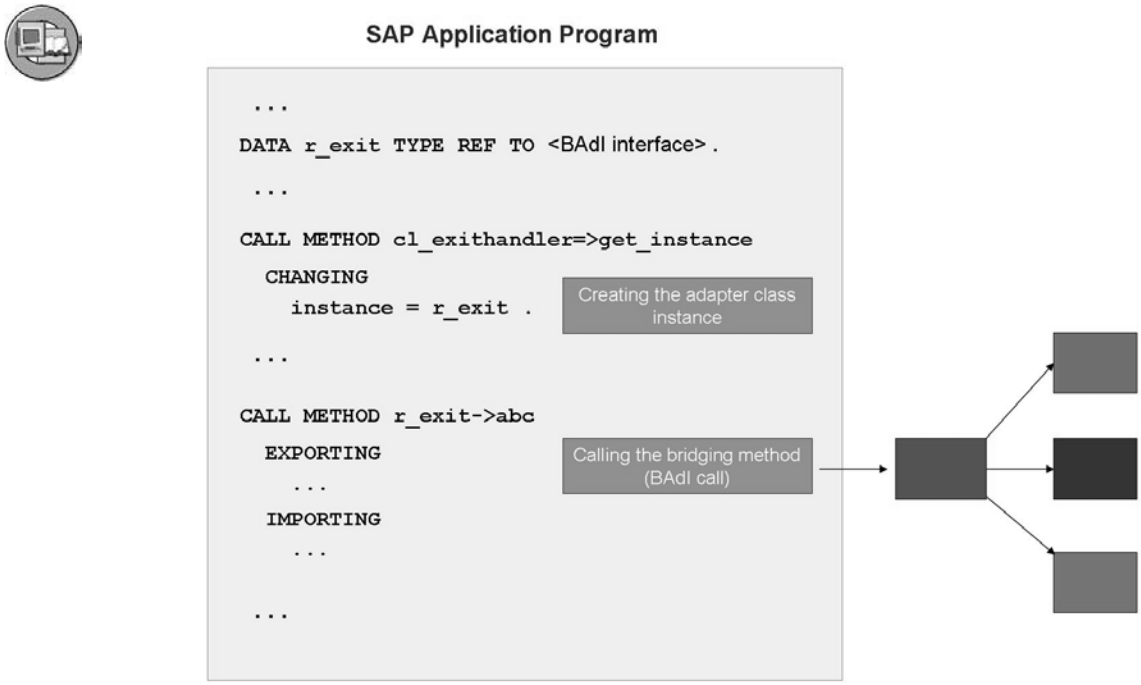


Figure 127: Classic BAdIs (Call Syntax in SAP Program)

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

The new BAdI technology works in the same way. However, adapter classes are no longer created, which means the SAP application program does not have to instantiate them. Instead, at the runtime for the application program, the system generates a BAdI handle in the kernel that performs the same function as the adapter class, but calls the available implementation methods much more efficiently.

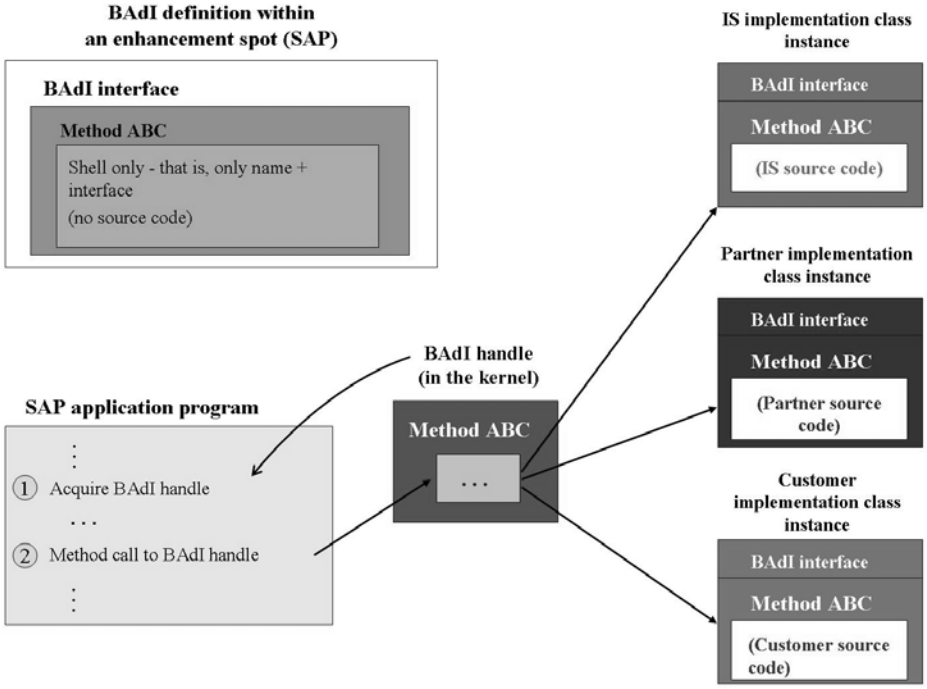


Figure 128: New BAdIs (Architecture)

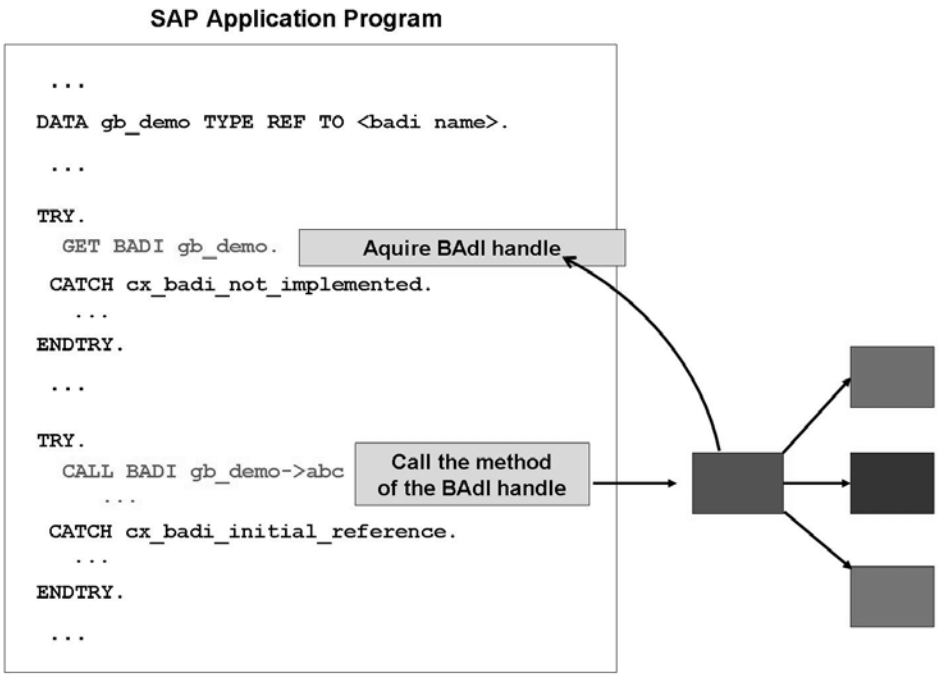


Figure 129: New BAdIs (Call Syntax in SAP Program)

The above graphic shows the call syntax for the new BAdIs.

If no active implementation of the BAdI is found at `GET BADI`, the exception *cx_badi_not_implemented* is raised.

If the handle reference is initial for `CALL BADI` (for example, because `GET BADI` failed), the exception *cx_badi_initial_reference* is raised.

The following graphic shows the procedure for searching for BAdIs.

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 function modules and methods that are called)
2. Double-click the reference variable to navigate to the variable definition.
3. Double-click the BAdI name to navigate to the display for the corresponding enhancement spot.

Figure 130: Searching for BAdIs

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.

Steps for creating a BAdI implementation (BAdI use):

1: Display the relevant enhancement spot.
2. Choose "Implement Enhancement Spot" (F6) to 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 navigation area for the BAdI you require, click the corresponding component "Implementing class".
7. Enter the name of the implementing class and choose "Change".
   (This can be inherited from the sample class or you can copy it.)
8. Double-click the method(s) to implement/adjust them.
9. Activate the method and the enhancement implementation.

Figure 131: Using BAdIs

The above description for searching for and using BAdIs relates to program exits (the most common exit type).

You can search for and implement menu and screen exits in exactly the same way as with the classic 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 dynpro for screen exits.

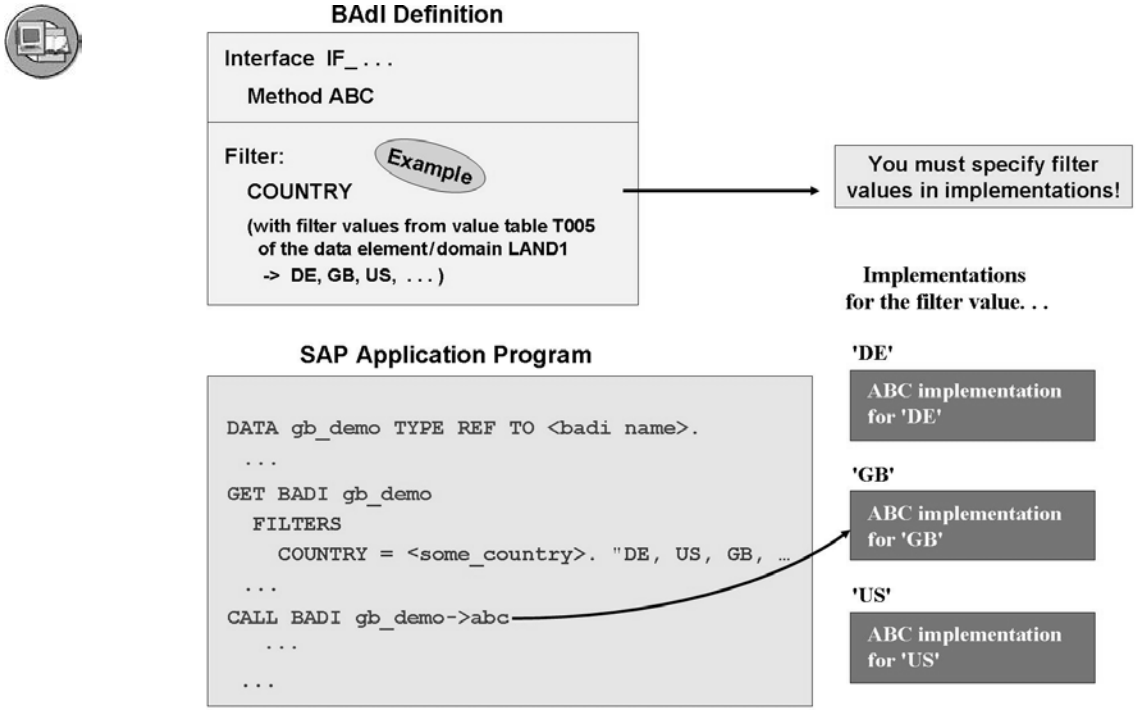


Figure 132: Filter-dependent BAdIs

The above graphic illustrates the concept of filter-dependent BAdIs, which corresponds to the classic BAdI concept. However, the functions have been enhanced. Numeric filters can now also be used. For implementations, you can now specify not only single filter values, but also filter conditions by using the operators <>, >, <, >=, <=, CP, NP. You can also define multiple filters for a BAdI.

Hint: 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"*

Switch Framework

The idea of the Switch Framework is that customers receive all industry solutions as a complete package and they can activate those they want to use. All the other solutions are available but they cannot be used. SAP has also decided to use enhancement packages to deliver future developments. Customers can then decide which new functions they want to activate.

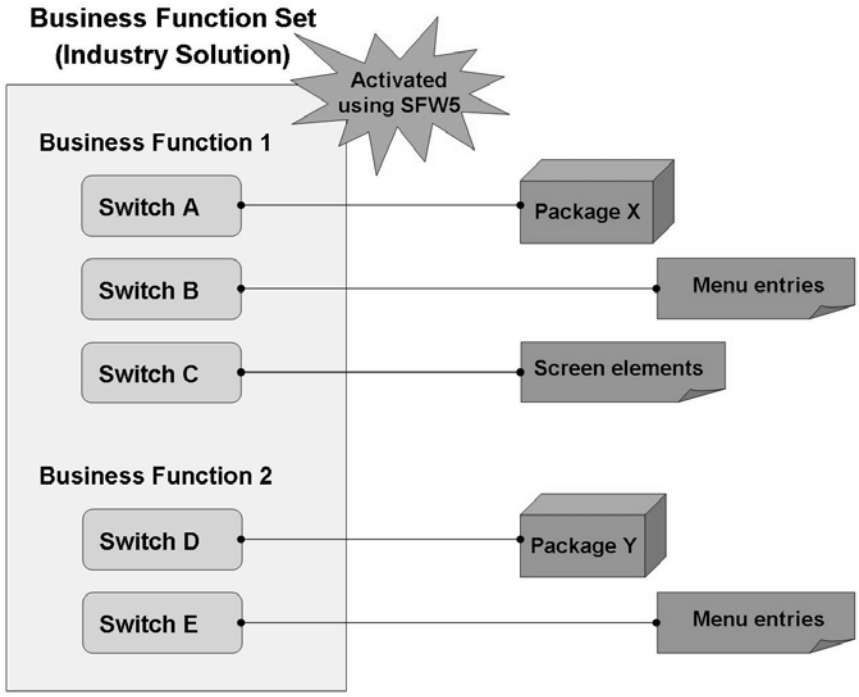


Figure 133: Switch Framework

You can use the Switch Framework to create switches and to assign packages, screen elements and menu entries to them. You can use a business function to group these switches.

One industry solution from SAP is a collection of business functions called a business function set. You can use transaction SFW5 to activate just one business function set, and/or to activate and deactivate business functions.

Demo: SFW5.

Customers can also use the Switch Framework to activate and deactivate enhancement implementations.

The customer defines a switch in transaction SFW1 and then assigns the package with the implementations to be activated to it. The customer also defines a business function (in transaction SFW2), to which the switch is assigned.

When you assign a switch to a business function, you must specify the assignment type. *Activation* and *Enabling/Standby* are possible options.

Activation assigns all switch/package objects to the business function, whereas *Enabling/Standby* assigns only the Dictionary objects. You should therefore choose *Activation*.

The business function can be activated and deactivated using transaction SFW5. When it is deactivated, all package objects that can be activated (this includes enhancements) become ineffective although they are still available in the system. For deactivation, however, the business function must be defined as reversible and the package must not contain any Dictionary objects.

As a demonstration, you should create a switch and assign it to your package with the previous enhancement implementations.

Then, you should create a business function and assign your switch to the business function.

Now show that your previous implementations are no longer effective although they are still available.

Activate your business function and check whether or not your previous implementations are effective.

 195

Exercise 9: Implicit Enhancement Points

Exercise Duration: 15 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Use implicit enhancement points to enhance SAP source code

Business Example

You want to use implicit enhancement points to enhance the definition of a structure variable and a subroutine in an SAP program without having to make modifications.

System Data

System:	Will be assigned
Client:	Will be assigned
User ID:	Will be assigned
Password:	Will be assigned
Set up instructions:	No special settings required in the standard training system

Task:

Enhance a structure definition and a subroutine

1. Analyze the source code of the SAP program **SAPDNW7AW_ENH_IMPLIC_##**. (## stands for your two-digit group number.)
2. Use the corresponding implicit enhancement point to add the fields *cityfrom* and *cityto* to the definition of the structural variable *gs_flight* (Use the data element S_CITY to type each field). Choose **ZDNW7AW_ENH_IMPLIC_##** as the name of your enhancement implementation.
3. In the subroutine, fill the additional fields *cityfrom* and *cityto* of the formal parameter *ps_flight* with any values before the fields *carrid* and *connid* are output (using the corresponding implicit enhancement point).
4. Use the corresponding implicit enhancement point in the subroutine after the fields *carrid* and *connid* have been output to output *fldate*, *cityfrom* and *cityto* as well.
5. Activate your enhancements and test the program.

Solution 9: Implicit Enhancement Points

Task:

Enhance a structure definition and a subroutine

1. Analyze the source code of the SAP program **SAPDNW7AW_ENH_IMPLIC_##**. (## stands for your two-digit group number.)

   a) -

2. Use the corresponding implicit enhancement point to add the fields *cityfrom* and *cityto* to the definition of the structural variable *gs_flight* (Use the data element S_CITY to type each field). Choose **ZDNW7AW_ENH_IMPLIC_##** as the name of your enhancement implementation.

   a) Show the program SAPDNW7AW_ENH_IMPLIC_## in display mode in the Object Navigator. Go to enhancement mode: *Program → Enhance*. Then choose *Edit → Enhancement Operations → Show Implicit Enhancement Options*. Place the cursor on the point indicated within the structure definition, call the context menu for the editor and choose *Enhancement Implementation → Create*. On the following screen, enter **ZDNW7AW_ENH_IMPLIC_##**. Maintain the short text and choose Enter. Save your entries.

   b) For the source code, define:

   ```
   DATA: cityfrom TYPE s_city,
     cityto TYPE s_city.
   ```

3. In the subroutine, fill the additional fields *cityfrom* and *cityto* of the formal parameter *ps_flight* with any values before the fields *carrid* and *connid* are output (using the corresponding implicit enhancement point).

   a) Place the cursor in the first row of the subroutine write_str. From the context menu, choose *Enhancement Implementation → Create*. On the following screen, choose *Declaration*. Then, on the next screen, select the implementation you have already created and confirm this selection with Enter.

   b) For the source code, define for example:

   ```
   ps_flight-cityfrom = 'Heidelberg'.
   ps_flight-cityto = 'Mainz'.
   ```

Continued on next page

4. Use the corresponding implicit enhancement point in the subroutine after the fields *carrid* and *connid* have been output to output *fldate*, *cityfrom* and *cityto* as well.

   a) See the equivalent approach above

   b) For the source code, define:

   ```
   WRITE: / ps_flight-carrid, ps_flight-connid.
   ```

5. Activate your enhancements and test the program.

   a) Press **Ctrl + F3** and **F8**.

199

Exercise 10: Implicit Enhancements of Classes

Exercise Duration: 30 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Enhance global classes

Business Example

SAP supplies a truck class that does not meet all your expectations. You therefore enhance (not modify!) the class CL_DNW7AW_ENH_IMPLIC_TRUCK_##. (## stands for your group number.)

System Data

System:	Will be assigned
Client:	Will be assigned
User ID:	Will be assigned
Password:	Will be assigned
Set up instructions:	No special instructions required when using a standard training system.

Task 1:

Create an additional attribute

1. Within an enhancement implementation ZDNW7AW_TRUCK##a, create a new type I private instance attribute for the class CL_DNW7AW_ENH_IMPLIC_TRUCK_##. It is to be used to record the maximum speed of a truck. (A necessary additional unit of measure such as km/h or mph has been dispensed with here.)

   Suggested name: *me_max_speed*. Initial value: 100.

2. Activate your enhancement implementation.

3. In accordance with the attribute, create a functional getter method GET_MAX_SPEED.

4. Activate your enhancement implementation.

Continued on next page

5. Test the class in the Class Builder test environment: Create a truck (in other words, an object of the class CL_DNW7AW_ENH_IMPLIC_TRUCK_##) and call your getter method for the maximum speed.

Task 2:

Create an additional parameter for the method

1. Enhance the class constructor with an optional parameter *ie_max_speed*, with which you can determine the speed when you create a truck.
2. Activate your enhancement implementation.
3. Use the implicitly available enhancement point at the end of the source code for the constructor to map the parameter *ie_max_speed* to the additional attribute *me_max_speed*. Activate your enhancement implementation.
4. Activate your enhancement implementation.
5. Test the class again in the Class Builder test environment: Create a truck (in other words, an object of the class CL_DNW7AW_ENH_IMPLIC_TRUCK_##) with a specific maximum speed and call your getter method for the speed.

Task 3:

Create a post method for the method

For the sake of safety (to reduce the risk of accidents), the latest legal requirement stipulates that the truck can only be loaded to 90% of its maximum weight.

1. Create a post method for the GET_CARGO method that reduces the actual return value to 90%.
2. Activate your enhancement implementation.
3. Test the class again in the Class Builder test environment: Create a truck (in other words, an object of the class CL_DNW7AW_ENH_IMPLIC_TRUCK_##) with a specific maximum weight and call your method GET_CARGO.

Solution 10: Implicit Enhancements of Classes

Task 1:

Create an additional attribute

1. Within an enhancement implementation ZDNW7AW_TRUCK##a, create a new type I private instance attribute for the class CL_DNW7AW_ENH_IMPLIC_TRUCK_##. It is to be used to record the maximum speed of a truck. (A necessary additional unit of measure such as km/h or mph has been dispensed with here.)

   Suggested name: *me_max_speed*. Initial value: 100.

   a) In the Class Builder of the Object Navigator, go to display mode for the class CL_DNW7AW_ENH_IMPLIC_TRUCK_##. Choose *Class* → *Enhance*. On the following screen, enter ZDNW7AW_TRUCK##a as the name of the implementation and create a short text for it. Save your entries. Go to the *Attributes* tab page for the class and create the additional attribute.

2. Activate your enhancement implementation.

   a) Press **Ctrl + F3**.

3. In accordance with the attribute, create a functional getter method GET_MAX_SPEED.

   a) Go to the *Methods* tab page and create GET_MAX_SPEED as a public instance method. Create *re_speed* as the type I returning parameter. Double-click on the name of the method to access its source code. Here, define: `re_speed = me_max_speed.`

4. Activate your enhancement implementation.

   a) Press **Ctrl + F3**.

5. Test the class in the Class Builder test environment: Create a truck (in other words, an object of the class CL_DNW7AW_ENH_IMPLIC_TRUCK_##) and call your getter method for the maximum speed.

   a) To test this, press **F8**. Create an instance (choose **F8** again) and call the method.

Continued on next page

Task 2:

Create an additional parameter for the method

1. Enhance the class constructor with an optional parameter *ie_max_speed*, with which you can determine the speed when you create a truck.

   a) Make sure that you are still in enhancement mode. Set the cursor on the CONSTRUCTOR method and go to its parameters. Here, create the type I import parameter you require.

2. Activate your enhancement implementation.

   a) Press **Ctrl + F3**.

3. Use the implicitly available enhancement point at the end of the source code for the constructor to map the parameter *ie_max_speed* to the additional attribute *me_max_speed*. Activate your enhancement implementation.

   a) Double-click on the method name CONSTRUCTOR to access its source code. Choose *Method → Enhance* and then *Edit → Enhancement Operations → Show Implicit Enhancement Options.* Place the cursor in the row before `ENDMETHOD`. Choose *Edit → Enhancement Operations → Create*, then *Source Code*. Since this is not an additional explicit enhancement, you have to create a new enhancement implementation. Enter ZDNW7AW_TRUCK##b. Create a short text. Save your entries.

   b) Define as the source code:

   ```
   me_max_speed = ie_max_speed.
   ```

4. Activate your enhancement implementation.

   a) Press **Ctrl + F3**.

5. Test the class again in the Class Builder test environment: Create a truck (in other words, an object of the class CL_DNW7AW_ENH_IMPLIC_TRUCK_##) with a specific maximum speed and call your getter method for the speed.

   a) See above

Continued on next page

Task 3:

Create a post method for the method

For the sake of safety (to reduce the risk of accidents), the latest legal requirement stipulates that the truck can only be loaded to 90% of its maximum weight.

1. Create a post method for the GET_CARGO method that reduces the actual return value to 90%.

   a) Make sure that you are still in enhancement mode. On the *Methods* tab page, place the cursor on the GET_CARGO method and choose *Edit* → *Enhancement Operations* → *Insert Post Method.* Click on the pushbutton that has just appeared in the *Post Exit* column.

   b) For the source code, define:

   ```
   re_cargo = re_cargo * '0.9'.
   ```

2. Activate your enhancement implementation.

   a) Press **Ctrl + F3**.

3. Test the class again in the Class Builder test environment: Create a truck (in other words, an object of the class CL_DNW7AW_ENH_IMPLIC_TRUCK_##) with a specific maximum weight and call your method GET_CARGO.

   a) See above


205

Exercise 11: Explicit Enhancement Points and Enhancement Sections

Exercise Duration: 15 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

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

Business Example

You want to use explicit enhancement points and enhancement sections to enhance or replace SAP source code without making modifications.

System Data

System:	Will be assigned
Client:	Will be assigned
User ID:	Will be assigned
Password:	Will be assigned
Set up instructions:	No special settings required in the standard training system

Task:

Enhance and replace SAP source code

1. Analyze the source code of the SAP program **SAPDNW7AW_ENH_EXPLIC_00** and execute it. (## stands for your two-digit group number.)
2. Use the explicit enhancement point **DNW7AW_ENHPO1_##** to output the fields *distance* and *distid* too. Choose **ZDN7AW_ENH_EXPLIC##** as the name of your enhancement implementation.
3. Use the explicit enhancement section **DNW7AW_ENHSEC1_##** to output a different text to that provided by SAP.

Solution 11: Explicit Enhancement Points and Enhancement Sections

Task:

Enhance and replace SAP source code

1. Analyze the source code of the SAP program **SAPDNW7AW_ENH_EXPLIC_00** and execute it. (## stands for your two-digit group number.)

   a) To execute it, press **F8**.

2. Use the explicit enhancement point **DNW7AW_ENHPO1_##** to output the fields *distance* and *distid* too. Choose **ZDN7AW_ENH_EXPLIC##** as the name of your enhancement implementation.

   a) To do this, go to enhancement mode: *Program → Enhance*. Place the cursor on the name of the enhancement point, call the context menu for the editor and choose *Enhancement Implementation → Create*.

3. Use the explicit enhancement section **DNW7AW_ENHSEC1_##** to output a different text to that provided by SAP.

   a) To do this, go to enhancement mode: *Program → Enhance*. Place the cursor on the name of the enhancement section, call the context menu for the editor and choose *Enhancement Implementation → Create*.

Exercise 12: New BAdIs

Exercise Duration: 30 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Search for new BAdIs in SAP programs and use them to enhance the SAP program.

Business Example

You want to find BAdIs implemented using the new BAdI technology in SAP programs and use them for enhancing SAP software.

System Data

System:	Will be assigned
Client:	Will be assigned
User ID:	Will be assigned
Password:	Will be assigned
Set up instructions:	No special settings required in the standard training system

Task:

Search for and use new BAdIs

1. In the SAP program **SAPDNW7AW_BADI_##**, search for a BAdI that is used and display the superordinate enhancement spot together with the BAdI definition. (## stands for your two-digit group number.)

2. Create an enhancement implementation (for the spot) with a BAdI implementation (for the BAdI).

   Name of the enhancement implementation: **ZDNW7AW_ESPOT_BADI_##**

   Name of the BAdI implementation: **ZDNW7AW_BADI_##**

Continued on next page

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

Hint: You can either

- copy the existing sample implementation class to your implementation class and adjust the copied method if required, **or**

- create an empty implementation class and implement the method yourself.

Since the BAdI is dependent on a filter, you can create various implementations, for example, one for DE (Germany) and one for GB (Great Britain).

Solution 12: New BAdIs

Task:

Search for and use new BAdIs

1. In the SAP program **SAPDNW7AW_BADI_##**, search for a BAdI that is used and display the superordinate enhancement spot together with the BAdI definition. (## stands for your two-digit group number.)

   a) Search for the `GET BADI` command. Double-click on the variable behind it to get to your definition. Double-click on the variable type (*DNW7AW_BADI_##*). This takes you to the relevant enhancement spot DNW7AW_ESPOT_BADI_##, where the BAdI DNW7AW_BADI_## is displayed.

2. Create an enhancement implementation (for the spot) with a BAdI implementation (for the BAdI).

   Name of the enhancement implementation: **ZDNW7AW_ESPOT_BADI_##**

   Name of the BAdI implementation: **ZDNW7AW_BADI_##**

   a) From the enhancement spot display, press **F6** or the pushbutton for creating an implementation.

   b) On the following screen, enter **ZDNW7AW_ESPOT_BADI_##** as the name of the implementation, create a short text for it and save your entries.

Continued on next page

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

Hint: You can either

- copy the existing sample implementation class to your implementation class and adjust the copied method if required, **or**

- create an empty implementation class and implement the method yourself.

Since the BAdI is dependent on a filter, you can create various implementations, for example, one for DE (Germany) and one for GB (Great Britain).

a) On the screen that appears after you have saved your data, enter **`ZDNW7AW_BADI_##`** as the name of the BAdI implementation and choose Enter.

b) Go to *Create Implementation* (pushbutton).

c) In the enhancement implementation tree, double-click on *Implementing Class*. Then enter **`ZCL_DNW7AW_BADI_IMPLEM_##`** as the implementing class and choose Change (pushbutton). Confirm that you want to create the class. The easiest way is to decide on the following screen to copy the sample class. Save the new class.

d) If you created the class from scratch, you still have to double-click to create the method IF_DNW7AW_BADI~WRITE_ADDITIONAL_COLS. For the source code, define for example:

```
WRITE: is_spfli-distance UNIT is_spfli-distid, is_spfli-distid.
```

e) Activate your class and your enhancement implementation.

f) To create a filter value for an implementation, double-click on *Filter Values* in the enhancement implementation tree. The click on the *Combination* pushbutton () and define the required filter values.

g) Execute the (unchanged) program SAPDNW7AW_BADI_##. Enter various different countries on the selection screen, if necessary, to test whether your filter-dependent implementations work properly.

Lesson Summary

You should now be able to:

- Explain the new enhancement options that exist and how they are managed
- List where implicit enhancement points exist and use these enhancement points for enhancing SAP software
- Find explicit enhancement points and enhancement sections and use these for enhancing or replacing SAP software
- Explain why SAP introduced the new BAdI technology
- Explain the concept and the runtime architecture of the new BAdI technology
- Search for new BAdIs and use these to enhance SAP software
- Explain how the existing enhancement implementation of industry solutions are integrated in the Switch Framework and can be activated

Unit Summary

You should now be able to:

- Explain the new enhancement options that exist and how they are managed
- List where implicit enhancement points exist and use these enhancement points for enhancing SAP software
- Find explicit enhancement points and enhancement sections and use these for enhancing or replacing SAP software
- Explain why SAP introduced the new BAdI technology
- Explain the concept and the runtime architecture of the new BAdI technology
- Search for new BAdIs and use these to enhance SAP software
- Explain how the existing enhancement implementation of industry solutions are integrated in the Switch Framework and can be activated

Unit 7

213

Web Dynpro: Introduction

The first task is to explain why SAP is rolling out another UI technology. In this chapter you could compare this technology with the classical Dynpro screens SAP ITS and BSP. This should result in the following evaluation:

Web Dynpro is the follow up of the classical Dynpro technology. It is not a direct competitor of BSP or SAP ITS.

SAP ITS is a mapping tool that allows you to translate existing Dynpro-based applications to a standard that a browser can interpret. SAP ITS is therefore the only technology that allows you to map existing programs based on Dynpros to HTML and therefore to a browser-based UI.

Web Dynpro is a programming model, which allows you to quickly design standard UIs. Anything that can be designed without source codeis designed without source code. However, the freedom in designing the UI is restricted. For example, it is not possible to include client-side JavaScript in the generated pages. The generated code is not restricted to HTML.

Finally, the BSP programming model allows all browser-based techniques to be inserted in the generated HTML pages. However, constructing a BSP UI takes a lot more time than developing a Web Dynpro UI, as there is no declarative approach. BSP can therefore be used for “free-style” programming.

Unit Overview

Web Dynpro is a programming model provided by SAP. It is implemented in Java and ABAP. It is suited to generating standardized user interfaces (UIs) using a declarative approach, which allows Web applications to be implemented over a minimal period of time.

The advantages of using the Web Dynpro programming model (compared to other established Web programming models) will be explained in this unit. In this context, the basic architecture and key functions of Web Dynpro will be summarized.

Unit Objectives

After completing this unit, you will be able to:

- Describe the declarative programming approach used to create Web Dynpro applications
- Explain the benefits of this metadata approach
- List the most important elements that are part of a Web Dynpro application and that can be defined declaratively

Unit Contents

Lesson: Web Dynpro: Introduction

 214

Lesson Duration: 120 Minutes

Lesson Overview

This lesson contains a brief overview of Web Dynpro functions. The main benefits of using Web Dynpro for designing Web applications are discussed.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the declarative programming approach used to create Web Dynpro applications
- Explain the benefits of this metadata approach
- List the most important elements that are part of a Web Dynpro application and that can be defined declaratively

---

It is beneficial if the trainer knows other UI programming models, to be able to list the benefits (and the restrictions) of ABAP Web Dynpro. In the ABAP area, there are two other programming models, SAP ITS and BSPs, which are still supported. The trainer should be prepared for questions related to the pros and cons of these three technologies. Here are some tips:

SAP ITS will be used in to future to map existing Dynpro-based transactions to HTML pages. This will be necessary, since it is impossible to convert all Dynpro-based transactions to Web Dynpro in an acceptable time frame. SAP ITS is the only Web-enabling technology. For all other technologies, the application has to be completely rewritten. The disadvantages are: Long response times, since an additional timeframe is necessary for mapping between HTML pages and Dynpros - in both directions.

Web Dynpro is the technology SAP uses for developing all future applications. Existing Dynpro-based transactions will be converted. Web Dynpro will replace the old Dynpro technology step by step. The disadvantages are: Although Web Dynpro does allow standardized interfaces to be developed quickly, it is not the right choice for the development of ornate Internet-like UIs. This is because the UI is rendered from metadata, and developers cannot place their own JavaScript code in the rendered HTML page. Web Dynpro has to be embedded in the portal environment, to be able to change the application design.

Finally BSP is the ABAP technology for “free-style” programming. Everything is possible. BSP can be compared to JSP on the Java side. The disadvantages are: Standardized UIs are not supported and everything has to be coded

(for example, navigation, and UI). It therefore takes a lot longer to develop BSP-based UIs than Web Dynpros. As SAP develops standard applications that require a standardized and consistent UI, BSP is not used by SAP. However, BSP is the only of these three technologies that allows sophisticate Internet applications to be developed using ABAP. This technology will therefore also be very interesting for the customer in the future.

Setting Up the System

It is easy to prepare the system for this course: For each user group, you have to define an SU01 user. For the sake of simplicity, assign this user SAP_ALL authorization (copy of your user). For all users, create a change request and assign each user a task. Check if the transaction ZUSR is available in the system, to perform these steps.

Business Example

You want to find the technology that is best suited for developing ABAP-based Web applications. The conditions for your projects are high speed, low cost, and standardized UIs. You have established that ABAP Web Dynpro is a good fit for your needs. You therefore want to obtain an initial overview of Web Dynpro functions.

What is Web Dynpro?

From a technological point of view, SAP Web Dynpro for Java and ABAP is a revolutionary step in the development of Web-based user interfaces. It is completely unlike any design paradigm previously used by SAP and represents a quantum leap in the development of Web-based enterprise resource planning (ERP) applications.

Web Dynpro applications are built using declarative programming techniques based on the Model View Controller (MVC) paradigm. That is, you specify which user interface elements you wish to have on the client, and where those elements will retrieve their data. You also define the possible navigation paths declaratively in your application. All the source code to create the user interface is then generated automatically in a standard runtime framework. This relieves you of the repetitive coding tasks involved in writing HTML and making it interactive with JavaScript.

ABAP Web Dynpro has been available since SAP NetWeaver 2004s (SAP NetWeaver Application Server 7.0). For developing the entities of a Web Dynpro application, the Object Navigator (transaction code SE80) has been enhanced.

Web Dynpro is designed to support structured development. The software modularization units are Web Dynpro components, which can be combined to build up complex applications.

Meta-Model Declarations vs. Custom Coding

A Web Dynpro application is developed using a declarative programming approach. Within the ABAP Workbench there are special tools that allow you to build an abstract representation of your application in the form of a Web Dynpro meta model. The necessary source code is then generated automatically and conforms to a standard architecture known as the Web Dynpro framework.

The Web Dynpro framework allows you to place the custom source code at predefined positions within the generated code.

All Web Dynpro applications are constructed from the same basic units. However, through the use of custom coding, the standard framework can be extended to supply any required business functions.

Not all implementation decisions need to be made at design time. It is possible to implement a Web Dynpro application in which the appearance of the user interface is decided at runtime. This allows a highly flexible application to be written without the need to directly write any HTML or JavaScript.

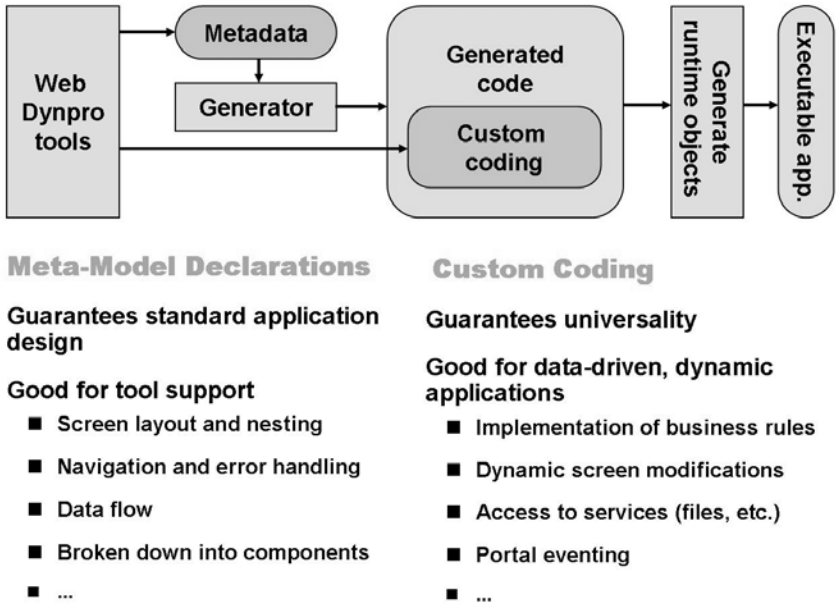


Figure 134: Meta-Model Declarations vs. Custom Coding

Application Scenarios with Web Dynpro

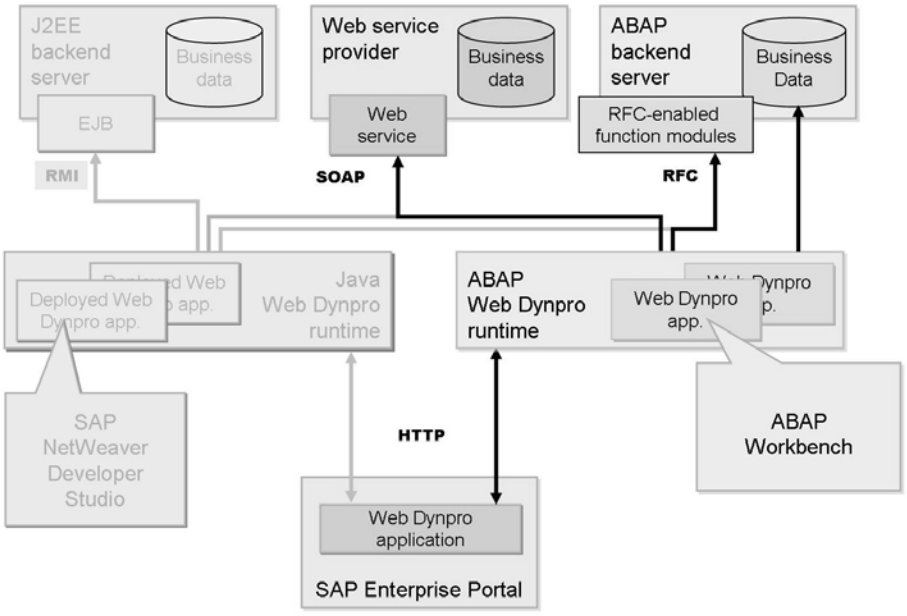


Figure 135: Application Scenarios with Web Dynpro

Web Dynpro applications can access different kinds of data sources:

- From an ABAP Web Dynpro application, all kinds of reusable components can be addressed directly (for example, function modules or methods). It is even possible to access the database via an ABAP SELECT. However, this leads to a mixing between flow logic and business logic and should therefore be omitted.
- Web services can be used after a Web Service client object has been generated.
- The SAP Java Connector (JCo) can be used to call methods of Enterprise JavaBeans located on a J2EE Engine.

Model objects are not yet supported in ABAP Web Dynpro. The best way to have reusable entities encapsulating business logic is to create ABAP classes containing the source code. It is also possible to develop UI-free (faceless) Web Dynpro components, which only offer reusable functions. These components can then be accessed by other Web Dynpro components by means of component reuse.

Advantages of Web Dynpro

Web Dynpro's main goal is to enable application developers to create powerful Web applications with minimum effort using descriptive tools in a structured development process.

One guiding principle in the Web Dynpro philosophy is: The fewer lines of hand-written code, the better. Web Dynpro pursues this goal in two ways.

- Web Dynpro uses a declarative, language-neutral meta model for defining a user interface. From this abstract definition, the development environment generates the required source code. Hand-written code still has its place, but is confined to that required to manipulate the business data, not the user interface.
- Web Dynpro provides technical features such as support for internationalization, flicker-free interaction, and a clean separation of the business logic and the user interface. This separation is achieved through a modified implementation of the Model View Controller (MVC) design paradigm.

As the repetitive tasks of UI coding have been eliminated, the developer can focus his attention on the flow of business data through the application.

Web Dynpro applications can run on a range of devices and on various types of networks – an important feature for collaboration scenarios.

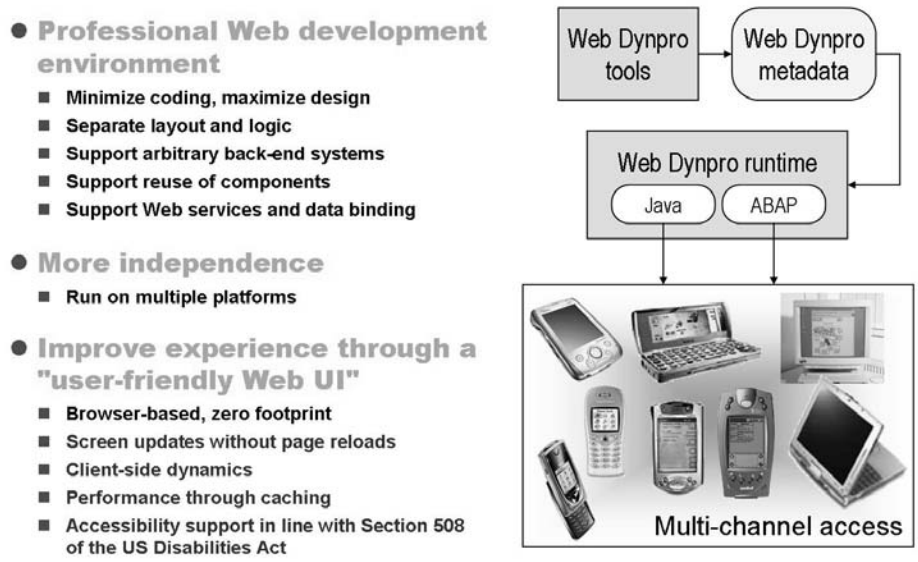


Figure 136: Advantages of Web Dynpro

Web Dynpro Architecture: Entities and Concepts

Main Parts of a Web Dynpro Component

- Windows
- Views
  - UI elements
  - Layout
  - Controller
  - Context
  - ....
- Component controllers
  - Context
  - ....

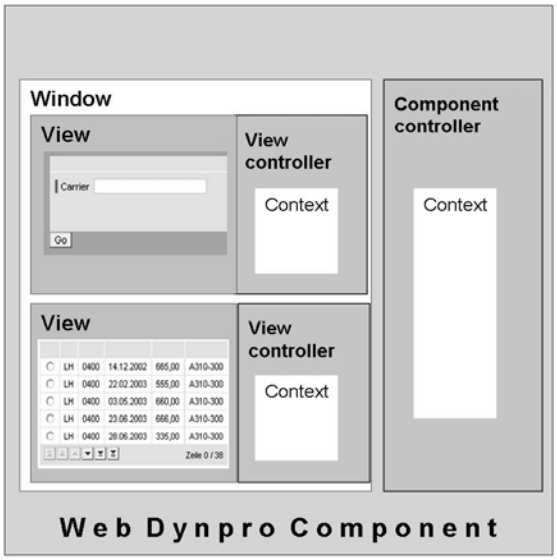


Figure 137: Web Dynpro Component

Web Dynpro components allow you to structure complex Web applications and develop reusable, interacting entities. This enables the nesting of large application sections.

Web Dynpro components are containers for other entities related to the UI and the Web Dynpro program.

Entities related to the UI are **windows** and **views**. The layout of a view represents a rectangular part of a page displayed by the client (for example, a browser). The view contains **UI elements** such as input fields and buttons. The complete page sent to the client can be set up by only one view, but can also be a combination of multiple views. The possible combinations of views and flow between the views is defined in a window. A window can contain an arbitrary number of views. A view can be embedded in an arbitrary number of windows.

The Web Dynpro source code is located in **Web Dynpro controllers**. The hierarchical storage for the global variables of controllers is called the **context**.

Here, you can log on to the system, start transaction SE80 and display package NET310. Open any Web Dynpro component (for example, NET310_UI_S2) and display the entities in the object tree. Do not go further into detail here.

Web Dynpro components can be addressed in three different ways:

- Using a Web Dynpro application, a Web Dynpro component can be related to a URL, which can be called from a Web browser or another Web Dynpro client.
- When reusing a Web Dynpro component as a sub-component, the visual interface of a Web Dynpro component can be combined with the visual entities of the main component to form the UI.
- When reusing a Web Dynpro component as a sub-component, all methods and data defined in the programming interface can be accessed by the main component.

Context Mapping and Data Binding

Context

- Like a data container that holds the data
- The data transport between the controllers can be implemented with mapping definition

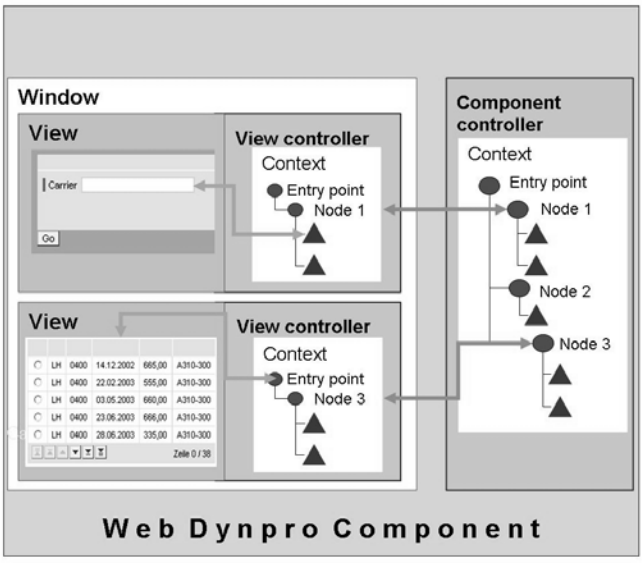


Figure 138: Context and Data Transport

The variables defined in a Web Dynpro controller context can be referenced from other Web Dynpro controllers. This is called **context mapping**. This allows common attributes to be shared between different controllers, so copying these attributes between the controller contexts is not necessary.

The value of UI elements that allow user input have to be connected to context attributes of the corresponding controller. This is called **data binding**. Through data binding, automatic data transport is established between the UI elements and the context attributes.

Combining these two concepts, the data transport between UI elements located in different views can be defined in a purely declarative way.

In the Web Dynpro component you have displayed before (for example, NET310_UI_S2), show the context of views and component controller. Show context mapping. Open views. Show UI elements that are bound to context attributes.

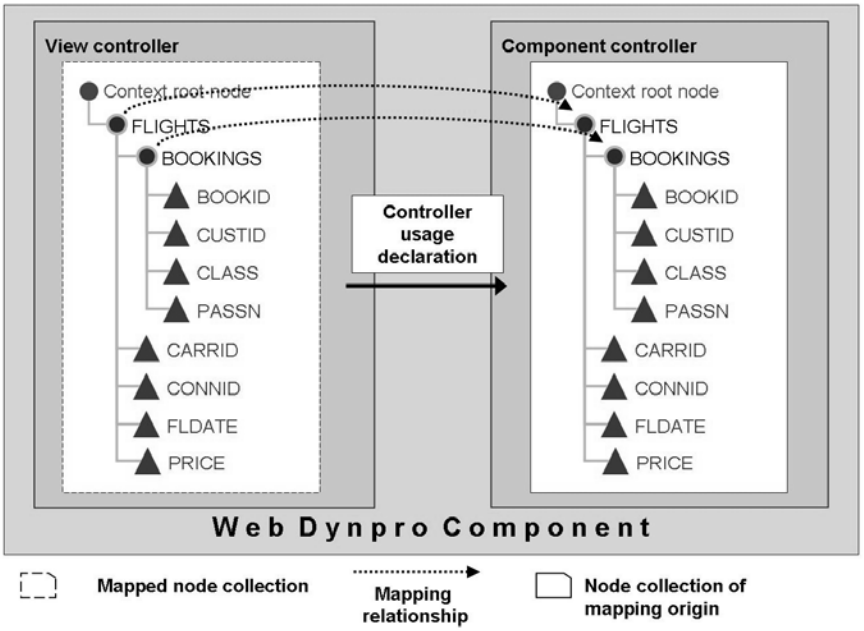


Figure 139: Context Mapping

Now you can go to the property tab of a view and show the entry for controller usage. The component controller usage is generated automatically when creating a new view.

Context mapping allows a context node in one controller to be supplied automatically with data from a corresponding context node in another controller. This is the primary mechanism for sharing data between controllers.

When two controllers within the same component share data through a mapping relationship, it is known as **internal mapping**.

The context node that acts as the data source is known as the **mapping origin node**, and the context node that is mapped is known as the **mapped node**.

The mapping between controller contexts located in different Web Dynpro components is known as **external mapping**.

Note: External mapping will not be covered in this lesson.

For a mapping relationship to be established, the following steps must first be performed:

- A node must exist in the context of the controller acting as the mapping origin. This node need not have any declared child nodes or attributes.
- The mapping origin controller must not be a view controller.
- The controller containing the mapped node must declare the use of the mapping origin controller as a used controller.

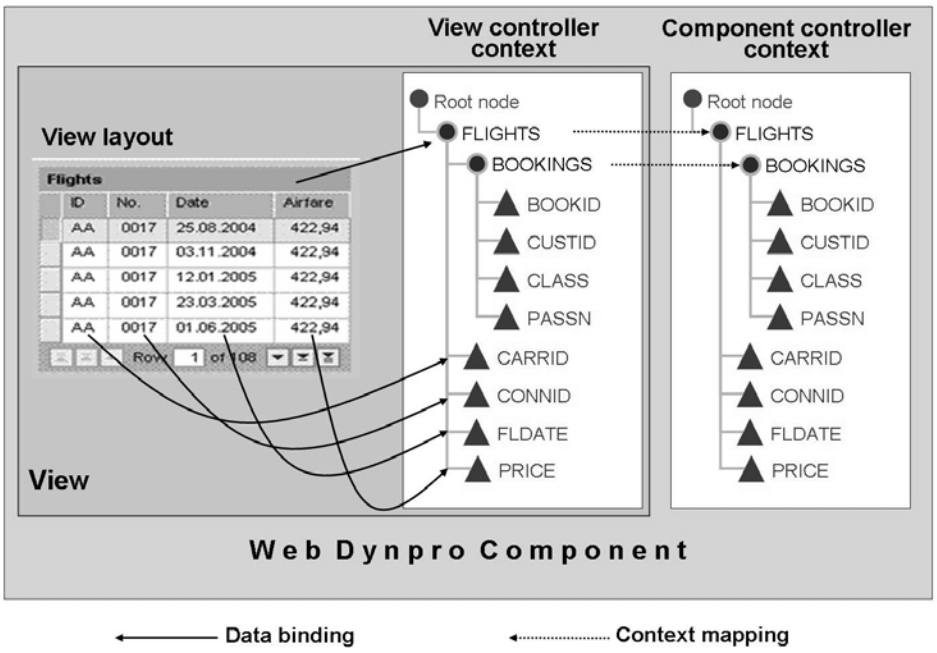


Figure 140: Putting Data on the Screen: Data Binding

Data binding is the means by which data is automatically transported from a view controller's context to a UI element in its layout, and visa versa.

You may not bind UI elements to context nodes or attributes defined in some other controller. UI elements are private to the view controller within which they are declared.

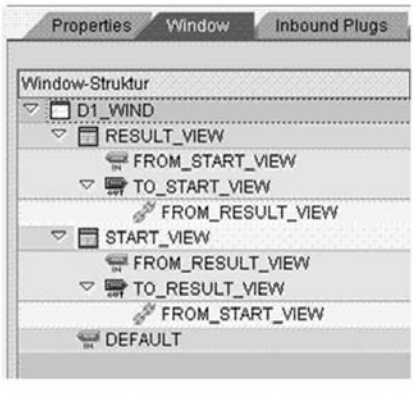
The data binding process decouples the UI element object from the application code within the view controller. Therefore, in order to manipulate UI element properties, the application code in the view controller needs only to manipulate the values of context nodes and attributes to which the UI elements are bound. The Web Dynpro framework then manages the following two tasks:

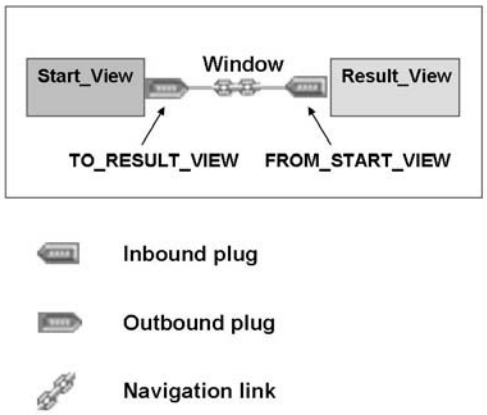
Open component NET310_UI_S2, to show the data binding of a TABLE UI element to a context node. Or open a view of component NET310_COND_S, to show the data binding between an input field and a context attribute.

- The transport of data from the context attribute to the UI element during the screen rendering process.
- Repopulating the context attribute from the UI element after data has been entered by the user and the next server round trip is initiated. Hereby, the values entered by the user are automatically converted and checked for type conformity. If an error occurs, a message is displayed.

Data binding is powerful, since not only the value of a UI element can be bound to a context attribute, but other UI properties such as visibility can also be bound. This way, UI element properties can be manipulated from the view controller by acting on context attributes.

Navigation Between Views





- **If you want to define the navigation between two views, you must create exit points (outbound plugs) and entry points (inbound plugs) for each view.**
- **You can then use navigation links to determine the navigation flow.**

Figure 141: Navigation Between Views

Navigation between views is triggered by firing **outbound plugs**. Firing an outbound plug causes a navigation event to be raised. Navigation events are special asynchronous events that are placed in a navigation queue. Multiple outbound plugs can be fired from one view. This can be used to define the next UI, which consists of multiple views. The navigation queue is processed at a certain point of time in the Web Dynpro processing phase. Up to this point of time, the navigation stack can be extended by firing additional outbound plugs, or the complete navigation stack can be deleted. Outbound plugs should be named according to the action that caused navigation away from the current view.

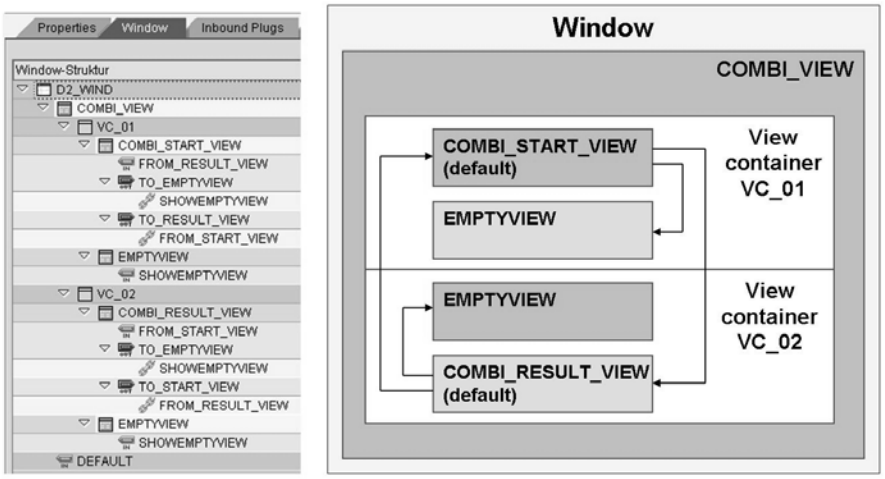
Inbound plugs are special event handler methods that subscribe to navigation events raised when outbound plugs are fired. Inbound plug methods are called only when the navigation queue is processed. This will only take place once the views in the current view assembly have fired their outbound plugs and no validation errors have occurred that would cause navigation to be cancelled. Inbound plugs should be named according to the reason why the view is being displayed.

Open any view of component NET310_UI_S2 and show plugs.

Outbound and inbound plugs are joined together using **navigation links**. Technically, linking an inbound plug to an outbound plug means registering the inbound plug event handler method to the navigation event, called by firing an outbound plug.

Outbound plugs and event handler methods related to the inbound plug can have parameters, allowing you to pass data between the views.

Windows and Nested Views



- A window is the set of all possible views that can make up a visible screen.
- A window can have zero or more views embedded within it.
- A view can contain UI elements of the type ViewContainerUIelement. This allows nesting views within a window.
- A ViewContainerUIelement can only display one view at a time.

Figure 142: Windows and Nested Views

A **window** defines which views are displayed in which combination and how the view combination may be changed by firing outbound plugs. Therefore, when creating a window, you define three things:

Open window D1_WIND of component NET310_NAVI_D1 and show how plugs are linked via navigation links. Open window D2_WIND of component NET310_NAVI_D1 and show how a view can be embedded in a view container. Explain view assembly.

- All the possible views that could exist in the component's visual interface must be embedded in the window.
- If multiple views must be displayed in parallel, the layout and position of these views is defined by a special view containing ViewContainerUIelements in its layout. This container view is embedded in the window and, inside each area defined by a ViewContainerUIelement, all possible views for this **view area** are embedded (nested embedding). For each ViewContainerUIelement, there is one default view displayed at startup.
- The navigation links between the different views must be defined.

Only one view can be displayed in the view area at a time. Navigation links must be defined between views in order for the contents of a view area to be replaced.

View areas can be blanked out by creating an empty view whose inbound plug responds to an appropriate navigation event.

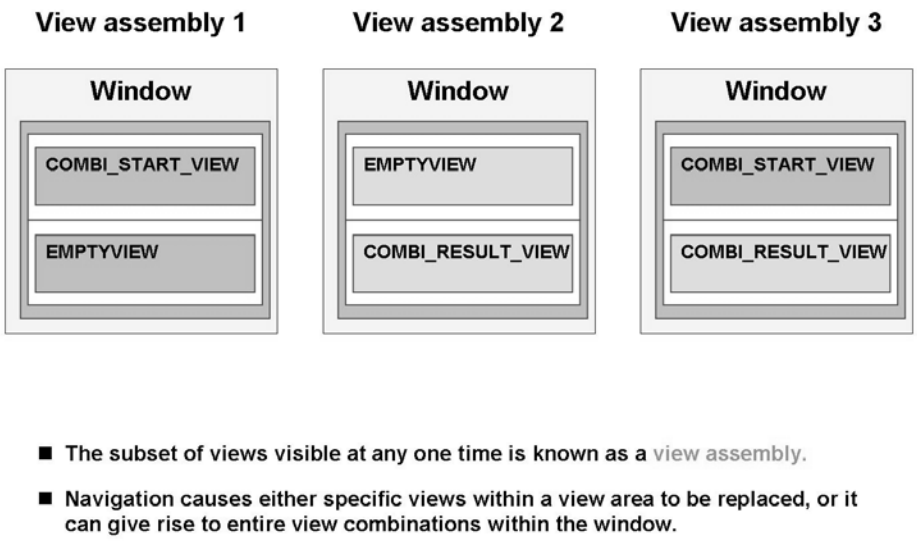


Figure 143: View Assembly

Outbound plugs are the triggers that cause a view area to contain a particular view.

For a given window definition in which multiple views are embedded into view areas, many permutations of views could be visible. The permutation that is visible depends on which navigation links the user follows.

The subset of views visible at any time is known as a **view assembly**.

At the end of this section, create your first WD component. Start by creating a package. Save this package in the transport request created prior to the course. Create a WD component with a window in your package. Create a view and embed it into the window. Create an application to access the component. Finally, add a TextView UI element to your view and enter text. Activate the component with all entities and start the application.

Exercise “Web Dynpro: Introduction”

Web Dynpro Architecture: Relationship Between Web Dynpro Entities

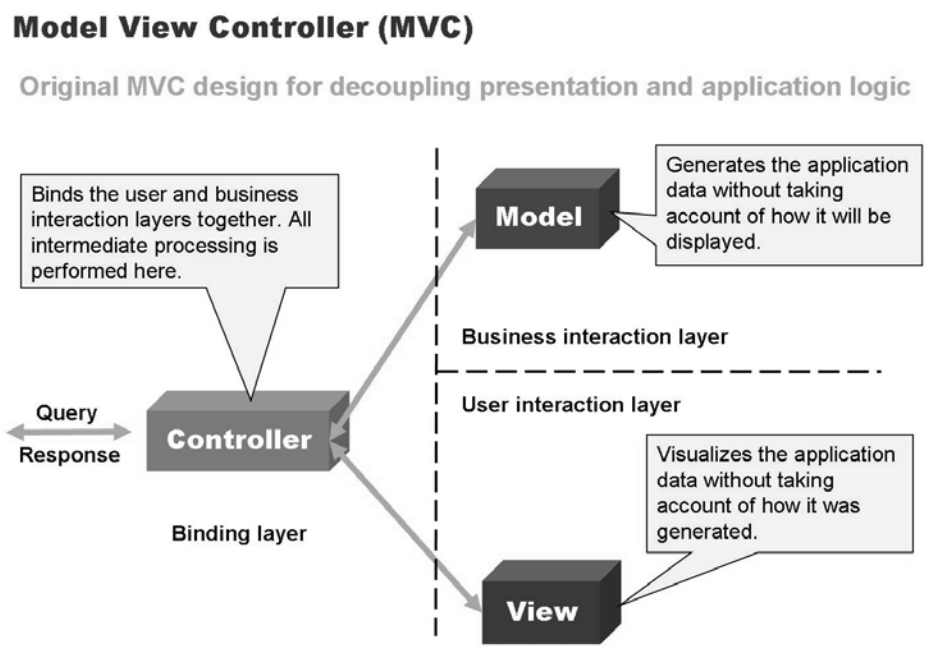


Figure 144: Model View Controller (MVC)

SAP's Web Dynpro is built on the foundation of the **Model View Controller** (MVC) design paradigm. MVC was originally invented by the Norwegian software designer Trygve Reenskaug while working at Xerox PARC in the late 1970s. The first implementation of this design paradigm was with the release of the Smalltalk-80 programming language.

MVC was a revolutionary design paradigm because it was the first to describe software components in terms of:

- The functional responsibilities each should fulfill
- The message protocols to which each component should respond

SAP has modified and extended the original MVC specification to create the Web Dynpro toolset.

Web Dynpro Component Architecture

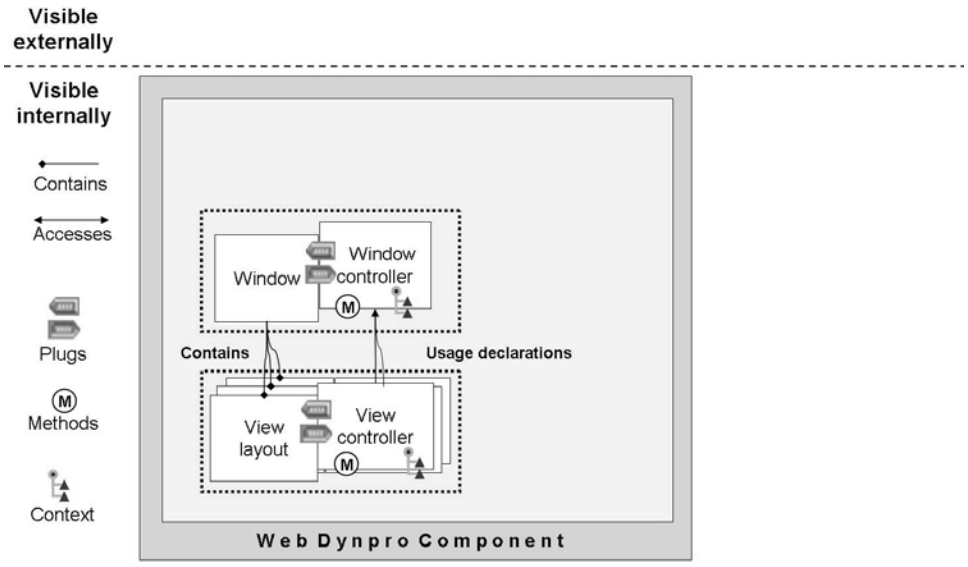


Figure 145: Internally Visible Web Dynpro Entities (1)

One view can be embedded into different windows. The first view that is embedded into a window will be the default view.

In your WD component: Create a second view and embed it into the window.

The architecture of a Web Dynpro component can be divided into two parts: **external and internal visibility**. The horizontal dashed line in the figure above separates the entities that are visible from outside the component from those that are only visible from within the component.

The internally visible parts can further be divided into visual entities and programming entities. Visual entities are those related to the UI, generated by the Web Dynpro framework, and passed to the client. The internally visible entities consist of windows and views.

A view consists of a view layout and the corresponding view controller. The view controller can contain navigation plugs, methods, and a context.

A window embeds one or more views and has a corresponding window controller. A window controller can contain navigation plugs, methods, and a context. Each view can be embedded in multiple windows.

The outbound plug of a window can be connected to any inbound plug of embedded views, and the outbound plug of a view can be connected to any inbound plug of the embedding window. However, navigation between windows of the same component is not possible.

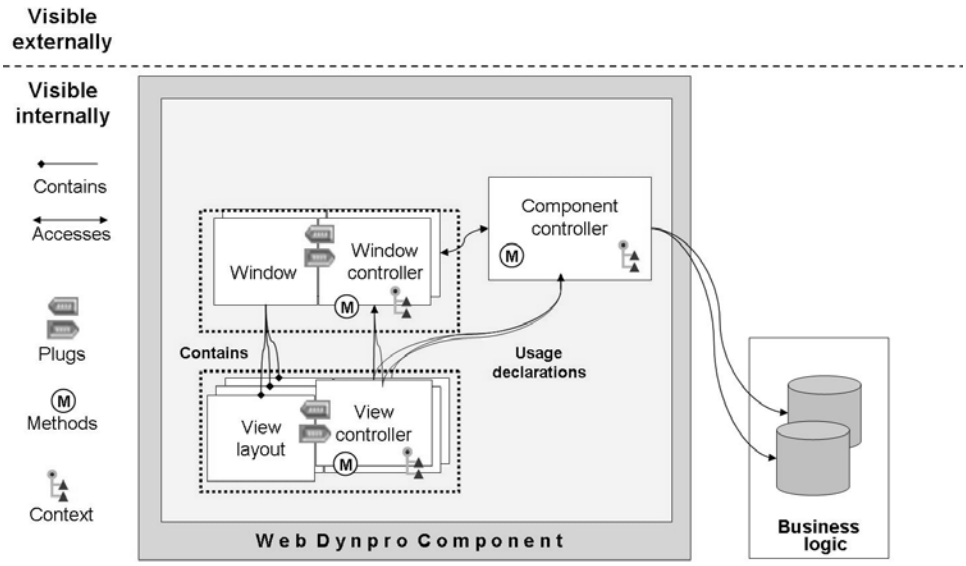


Figure 146: Internally Visible Web Dynpro Entities (2)

Multiple windows may exist (for example, NET310_NAVI_D1). Sometimes it is useful to access business logic from a view (input check: does a data record exist that fits the user's input?).

In your component: Create a second window. Embed both views that have been embedded before into the first window.

The component controller acts as a component-wide controller. Program logic that is related to only a certain view (for example, the checking of user input) should be coded in the relevant view controller. Usage declarations between the controllers allow you to access the context data and methods of the declared controller (**used controller**). A view controller can not be declared as a used controller for other controllers, as this would violate the principles of good programming (MVC programming paradigm).

Business logic should not be part of the Web Dynpro component, but should be defined outside of the component, to ensure a high degree of reusability. It is preferable that ABAP classes be used to encapsulate the related source code.

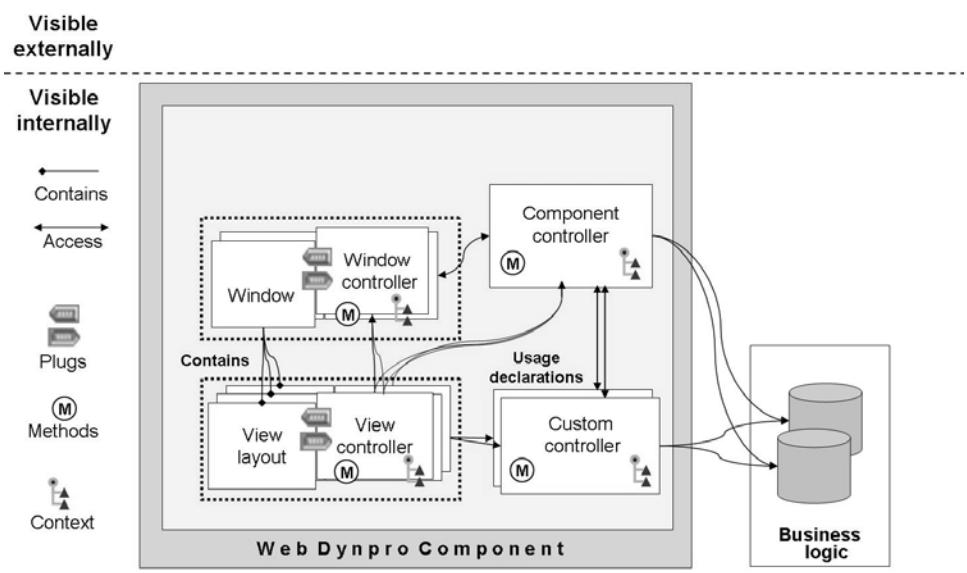


Figure 147: Internally Visible Web Dynpro Entities (3)

In your component: Create a custom controller. In the custom controller, create a usage declaration for the component controller and vice versa.

Custom controllers are optional controllers that must be defined by the developer. These controllers can be used to modularize the component content. For example, custom controllers may serve as local controllers for some views, or they can be used to encapsulate logic that belongs to a specific model class (business logic). This allows you to reduce the content of the component controller by populating sub-functions.

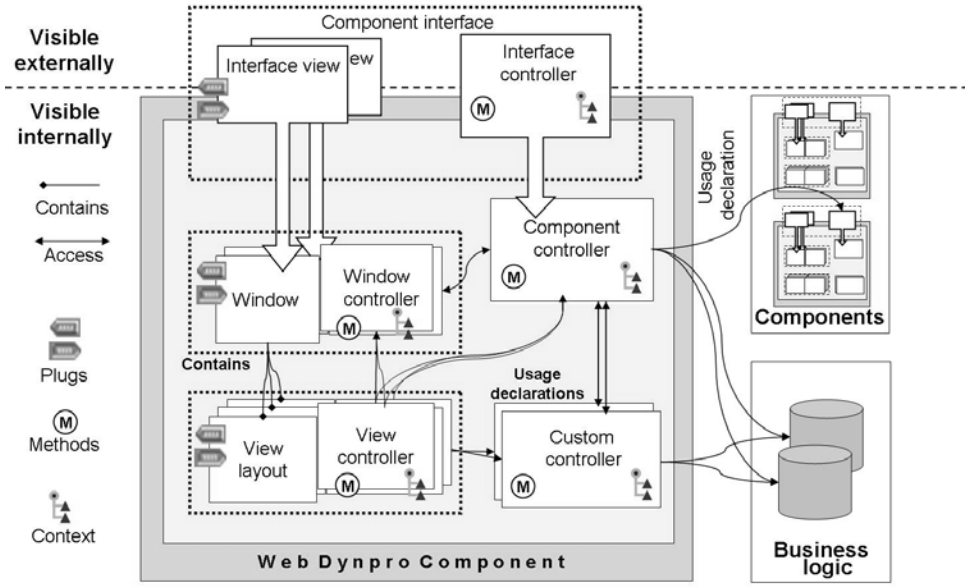


Figure 148: Externally Visible Web Dynpro Entities

If one Web Dynpro component (**parent component**) needs access to another Web Dynpro component (**child component**), the parent component can declare the use of the child component. A specific component usage instance is then created and the parent component accesses the functions of the child component using its component interface controller.

The only parts of a Web Dynpro component that are visible to the user, are the **interface controller** and the **interface view(s)**.

In your component: Display the component interface. An interface view has been created for each window. An interface controller has also been created automatically.

- All Web Dynpro components have only one interface controller. Via the interface controller, data, methods, and event handlers can be exposed to other components.
- Interface views represent the visual interface of a Web Dynpro component. There is a one-to-one relationship between a Window and an interface view. Each time a window is defined, a related interface view is automatically generated, which makes the window accessible from outside the component. The interface view only exposes to the component user the inbound and outbound plugs for which the *interface* property is activated. Methods and context data of the window are not accessible from the related interface view.
- If the component has no views, there is no need to have Windows. In this case, the component will not implement an interface view. Components that have no visual interface are known as **faceless components**.

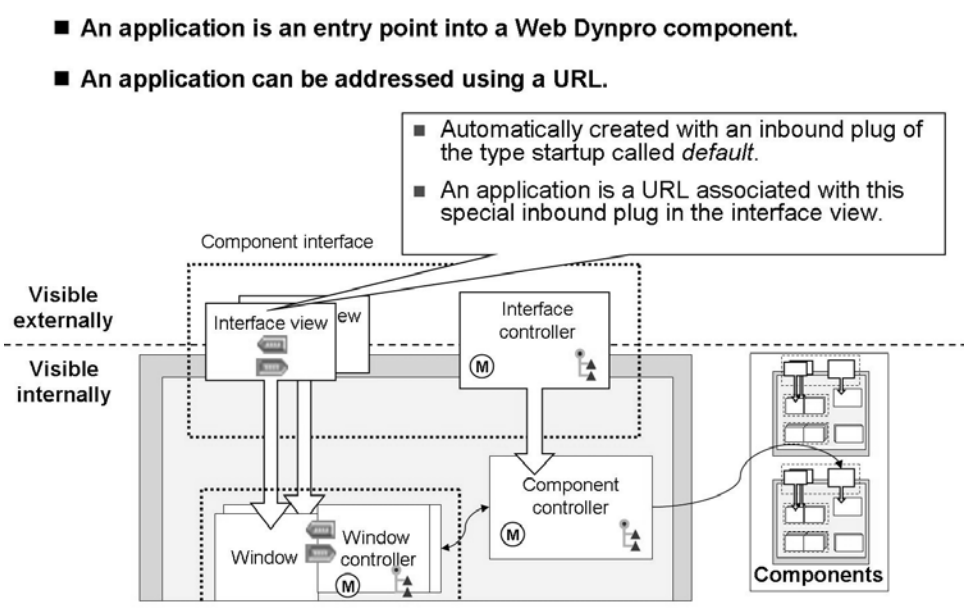


Figure 149: Web Dynpro Application

A **Web Dynpro application** is an entry point into a Web Dynpro component and is the only Web Dynpro entity that can be addressed using a URL.

In your component: Display the application characteristics. An application is pointing to a startup plug of an interface view located in a certain WD component.

There is often (but not always) a one-to-one relationship between an interface view and an application.

In the same way that the functionality in an ABAP module pool can be accessed by defining multiple transaction codes, the functions of a single Web Dynpro component can be accessed by defining multiple applications, each addressing a different interface view and/or a different inbound plug of the interface view.

In order to define a Web Dynpro application, you must specify:

- The component to be called. This component is then known as the **root component**
- The interface view of the root component to be used. The default view(s) in this interface view define(s) the default view assembly.
- Which inbound plug will act as the entry point for the nominated interface view (this inbound plug must be of the type *Startup*)

231

Exercise 13: Web Dynpro: Introduction

Exercise Duration: 20 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Create a Web Dynpro component with a Web Dynpro window containing a single Web Dynpro view
- Place a simple UI element on the view layout and edit its properties
- Create a Web Dynpro application to make the view available for users

Business Example

You want to develop a Web Dynpro application. You will start by creating the Web Dynpro entities that are needed for an application that shows a view with a simple text field.

Template: None

Solution: NET310_INTR_S

Task 1:

Create a package that will contain all the repository objects you are going to develop.

1. Create package **ZNET310_##**. Assign application component **BC-WD** and software component **HOME**.

Task 2:

Create a Web Dynpro component with a Web Dynpro window.

1. Create the Web Dynpro component **ZNET310_INTR_##** with a main window **MAIN_WINDOW**.

Continued on next page

2. What other Web Dynpro entities have been created automatically in this step?

Task 3:

Create a Web Dynpro view with a simple TEXTVIEW element and make it part of the main window.

1. Create a Web Dynpro view (suggested name: **MAIN_VIEW**) in your component.
2. Create a simple UI element of the type TEXTVIEW (suggested name: **TEXT_VIEW_1**) in the view layout and edit its properties.
3. Maintain the text to be displayed in the TEXTVIEW element. You may want to change some other properties of the element and see the result in the layout preview.
4. Make the Web Dynpro view part of the Web Dynpro window.

Task 4:

Create a Web Dynpro application to access your Web Dynpro component and test it.

1. Create a Web Dynpro application (suggested name: **ZNET310_INTR_##**) that accesses the main window of your Web Dynpro component.
2. Activate the Web Dynpro component and test the Web Dynpro application.

Solution 13: Web Dynpro: Introduction

Task 1:

Create a package that will contain all the repository objects you are going to develop.

1. Create package **ZNET310_##**. Assign application component **BC-WD** and software component **HOME**.

   a) Perform this step like you (hopefully) have done many time before.

Task 2:

Create a Web Dynpro component with a Web Dynpro window.

1. Create the Web Dynpro component **ZNET310_INTR_##** with a main window **MAIN_WINDOW**.

   a) In the navigation area, open the context menu for the package and choose *Create → WebDynpro → WebDynpro Component (Interface)*.

   b) In the dialog box, enter the name of the component, a description, and the name of the main window.

2. What other Web Dynpro entities have been created automatically in this step?

   Answer:

   A component controller

   An interface controller

   An interface view for the window with its ABAP objects interface ZIWCI_NET310_INTR_##

Task 3:

Create a Web Dynpro view with a simple TEXTVIEW element and make it part of the main window.

1. Create a Web Dynpro view (suggested name: **MAIN_VIEW**) in your component.

   a) In the context menu for the Web Dynpro component choose *Create → View*.

   b) Enter the name of the view and a short description.

Continued on next page

2. Create a simple UI element of the type TEXTVIEW (suggested name: **TEXT_VIEW_1**) in the view layout and edit its properties.

   a) Edit your Web Dynpro view MAIN_VIEW and open the *Layout* tab.

   b) In the display of the UI element hierarchy (on the upper right corner), open the context menu for ROOTUIELEMENTCONTAINER and choose *Insert Element*.

   c) Enter the name of the element and the element type: **`TEXTVIEW`**.

3. Maintain the text to be displayed in the TEXTVIEW element. You may want to change some other properties of the element and see the result in the layout preview.

   a) Double-click the TEXTVIEW element in the UI element hierarchy.

   b) In the Properties list (below the UI element hierarchy) maintain the text in the *text* field.

4. Make the Web Dynpro view part of the Web Dynpro window.

   a) Edit the Web Dynpro window and open the *Window* tab.

   b) In the window structure, open the context menu for the Web Dynpro window and choose *Embed View*.

   c) In the dialog box, enter the name of the view.

Task 4:

Create a Web Dynpro application to access your Web Dynpro component and test it.

1. Create a Web Dynpro application (suggested name: **ZNET310_INTR_##**) that accesses the main window of your Web Dynpro component.

   a) In the navigation area, open the context menu for your Web Dynpro component and choose *Create → Web Dynpro Application*.

   b) In the dialog box, check the name of the application (same name as the component) and enter a description.

2. Activate the Web Dynpro component and test the Web Dynpro application.

   a) Open the Web Dynpro **Component** and activate it.

   b) Open the Web Dynpro **Application** and test it.

Lesson Summary

You should now be able to:

- Describe the declarative programming approach used to create Web Dynpro applications
- Explain the benefits of this metadata approach
- List the most important elements that are part of a Web Dynpro application and that can be defined declaratively

Unit Summary

You should now be able to:

- Describe the declarative programming approach used to create Web Dynpro applications
- Explain the benefits of this metadata approach
- List the most important elements that are part of a Web Dynpro application and that can be defined declaratively

Unit 8

237

Web Dynpro Controllers

Not every detail of Web Dynpro controllers is covered in this chapter. Some topics such as the special hook methods that only exist for view controllers or component controllers will be discussed in the relevant chapter (for example, Controller and Context Programming, Dynamic Modifications at Runtime). So focus on giving an overview of the entities of a controller and describe what these entities are used for.

Unit Overview

Controllers contain the source code of a Web Dynpro application; they hold the context data and care are responsible for event handling and navigation. You can only program flow logic if you know how controllers are defined and what kind of entities they offer. This chapter provides a brief introduction to ABAP Web Dynpro controllers, focusing on the differences between the controller types that can exist in a Web Dynpro component.

Unit Objectives

After completing this unit, you will be able to:

- Name the contents of a Web Dynpro controller
- Define the function of each controller entity
- List the differences between the component controller, custom controllers, view controllers, and window controllers

Unit Contents

Lesson: Web Dynpro Controllers

Lesson Duration: 40 Minutes

Lesson Overview

Controllers are the heart of Web Dynpro components, since program logic and data are defined in these entities. This lesson gives an overview of the entities that make up a controller.

Lesson Objectives

After completing this lesson, you will be able to:

- Name the contents of a Web Dynpro controller
- Define the function of each controller entity
- List the differences between the component controller, custom controllers, view controllers, and window controllers

Do not go into detail about the order of hook methods, special hook methods, and so on, because these details will be explained in the relevant chapters of course NET310.

For the demos, it is assumed that you have created a WD component before, with a view embedded in one window, a custom controller and a WD application.

Business Example

You know about windows, views, and controllers that make up a Web Dynpro program. However, to be able to implement the flow logic, which comprises event handling, inbound plug method implementation reaction to client side events, and so on, you must know about the inner structure of controllers.

Controller Types

There are four types of controllers in an ABAP Web Dynpro component. These different controller types comprise different entities:

- Component controller

  For each Web Dynpro component there is only one component controller. This is a global controller, visible to all other controllers. The component controller drives the functions of the entire component. This controller has no visual interface.

- Custom controllers

  Custom controllers are optional. They have to be defined at design time and can be used to encapsulate sub-functions of the component controller. Multiple custom controllers can be defined in a component. Custom controllers are automatically instantiated by the Web Dynpro framework, without the sequence of instantiation being defined. The source code in a custom controller should therefore not depend on the existence of other custom controllers.

- Configuration controller

  This is a special custom controller. It is only necessary if the corresponding component implements special configuration and personalization functionality. Only one configuration controller may exist in any component. Any other controller can access the configuration controller, but the configuration controller cannot access any other controller.

- View controllers

  Each view consists of the layout part and exactly one view controller. This controller cares for view-specific flow logic, for example, checking user input and handling user actions.

- Window controllers

  Each window has exactly one window controller. This controller can be used to care for the data passed via the inbound plugs when being reused as a child controller. Methods of this controller can be called from the window"s inbound plug methods.

At runtime, all controller instances are singletons in respect to their parent component. This is also true for view controllers. Therefore, it is not possible to embed a view in a view assembly more than once.

The global data of a controller is stored in a hierarchical data storage, the **controller context**. This context and the methods defined in a controller are private unless another controller explicitly declares the usage of this controller. However, a view controller cannot be declared as a used controller, so the context data and the methods of a view controller are always private.

Web Dynpro is a stateful implemented technology, meaning that the lifetime of controller instances is not limited to the time used to process the program code and to process the UI. Depending on the controller type, the controller instance lifetime is as follows:

Component controller

The lifetime of the component controller equals the lifetime of the component. When starting a Web Dynpro application, the component controller is instantiated by the Web Dynpro runtime.

Custom controllers

The instantiation of a custom controller is delayed until the first method of the controller is called. Custom controller instances can not be deleted explicitly.

Configuration controllers

This controller is instantiated as the first controller of the component. It lives as long as the component lives.

View controllers

The instantiation of a view controller is delayed until the first method of the controller is called. The lifetime of a view controller can be controlled by the view properties:

If *framework controlled* is selected, the view instance will be deleted with the component.

If *when visible* is selected, the view instance is deleted as soon as the view no longer belongs to the view assembly.

Window controllers

The instantiation of a window controller is delayed until the first method of this controller is called. This is done by starting a Web Dynpro application or by embedding the related interface view in the parent component's window. Window controller instances cannot be deleted explicitly.

Common Controller Entities

Each controller has its own **context**. The context root node already exists. All other nodes and attributes have to be defined statically or by source code.

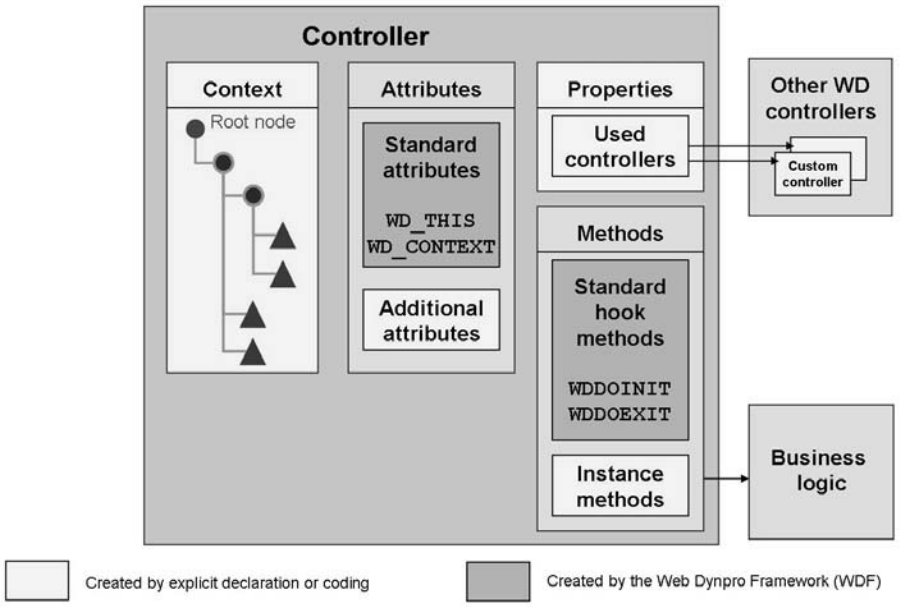


Figure 150: Constituents for All Controllers

Display the component controller of the component NET310_EVEN_D1. Click the single tabs to display the common controller entities. Open a view controller and the window controller of this component, to demonstrate which entities are contained in all of these controllers.

Add an attribute and a method to any of the controllers located in your demo component. Show how to add a controller usage declaration to one of the controllers.

For all controllers, there are methods that are called by the Web Dynpro framework in a predefined order. These are called **hook methods**. Depending on the controller type, there are different hook methods available. At least two hook methods are contained in all controller types. These methods are processed only once during the lifetime of a controller instance: When a controller instance is created (*wddoinit( )*) and when a controller instance is deleted (*wddoexit( )*).

Instance methods can be defined using the *Methods* tab.

Attributes that are not related to the value or property of UI elements can be declared using the *Attributes* tab. These attributes are then displayed in all methods of this controller. There are two predefined attributes, which are used to access the functions of the controller (*WD_THIS*) and of the context (*WD_CONTEXT*).

For information to be shared between different controllers, one controller must declare the use of another controller. This is done on the *Properties* tab of the controller that needs to access another controller. The most frequent requirement for this kind of data sharing is when you wish to create a mapped context node or you wish to access another controller's instance methods.

Do not, under any circumstance, enter the name of a view controller as a used controller, as this would violate good MVC design principles. A view controller should be written such that it is responsible for nothing more than the display of data and user interaction. A view controller is not responsible for generating the data it displays. That is the role of a custom controller.

Business logic, such as function modules, BAPIs, or methods in helper classes can be accessed from the methods of all controllers.

Special Entities of Component / Custom Controllers

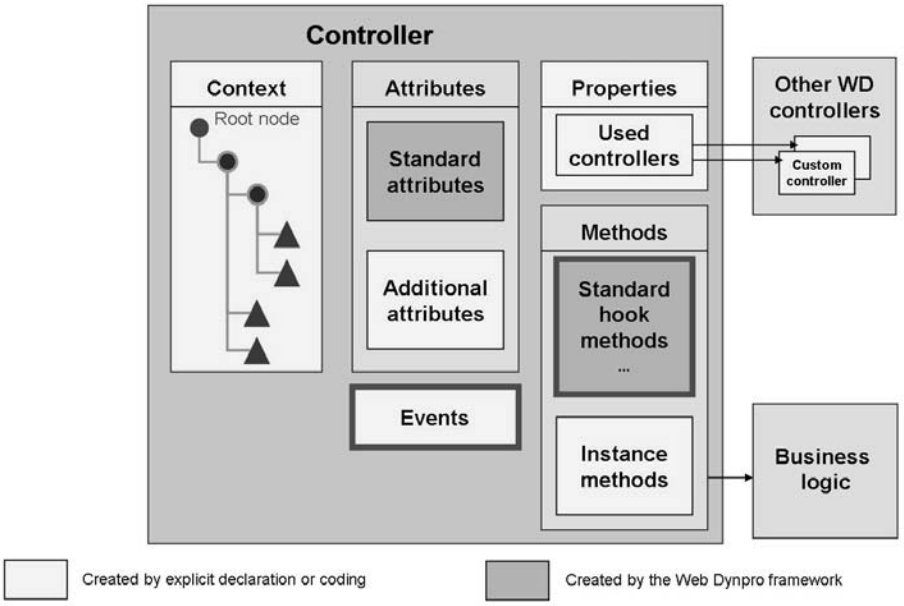


Figure 151: Component/Custom Controllers: Special Entities

Create an event in the custom controller of your component. Define a controller usage of the custom controller in the component controller. Define an event handler method in the component controller and register this method to the custom controller event.

For both component and custom controllers, **events** can be defined with arbitrary parameters. Any method of any other controller (also view and window controllers) can register to these events if this method is defined as an event handler method. A typical use of such events is the triggering of processing in a view controller after processing in the component controller has been completed. This can be achieved when a method in the view controller subscribes to an event raised by the component controller.

Using your design time declarations, the Web Dynpro framework will automatically manage the definition, triggering, and handler subscription to such events for you. You also have the additional option of implementing dynamic event registration at runtime.

Caution: If two or more methods are registered to the same event, the order in which they will be executed is not defined.

The component controller and **only** the component controller has the following additional standard hook methods. These are called directly before the navigation requests are processed (*wddobeforenavigation( )*) and after all views of the view assembly to be sent have been processed (*wddopostprocessing( )*).

Attributes, methods, context elements, and events can be marked as interface elements. These elements are then exposed to other components by means of the interface controller.

Special Entities of View Controllers

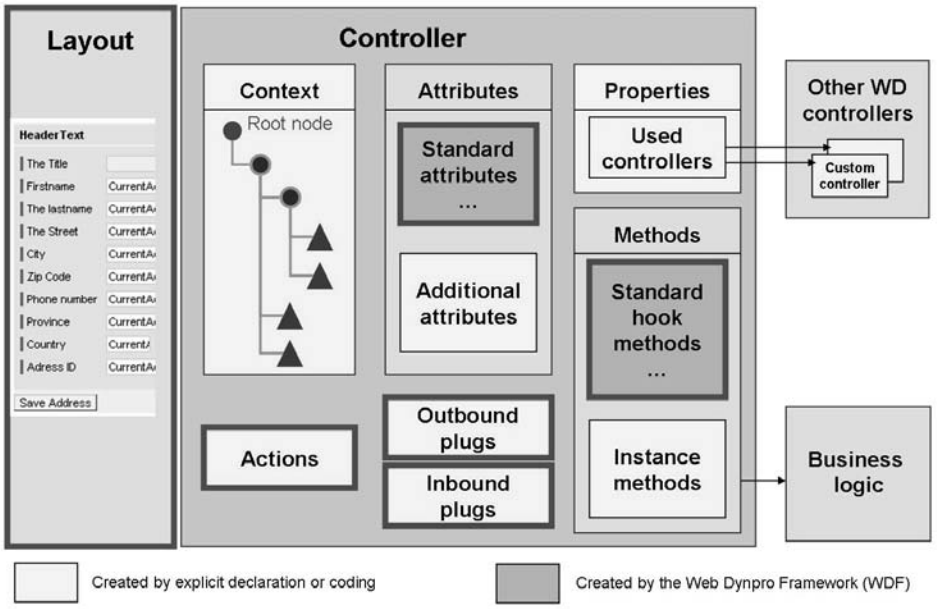


Figure 152: View Controllers: Special Entities

Your application: Create a second view and embed this view in your window. Add an outbound plug and an inbound plug to each of the views (no plug parameters). Go back to the window and add navigation links, so navigation back and forth between the views is possible. Add a button to each of the views to trigger the navigation. For each of the views: Create the action and the fire_plug statement by entering the action's name to the onAction property of the button. Test the application.

Demonstrate that the WDCOMPCONTROLLER attribute is generated automatically for each controller if the component controller is added to the used controller list.

A view controller is a visual building block of a Web Dynpro component and is designed to handle all aspects of data display and user interaction.

For navigation to take place between different Web Dynpro view controllers, special navigation events and navigation event handlers have been created. These are called **navigation plugs**.

A navigation event is raised when an **outbound plug** is fired.

The declaration of an outbound plug triggers the generation of a method in the view's component controller. If the generic name *<Outbound-Plug>* is used for an outbound plug, this method is called *FIRE_<Outbound-Plug>_PLG*. This method is only visible for the Web Dynpro framework. It is not visible to the developer.

An **inbound plug** is the navigation event handler that can be registered to a navigation request. The declaration of an inbound plug triggers the generation of a method in the view's component controller. If the generic name *<Inbound-Plug>* is used for an inbound plug, this method is called *HANDLE_<Inbound-Plug>*.

A static registration of an inbound plug (method *HANDLE<Inbound plug>*) to the navigation event raised by an outbound plug (method *FIRE_<Outbound plug>_PLG*) is called a **navigation link**. Navigation links are not part of a view but are defined in a window embedding the view. Therefore, in different windows, the event registration can be defined differently.

An **action** links a client-side event (for example, selecting a button in a browser) to an event handler method defined in the corresponding view controller. When defining an action having the name *<Action>*, an event handler method (*ONACTION<Action>*) is automatically generated. If a view will be replaced as a result of a client event, the related outbound plug can be fired in this action event handler method.

There are two special hook methods found in view controllers:

- *wddobeforeaction( )* is processed if a client event is fired in any view.

  Before the action event handler methods are processed, the*wddobeforeaction( )* methods are executed for all views that are part of the last view assembly.

- *wddomodifyview( )* is a method that allows you to programmatically manipulate the layout of the view. This event can be used to define the UI dynamically.

In addition to the standard attributes of all controllers, a view controller has a reference to the component controller of its component: ***WD_COMP_CONTROLLER***. This reference can be used to access functionality related to the component controller. It is used when accessing messages or when calling a method declared in the component controller.

Start demo NET310_EVEN_D1. Discuss the order in which the events are processed. Repeat the discussion with demo NET310_EVEN_D2

Here, you could also demonstrate the influence of the view's lifetime property on the lifetime of the view. Start the WD applications NET310_NAVI_D2 and NET310_NAVI_D2A. The related components are identical, only the lifetime of the views is different. The layout displays how often the WDDOINIT hook method is processed. If the view survives a navigation request, this method will not be processed again. WDDOEXIT is only processed if the view is deleted.

Special Entities of Window Controllers

Window controllers are very similar to view controllers. Technically, it is like a view controller without a UI (view layout). All views that are to be displayed when using a Web application have to be embedded in the window that is referred to by the application or the calling component.

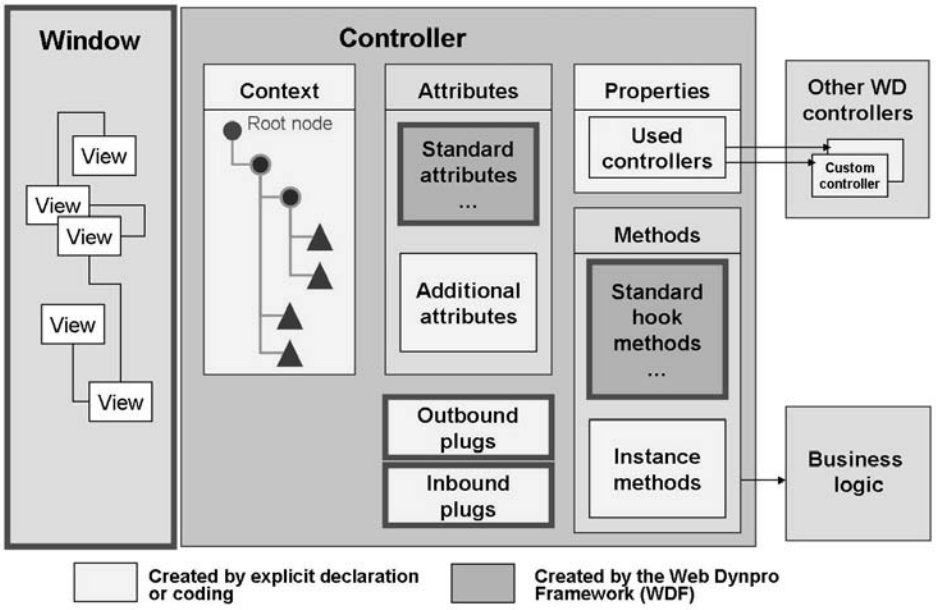


Figure 153: Window Controller Architecture

Your component: Create a new view consisting of two view containers, VC1 and VC2. Embed this view in the window and make it the default view. Embed the existing start view in VC1 and embed the existing result view in VC2. Also include an empty view in VC2 and make it the default view in this view container. Define

the navigation so that clicking on the button of the start view in VC1 will replace the empty view in VC2 with the result view. When you select the Back button in the result view in VC2, the empty view will be displayed again in VC2.

Having demonstrated this, you should discuss the applications NET310_NAVI_D1 and NET310_NAVI_D1_, followed by NET310_NAVI_D1A (using window plugs for navigation) and NET310_NAVI_D1B (using main view plugs for navigation, pass state information using plugs).

The Web Dynpro window contains the structure of all views to be displayed and (if statically defined) the navigation links defining the possible view assemblies.

Each Web Dynpro window contains outbound and inbound plugs, and views. You can use the plugs to set up cross-component navigation. To expose the plugs to the component interface, activate the *Interface* property for each plug. These plugs will then be part of the related interface view.

The window controller also has the special attribute *WD_COMP_CONTROLLER*, which holds the reference to the component controller.

Exercise “Web Dynpro Controllers”

247

Exercise 14: Web Dynpro Controllers

Exercise Duration: 30 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Create buttons on Web Dynpro views
- Trigger actions by the selection of buttons by the user
- Implement navigation between Web Dynpro views

Business Example

You want to develop a Web Dynpro application with more than one view. On your views, you wish to provide buttons to allow the user to initiate navigation to the text view.

Template: None

Solution: NET310_CTRL_S

Task 1:

Create a Web Dynpro component with a Web Dynpro window and two Web Dynpro views.

1. Create Web Dynpro component **ZNET310_CTRL_##** with a main window **MAIN_WINDOW**.
2. Create two Web Dynpro views (suggested names: **INPUT_VIEW** and **OUTPUT_VIEW**) in your component and embed them into your window.

Task 2:

Create an outbound plug and an inbound plug for each view. Define two navigation links between the two views – each going from the outbound plug of one view to the inbound plug of the other view.

1. For your first view (**INPUT_VIEW**), create an outbound plug (suggested name: **TO_OUTPUT_VIEW**) and an inbound plug (suggested name: **FROM_OUTPUT_VIEW**).

Continued on next page

2. What new parts have been created in the view controller by completing the previous step (refer to the various tabs of the view editor)?

3. Do the same for your other view (suggested names for the plugs: **TO_INPUT_VIEW** and **FROM_INPUT_VIEW**).
4. Define a navigation link from the outbound plug of the first view (TO_OUTPUT_VIEW) to the inbound plug of the second view (FROM_INPUT_VIEW).
5. Define a navigation link from the outbound plug of the second view (TO_INPUT_VIEW) to the inbound plug of the first view (FROM_OUTPUT_VIEW).

Task 3:

Create a button on each of the two views and make sure the outbound plug of the respective view is fired when the button is selected by the user.

1. Create a simple UI element of type *BUTTON* (suggested name: **BUTTON_GO**) on the first view and maintain the text to be displayed on the button.
2. Create an action for the button (suggested name: **GO**) and relate it to the outbound plug of the view (TO_OUTPUT_VIEW).

3. What new parts have been created in the view controller by completing the previous step (refer to the various tabs of the view editor)?

4. Create a button on the second view (suggested name: **BUTTON_BACK**) with text and action (suggested name: **BACK**), and link to the outbound plug of that view.
5. Create a Web Dynpro application for your component, activate, and test it.

Solution 14: Web Dynpro Controllers

Task 1:

Create a Web Dynpro component with a Web Dynpro window and two Web Dynpro views.

1. Create Web Dynpro component **ZNET310_CTRL_##** with a main window **MAIN_WINDOW**.

   a) Perform this step as in the previous exercise.

2. Create two Web Dynpro views (suggested names: **INPUT_VIEW** and **OUTPUT_VIEW**) in your component and embed them into your window.

   a) Perform this step as in the previous exercise.

Task 2:

Create an outbound plug and an inbound plug for each view. Define two navigation links between the two views – each going from the outbound plug of one view to the inbound plug of the other view.

1. For your first view (**INPUT_VIEW**), create an outbound plug (suggested name: **TO_OUTPUT_VIEW**) and an inbound plug (suggested name: **FROM_OUTPUT_VIEW**).

   a) Edit the Web Dynpro view INPUT_VIEW.

   b) Select the *Outbound plugs* tab and enter the outbound plug name and a description in the topmost table.

   c) Select the *Inbound plugs* tab and enter the inbound plug name and a description.

2. What new parts have been created in the view controller by completing the previous step (refer to the various tabs of the view editor)?

   Answer:

   Event handler method “HANDLEFROM_OUTPUT_VIEW” (empty).

3. Do the same for your other view (suggested names for the plugs: **TO_INPUT_VIEW** and **FROM_INPUT_VIEW**).

   a) Perform this step as before.

Continued on next page

4. Define a navigation link from the outbound plug of the first view (TO_OUTPUT_VIEW) to the inbound plug of the second view (FROM_INPUT_VIEW).

   a) To edit the Web Dynpro window, choose the *Window* tab and expand all nodes of the window structure.

   b) Drag the outbound plug of the first view and drop it on the inbound plug of the second view. Confirm the data in the dialog box.

5. Define a navigation link from the outbound plug of the second view (TO_INPUT_VIEW) to the inbound plug of the first view (FROM_OUTPUT_VIEW).

   a) Perform this step as before.

Task 3:

Create a button on each of the two views and make sure the outbound plug of the respective view is fired when the button is selected by the user.

1. Create a simple UI element of type *BUTTON* (suggested name: **BUTTON_GO**) on the first view and maintain the text to be displayed on the button.

   a) Edit the view and choose the *Layout* tab.

   b) Open the context menu for ROOTUIELEMENTCONTAINER and choose *Insert Element*.

   c) Enter the name of the button and choose the element type *BUTTON*.

   d) Go to the *Text* property and maintain the text to be displayed in the *Value* column.

2. Create an action for the button (suggested name: **GO**) and relate it to the outbound plug of the view (TO_OUTPUT_VIEW).

   a) Go to the *OnAction* property for the button. Choose *Create* ❑ in the *Binding* column.

   b) In the dialog box, enter the name of the action, a description, and the name of the outbound plug.

Continued on next page

3. What new parts have been created in the view controller by completing the previous step (refer to the various tabs of the view editor)?

   Answer:

   Action “GO”

   Event handler method “ONACTIONGO” (with coding to fire the outbound plug)

4. Create a button on the second view (suggested name: **BUTTON_BACK**) with text and action (suggested name: **BACK**), and link to the outbound plug of that view.

   a) Perform this step as before.

5. Create a Web Dynpro application for your component, activate, and test it.

   a) Perform this step as in the previous exercise.

Lesson Summary

You should now be able to:

- Name the contents of a Web Dynpro controller
- Define the function of each controller entity
- List the differences between the component controller, custom controllers, view controllers, and window controllers

Unit Summary

You should now be able to:

- Name the contents of a Web Dynpro controller
- Define the function of each controller entity
- List the differences between the component controller, custom controllers, view controllers, and window controllers

Unit 9

255

The Context at Design Time

It is a good idea to relate the node properties such as *cardinality* and *singleton* to elements with which students are familiar. You could use the following relationship, for example:

Node <-> internal table

Element <-> column of internal table

Then the cardinality can be related to the restrictions on the number of lines allowed in the internal table. In this context, the singleton property can be declared using the memory type of dependent internal tables in a classic ABAP environment. The dependent table could be the content of a cell in each table line (*singleton=false* <-> nested internal tables) or the dependent table could be an independent element that contains the dependent data of just one line of the parent table (*singleton=true* <-> independent internal tables).

Unit Overview

The context is the data storage of a Web Dynpro component. Data is stored here in a hierarchical structure, which can be defined statically or at runtime. Understanding the context structure elements and their properties is the prerequisite for working with the context. In this unit, we will explain how to statically define the component structure.

Unit Objectives

After completing this unit, you will be able to:

- Define nodes and attributes in a controller's context
- Explain how the node property's cardinality and singleton influence the memory allocation at runtime
- Define the mapping between contexts of different controllers located in the same Web Dynpro component

Unit Contents

Lesson: The Context at Design Time

256 Lesson Duration: 70 Minutes

Lesson Overview

At design time, the developer can statically define the context structure of any controller. Context elements have to be added to the structure hierarchy and properties have to be set for each context element. At runtime, these properties rule the memory allocation, for example, the type of the content and the dimension of data arrays to be stored. It is therefore essential to understand the static definitions and the runtime properties related to the context. This lesson gives a basic introduction to the context.

Lesson Objectives

After completing this lesson, you will be able to:

- Define nodes and attributes in a controller's context
- Explain how the node property's cardinality and singleton influence the memory allocation at runtime
- Define the mapping between contexts of different controllers located in the same Web Dynpro component

Depending on how much has been shown in the chapter introducing Web Dynpro, some topics can be skipped (for example, context mapping).

In this lesson you should continue to develop your WD component. You should create a context node in the component controller (with some attributes), copy this node to both views, and map the context nodes of both views to the component controller context node. You should create container forms that allow data to be entered in one view and displayed in another view.

Business Example

You have created your first Web Dynpro component with views that are embedded by windows. You have created a Web Dynpro application to access your Web Dynpro component from the browser. However, you want your application data from a backend system and not just static data. You therefore need to define variables that can be bound to UI elements and that can be easily interchanged between different controllers. These variables must be defined in the controller context, so you want to learn about how to define this hierarchical data storage.

Defining the Context Structure

Each Web Dynpro controller has exactly one hierarchical data storage structure, known as the context. The data held in the context exists only for the lifespan of the controller. Once the controller instance has been terminated, all data held within its context is lost.

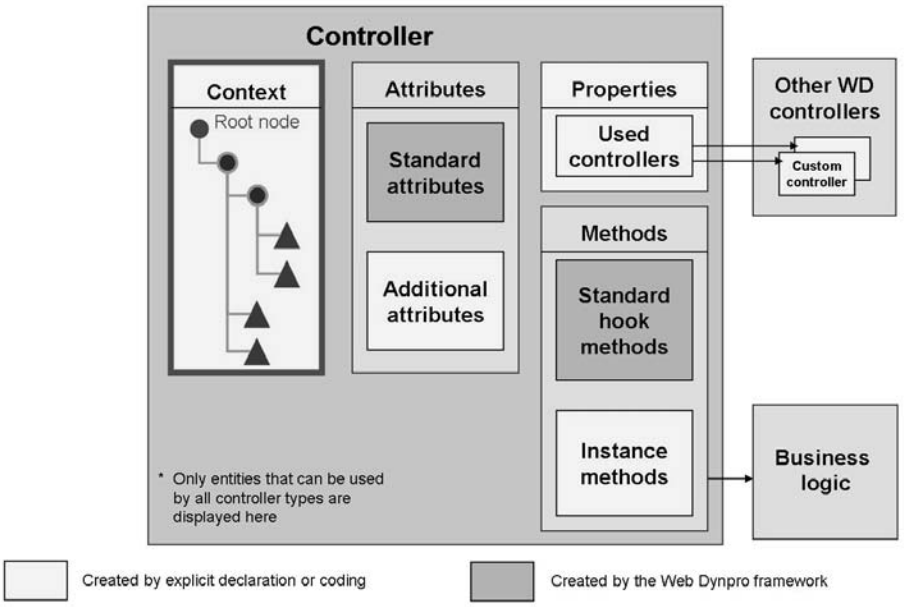


Figure 154: Context: The Heart of a Controller

The structure (that is, the metadata) of a context will typically be defined at design time. However, at runtime, it is possible not only to modify the contents of the context, but also to modify its structure itself.

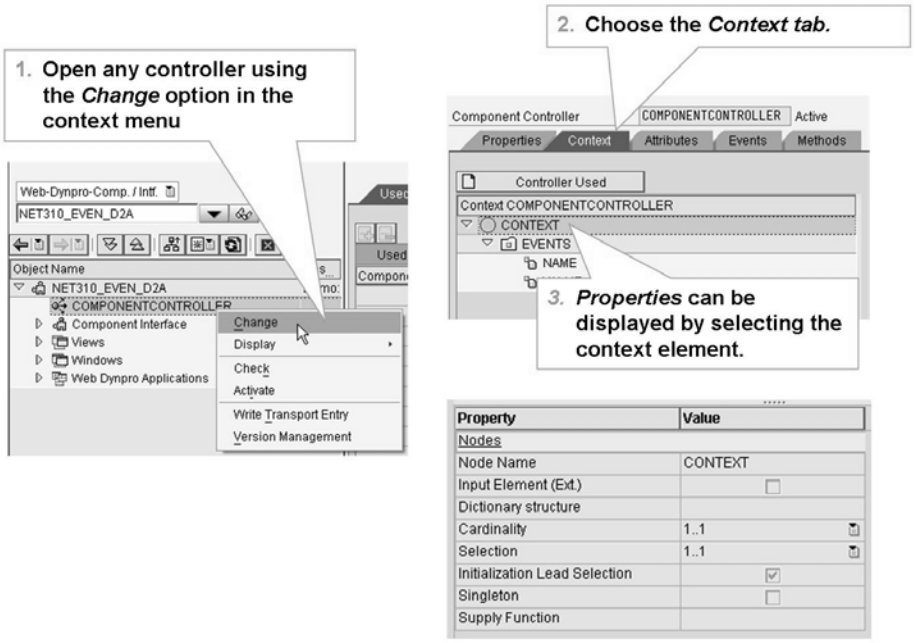
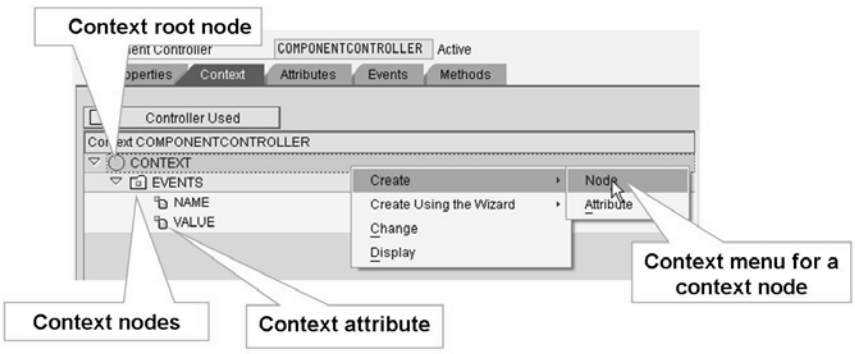


Figure 155: Changing the Context

Unless declared differently, all runtime data within a controller's context is private to that component.

Information held in the context of a custom controller can be made easily accessible to the context of another controller (view or custom) by a technique known as **context mapping**. Using this technique, two or more controllers can access the same runtime data. This is the primary mechanism for sharing data between controllers within a single component.

It is not possible for a view controller to share its context data.



Context nodes:

- Arranged hierarchically
- Permitted to have children (nodes or attributes)
- Metadata description declared manually or derived from a DDIC structure

Context attribute:

- Stores runtime data or references to runtime data
- Based on DDIC types

Figure 156: Creating New Context Elements

Demo step 1: Create a context node with attributes in the component controller.

Show how to define a node without attributes and add the attributes one by one. Show also how a node related to a DDIC structure and some attributes related to structure fields can be defined in one step. Show how to add or delete some of the attributes after having created the node (use *Change* from the context menu of the node). Finally, show how to copy a node from another controller (choose *Create Using the Wizard -> Copy Nodes of Different Context).*

All controller contexts are constructed from a hierarchical arrangement of entities known as **nodes** and **attributes**. A context always has a parent node known as the **context root node**. The root node is created automatically when the controller is initialized and always has fixed properties. The context root node cannot be deleted or modified in any way.

A **context node** is the main abstraction class used for runtime data storage within the Web Dynpro framework. Context nodes are arranged hierarchically. The node may have attributes or other nodes as children.

All the child entities of a node are aggregated into a unit known as an **element**. A node can then be thought of as a collection of such elements in the same way that a table is a collection of rows.

Caution: The name of a context node must be unique within the complete structure of a controller's context.

A **context attribute** is an entity within the context that is *not* permitted to have children. A context attribute can only exist if it is assigned to a parent node. This parent node can be a context root node or another node.

If a collection of structured data objects is to be stored in a controller context at runtime, a context node must be defined to store the collection itself. Context attributes or other context nodes must be defined as sub-elements that store the elements of each structured data object.

Context Element Properties

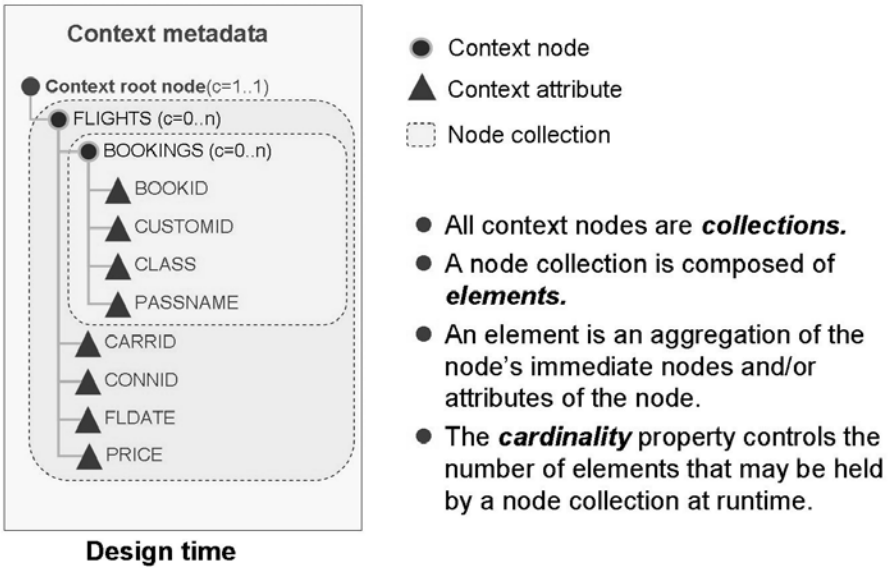


Figure 157: Context Structure at Design Time

The Collection Cardinality

At design time, you create the metadata structure within which the runtime data will be stored. The diagram above gives the impression that the context structure is a flat, two-dimensional tree, much like the display of directories and files shown in Windows Explorer. However, all context nodes are collections, so there could potentially be multiple instances of each child node and attribute within a nodes collection.

Each context node has a property called *cardinality*. This property is composed of two values that, taken together, describe the maximum and minimum number of elements the node collection may hold at runtime:

Cardinality minimum: **0** or **1**

Cardinality maximum: **1** or **n**

Therefore, there are four possible cardinality values (specified as <*Min*>..<*Max*>):

- **0..1**: Zero or one element permitted
- **0..n**: Zero or more elements permitted
- **1..1**: Exactly one element permitted
- **1..n**: One or more elements permitted

If the cardinality is 0:1 or 0:n and no element is created and explicitly added to the collection, input fields bound to these attributes are set as read-only by the WD runtime.

All nodes contain an element collection, even if the maximum number of elements within the collection is limited to one.

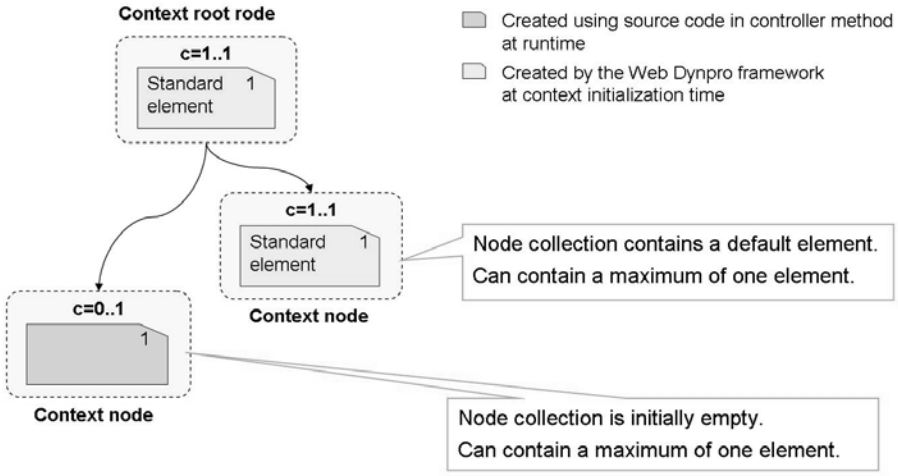


Figure 158: Context Structure at Runtime: Cardinality Property (1)

The context root node always contains exactly one element (notice the cardinality is **1..1**). This is known as the default element and it cannot be deleted. This also implies that any node collection that is a direct child of the context root node does exist at runtime as an empty collection (*cardinality = 0..<something>*) or with one

default element (*cardinality = 1..<something>*). These child nodes of the context root node are therefore called **independent nodes**. All other node collections exist only if their parent collection contains at least one element. The existence of these node collections at runtime is therefore not guaranteed. These nodes are therefore called **dependent nodes**.

If you attempt to perform any action on a node collection that would violate its cardinality, you will get a runtime error from the Web Dynpro framework. Such actions include:

- Trying to delete the default element from a node of cardinality *1..<something>*
- Trying to add a second element to a node of cardinality *<something>..1*

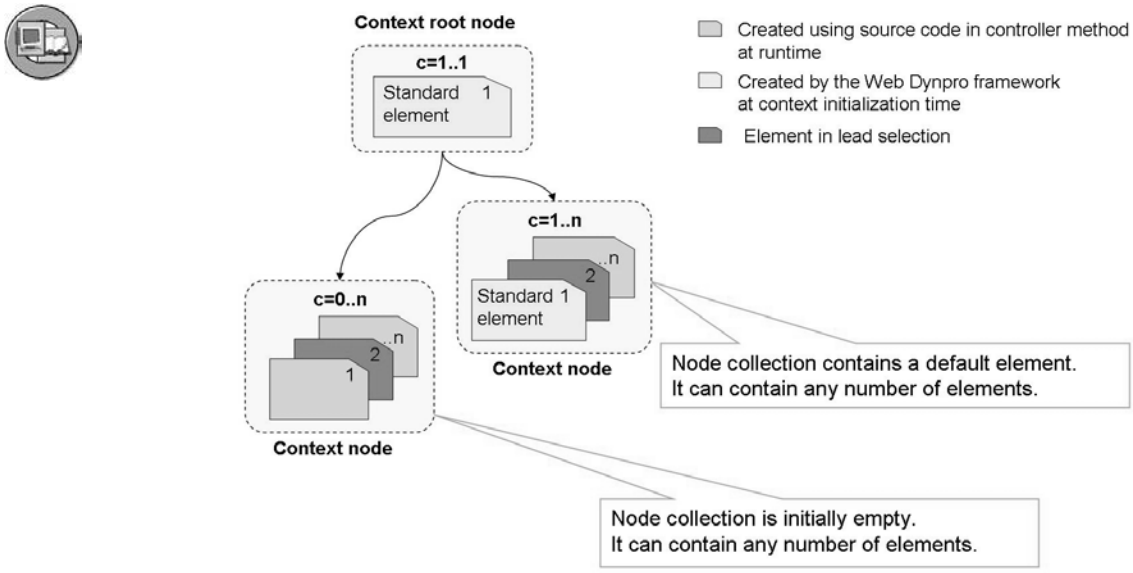


Figure 159: Context Structure at Runtime: Cardinality Property (2)

You can now see that the structure of the context at runtime will not be the flat, two-dimensional hierarchy seen at design time. Instead, a context node takes on depth.

This can be compared to an internal table in ABAP at design time and runtime. At design time, the metadata of the internal table comprises the structure given by the line type. The dimension of the internal table (how many lines the data object will hold at runtime) is not part of the definition. At runtime, multiple lines can be appended to the internal table data object (as multiple elements can be appended to a collection originating from a context node at runtime). However, since there are no restrictions on the minimum and maximum number of lines an internal table can hold, there is no equivalent for the *cardinality* property. Internal tables always have zero lines after the declaration of the object, and they can hold an unrestricted number of lines at runtime.

The Lead Selection

A node's element collection can be accessed using a 1-based index value. Exactly one element of the node collection can be marked as the element at lead selection. The lead selection of a context node points to either a single selected node element (value of the lead selection = number of the selected node element) or, if no element is selected, it has the value of the constant *IF_WD_CONTEXT_NODE=>NO_SELECTION*. The lead selection can be set automatically by the Web Dynpro framework if the context node property *Initialize Lead Selection* is set to *true*. In this case, the first element in a collection will automatically be marked as the element at lead selection. The lead selection can also be set by program source code or it can be set by user actions related to UI elements (for example, by selecting a line in the table view element that is bound to the node).

Your component: Open one of the views, navigate to the context and show the property of the previously created context node (*cardinality*, *Initialization Lead Selection*, ...). Show that these properties can only be manipulated for the mapping origin, and not for the mapped nodes.

If the lead selection is set, the following is true:

- In the controller code, special methods can be used to access the lead selection of a node, as well as its position.
- UI elements such as input fields can be bound to the attributes of this element.

The Singleton Property

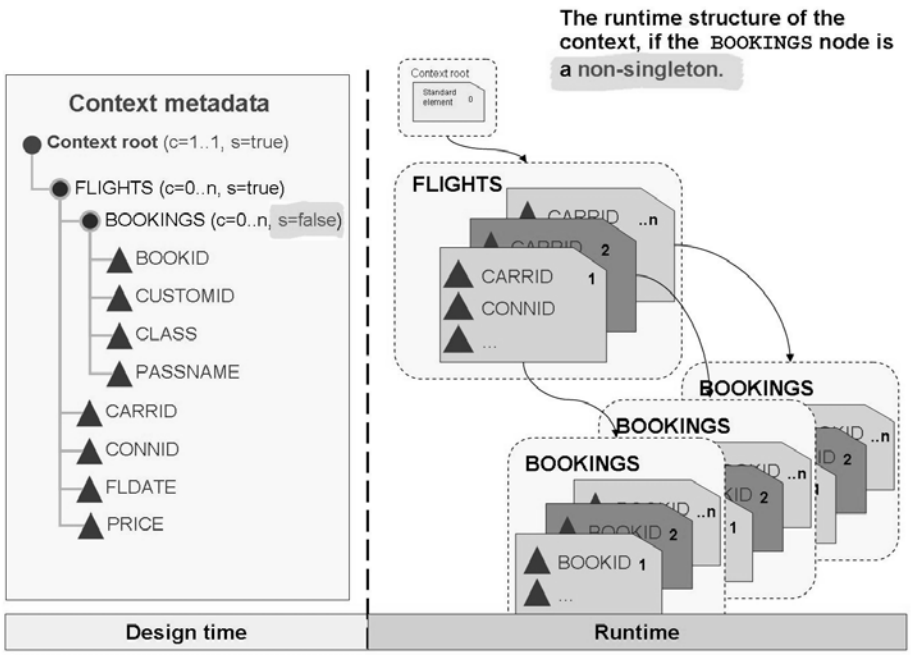


Figure 160: Context Structure at Runtime: Singleton Property (1)

Notice that the context node FLIGHTS has a child node called BOOKINGS. The BOOKINGS node is a separate node with its own element collection.

The Boolean property ***Singleton*** critically affects the relationship between a dependent node and its parent node.

If the BOOKINGS node has its singleton property set to false (node BOOKINGS at runtime will be a **non-singleton**), then for every element in the parent node collection (FLIGHTS, in this case), there will be a distinct instance of the child node BOOKINGS.

The most important thing to understand here is that each instance of the BOOKINGS node is related to the respective element in the parent node collection. Notice that the arrows pointing to each of the BOOKINGS node collections originate from each element in the parent node.

Therefore, if there are *n* elements in the parent node, then there will be *n* distinct instances of a non-singleton child node.

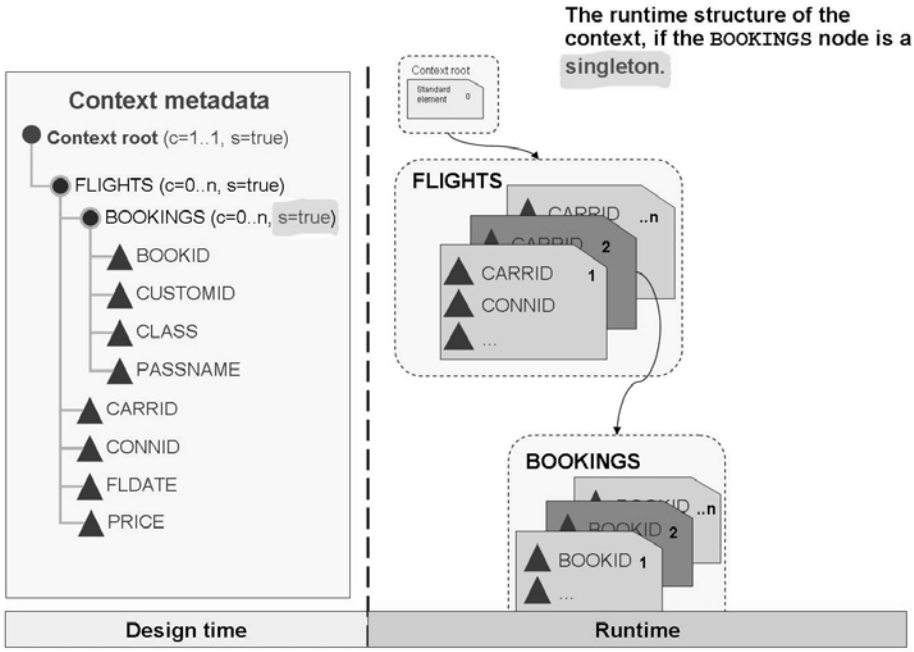


Figure 161: Context Structure at Runtime: Singleton Property (2)

If the node BOOKINGS now has its *Singleton* property set to true (which is the default), it does not matter how many elements are present in the parent node collection (FLIGHTS, in this case). There will only ever be one instance of the child node BOOKINGS, so the BOOKINGS collection will be a singleton at runtime.

Supply Function

In the above example, there could be many different elements in the FLIGHTS node collection. However, since there is only ever one instance of the singleton child node BOOKINGS, we need to ensure that when this child node is accessed, it contains the correct data for the selected element in the parent node.

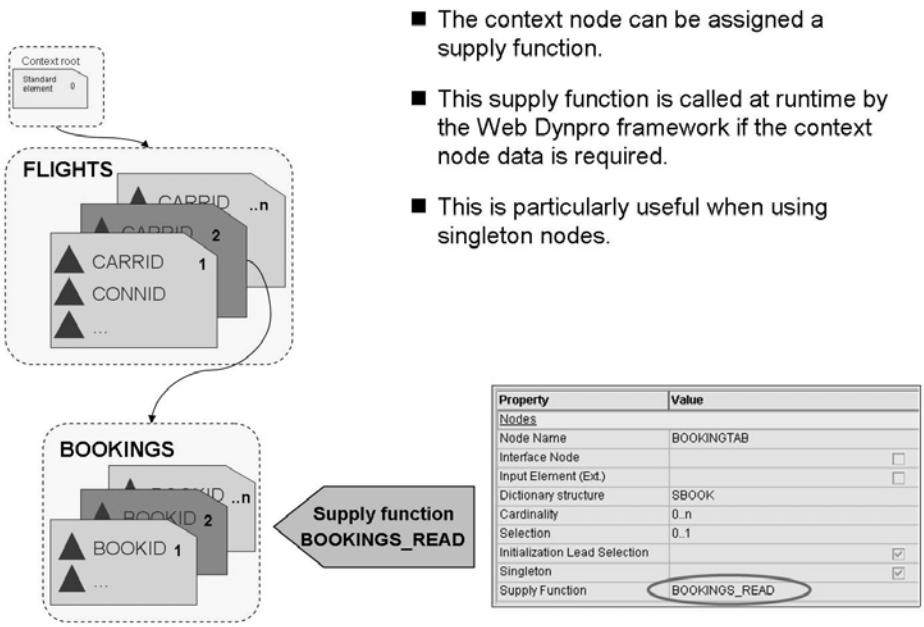


Figure 162: Repopulating a Dependent Singleton Node: Supply Function

Supply functions are a mechanism for repopulating child nodes. A supply function can be assigned to each context node of a controller. This supply function is called by the runtime automatically when the data of an **invalid context node** is accessed. Invalidation of a context node occurs under the following conditions:

- The node collection is initial.
- The lead selection in the parent node collection is changed.
- The node collection is invalidated by source code.

The supply function is especially useful when combined with singleton nodes. The values of child node elements of the type *Singleton* depend on the lead selection element of the parent node. If the lead selection is changed by the user, the supply function can access the new lead selection element and recalculate the values of the child node elements accordingly.

The name of a supply function can be entered when defining or changing a context node from its context menu. As a result, an instance method is generated with a given signature:

- A reference to the element at lead selection in the parent node allows access to the attributes of this element.
- A reference to the node to which the supply function is related allows you to store the dependent data in the child nodes collection.

Your component: Edit the component controller context. You have already created a node with some attributes. Now add a subnode to this node. Edit the node's properties. Enter any name in the input field for the property *Supply Function*. Select *Enter*. Open the *Methods* tab. Explain that a related method has been generated.

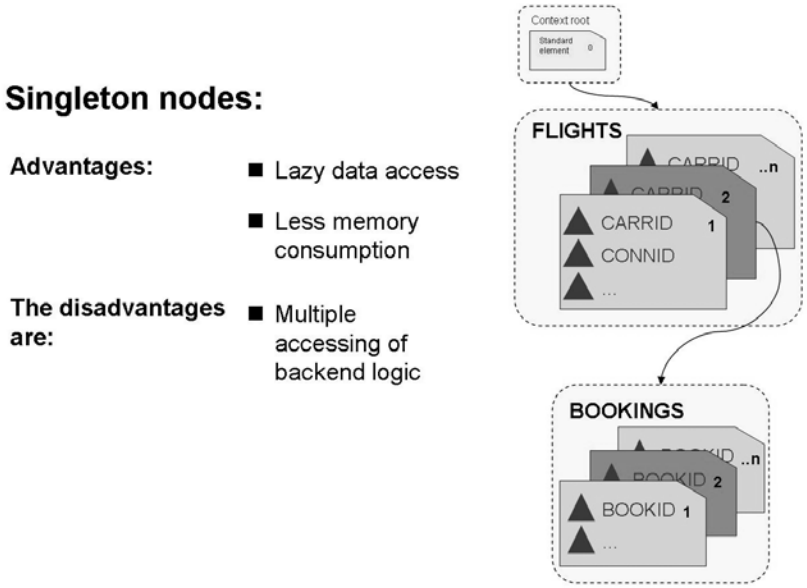


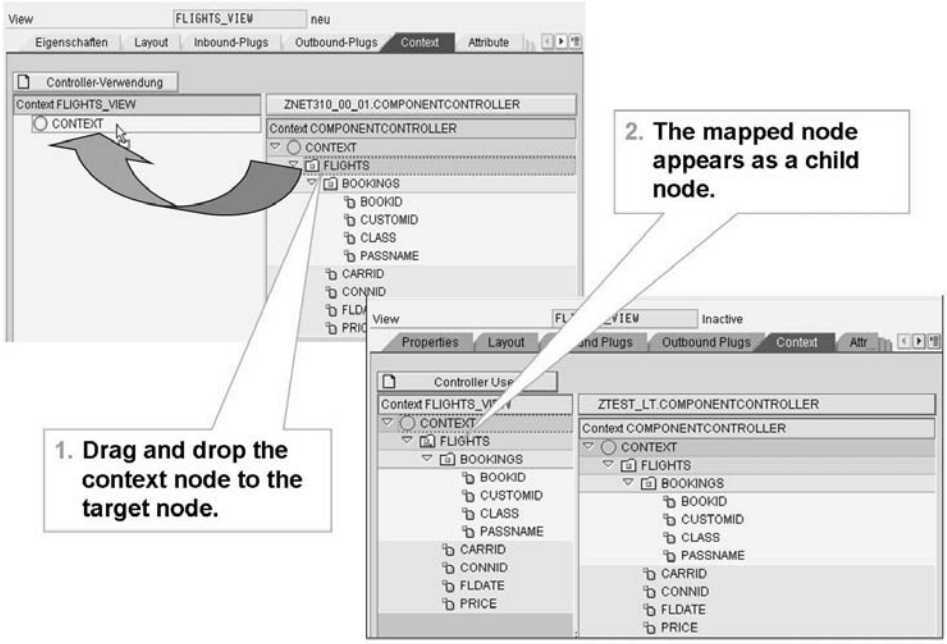
Figure 163: Context Structure: Singleton Nodes

When a typical business transaction is being used, information will often be presented in the form of a list of header records of some sort (for instance, sales order headers). From this list, the user will typically select one order header and then look at its line items. It makes much more sense to read the line item information only when it is needed, rather than to read all the line items for all the sale orders in the hope that the user might want to look at the information. This would be a highly inefficient application architecture both in terms of processing performance and memory usage.

Web Dynpro therefore follows the principle of **lazy data instantiation**. This means that data is only created when it is actually needed. When this principle is applied to the context architecture, it means that the child node will remain unprocessed until the program accesses its data. There is therefore no need to hold multiple instances of data the user has not requested.

Lazy data instantiation also means that dependent collections are not automatically created for all elements of the parent node. The creation of a collection of the dependent node is delayed until the related element of the parent collection gets the lead selection.

Context Mapping



In the property of the node, the mapping path is visible.

Figure 164: Context Mapping

Context mapping allows a controller (typically a view controller) to access data that has been pre-processed by some other controller. Since a mapping relationship allows a direct reference to be made to data in another controller, there is no need for this data to be copied or moved.

Note: A mapped node does not have its own element collection. Instead, it references the element collection of the mapping origin node. A view controller cannot act as a data source for a mapping relationship because this would violate the principles of MVC program design.

Hint: In programs designed according to the MVC principle, a view controller should be written in such a way that it plays no part in the generation of the data it displays. Instead, the view controller should only be concerned with displaying pre-generated data and then handling the resulting user interaction (validation, error handling, and so on). It is the custom controller's job to interact with the backend logic, to generate the required data.

Once the data has been generated, the custom controller then supplies it to a view controller for display. If a view controller were allowed to act as the data source (origin) for a mapping relationship, then a situation could be created in which a custom controller is dependent upon functions found in a view controller. This would be constitute poor MVC design and is therefore not permitted.

To establish the context mapping between two controllers, the target controller has to declare the source controller as a used controller in its properties. If the *Context* tab is chosen in the edit mode of the target controller, the contexts of all used controllers are displayed on the right side. A node of a source context can then be referenced as follows:

Your component: Now edit your views. To create the context structures, drag and drop the node from the component. This also implements context mapping. You can also copy the context node first and implement mapping afterwards by dragging and dropping the view node to the node that serves as the mapping origin. At the end, demonstrate how input fields can be created automatically. Use the WD Code Wizard to create a group that contains the form elements. However, only input fields can be created this way. Using the *Create Container Form* function from the context menu of the ROOTUIELEMENTCONTAINER will not create a group that embeds the form fields. However, the developer can decide which context attribute will be bound to what kind of UI element.

You should also show how changes in the context structure of the mapping origin can be taken into account by mapping (using the functions *Update Mapping* and *Delete Mapping*).

- If an independent node to be mapped is not defined yet in the target controller's context, the source node structure is mapped by dragging and dropping it from the source controller context to the root node of the target controller context.
- If the structure of a node is already defined in the context of the target controller, the target node is dragged and dropped from the context of the target controller to the mapping origin in the source controller context. However, for dependent nodes this is only possible if the related independent nodes are already mapped.

Caution: Nodes can only be mapped if the mapped node does not contain more attributes than the mapping origin.

Hint: Having established the mapping between two nodes, all of their elements are mapped – both attributes and child nodes.

If the node structure of the mapping origin changes (elements are added, deleted, modified), the mapping can be updated (synchronized) using the context menu of the mapped node.

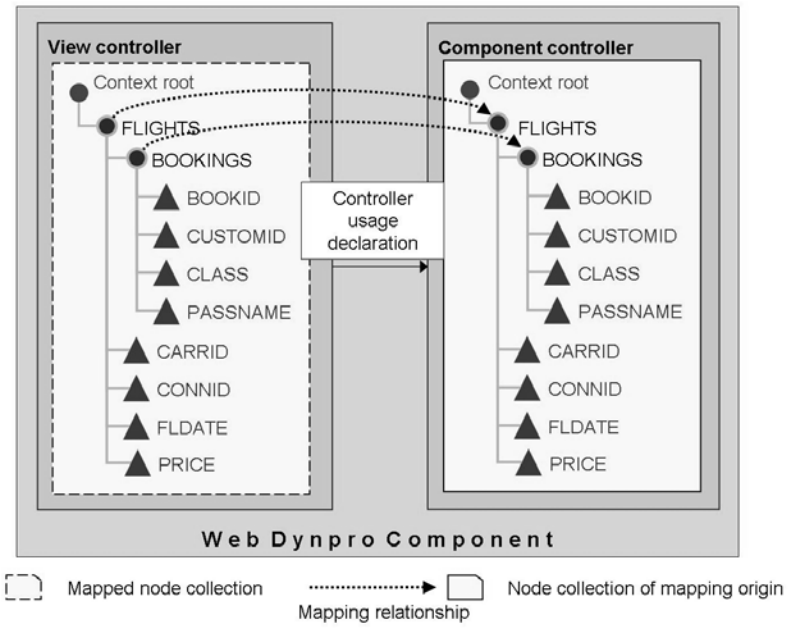


Figure 165: Context Mapping: Internal Mapping

Internal mapping is the name given to a mapping relationship in which both the mapped node and the mapping origin node lie within the boundaries of one component. This means that the mapping relationship can be fully established when developing the current component.

There is however, one special case where a mapping relationship can only be partially established when writing the current component. This is known as **external mapping**. As is implied by the name, the mapping origin node is outside the boundaries of the current component. External mapping will be discussed at a later stage.

Exercise "The Context at Design Time"

271

Exercise 15: The Context at Design Time

Exercise Duration: 30 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Create nodes in contexts of components and views
- Map view context nodes to component context nodes
- Create input/output fields on Web Dynpro views
- Bind UI elements to view context nodes

Business Example

You want to develop a Web Dynpro application with input/output fields on the views. The data the user enters on one view (a carrier ID and a connection ID) should be available as output on the next view.

Template: NET310_CTRL_S

Solution: NET310_COND_S

Task 1:

Copy your solution from the previous exercise or use the template component.

1. Copy your solution from the previous exercise (ZNET310_CTRL_##) or the Web Dynpro component NET310_CTRL_S to the new component, **ZNET310_COND_##**.

Task 2:

In the context of the component controller, enter a context node that contains attributes for an airline ID and a connection ID. Create the same context node in each of the two views, and map the view controller contexts to the context of the component context.

1. In the context of the component controller, create a context node (suggested name: **FLIGHTINFO**) based on the dictionary type SFLIGHT and with the cardinality 1...1. Add two attributes from the structure components CARRID and CONNID.

Continued on next page

2. Create the same node in the context of each view controller and map the contexts node of the view controllers to the context node FLIGHTINFO of the component controller.

Task 3:

For both views, create input fields that correspond to the context node attributes and bind them to these attributes.

1. On the layout of the view INPUT_VIEW, create a form that refers to the context attributes CARRID and CONNID. Use the Web Dynpro Code Wizard to create the form.
2. Repeat the previous step for the view OUTPUT_VIEW.
3. Create a Web Dynpro application for your component, activate, and test it.

Solution 15: The Context at Design Time

Task 1:

Copy your solution from the previous exercise or use the template component.

1. Copy your solution from the previous exercise (ZNET310_CTRL_##) or the Web Dynpro component NET310_CTRL_S to the new component, **ZNET310_COND_##**.

   a) One possible solution: In the Object Navigator, choose *Edit Object*, and select the *Web Objects* tab.

   b) Enter the name of the Web Dynpro component to copy and choose *Copy*.

   c) Enter a name for the new WebDynpro component.

Task 2:

In the context of the component controller, enter a context node that contains attributes for an airline ID and a connection ID. Create the same context node in each of the two views, and map the view controller contexts to the context of the component context.

1. In the context of the component controller, create a context node (suggested name: **FLIGHTINFO**) based on the dictionary type SFLIGHT and with the cardinality 1...1. Add two attributes from the structure components CARRID and CONNID.

   a) Edit the component controller and choose the *Context* tab.

   b) Open the context menu of the root context node and choose *Create → Node*.

   c) Enter the node name, the name of the dictionary type (structure), and the cardinality.

   d) Choose *Add Attributes from Structure*, select the structure components CARRID and CONNID, and select .

Continued on next page

2. Create the same node in the context of each view controller and map the contexts node of the view controllers to the context node FLIGHTINFO of the component controller.

   a) Edit one of the WebDynpro views and choose the *Context* tab.

   b) Drag the context node FLIGHTINFO from the component controller context (right-hand side) and drop it on the root node of the view controller context (left-hand side). Confirm that you want to copy and map the node.

   c) Repeat this step for the second view.

Task 3:

For both views, create input fields that correspond to the context node attributes and bind them to these attributes.

1. On the layout of the view INPUT_VIEW, create a form that refers to the context attributes CARRID and CONNID. Use the Web Dynpro Code Wizard to create the form.

   a) Edit the Web Dynpro view and choose the *Layout* tab.

   b) Start the Web Dynpro Code Wizard by clicking the button in the application toolbar of the Web Dynpro Explorer. Select the template for creating forms.

   c) Choose *Context* and, in the dialog box, double-click the context node (FLIGHTINFO).

   d) Confirm the entries by choosing ✔.

Hint: You can rearrange the UI elements as follows:

- If you move the button to the UI element group using the drag and drop function, this button will appear as the last element in the group.
- If you set the *Layout Data* button property to *MatrixHeadData*, this will move the button to a new row in the group.
- If you maintain the *text* property of the groups caption, a heading is added to the group.

Continued on next page

2. Repeat the previous step for the view OUTPUT_VIEW.
   a) Perform this step as before.
   b) If you want to change the read-only property of the input fields of this view, toggle the *readOnly* checkbox for each input field.
3. Create a Web Dynpro application for your component, activate, and test it.
   a) Perform this step as in previous exercises.

Lesson Summary

You should now be able to:

- Define nodes and attributes in a controller's context
- Explain how the node property's cardinality and singleton influence the memory allocation at runtime
- Define the mapping between contexts of different controllers located in the same Web Dynpro component

Unit Summary

You should now be able to:

- Define nodes and attributes in a controller's context
- Explain how the node property's cardinality and singleton influence the memory allocation at runtime
- Define the mapping between contexts of different controllers located in the same Web Dynpro component

Unit 10

279

Defining the User Interface (UI)

The focus of this chapter is NOT to discuss all UI elements and their properties. This would be very boring as it would be difficult to remember all these details. Instead, this chapter looks at the concepts of data binding, changing UI element properties at runtime and the complex *Table* element . Detailed documentation is available is available for each UI element in the context menu of the UI element hierarchy.

Unit Overview

One of the strengths of Web Dynpro is that a user interface can be easily defined. Like designing a traditional screen, a WYSIWYG tool is used to design a view layout. At runtime, the layout is generated from the metadata, depending on the client used. All aspects of the user interface definition are discussed in this unit.

Unit Objectives

After completing this unit, you will be able to:

- Define the views layout by arranging UI elements and setting their properties
- Bind UI element properties to context attributes
- Use the table UI element as an example of using composite UI elements

Unit Contents

Lesson: Defining the User Interface (UI)


280

Lesson Duration: 160 Minutes

Lesson Overview

Defining the user interface comprises the definition of the UI elements and their arrangement in the view layout, and the binding of the UI element properties to context attributes. The UI element properties can then be controlled from the controller source code by acting on the attributes, guaranteeing the best separation between UI and program logic.

This lesson explains how to arrange the various UI elements, to design the layout of a view. It discusses standard UI elements and complex elements – that is, nested UI elements (for example, *TABLE* or *TREE*). We will also discuss how to define the data binding between UI element properties and context attributes.

Lesson Objectives

After completing this lesson, you will be able to:

- Define the views layout by arranging UI elements and setting their properties
- Bind UI element properties to context attributes
- Use the table UI element as an example of using composite UI elements

---

Only the simple elements and the container elements are discussed here. Some students may also ask for trees, Gantt charts, network charts or office integration UI elements. Some of these elements - for example, information on integrating interactive forms (of Adobe), will be explained in the special topics chapter at the end of this handbook. Demos on business graphics are also contained in the NET310 package. A standard SAP Web Dynpro application (**WDR_TEST_UI_ELEMENTS**) can also be used to check out the functions of elements not covered in this training material.

Business Example

You want to create a Web Dynpro application that has an optimized user interface (for example, you want to arrange the elements on the screen in groups or trays). Mass data should be displayed in tables, which allows you to mark one or multiple rows, and filter or sort the data. In addition, certain properties of the UI elements (such as visibility) should be controlled from the application source code. You therefore want to learn more about UI elements.

Defining a View Layout

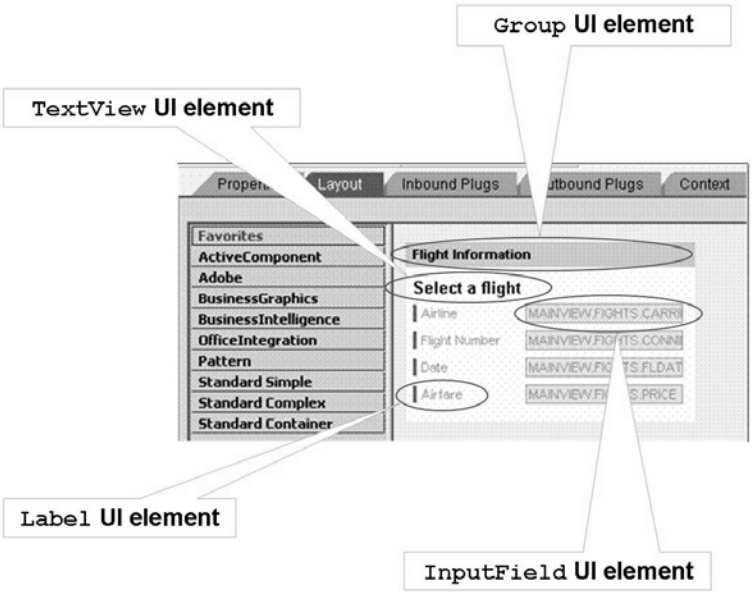
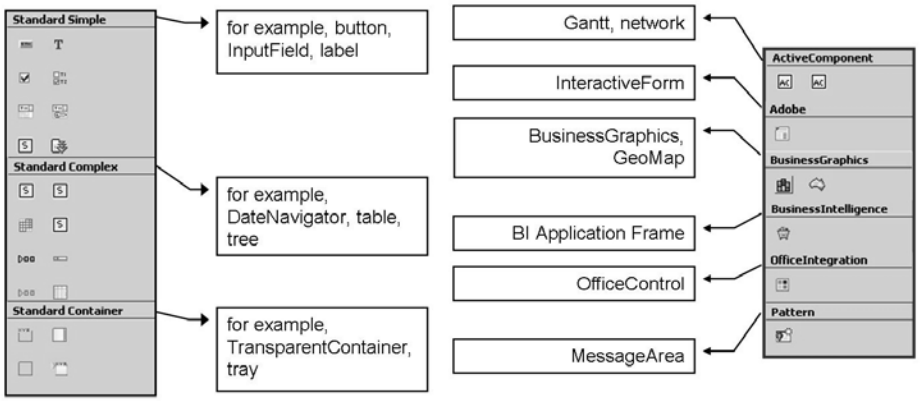


Figure 166: UI Elements

A **UI element** is any graphical entity that occupies a position within a view layout. However, that does not mean that all UI elements are visible on the screen. There are certain UI elements that are not visible on the screen, such as the *TransparentContainer* or the *ViewUIElementContainer*. These elements are used to structure the UI without being visible, but they occupy a position in the UI element hierarchy just like any other visible UI element. In addition, all UI elements can be set to invisible at runtime without freeing the space they would occupy as visible UI elements. For example, hiding the *Label* UI element located left of an *InputField* UI element does not automatically mean that the *InputField* UI elements moves left and appears at the position where the *Label* UI element was previously displayed.

Web Dynpro has been designed to operate with any form-based user interface. This is probably going to be a standard browser, but it could also be a Smart Client, a Pocket PC™, or a Blackberry™ in a future release. This implies that any UI element is only an abstract description of the source code to be rendered at runtime, and not just an HTML or WML representation of this element.



Each UI element object is represented as an abstract class that is independent of any client presentation layer.

Figure 167: UI Element Categories

There are numerous user interface elements available for designing the user interface of a Web Dynpro application. All available user interface elements are divided into categories, which are displayed in the view designer, if the layout preview is visible.

The *Standard Simple* category contains elements that are used frequently in Web Dynpro applications and that are also known from other UI technologies like *Button*, *Label*, or *InputField*.

The *Standard Complex* category contains elements that need to have child elements to define a valid, renderable UI element. A good example is a *Table*, which needs to have a *TableColumn* element as a child for each column to be displayed.

The *Standard Container* category comprises elements that may have child elements. Container UI elements structure the layout visibly (for example, *Group*) or non visibly (for example, *TransparentContainer*).

Other categories contain elements to display ActiveX-based diagrams (*Active Component*), Adobe interactive forms (*Adobe*), or business graphics rendered by the Internet Graphics Server (*BusinessGraphics*), or to embed Microsoft Office documents like Microsoft Word or Excel documents (*OfficeIntegration*) and some other special UI elements.

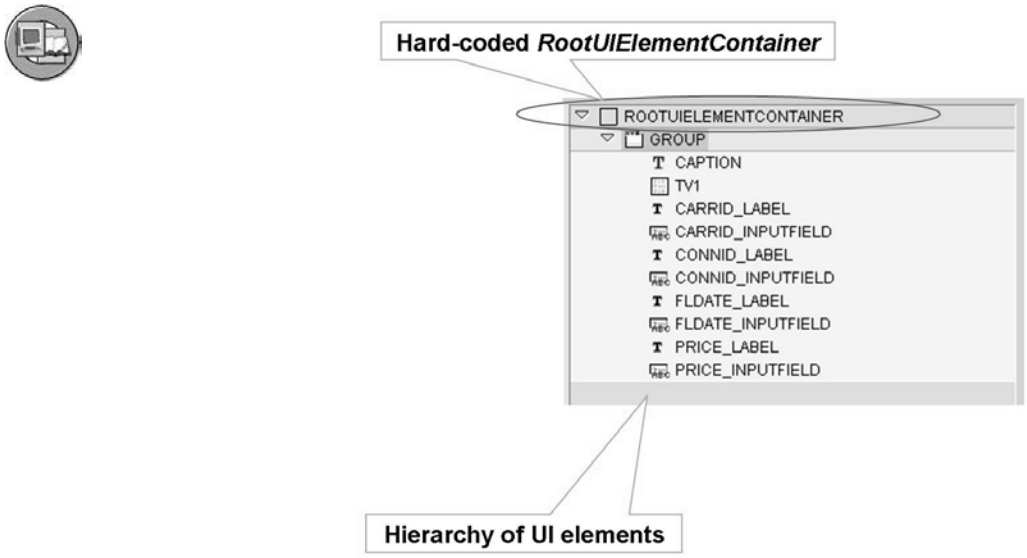


Figure 168: Arrangement of UI Elements

Show documentation available for each UI element (right mouse click on any element defined in UI element hierarchy)

All view layouts are composed from a hierarchy of UI elements. The root node is always of the type *TransparentContainer*, and is always called ***RootUIElementContainer***. You cannot change this, it is hard-coded. All subsequent UI elements added to a view layout are hierarchically subordinate to *RootUIElementContainer*.

Container Elements and Layout Managers

Container elements are UI elements that may have child elements. They occupy a rectangular area in a view's layout. All UI elements that are children of a container element are located in this rectangular area. All container elements define how their children will be arranged. This is done by evaluating the *Layout* property, which

assigns a **layout manager** to the UI element. All child elements of a container UI element inherit a set of properties that are related to the value of the container's *Layout* property section *Layout Data*. The *Layout* property may have four values:

- *FlowLayout*
- *RowLayout*
- *MatrixLayout*
- *GridLayout*

To demonstrate the differences between the layout managers, start the demo NET310_UI_D1. Resize your browser window.

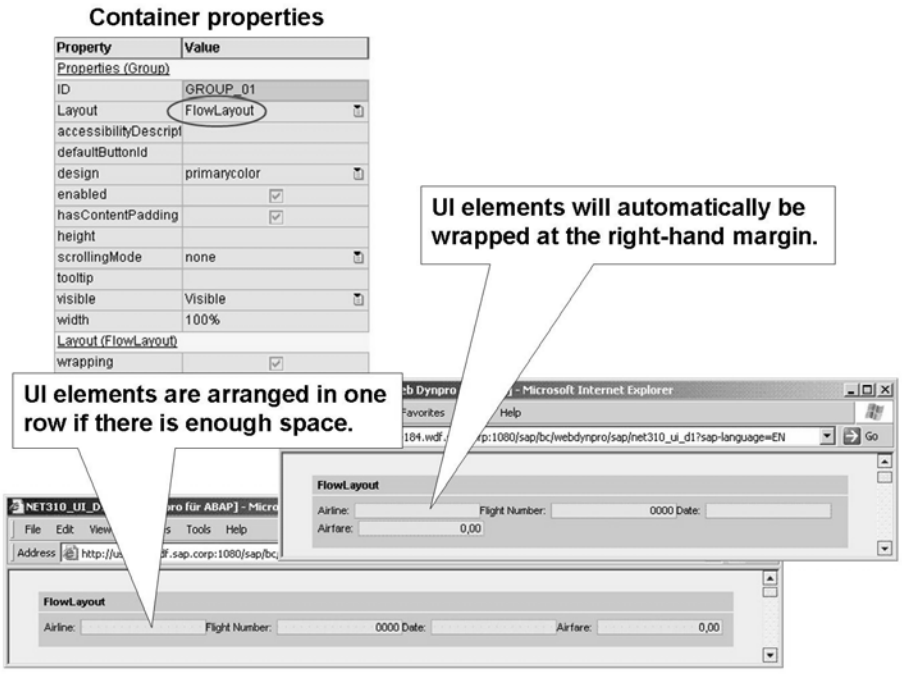


Figure 169: Layout Managers:FlowLayout

The default layout manager is the *FlowLayout* layout manager. All child attributes of this container will be displayed in a row, as long as the container is wide enough. If the container UI element is too narrow for all child elements to be displayed in one row (for example, the browser window is narrowed), they will be automatically

wrapped to the next line(s). This wrapping cannot be forced at design time. Elements in different lines are not related to each other. This kind of container can be used to arrange sub-containers.

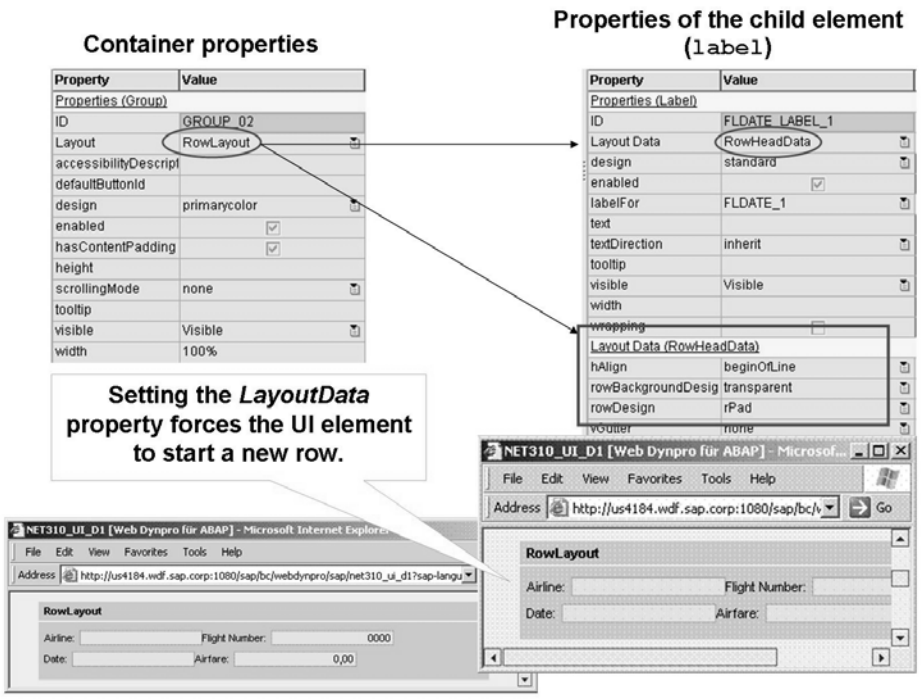


Figure 170: Layout Managers: RowLayout

If the *RowLayout* layout manager is used with the container UI element, all children inherit the property *LayoutData*, which can have the values *RowData* and *RowHeadData*. By setting this property to *RowHeadData*, a line break is forced. If you set the property to *RowData*, this child element will appear in the same line as the previous element, even if the right-hand margin is reached. UI elements located in different rows are not linked to each other and thus are not aligned in columns.

The width of each cell can be set by the *width* attribute of each child element.

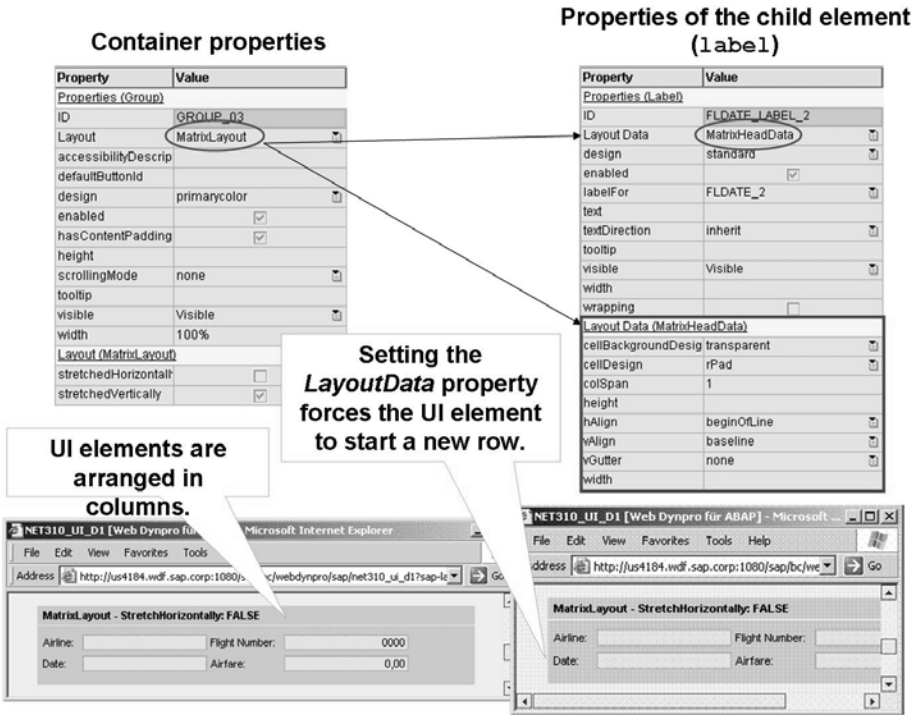


Figure 171: Layout Managers: MatrixLayout

If the *MatrixLayout* layout manager is used with the container UI element, all children inherit the property *LayoutData*, which can have the values *MatrixData* and *MatrixHeadData*. By setting this property to *MatrixHeadData*, a line break is forced. When the property is set to *MatrixData*, the child elements will appear in the same line as the previous element, even if the right-hand margin is reached. The child elements in this container are arranged in columns. Using this layout manager, the number of columns is not defined statically, but it is defined by the maximum number of child elements in any row. The number of elements in different rows do not have to match.

UI elements arranged in a *MatrixLayout* can occupy multiple cells - *colSpan*(property).

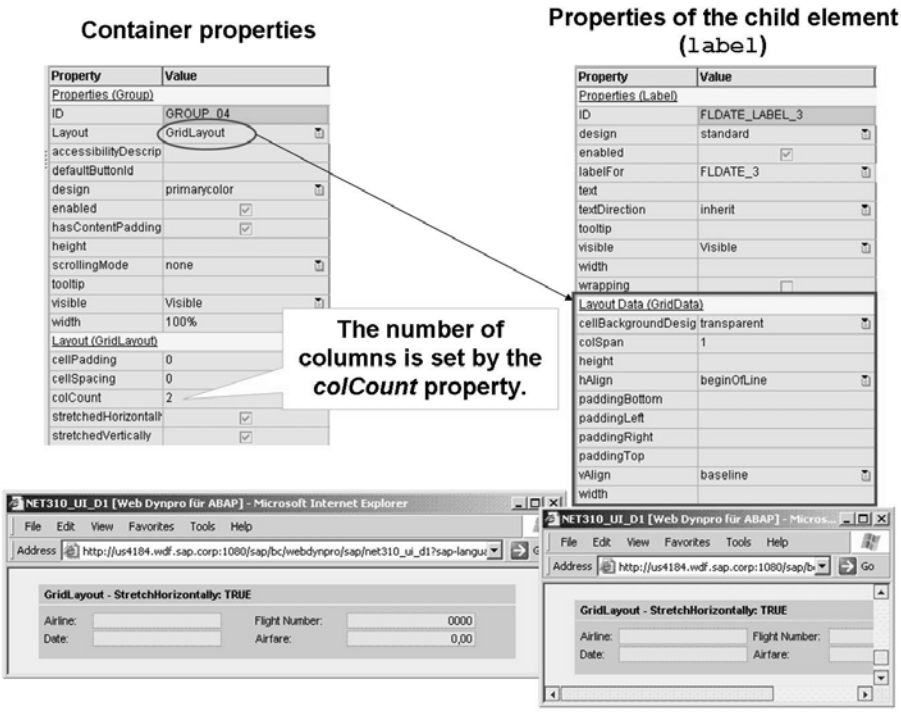


Figure 172: Layout Managers: GridLayout

Like the *MatrixLayout*layout manager, the *GridLayout* layout manager can be used if a vertical alignment of the elements is desired. However, here the number of columns is defined via the *colCount* property of the container element. Thus, the single child element does not control whether it is the first element of a new row. A line break will take place when all cells of a row are occupied. If an element is removed from the hierarchy, the complete arrangement will change since all elements following in the hierarchy will move as many cells to the left as were occupied by the removed element.

This kind of layout manager should be used if all rows occupy the same number of columns and if only complete rows are inserted or deleted. When using this layout manager, UI elements should not be removed completely but replaced by an *InvisibleElement* in order to retain the original element arrangement.

Your component: Change the layout manager of the ROOTUIELEMENTCONTAINER in one of the views. Rearrange the UI elements.

Adding UI Elements to the Layout

The **View Editor** is a Web-Dynpro-specific tool that allows you to edit a view layout. The View Editor is only available when you are editing a view controller. It will not appear when you edit a custom controller because these controllers have no visual interface.

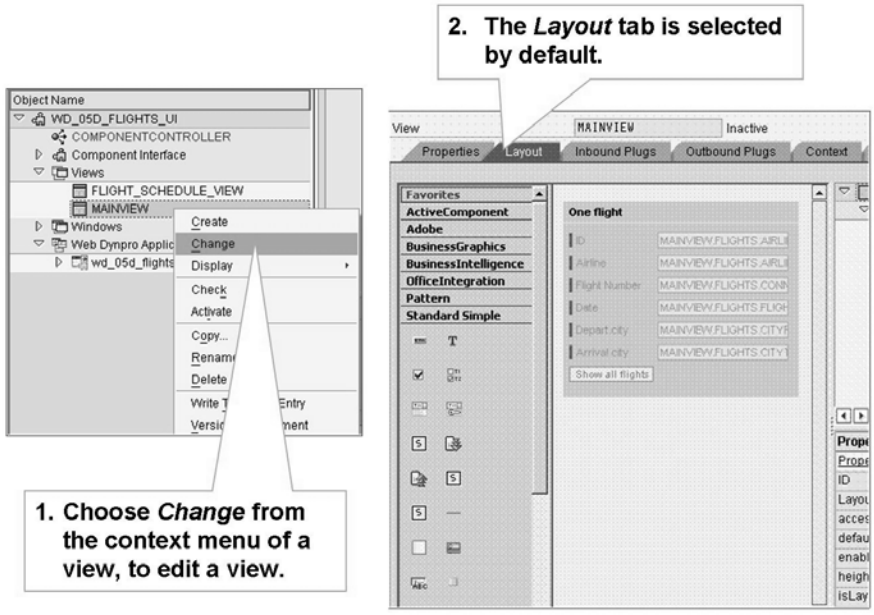


Figure 173: Using the View Editor

The view editor can be used with or without the layout preview. The following description assumes that you work with the layout preview.

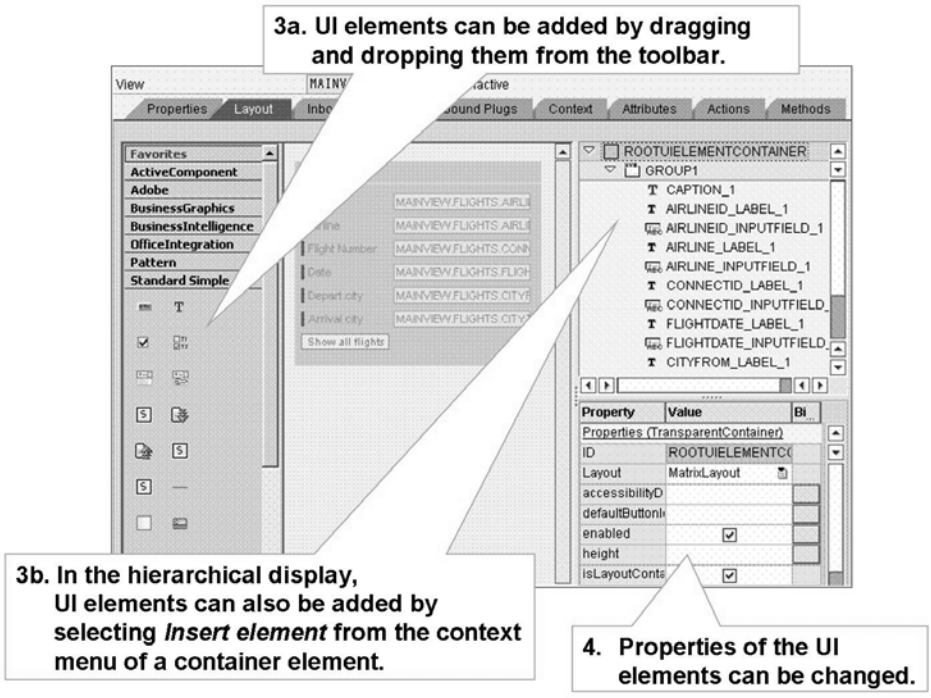


Figure 174: View Editor

To add any UI element, you can drag and drop it from the toolbar on the left-hand side of the View Editor. Adding a new UI element in the hierarchical representation is also possible. This can be done from the context menu of any element in the hierarchy that can have child elements (for example, a *TransparentContainer* element or a *Group* element).

To change the position of a UI element in the element hierarchy, you can move it up or down using the related functionality from the element's context menu. The element can also be moved by dragging and dropping it to the new position in the hierarchical representation or in the layout preview.

The *Properties* tab displays all properties of a selected UI element. If a UI element supports events, the supported client-side events are listed in the *Events* properties section. Properties related to client-side events begin with the prefix *on* (for example, *onFilter*, *onSort*, or *onAction*). To handle these client-side events, **actions** have to be associated with each of the events.

Hint: Client-side events are events related to UI elements that are predefined by the Web Dynpro framework. It is not possible to handle additional events at the client side (for example, by using JavaScript).

Data Binding

Once a UI element property is bound to a context node or attribute, the context data is used to supply a value to the UI element property. If the UI element property is one that the user can update (such as the value property of an *InputField* UI element), the context is automatically updated with the new value during the next round trip.

Almost all of the properties of a UI element can be bound either to a context node or to a context attribute having the correct data type.

Note: A binding relationship can only exist between the context and UI elements of the **same view controller**.

The Web Dynpro view controller has been constructed in such a way that it is usually possible to have full control over the appearance of the screen layout without ever needing direct access the UI element objects. Any property over which you wish to have programmatic control should be bound to an appropriate context node or attribute.

To control the behavior of UI elements, you should manipulate the context nodes or attributes to which the UI elements are bound.

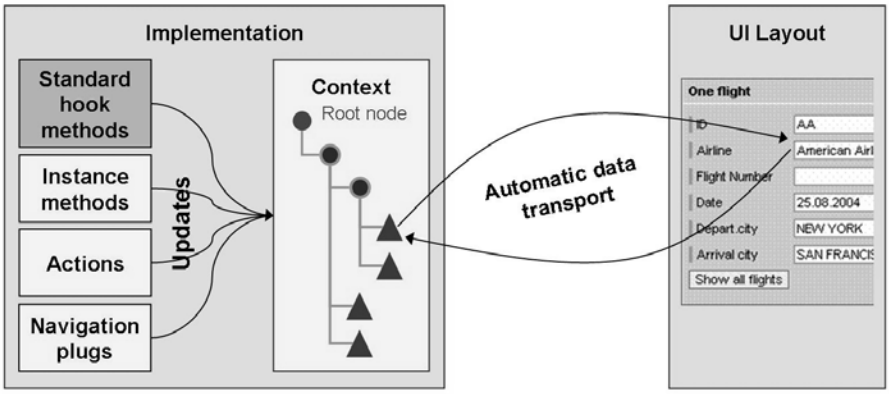


Figure 175: Data Binding

Defining the Data Binding

For a UI element to display any data, at the very least its value property must be bound to an appropriate context node or attribute (there are very few UI elements that do not have a value property – *HorizontalGutter* for instance.

The following steps are the minimum requirement for putting some data on the screen:

1. Create a node or attribute in the view controller's context that will contain the data. It is not important whether this is a mapped context node or not.
2. Create the UI element in your view layout.
3. For all properties requiring a context binding, a button with a yellow icon and an empty circle is displayed on the right hand-side of the property. Assign the required binding by clicking this button. A dialog box will be displayed showing the view controller's context. All nodes or attributes of the appropriate type to be bound to the UI element property are displayed. Select an appropriate node or attribute.

The context path to the node or attribute will then be displayed as the property's value. In addition, the empty circle will be replaced with a green check mark icon. The UI element on the layout preview will also display the context path of the node or attribute to which it is bound.

The establishment of a binding relationship instructs the Web Dynpro screen renderer to obtain the value for a UI element property from the context node or attribute to which it is bound.

Context binding is not limited to simply supplying an *InputField*, for example, with a value. A UI element's value property is just one of the many properties that can be supplied with data through a binding relationship.

This is the mechanism by which a view controller can adjust the appearance and behavior of its view layout without ever needing to access the UI element objects themselves.

It is to early to explain how UI element properties can be changed from the controller source code. However, binding the properties to context attributes can be shown.

You can use the demo NET310_UI_D2 to show how to change the UI element properties.

Your component: In this context, you should explain that depending on the property, the context attribute has to have a certain type. Show how to type a context attribute accordingly (property *visible* of a *Label* and/or an *InputField*). Add a context node containing such attributes and bind the corresponding UI element properties.

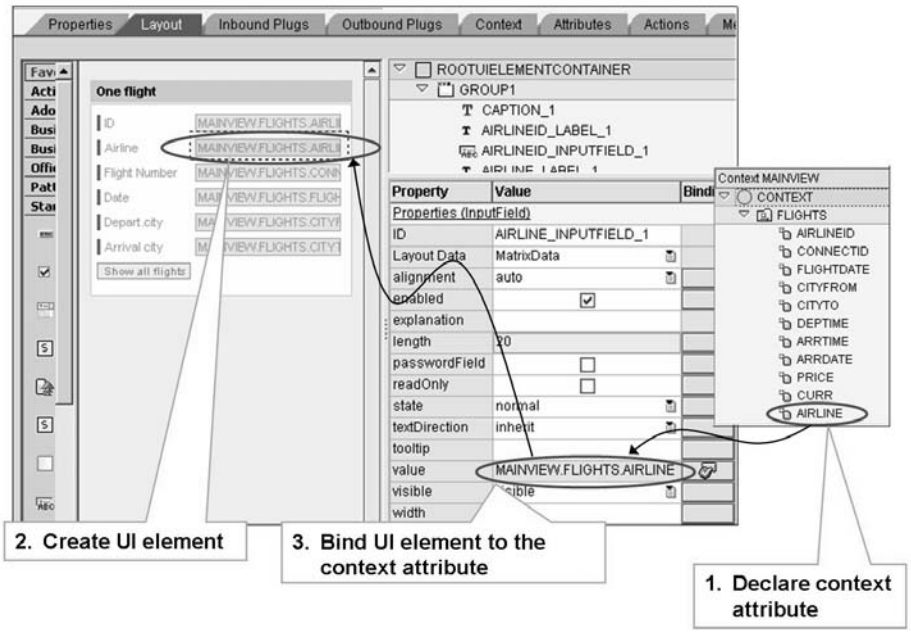
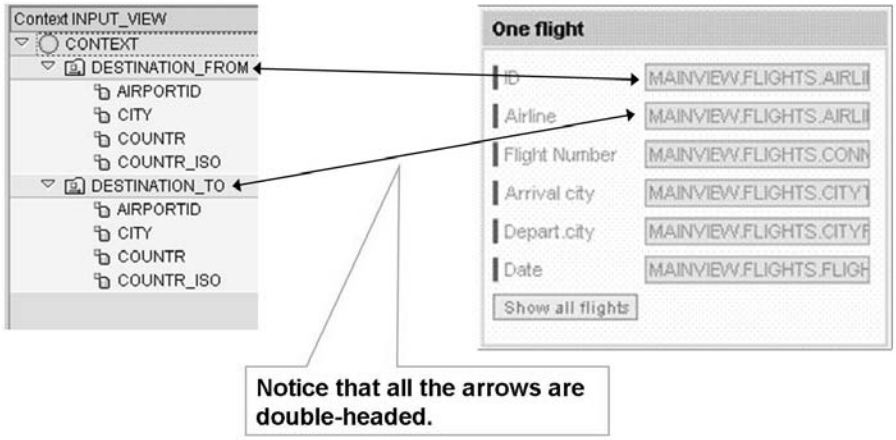


Figure 176: Putting Data on the Screen (1)



The binding between a UI element and a context attribute is a *two-way* relationship:

- **Data from the context is transported to the client during screen rendering.**
- **Data entered by the user is transported back to the context when the HTTP round trip is processed.**

Figure 177: Putting Data on the Screen (2)

As you can see in the figure above, there is no requirement to have the UI elements in the same order as the context attributes to which they are bound.

Data binding is a **two-way** relationship.

Once a binding relationship has been declared, the data in the bound nodes and attributes are transported automatically to the corresponding UI elements. After the user has interacted with the screen and initiated an HTTP round trip, the new or modified data in the UI elements is transported back to the same nodes and attributes in the view controller's context. By the time the Web Dynpro framework hands over control to your action handler, the context already holds the updated information.

This two-way transport process is entirely automatic, and requires no action on the part of the application developer. All you need to do is declare a binding relationship.

Controlling the UI Element Behavior

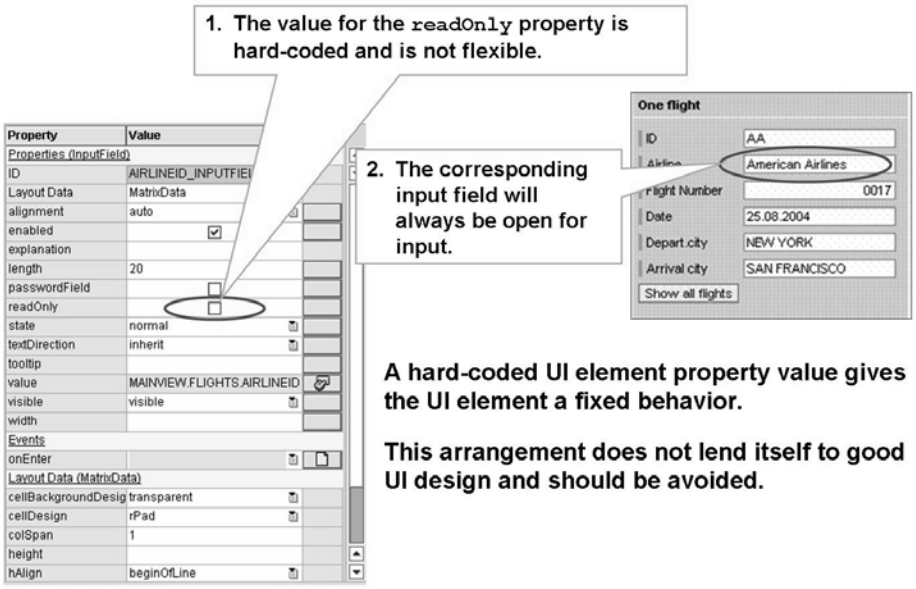


Figure 178: Defining UI Element Properties Statically

The value of a UI element property can either be hard-coded or it can be bound to a context attribute of a suitable data type. If a property value is hard-coded at design time, it can only be changed at runtime by accessing the UI element directly from the view controller's source code. This can only be done from the hook method *wddomodifyview( )*, since only this method provides a reference to the UI element hierarchy. However, accessing the UI element hierarchy directly from a controller's method is considered poor design because the separation between flow logic and UI is no longer guaranteed. This technique should be avoided.

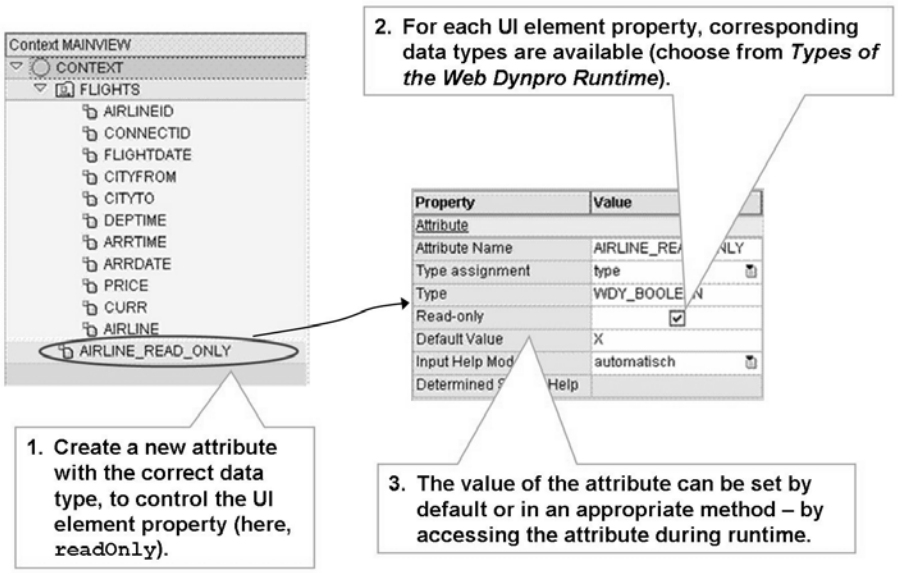


Figure 179: Controlling UI Element Properties (1)

In order to programmatically control the behavior of a UI element, you should create a context attribute having a data type that fits the property you wish to control. This allows you to control the behavior of the UI element by modifying the related attribute value in any method of any controller that has access to this context attribute. Directly accessing the UI element object from the controller source code is no longer necessary.

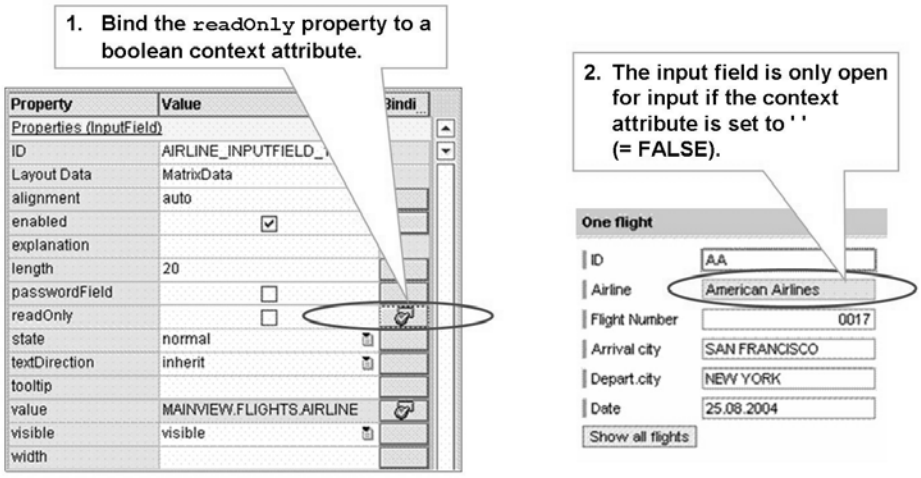


Figure 180: Controlling UI Element Properties (2)

Once the context attribute has been created, it must be bound to the appropriate UI element property. In this case, the *readOnly* property of an *InputField* UI element has been bound to a Boolean value attribute.

The value of the context attribute can now be manipulated from any controller hook method, or from your instance methods or action handler methods.

This technique is applicable to most of the UI element properties. Appropriate data types can be found using the *Types of the Web Dynpro Runtime* tab on the dialog box displayed when typing a context attribute.

Texts from the ABAP Dictionary

Many UI elements (*TextViews*, *Labels*, *Captions*, and so on) display texts in the rendered UI. These texts can be obtained from the ABAP Dictionary in two ways:

- The property related to the text is bound to a context attribute, which itself is typed with a data element defined in the ABAP Dictionary.

  Example: The *Text* property of a *Button* is bound to a context attribute. The button text then originates from the data element related to the context attribute.

- The UI element is related to a second UI element, and this second element is bound to a context element that is typed with a data element. In this case, the property related to the text must be left blank in order to use the dictionary text.

  Example: A *Label* is related to an *InputField* and the *Text* property of the *Label* is left blank. The label text then originates from the data element related to the *InputField*.

Your component: Show, how to use DDIC texts. Get the texts related to the *text* property of some view elements (for example, the *Labels*) from the DDIC (data elements or structure fields).

Test Page for UI Elements

In each system based on SAP NetWeaver 2004s, the Web Dynpro application *WDR_TEST_UI_ELEMENTS* is delivered. This can be used to check each UI element's functions.

The Web Dynpro test application `wdr_test_ui_elements` allows you to investigate and check out the function of the available UI elements.

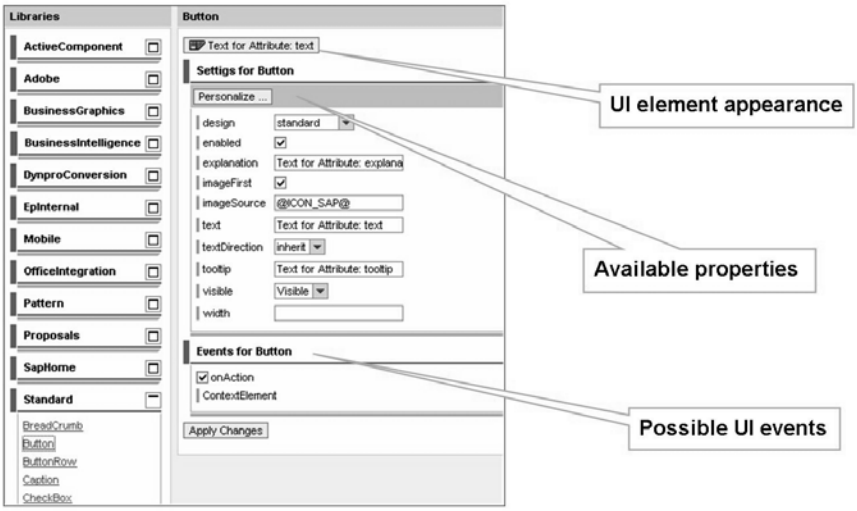


Figure 181: Test Page for UI Elements

Here you should ask the students, if they know the MIME Repository. If this is not the case, give a brief introduction. This should comprise the following issues:

Explain that the path visible in the MIME Repository (for example, sap/bc/webdynpro/sap/public/NET310/explosion.gif) corresponds to the URL path for accessing the MIME object. Check it out by using this path as part of the URL in the browser.

Use existing MIME objects in Web Dynpro (for example, drag and drop an image from the MIME Repository to an Image UI element in the UI element hierarchy).

Add a MIME object to your component using the context menu for your component. After importing the MIME object, switch to the MIME Repository and localize the object (sap/bc/webdynpro/sap/z....).

Exercise “Displaying MIME Repository Objects”

Composite UI Elements

Certain UI elements are displayed on the screen as aggregations of simpler, more basic, UI elements. The *Table* UI element is a good example.

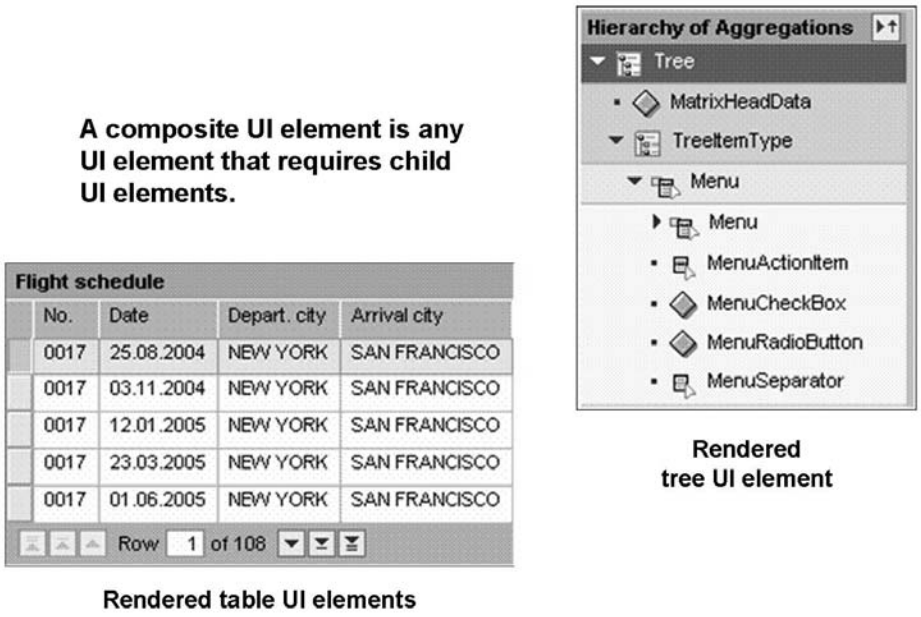


Figure 182: Composite UI Elements (1)

Without the subordinate, or child, UI elements, a composite UI element is not capable of displaying any information. Composite UI elements such as *Group* and *Tray* have a mandatory *Caption* child UI element, but beyond that, their structure is entirely user defined. Composite UI elements such as *Table* and *Tree*, however, require a more complex mandatory child structure.

Composite UI elements are incapable of displaying information on their own.

They must have child UI elements, to function correctly.

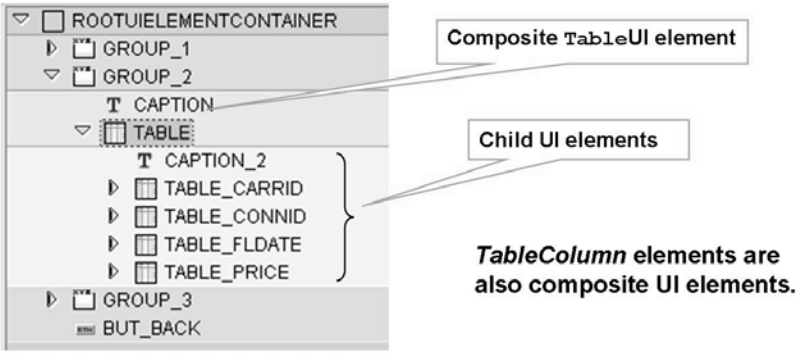


Figure 183: Composite UI Elements (2)

The *Table* UI Element

The *Table* UI element acts as a parent of several *TableColumn* UI elements. Each of these elements in turn acts as the parent for a header (implemented as a *Caption* UI element) and a cell editor. **Cell editor** is an abstract expression for all kinds of UI elements that can serve as cell elements in a given column. The default for the cell editor is the *TextView* UI element. However, depending on the cell value and the necessity that the cell value should be changeable, other UI elements can be used as cell editors (for example, *InputField*, *DropDownByKey*, *Checkbox*, or *Button*).

Show Demo NET310_UI_D3 here. Focus on: UI element hierarchy in view layout.

The `Table`UI-Element is an example of a composite UI element.

The child elements of a `Table` UI element are also composite.

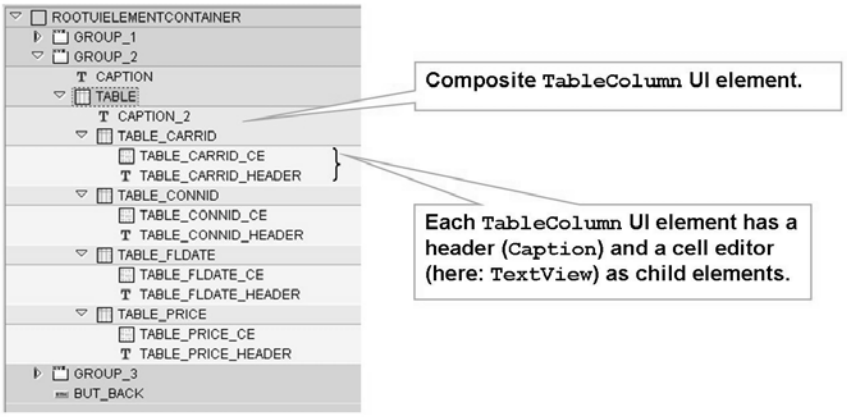


Figure 184: The Table UI Element

Binding a *Table* to the Context

The *Table* UI element allows a two-dimensional display of data in cells arranged in rows and columns. The UI element consists of an optional header area, zero or more rows, and a footer area. The *Table* UI element must be bound to a context node of cardinality **0..n** or **1..n**. The element at the lead selection in the context node becomes the highlighted row when displayed on the screen (this does not depend on having a selection column).

The *Table* UI element with its child elements can be created using the Web Dynpro Code Wizard. In this case defining the complete context binding is part of the wizard process. However, if the *Table* UI element has been added to the element hierarchy manually, the context menu entry *Create Binding* should be used to create the complete binding.

The caption appearing at the top of the table (*Flights* in the figure below) is optional.

Show Demo NET310_UI_D3 here. Focus on: Data binding of TABLE UI element.

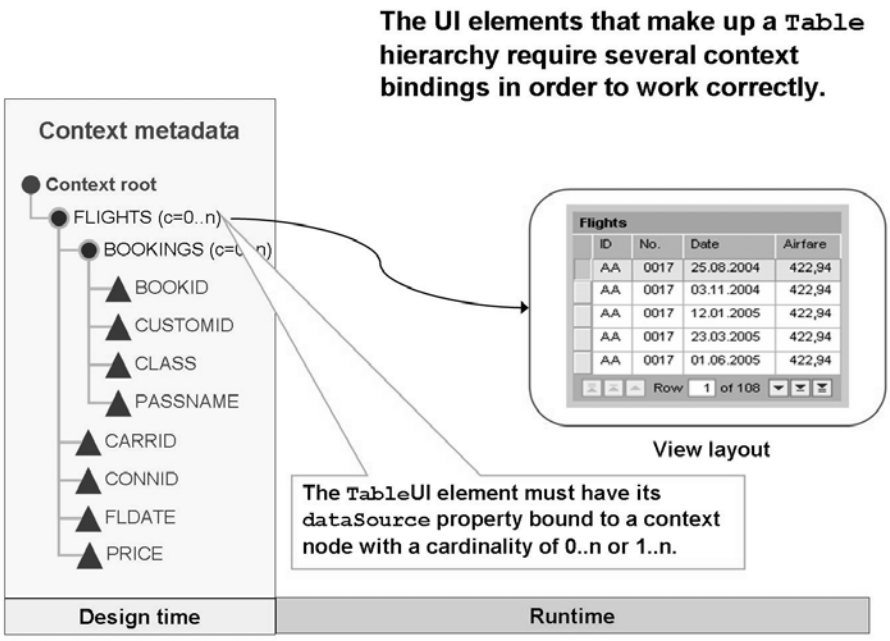


Figure 185: Binding a Table UI Element to the Context

TableColumn UI Elements

The *Table* UI element must contain at least one *TableColumn* UI element. *TableColumn* UI elements should be bound to attributes of the same node to which the parent *Table* UI element is bound. Any node attribute of a simple data type is a candidate for becoming a table column.

Hint: A *TableColumn* UI element can be bound to an attribute that is not a child of the node to which the *Table* UI element is bound. In this case, the values of all cells in the column are equal.

TableColumn UI elements are composite UI elements. This means that they must have child UI elements in order to function correctly.

The column header is created using a *Caption* element. The text displayed in the column header can be obtained from the ABAP Dictionary if the *Text* property of the *Caption* UI element is left blank and if the context attribute displayed in this column is typed with an ABAP Dictionary data element.

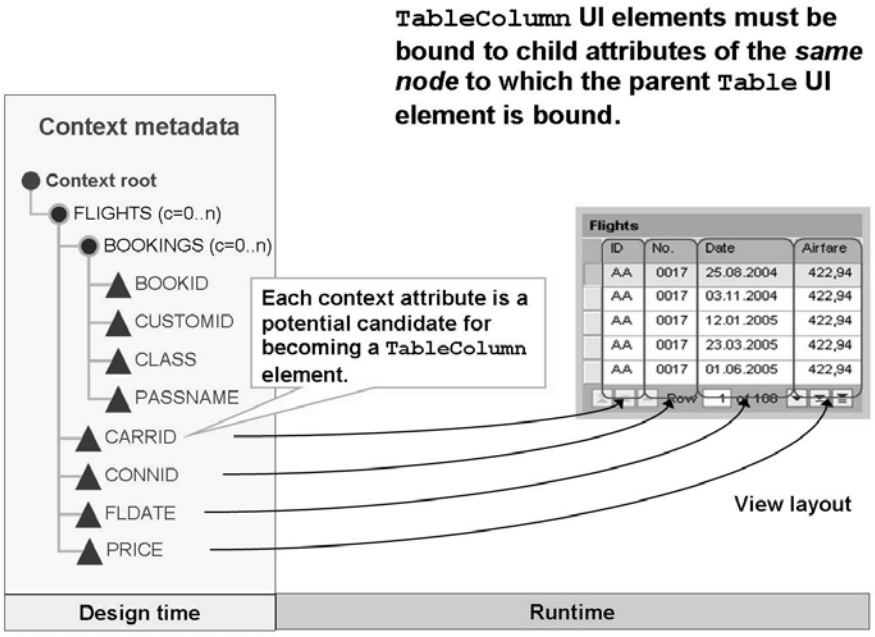


Figure 186: TableColumn UI elements

Child UI Elements of a *TableColumn*

To present information to the user, a *TableColumn* UI element must have a child UI element that will act as the **cell editor**. In order to decide which UI element to use, the kind of interaction between the user and the data in each column has to be clarified.

Do not think that because the name of this role contains the word “editor” that the user can necessarily change the data. If you select some sort of display-only UI element such as a *TextView*, then the user will not be able to modify the data; the UI element you chose does not allow it. If you choose a *Button*, then the cell value is displayed as the button’s text. Here, users cannot input data, but they can fire a client event. If, on the other hand, you chose an *InputField* to be the table cell editor, then all the cells appearing in that column would, by default, be open for input.

The caption appearing as the column header is optional, but if defined will always be of type *Caption*.

Show Demo NET310_UI_D3 here. Focus on: Data binding of cell editors.

Your component: Now you should show how to create a table. Gather data by creating a service call in the component controller (this has to be explained anyway before the related exercise can be solved. Be sure you know which BAPI you want to call. Copy the generated context node containing the mass data to the start view and map the node. Use the WD Code Wizard to create the table display. Call the service method from WDDOINIT of the view. If necessary provide data statically.

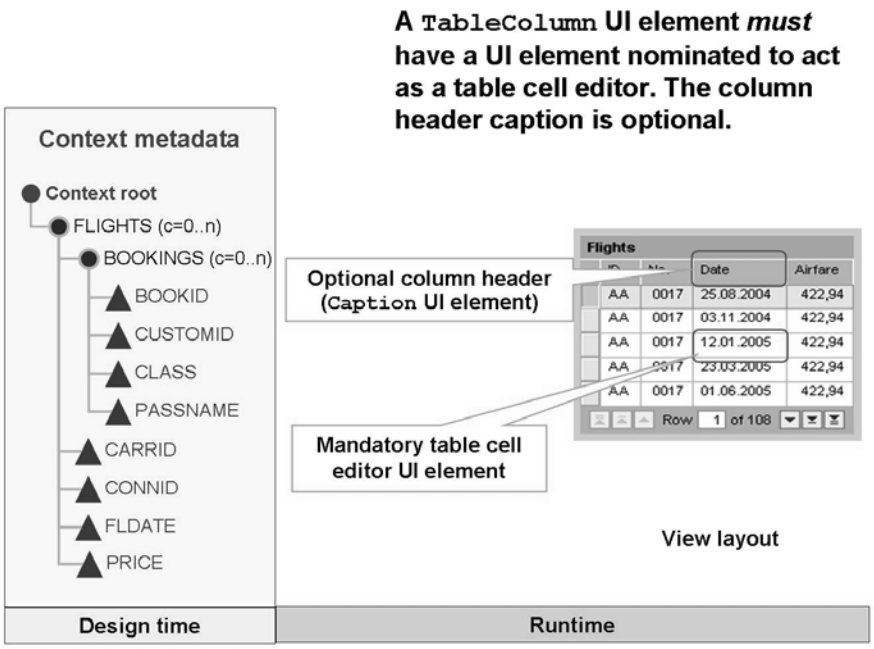


Figure 187: Defining Child UI Elements for a TableColumn

Selecting a Table Row and the Effect on the Node's Lead Selection

When you click on the selection button of a table row on the rendered screen, a round trip is initiated. This round trip will cause the lead selection of the corresponding context node to which this table is bound to be adjusted.

In the figure below, the node collection contains two elements, and the user has clicked on the second table row. This corresponds to element 2 of the FLIGHTS node, and the lead selection of this node is altered to reflect the user's selection.

There will be as many rows in the table as there are elements in the node collection.

You can also define a Web Dynpro action and associate it with the table's *onLeadSelect* event. Then, not only will the node collection's lead selection element be adjusted when the user clicks on a table, but the hook method *wddobeforeaction( )* and the corresponding action handler method of the view containing the *TableView* UI element will be processed.

Show Demo NET310_UI_D3 here. Focus on: Selecting a row sets the lead selection (displayed in the view in the input field).

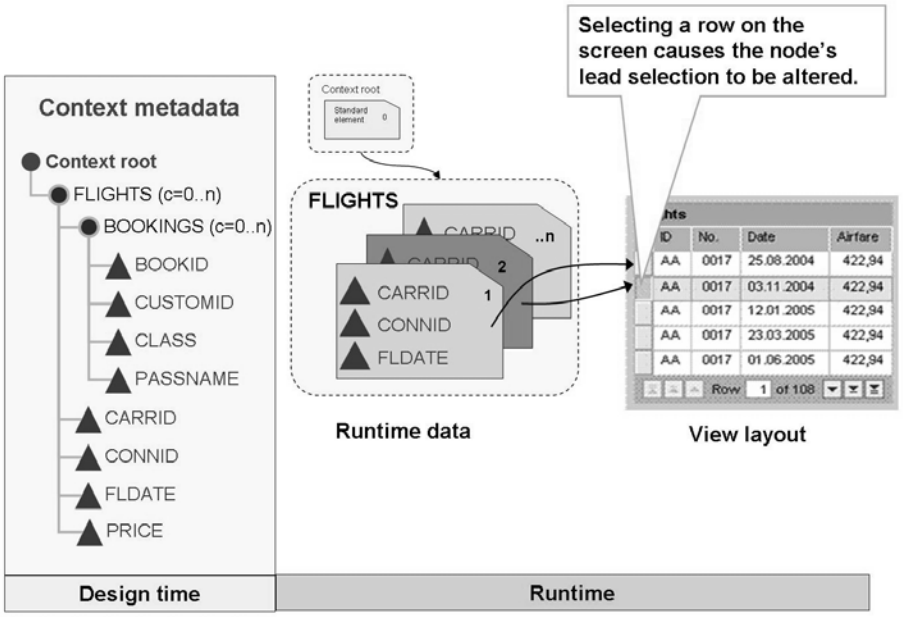


Figure 188: Table Row Selection

Multiple Selection of Rows from a Table

Before a user is permitted to select multiple rows from a table (by holding down **SHIFT** or **CTRL**, and clicking on the required rows), the context node to which the *Table* UI element is bound must have a **selection cardinality** of either **0..n** or **1..n**. The default selection cardinality setting for any context node is **0..1**, meaning that zero or one element may be selected at any one time.

Show Demo NET310_UI_D3 here. Focus on: Selecting multiple rows (this is allowed, since the selection cardinality is set to 0...n for this demo). Choose CTRL and select some lines. Fire the request using the button above the table. The lead selection does not change.

Each node provides the method *get_selected_elements( )*, which returns all selected elements in an internal table (type *WDR_CONTEXT_ELEMENT_SET*).

Note: A table line in a view that is selected with the lead selection will be only part of the result of the method *get_selected_elements( )* if the *INCLUDING_LEAD_SELECTION* parameter of the method is set to true.

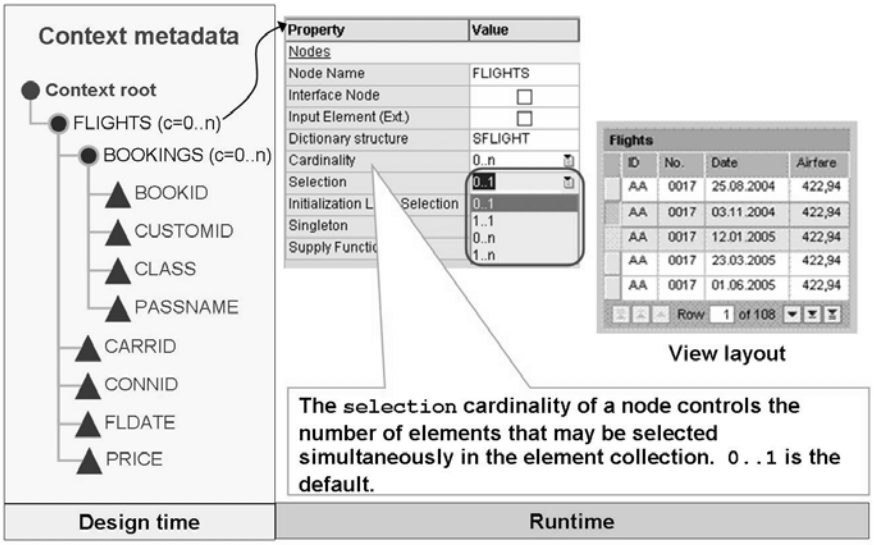


Figure 189: Multiple Selection of Rows from a Table

The Table UI Element Property *SelectionMode*

The topics discussed in the last two sections above are only true for the default value of the *Table* property *selectionMode* (*auto*). However, this property can have six values, which influences the possibility to select rows and set the lead selection as follows:

Show Demo NET310_UI_D3 here. Focus on: Choose different settings for the Selection Mode and verify the influence on choosing rows and setting lead selection as described below.

Other interesting demos:

NET310_UI_D4 shows that selecting a row leads to an HTTP round trip. However, there is no client-side event handling involved, so the related hook methods in the view controller are not processed (table on bottom). If the client event of selecting a row is bound to an action, the related hook methods are processed (table on top).

The Property SelectionMode

Value	Minimum of Selected Rows	Maximum of Selected Rows	Selecting Table Row Triggers Round Trip	Selecting Table Row Sets Lead Selection
auto	Selection cardinality	Selection cardinality	X	X
Single	Selection cardinality	1	X	X
multi	Selection cardinality	Selection cardinality	X	X
none	0	0	-	-
singleNoLead	Selection cardinality	1	-	-
multiNoLead	Selection cardinality	Selection cardinality	-	-

Exercise “Displaying Tables”

307

Exercise 16: User Interface: Displaying MIME Repository Objects

Exercise Duration: 20 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Create UI elements of the type TRAY
- Create UI elements of the type IMAGE
- Display graphic files stored in the MIME Repository

Business Example

You want to develop a Web Dynpro application with a view that includes the display of a graphics file. The graphics file is stored in the MIME Repository.

Template: None

Solution: NET310_UI_S1

Task:

Create a Web Dynpro component. Create one view and embed it into the main window of the component. On the view layout, create a UI element of type TRAY and, on the tray, a UI element of type IMAGE that displays file *explosion.gif* from the MIME Repository folder *SAP → BC → WebDynpro → SAP → PUBLIC → NET310*.

1. Create Web Dynpro component **ZNET310_UI1_##** with one view embedded into the main window.
2. On the view, create a UI element of type TRAY (suggested name: **TRAY_1**) and maintain the caption text.
3. On the tray element, create a UI element of the type IMAGE (suggested name: **IMAGE_EXPLOSION**) and maintain the source of the IMAGE element. The source should be file *explosion.gif* from MIME Repository folder *SAP → BC → WebDynpro → SAP → PUBLIC → NET310*.
4. Activate your component. Create a Web Dynpro application and test it.

Solution 16: User Interface: Displaying MIME Repository Objects

Task:

Create a Web Dynpro component. Create one view and embed it into the main window of the component. On the view layout, create a UI element of type TRAY and, on the tray, a UI element of type IMAGE that displays file *explosion.gif* from the MIME Repository folder *SAP → BC → WebDynpro → SAP → PUBLIC → NET310*.

1. Create Web Dynpro component **ZNET310_UI1_##** with one view embedded into the main window.

   a) Perform this step as before.

2. On the view, create a UI element of type TRAY (suggested name: **TRAY_1**) and maintain the caption text.

   a) Open the context menu for ROOTUIELEMENTCONTAINER and choose *Insert Element*.

   b) Enter the name of the UI element and choose the element type *TRAY*.

   c) Maintain the caption in the *text* property of the sub-element *CAPTION*.

3. On the tray element, create a UI element of the type IMAGE (suggested name: **IMAGE_EXPLOSION**) and maintain the source of the IMAGE element. The source should be file *explosion.gif* from MIME Repository folder *SAP → BC → WebDynpro → SAP → PUBLIC → NET310*.

   a) Open the context menu for TRAY_1 and choose *Insert Element*.

   b) Enter the name of the IMAGE element and choose the element type *IMAGE*.

   c) Open the MIME Repository (choose the *MIME Repository* button at the top of the navigation area of the ABAP Workbench).

   d) Open the specified folder. Drag *explosion.gif* and drop it on the UI element IMAGE_EXPLOSION in the UI element hierarchy.

4. Activate your component. Create a Web Dynpro application and test it.

   a) Perform this step as in previous exercises.

309

Exercise 17: User Interface: Displaying Tables

Exercise Duration: 35 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Create a service call to call a function module and bind the parameters to the controller context
- Create a UI element of the type TABLE, to display mass data

Business Example

Template: None

Solution: NET310_UI_S2

Task 1:

Create a Web Dynpro component with two views (input view and display view) embedded into the main window.

1. Create the Web Dynpro component **ZNET310_UI2_##** with two views (suggested names: **INPUT_VIEW** and **DISPLAY_VIEW**) and embed the views into the main window.

Task 2:

Create a service call of function module BAPI_FLIGHT_GETLIST in the component controller. This will create a method encapsulating the BAPI call. In addition, the BAPI's interface parameters will be bound to the component controller context.

1. Create a service call in the existing component controller for the function module BAPI_FLIGHT_GETLIST.

   When asked for the parameters to be stored in the controller's context, select DESTINATION_FROM, DESTINATION_TO, and FLIGHTLIST.

   When asked for the name of the service method, accept the default name (EXECUTE_BAPI_FLIGHT_GETLIST).

Continued on next page

2. What entities of the component have been created?

3. What are the cardinalities of nodes DESTINATION_FROM and FLIGHTLIST? Why are they different?

Task 3:

On the input view layout, create a group with input fields for departure city and destination city. To bind the input fields to view context attributes, copy the relevant nodes of the component context to the context of the input view and define the context mapping.

1. Copy the nodes DESTINATION_FROM and DESTINATION_TO from the component context to the context of the input view and define the mapping between the context nodes of the different controllers.
2. On the view layout, use the Web Dynpro Code Wizard to create a form that provides an input field that corresponds to the attribute CITY of the context node DESTINATION_FROM. The wizard will also create a group UI element embedding the form fields. Adjust the text to be displayed by the UI element CITY_LABEL.
3. Repeat the last step for the attribute CITY of the context node DESTINATION_TO. This will also create a second group. Adjust the text to be displayed by the UI element CITY_LABEL_1.

   Move all labels and input fields to one of the groups and delete the second group.
4. Adjust the properties of the UI elements to make sure that the input fields (with their labels) are displayed one below the other.

Continued on next page

Task 4:

Copy the relevant nodes of the component context to the context of the display view and map the context nodes of the different controllers. Use the Web Dynpro Code Wizard to create a table that displays the flight info on the display view.

1. Copy node FLIGHTLIST from the component context to the context of the display view and map the two nodes.
2. Use the Web Dynpro Code Wizard to create a table display with binding to the context node FLIGHTLIST.

Task 5:

Provide a button on each view and implement navigation between the two views.

1. Create an inbound plug and an outbound plug for each of the two views.
2. Create navigation links between the plugs of the views (both directions).
3. Provide a button on each of the views. Create actions related to these buttons and make sure the respective outbound plug is fired when the user selects the button.

Task 6:

Use the Web Dynpro Code Wizard to implement a call of the service method EXECUTE_BAPI_FLIGHT_GETLIST in the event handler for the inbound plug of the display view.

1. In the display view, implement the method to handle the inbound plug (method with the name HANDLE...). Start the Web Dynpro Code Wizard to implement a call of method EXECUTE_BAPI_FLIGHT_GETLIST of the component controller (method call In used controller).
2. Activate your component. Create a Web Dynpro application and test it.

Solution 17: User Interface: Displaying Tables

Task 1:

Create a Web Dynpro component with two views (input view and display view) embedded into the main window.

1. Create the Web Dynpro component **ZNET310_UI2_##** with two views (suggested names: **INPUT_VIEW** and **DISPLAY_VIEW**) and embed the views into the main window.

   a) Perform this step as in previous exercises.

Task 2:

Create a service call of function module BAPI_FLIGHT_GETLIST in the component controller. This will create a method encapsulating the BAPI call. In addition, the BAPI's interface parameters will be bound to the component controller context.

1. Create a service call in the existing component controller for the function module BAPI_FLIGHT_GETLIST.

   When asked for the parameters to be stored in the controller's context, select DESTINATION_FROM, DESTINATION_TO, and FLIGHTLIST.

Continued on next page

When asked for the name of the service method, accept the default name (EXECUTE_BAPI_FLIGHT_GETLIST).

a) Open the context menu for the Web Dynpro component in the navigation area of the ABAP Workbench (left-hand side) and choose *Create* → *Service Call*.

b) Choose *Continue*, to confirm the first window of the wizard.

c) Select *Use Existing Controller*, enter the component controller of your component, and choose *Continue*.

d) Select *Function Module* and choose *Continue*.

e) On the next screen, enter the name of the function module in the *Function Module* field and choose *Continue*.

f) Check the parameters to be stored in the controller's context and choose *Continue*.

g) Enter the name of the service method and a description and choose *Continue*.

2. What entities of the component have been created?

Answer:

- Context node BAPI_FLIGHT_GETLIST with several subnodes for the selected IMPORTING and TABLES parameters of the function module
- Method EXECUTE_BAPI_FLIGHT_GETLIST in the component controller, which encapsulates the call of the function module

3. What are the cardinalities of nodes DESTINATION_FROM and FLIGHTLIST? Why are they different?

Answer:

- DESTINATION_FROM: 1...1
- FLIGHTLIST: 0...n

The parameter DESTINATION_FROM is a structure. The parameter FLIGHTLIST is an internal table.

Continued on next page

Task 3:

On the input view layout, create a group with input fields for departure city and destination city. To bind the input fields to view context attributes, copy the relevant nodes of the component context to the context of the input view and define the context mapping.

1. Copy the nodes DESTINATION_FROM and DESTINATION_TO from the component context to the context of the input view and define the mapping between the context nodes of the different controllers.

   a) Perform this step as in previous exercises.

2. On the view layout, use the Web Dynpro Code Wizard to create a form that provides an input field that corresponds to the attribute CITY of the context node DESTINATION_FROM. The wizard will also create a group UI element embedding the form fields. Adjust the text to be displayed by the UI element CITY_LABEL.

   a) Perform this step as in previous exercises.

3. Repeat the last step for the attribute CITY of the context node DESTINATION_TO. This will also create a second group. Adjust the text to be displayed by the UI element CITY_LABEL_1.

   Move all labels and input fields to one of the groups and delete the second group.

   a) Create the group as before. Move all elements from this group to the first group by dragging and dropping them in the UI element hierarchy.

4. Adjust the properties of the UI elements to make sure that the input fields (with their labels) are displayed one below the other.

   a) Set the *Layout* property of ROOTUIELEMENTCONTAINER to *MatrixLayout*.

   b) Set the *Layoutdata* property of the UI Element group to *MatrixHeadData*.

   c) Set the *Layoutdata* property of CITY_LABEL to *MatrixHeadData*.

   d) Set the *Layoutdata* property of CITY_LABEL_1 to *MatrixHeadData*.

Continued on next page

Task 4:

Copy the relevant nodes of the component context to the context of the display view and map the context nodes of the different controllers. Use the Web Dynpro Code Wizard to create a table that displays the flight info on the display view.

1. Copy node FLIGHTLIST from the component context to the context of the display view and map the two nodes.

   a) Perform this step as in previous exercises.

2. Use the Web Dynpro Code Wizard to create a table display with binding to the context node FLIGHTLIST.

   a) Place the cursor on **ROOTUIELEMENTCONTAINER**, press the *WebDynpro Code Wizard* button, and choose *Standard* → *Table*.

   b) Choose *Context* and double-click the context node **FLIGHTLIST**.

   c) Accept the default (standard cell editor *TextView* and binding for all fields) and choose .

Task 5:

Provide a button on each view and implement navigation between the two views.

1. Create an inbound plug and an outbound plug for each of the two views.

   a) Perform this step as in previous exercises.

2. Create navigation links between the plugs of the views (both directions).

   a) Perform this step as in previous exercises.

3. Provide a button on each of the views. Create actions related to these buttons and make sure the respective outbound plug is fired when the user selects the button.

   a) Perform this step as in previous exercises.

Continued on next page

Task 6:

Use the Web Dynpro Code Wizard to implement a call of the service method EXECUTE_BAPI_FLIGHT_GETLIST in the event handler for the inbound plug of the display view.

1. In the display view, implement the method to handle the inbound plug (method with the name HANDLE...). Start the Web Dynpro Code Wizard to implement a call of method EXECUTE_BAPI_FLIGHT_GETLIST of the component controller (method call In used controller).

   a) Open the method in the editor. Choose *Web Dynpro Code Wizard* and then *Method Call in Used Controller*.

   b) Select the component controller and the method to be called from the input help and choose ✔.

2. Activate your component. Create a Web Dynpro application and test it.

   a) Perform this step as in previous exercises.

Lesson Summary

You should now be able to:

- Define the views layout by arranging UI elements and setting their properties
- Bind UI element properties to context attributes
- Use the table UI element as an example of using composite UI elements

Unit Summary

You should now be able to:

- Define the views layout by arranging UI elements and setting their properties
- Bind UI element properties to context attributes
- Use the table UI element as an example of using composite UI elements

Unit 11


319

Controller and Context Programming

This is the first chapter that deals with the source code of controller methods. ABAP programmers do not usually work with frameworks and they may not be familiar with the need for using framework methods. In particular, programmers may not know how to access the controller's global data (defined in the context), as in the classical ABAP programming context, all data objects are defined using the `data` statement. Here, the data cannot be accessed if the API methods for context accesses are not known.

Therefore, point out the advantages of such an approach: For example, code wizards can be used to define the source code, data binding between UI properties and context fields requires a standard method for storing data, and data sharing (context mapping) between controllers requires standard data storage.

Unit Overview

One of the strengths of Web Dynpro is the declarative metadata approach. This approach allows you to define the complete static UI and the navigation between the different view assemblies without having to program source code. However, defining programming logic and back-end access requires the programming of ABAP source code. The source code is defined in methods of custom controllers, view controllers, or window controllers. This unit deals with controller programming, and focuses on how to access the controller's context.

Unit Objectives

After completing this unit, you will be able to:

- List the hook methods that exist for the different controller types
- Explain in which order these hook methods are processed
- Create and call your own controller methods
- Use the standard controller attributes to access the controller functions, especially the controller context

- Define your own controller attributes and use them in the controller methods
- Access controller context and read, delete, change, or add collection elements

Unit Contents

Lesson: Controller and Context Programming

 320

Lesson Duration: 130 Minutes

Lesson Overview

Defining the source code of controllers requires an understanding of how predefined methods (hook methods) should be used, when and in which order they are processed, how additional user defined methods can be created, and how these methods can be accessed from the same or from another controller. In addition, the definition of events, raising these events, and registering the corresponding event handler on such events must also be understood. Finally, accessing the controller context (reading, changing, and deleting data stored in the controller) is essential to implement controllers. In this lesson, all of these aspects will be discussed in detail.

Lesson Objectives

After completing this lesson, you will be able to:

- List the hook methods that exist for the different controller types
- Explain in which order these hook methods are processed
- Create and call your own controller methods
- Use the standard controller attributes to access the controller functions, especially the controller context
- Define your own controller attributes and use them in the controller methods
- Access controller context and read, delete, change, or add collection elements

References to demos can be found in this lesson.

Business Example

You have created a Web Dynpro component. Up to now, you have worked solely on entities that can be defined in a declarative manner. However, the flow logic (including access to backend logic) needs to be implemented by defining ABAP source code in controller methods. You want to know how to work with controller methods and controller attributes.

Controller Methods

Each Dynpro Controller is a separate ABAP class. The definition of these classes is automatically generated when a new component or controller is created. The source code that implements these controllers is generated automatically. Each controller provides a number of predefined methods that are called from the Web Dynpro runtime in a predefined order. These methods are known as **hook methods**. At the time of controller creation, these methods are empty and can hold any coding the developer wishes to place within them. In addition to these hook methods, the developer can define additional methods: Instance methods, event handler methods, or supply functions.

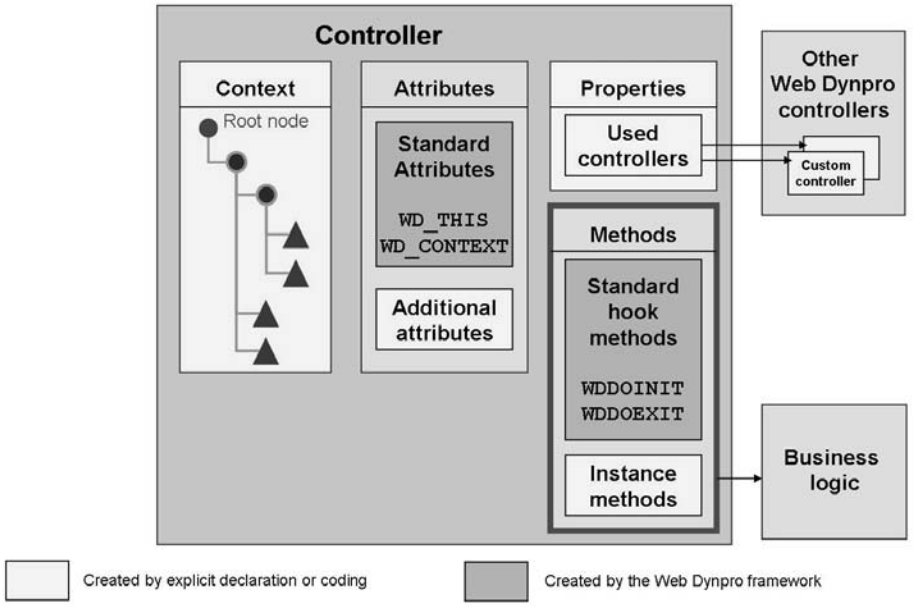


Figure 190: Controller Methods

From all controller methods, hook methods, and user-defined methods, the controller class is generated each time the controller is activated. The ABAP Workbench allows you to enter source code inside the controller methods only; the generated controller class is not accessible.

Hook Methods

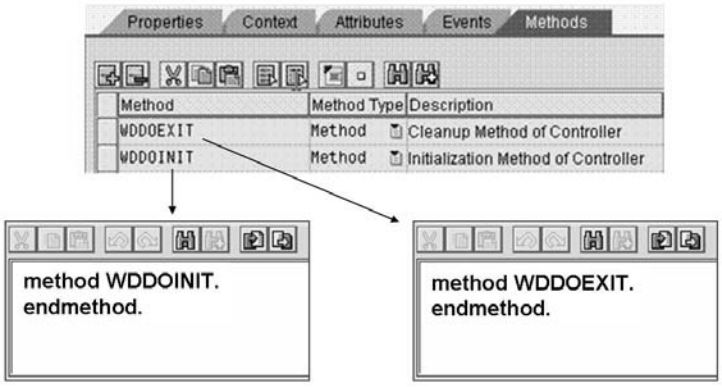
There are two hook methods that exist for all controller types:

- ***wddoinit( )***:

  wddoinit( ) is the first method processed in the controller's lifetime. It is only called once in the controller's life cycle. All your initialization code should go here, since this method is called immediately after the controller has been instantiated.

- ***wddoexit( )***:

  wddoexit( ) is the last method that is processed at the end of a controller's life cycle. All your cleanup code should go here.



- **All controllers have these two standard hook methods.**
- **All methods will only be called during the controller's life cycle if they contain coding.**

Figure 191: Standard Hook Methods for All Controllers

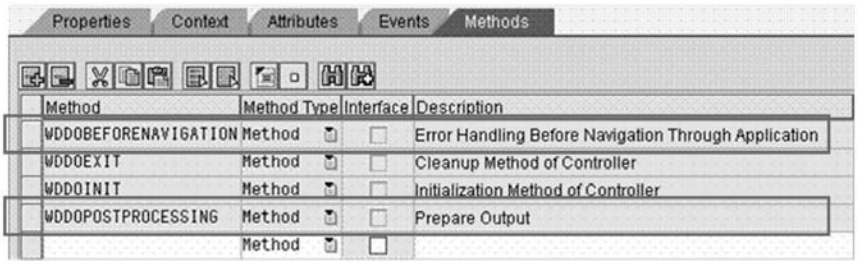
Depending on the controller type, additional hook methods are available. For the component controller, there are three additional standard methods. Two of them are particularly significant:

- ***wddobeforenavigation( )***

  When a client event is raised, the corresponding action method in the view controller is processed. The method *wddobeforenavigation( )* is called after the action method has been processed and just before the Web Dynpro framework processes the events in the navigation queue.

- ***wddopostprocessing( )***

  In complex Web Dynpro applications, it is possible that the data from multiple components must be validated before the next step in the business process can be taken. This method has been created so that cross-component validation can take place. It is the last controller method that is processed before the UI is sent to the client.



- **WDDOBEFORENAVIGATION**
  Executed before the navigation stack is processed
- **WDDOPOSTPROCESSING**
  Data from multiple components can be validated before the next step is executed

Figure 192: Standard Hook Methods: Component Controller

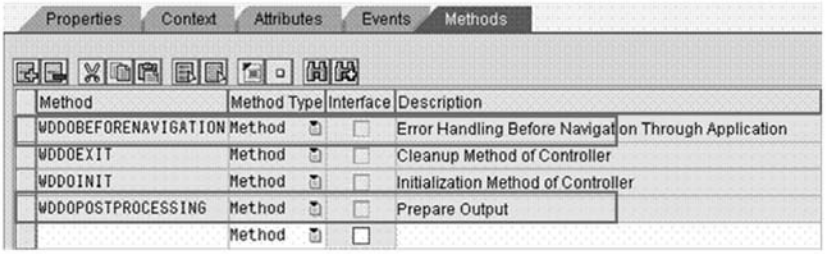
For view controllers, there are two additional hook methods:

- ***wddobeforeaction( )***

  After the Web Dynpro application has been started and the user has raised a client event, the first methods to be processed in the Web Dynpro application are the *wddobeforeaction( )* methods of all view controllers of the view assembly previously rendered. This happens even before the action handler method for the client event is processed.

- ***wddomodifyview( )***

  The only method that allows access to the UI element hierarchy is the view method *wddomodifyview( ).* The interface parameter *VIEW* is a reference to the UI element hierarchy; the *FIRST_TIME* parameter can be used to determine if this method has been previously processed in the view controller's life cycle. This method can be used to manipulate the UI element hierarchy dynamically.



- **WDDOBEFOREACTION**
  The first method processed after a client event has been triggered
- **WDDOMODIFYVIEW**
  Can be used to define the UI dynamically

Figure 193: Standard Hook Methods:View Controller

Here, the phase model for controller processing can be visualized. Here, the following demos should be shown:

NET310_EVEN_D1: Order of hook methods for one view in one window.

NET310_EVEN_D2: Order of hook methods for two views in one window.

NET310_EVEN_D2A: Order of hook methods for two views in one window. The views lifetime is restricted.

NET310_EVEN_D3: Order of hook methods for three views in one window. One view contains two view containers. The other views are embedded in these view containers, so they are displayed in parallel.

Controller Instance Methods

For all controllers, you can create additional methods by declaring the method name and its parameters in the *Methods* tab of the controller editor window. To define an instance method, choose *Method Type = Method*. If you choose *Method Type = Event Handler*, an event handler method is created. In the *Event* column, the method can be registered statically for each event that is triggered in a controller that is defined as the *Used Controller* on the *Properties* tab. Finally, the *Method Type = Supply Function* can be set, to define methods that are bound to context nodes (property *Supply Function*). These methods are automatically called from the Web Dynpro framework if the node is accessed and it is marked as invalid.

After double-clicking the method name, the signature of the method and the source code of the method can be defined.

All methods are public and can be used by any other controller of the same component. As a prerequisite, the controller that will call the method defines a usage relation to the controller that owns the methods. The ABAP source code to call a method in the used controller can be generated using the Web Dynpro Code Wizard.

If the *Interface* indicator is set for a user-defined method, this method will appear in the component interface, so the method is also visible to other components.

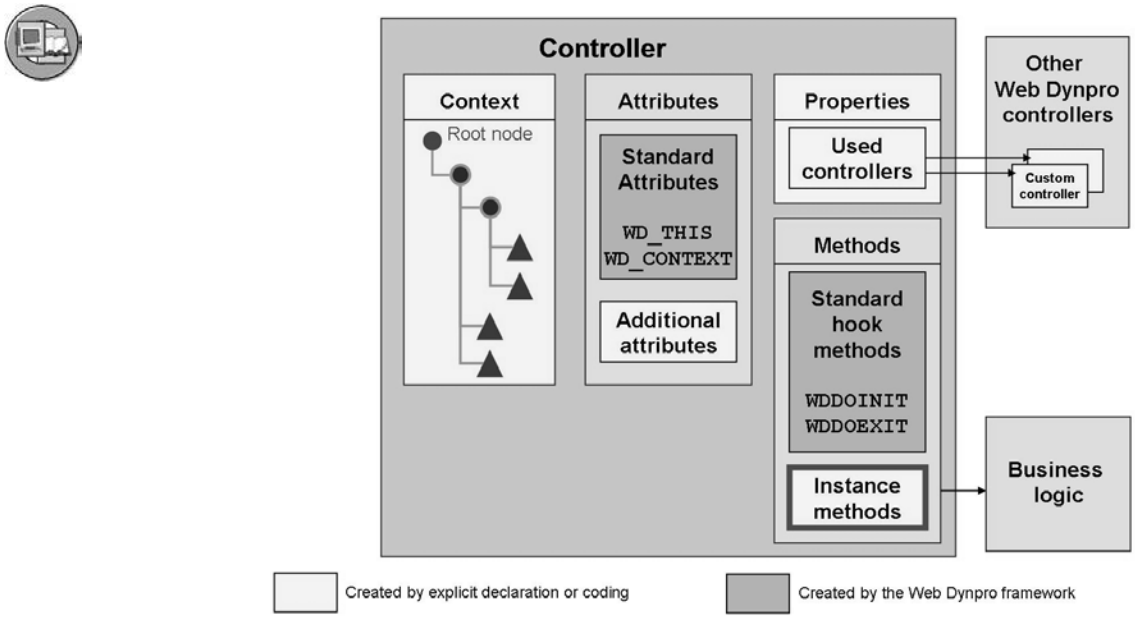


Figure 194: Controller Instance Methods

Your component: Create a new method in the custom controller. Define this controller to be a used controller for one of your view controllers. Call the custom controller method from this view controller. Look at source code.

Controller Attributes

Each controller is a separate ABAP program, having not only predefined and user-defined methods, but also attributes, which are at least visible for each method of the controller.

Standard Attributes

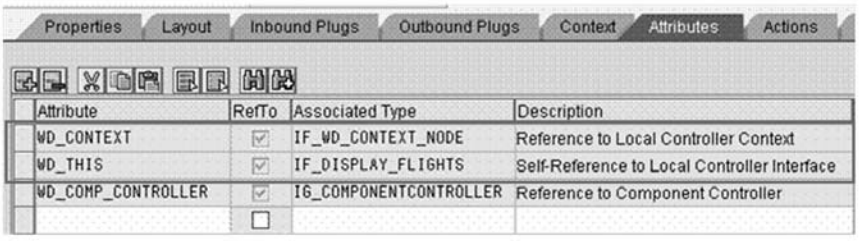
After having defined a controller, at least two attributes are predefined. These attributes have a visibility, which is restricted to the controller methods. The standard attributes are:

- ***WD_THIS***

  WD_THIS is a self reference to the controller interface. This attribute must be distinguished from the standard ABAP self reference *ME*, which should not be used in the source code of any controller. *WD_THIS* is a reference to the current controller interface *IF<Controller-Name>*, and represents all functions implemented in the generated class. This also allows you to access standard Web Dynpro functions, such as validation.

- ***WD_CONTEXT***

  WD_CONTEXT is the reference to the controller's context root node, and therefore constitutes a reference to the entire context. Any access to the controller's context starts with this reference.



- **WD_CONTEXT and WD_THIS**
  - **Present in any Web Dynpro controller (except the interface controller and the interface view controller)**
  - **WD_THIS: Self reference of the local interface - type depends on the controller type**
  - **WD_CONTEXT: Reference to the context of the associated controller**

Figure 195: Standard Controller Attributes WD_CONTEXT and WD_THIS

If the component controller is declared as a used controller on the *Properties* tab of any other controller, an additional attribute is automatically created for the controller that declared the usage:

- ***WD_COMP_CONTROLLER***

  WD_COMP_CONTROLLER is the reference to the component controller. If you use this reference, you can access all methods and public attributes of the component controller (`wd_comp_controller-><meth>`, whereby <*meth*> is a placeholder for the method name).

Create an instance method in the component controller and call this method from the view controller. Look at source code. Compare with the coding to call the custom controller method (for the component controller method, the shortcut variable WD_COMP_CONTROLLER is used).

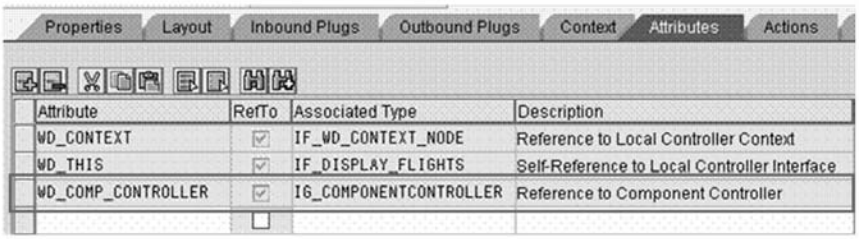


Figure 196: Standard Controller Attribute WD_COMP_CONTROLLER

For all other controllers, even if declared as used controllers, such a reference is not available. However this does not mean that user-defined methods and public attributes are not available, but that the reference must be evaluated first instead. To have the reference to the <*ctrl*> controller defined as a used controller, the following statement must be used:

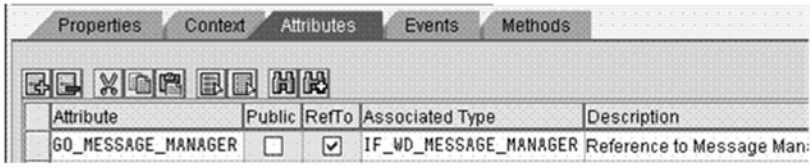
```
DATA: r_ctrl TYPE REF TO ig_<ctrl> .
  r_ctrl = wd_this->get_<ctrl>_ctr( ).
```

User-Defined Attributes

On the *Attributes* tab, additional attributes can be defined for the related controller. If Visibility is set to *Public*, these attributes are also visible for other controllers of the same Web Dynpro component. Attributes cannot be exposed to the components interface.

To access public controller attributes in one of the controller's methods, the self-reference variable *WD_THIS* has to be used. Accessing public attributes defined in other controllers of the same component is implemented similarly to accessing methods defined in other controllers.

- **Attributes for the controller can be created (public or private).**



- **Arbitrary methods, that may or may not be exposed to the component interface, can also be created.**

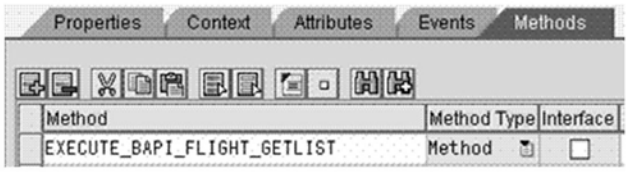


Figure 197: User-Defined Controller Attributes and Controller Methods

Your component: In the component controller, create two attributes (one public, one private). Access these attributes from any method of the component controller. Try to access these attributes from any method of the view controller (WD_COMP_CONTROLLER-><attribute>). Result: Result: Only the public attribute can be used by another controller.

Accessing the Context Nodes and Node Elements at Runtime

In the last section, we saw that we can use controller attributes to provide data objects that are visible throughout the controller. However, it is not possible to bind UI element properties to these controller attributes. UI element properties can only be bound to variables defined in the controller context. This hierarchical data storage is also preferable if information has to be shared between controllers.

Accessing the controller context at runtime requires knowledge of the appropriate Web Dynpro methods. This section deals with how to read, change, add, or delete information stored in the controller context.

Accessing a Context Node

Accessing a context element or a context attribute requires you to first have a reference to the related context node.

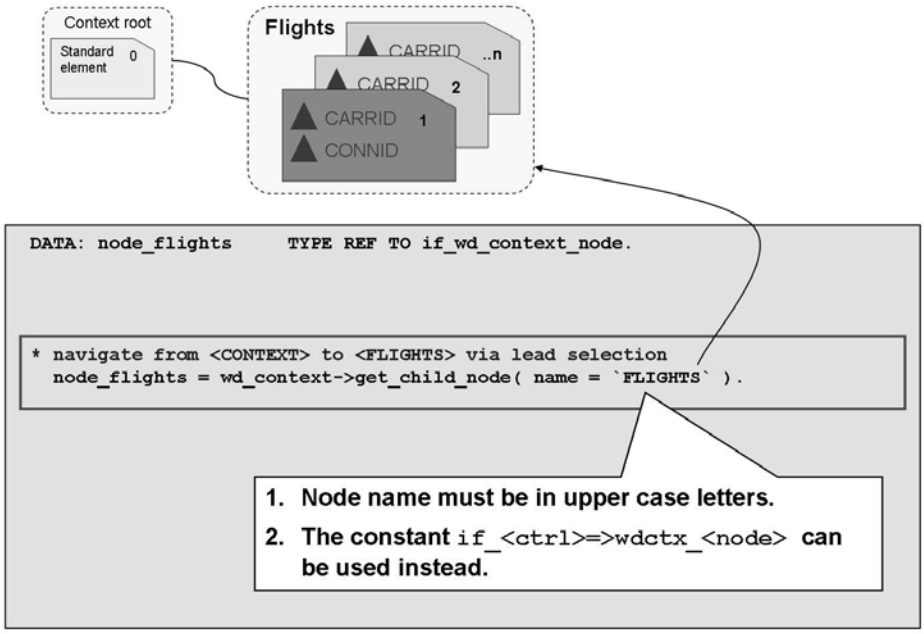


Figure 198: Accessing a Context Node

The following information is very important:

- For each controller (<*ctrl*>), an interface with the name *IF_<ctrl>* is generated.
- For each node <*node*> of a controller context, a constant (*WDCTX_<node>*) is generated in this interface. It has the same name as the node (in upper case), and uses this name as its value. This constant can be used when accessing the context node.

The context root node can be accessed by the standard attribute *WD_CONTEXT*. Child nodes of the context root node can be determined using the *get_child_node( )* method. This method returns a reference to the node instance of the type *IF_WD_CONTEXT_NODE*. The *get_child_node( )* method requires the name of the node and, optionally, the index of the element in the parent node to which the desired node instance belongs. In this case, the parent node is the context root, which only ever has one element. Therefore, the index parameter is 1 (default = *lead selection* of the parent element).

Accessing a Node Element

After having accessed a context node, the reference to the element at lead selection of this node can be obtained by calling the method *get_element( )*. This method returns a reference to the node instance of the type *IF_WD_CONTEXT_ELEMENT*.

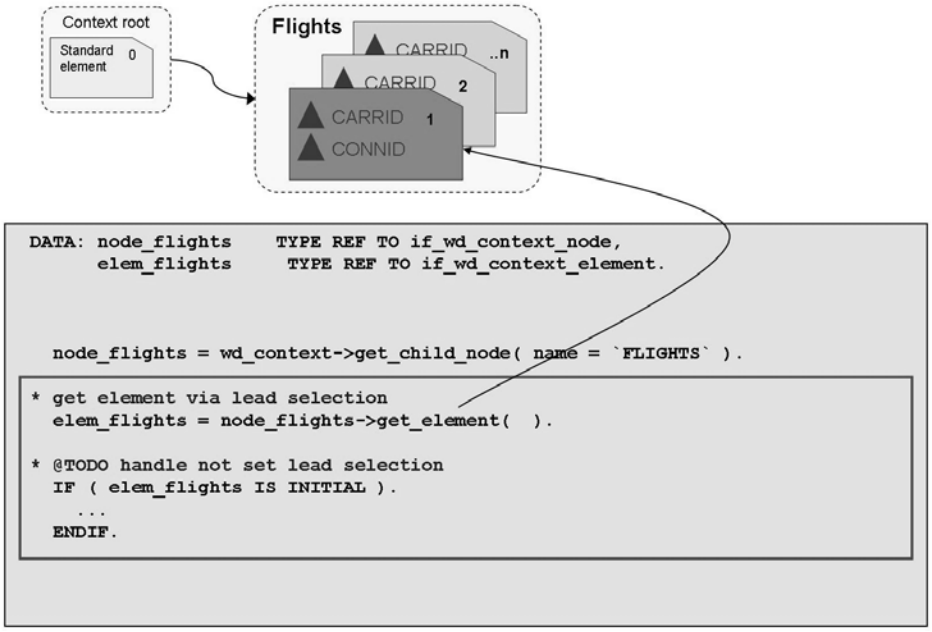


Figure 199: Accessing the Node Element at Lead Selection

The element with the index *n* can be accessed using the method *get_element(index = n).* The number of elements in a collection can be obtained from the method *get_element_count( )*

Conclusion

Summary: Accessing Context Nodes and Node Elements

Action	Method
Reference to the node <*node*> of a controller <*ctrl*>	`r_node = wd_context->get_child_node( name = if_<ctrl>=>wdctx_<node> ).`
Reference to the lead selection element	`r_element = r_node->get_element( ).`
Reference to the element with the index *n*	`r_element = r_node->get_element( index = n ).`
Get the number of elements in the collection	`n = r_node->get_element_count( ).`

Your component: in your component controller, method DOINIT, access the element at lead selection of any of your nodes. Use the code wizard to access attributes, but only show the coding discussed up to now. Check what happens if the node's cardinality is 0..<something> and no elements have been added to the collection yet (Result: get_element( ) will return an initial element reference; this should be handled).

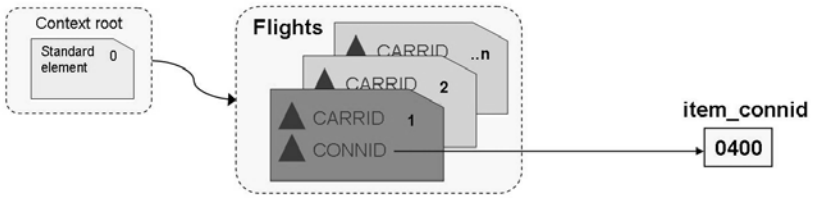
Reading and Changing Attribute Values

The Web Dynpro framework offers a variety of methods to access the attributes of one node element or to access the attributes of all elements of a context node.

Accessing Attributes of a Single Node Element

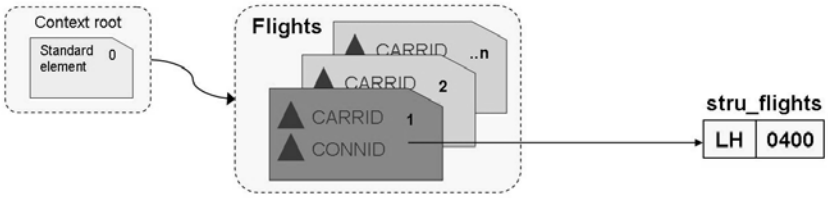
Once you have obtained the reference to a node element, there are two ways of obtaining the attribute values of this element:

1. Any attribute of a node element can be accessed using the method *get_attribute( ).* Here, the name of the attribute must be exported and the attribute value is returned in an import parameter.
2. Statically defined attributes can be obtained by calling the method *get_static_attributes( ).* Here, a structure is returned in an import parameter. The target structure can be different from the node structure. This is possible because the structure fields are copied individually to the target structure.




```
DATA: node_flights     TYPE REF TO if_wd_context_node,
      elem_flights     TYPE REF TO if_wd_context_element,
      item_connid      TYPE if_componentcontroller=>element_flights-connid.


  node_flights = wd_context->get_child_node( name = `FLIGHTS` ).

  elem_flights = node_flights->get_element(  ).

* get single attribute
  elem_flights->get_attribute(
    EXPORTING
      name =  `CONNID`
    IMPORTING
      value = item_connid ).
```

Figure 200: Accessing a Single Attribute of a Node Element




```
DATA: node_flights     TYPE REF TO if_wd_context_node,
      elem_flights     TYPE REF TO if_wd_context_element,
      stru_flights     TYPE        if_componentcontroller=>element_flights.


  node_flights = wd_context->get_child_node( name = `FLIGHTS` ).

  elem_flights = node_flights->get_element(  ).

* get all statically declared attributes
  elem_flights->get_static_attributes(
    IMPORTING
      static_attributes = stru_flights ).
```

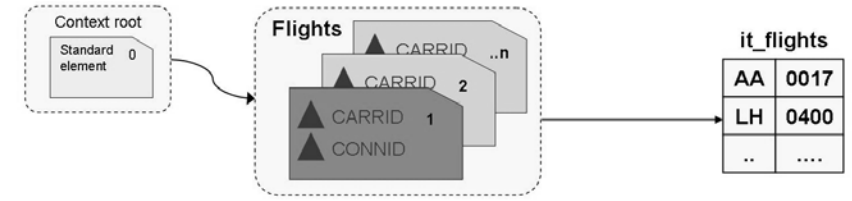
Figure 201: Accessing All Statically Defined Attributes of a Node Element

Your component: Use the code wizard to generate coding to read a single attribute or all static attributes of a node element at lead selection.

The following information is of importance in this context:

- For each node *<node>* of a controller context, a structure type *element_<node>* is implicitly generated in the interface *IF_<ctrl>*. The structure fields correspond to the attributes that make up a node element. This data type can be used to type a variable, which is filled by the methods listed above.
- In addition, for each node *<node>* of a controller context, a standard table type *elements_<node>* is implicitly generated in the interface *IF_<ctrl>*. The line type of this table is *element_<node>*. This data type can be used to type an internal table that can hold the attributes of multiple node elements.

Accessing the Attributes of all Node Elements




```
DATA: node_flights     TYPE REF TO if_wd_context_node,
      it_flights       type        if_componentcontroller=>elements_flights.


  node_flights = wd_context->get_child_node( name = `FLIGHTS` ).

* get all static attributes of all node elements
  node_flights->get_static_attributes_table(
    IMPORTING
      table = it_flights ).
```

Figure 202: Access to the Static Attributes of All Node Elements

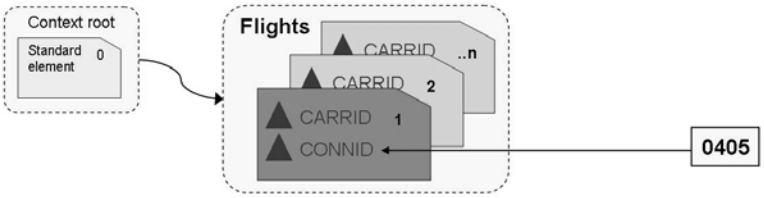
The method *get_static_attributes_table( )* can be used to retrieve the attributes of all elements as an internal table.

Your component: You have created a service call to bind mass data to a related context node. After calling the service method, you could read the mass data from the context, using the get_static_attributes_table() method.

Changing Attribute Values of a Node Element

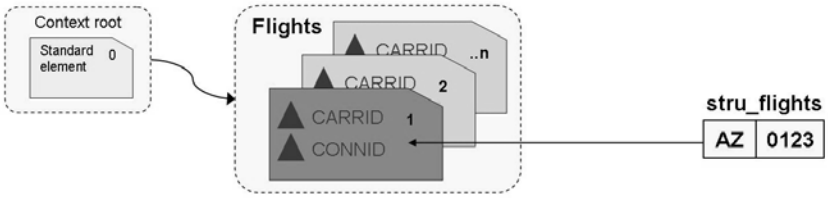
After the reference to a certain node element has been determined, the attribute values cannot only be read, using the appropriate getter methods, but it is also possible to change existing attribute values by calling related setter methods.

The method *set_attribute( )* can be used to change the value of any attribute of the node element. Multiple attributes can be changed if they are statically set using the method *set_static_attributes( ).*




```
DATA: node_flights     TYPE REF TO if_wd_context_node,
      elem_flights     TYPE REF TO if_wd_context_element,
      item_connid      TYPE if_componentcontroller=>element_flights-connid.


  node_flights = wd_context->get_child_node( name = `FLIGHTS` ).

 elem_flights = node_flights->get_element( ).

* set single attribute
  elem_flights->set_attribute(
    EXPORTING
      name =  `CONNID`
      value = `0405` ).
```

Figure 203: Changing a Single Attribute of a Node Element




```
DATA: node_flights     TYPE REF TO if_wd_context_node,
      elem_flights     TYPE REF TO if_wd_context_element,
      stru_flights     TYPE        if_componentcontroller=>element_flights.


  node_flights = wd_context->get_child_node( name = `FLIGHTS` ).

 elem_flights = node_flights->get_element(  ).

 stru_flights-carrid = `AZ`.
  stru_flights-connid = `0123`.

* set statically declared attributes
  elem_flights->set_static_attributes(
    EXPORTING
      static_attributes = stru_flights ).
```

Figure 204: Changing Multiple Attributes of a Node Element

Conclusion

Summary: Accessing and Changing Attributes in the Node <node> of the Controller <ctrl>

Action	Method

Read value of attribute <*attr*>	`DATA:`
	` v_value TYPE`
	`  if_<ctrl>=>wdctx_element_<node>-<attr>.`
	`r_element->get_attribute(`
	`  EXPORTING`
	`  name = '<attr>'`
	`  IMPORTING`
	`  value = v_value ).`

Read value of multiple static attributes	`DATA:`
	` s_struc TYPE`
	`  if_<ctrl>=>wdctx_element_<node>.`
	`ref_element->get_static_attributes(`
	`  IMPORTING`
	`  static_attributes = s_struc ).`

Read static attribute values for all node elements	`DATA:`
	` t_tab TYPE`
	`  if_<ctrl>=>wdctx_elements_<node>.`
	`ref_node->get_static_attributes_table(`
	`  IMPORTING`
	`  table = t_tab ).`

Change value *v_value* of a single attribute <*attr*>	`ref_element->set_attribute(`
	`  EXPORTING`
	`  name  = '<attr>'`
	`  value = v_value ).`

Changing Multiple Attributes of a Node Element	`DATA:`
	` s_struc TYPE`
	`  if_<ctrt>=>wdctx_element_<node>.`
	`s_struc-<attr1> = ...`
	`...`
	`ref_element->set_static_attributes(`
	`  EXPORTING`
	`  static_attributes = s_struc ).`

Exercise “Accessing the Context at Runtime”

Adding New Elements to a Context Node

Up to now we have discussed the reading and changing of data that is already stored in the controller context. Now we will learn how to add new elements to a context node.

Adding a new element to a node is performed in two steps. The first step is to create an element that can be added to a certain context node at a later stage. After the attribute values have been defined, the new element can be added to the context node. This is similar to the process for adding a new line to an internal table. The first step is to define the cell values of a work area that has the correct line type, and the second step is to insert the work area into the internal table.

Creating a New Node Element

To create an element that can be added to a context node, the reference to this node must first be determined. This is done using the *get_child_node( )* method of the standard attribute *WD_CONTEXT*, which points to the context root node.

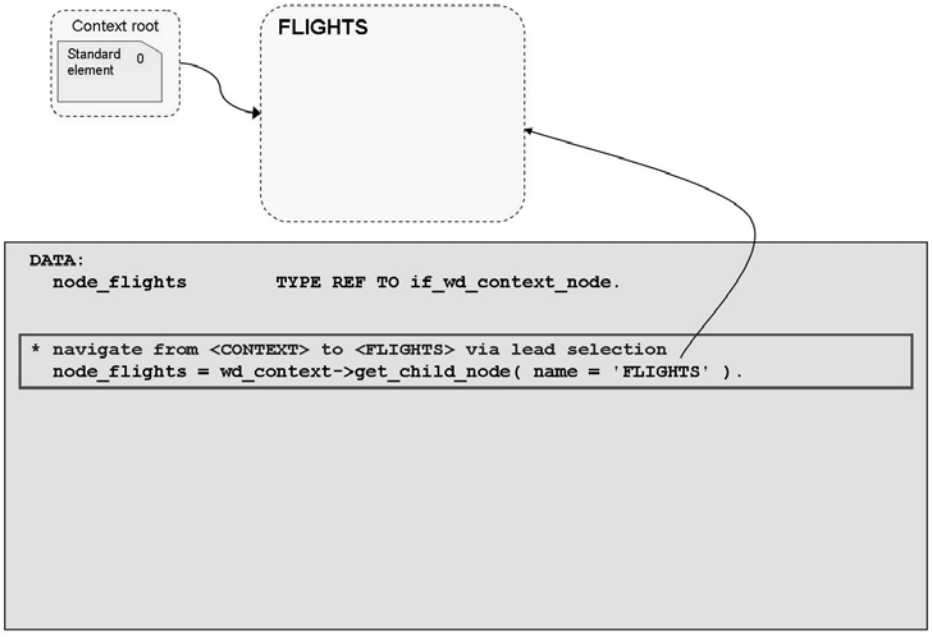


Figure 205: Determining the Reference to a Context Node

Once you have determined this reference, the method *create_element( )* is used to create the new element. Initial values for the static attributes can be submitted using the *static_attribute_values* parameter or the attribute values can be defined using the setter methods *set_attribute( )* or *set_static_attributes( )*.

Caution: The new element is not yet part of the context node.

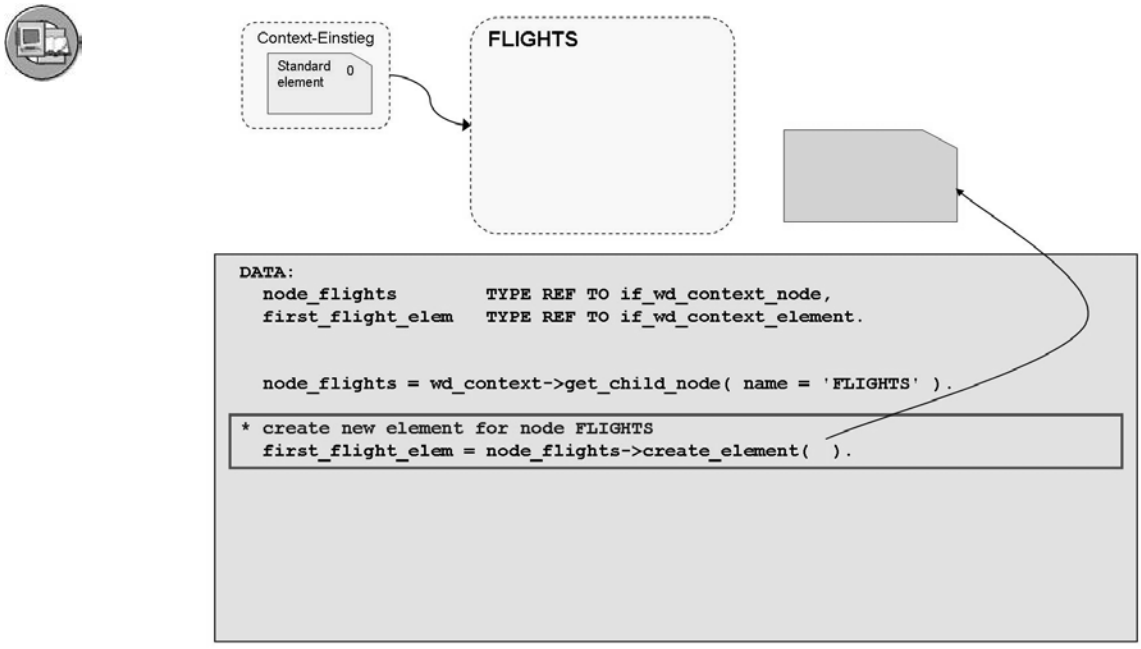


Figure 206: Creating a New Node Element

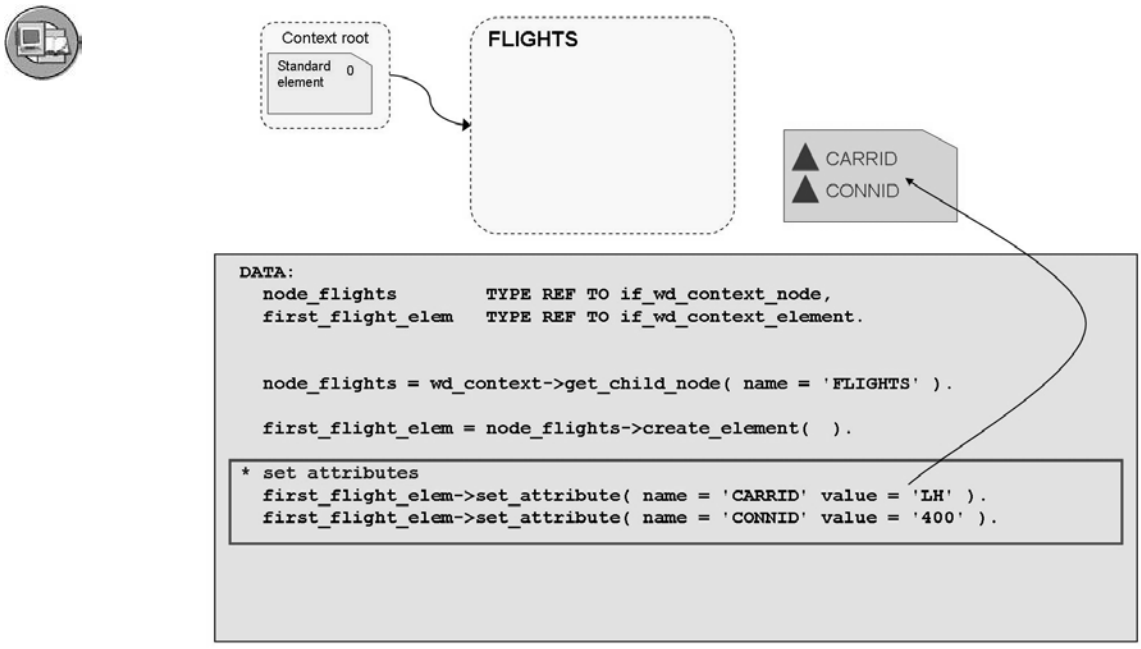
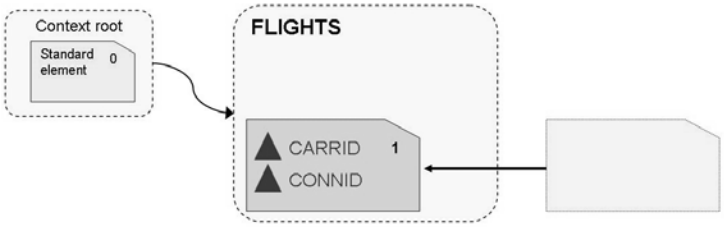


Figure 207: Setting the Attribute Values of the New Element

Adding Elements to a Context Node

Finally, an element that is not yet part of the context node can be added to the node using the method *bind_element( )*, which refers to the node reference. This method has two import parameters:

- The element reference is submitted using the parameter *new_item*.
- The parameter *set_initial_elements* defines if the new element is just added to the element collection (value = *abap_false*) or if the new element replaces all existing elements in the collection (value = *abap_true*).




```
DATA:
  node_flights        TYPE REF TO if_wd_context_node,
  first_flight_elem   TYPE REF TO if_wd_context_element.


  node_flights = wd_context->get_child_node( name = 'FLIGHTS' ).

  first_flight_elem = node_flights->create_element(  ).

  first_flight_elem->set_attribute( name = 'CARRID' value = 'LH' ).
  first_flight_elem->set_attribute( name = 'CONNID' value = '400' ).

* bind first element to node
  node_flights->bind_element(
    new_item             = first_flight_elem
    set_initial_elements = abap_false ).
```

Figure 208: Binding an Element to a Context Node

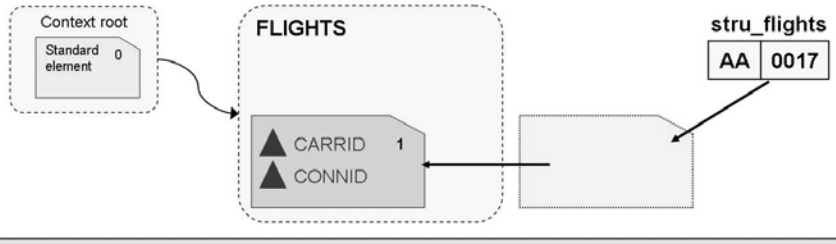
Your component: On your first view, you have some input fields bound to context attributes. Change the cardinality of the related context node to 0..1. Start the application. The application terminates.

In the WDDOINIT method of this view, create a context element, set the context attributes displayed in the layout and add the element to the collection. Start the application again. Input fields are ready for input and attribute values are displayed.

Binding a Structure to a Context Node

In ABAP programs, data records are handled as structures. However, to display the structure elements in the UI, the structure content must be copied to a context element. This means that a new element has to be defined, the attribute values have to be set, and this element has to be bound to the appropriate context node.

There is an easier way to copy the structure content as a new element to the context node. Instead of using the method *bind_element( )* and submitting the element reference, the method *bind_structure( )* can be used with the parameter *new_item*, to submit the structure. As for the *bind_element( )* method, the existing collection can be extended or replaced (parameter *set_initial_elements*).



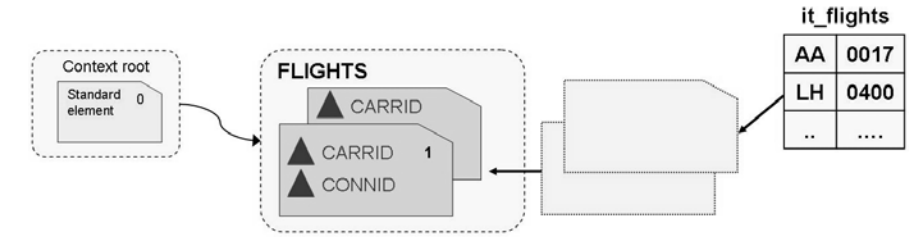

```
DATA:
  node_flights        TYPE REF TO if_wd_context_node,
  stru_flights     TYPE        if_componentcontroller=>element_flights.


  node_flights = wd_context->get_child_node( name = 'FLIGHTS' ).

* Set field values of local structure
  stru_flights-carrid = 'AA'.
  stru_flights-connid = '0017'.

* bind structure to node
  node_flights->bind_structure(
    new_item             = stru_flights
    set_initial_elements = abap_false ).
```

Figure 209: Binding a Structure to a Context Node

Binding an Internal Table to a Context Node

Multiple data records that have an identical structure are handled as internal tables in ABAP programs. However, to be able to display the data sets in the UI, the internal table content has to be copied to as many context elements as there are lines in the internal table.

The best way to do this is to use the method *bind_table( )*. Here, the internal table is submitted using the parameter *new_items*. The existing collection can be extended or replaced (parameter *set_initial_elements*).




```
DATA:
  node_flights     TYPE REF TO if_wd_context_node,
  stru_flights     TYPE if_componentcontroller=>element_flights,
  it_flights       TYPE if_componentcontroller=>elements_flights.

  node_flights = wd_context->get_child_node( name = 'FLIGHTS' ).

* append lines to local table
  stru_flights-carrid   = `AA`.
  stru_flights-connid   = `0017`.
  APPEND stru_flights TO it_flights.

  stru_flights-carrid   = `LH`.
  stru_flights-connid   = `0400`.
  APPEND Stru_flights TO it_flights.

* bind internal table to node
  node_flights->bind_table( new_items = it_flights ).
```

Figure 210: Binding an Internal Table to a Context Node

Here, you should show demo NET310_UI_D5. Here, the source code of the supply function is of interest. The reference to the node to be altered by the supply function is available using the interface parameter NODE and the reference to the element at lead selection in the parent collection is provided by the interface parameter PARENT_ELEMENT. Only data retrieval has to be implemented. You can also implement a supply function for any node of your component controller context.

Deleting Elements from a Context Node

To remove an element from a collection, call the method *remove_element( )*. The reference to the element must be submitted using the parameter *element.*

Two demos can be shown:

NET310_CONR_D1: The following method calls can be discussed while debugging the source code of this demo:

node->get_element(), node->create_element(), node->bind_element(), node->get_lead_selection(), node->get_element_count(), node->bind_structure(), element->set_static_attributes() and *element->set_attribute()*.

NET310_CONR_D2: In addition, the method call *node->remove_element()* is used in this demo.

Conclusion

Summary: Adding Elements to / Deleting Elements from a Context Node

Action	Method
Create a new element	`r_element = r_node->create_element( ).`

Add element to collection	`r_node->bind_element(`
	`  new_item = r_element`
	`  set_initial_elements = abap_false`
	`).`

Bind structure *s_struc* to collection	`DATA:`
	`  s_struc TYPE`
	`  if_<ctrt>=>wdctx_element_<node>.`
	`...`
	`r_node->bind_structure(`
	`  new_item = s_struc`
	`  set_initial_elements = abap_false`
	`).`

Bind internal table *t_tab* to collection	`DATA:`
	`  t_tab TYPE`
	`  if_<ctrt>=>wdctx_elements_<node>.`
	`...`
	`r_node->bind_table(`
	`  new_items = t_tab`
	`  set_initial_elements = abap_false`
	`).`

Remove element from the collection	`r_node->remove_element(` `  element = r_element ).`

Exercises “Binding Internal Tables to Context Nodes” and “Lead Selection and Supply Functions”

343

Exercise 18: Accessing the Context at Runtime

Exercise Duration: 15 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Set the value of context node attributes dynamically
- Set default values for input fields dynamically

Business Example

You want to develop a Web Dynpro application where input fields of the first view are filled with default values at runtime.

Template: NET310_COND_S

Solution: NET310_CONR_S1 / NET310_CONR_S1_OPT

Task 1:

Copy your Web Dynpro component **ZNET310_COND_##** or the template **NET310_COND_S** to Web Dynpro component **ZNET310_CONR1_##**.

Change the layout of both views and make sure that the two input fields (with their labels) and the buttons are placed below each other.

1. Copy the template.
2. Change the layout of the views.

Task 2:

Make **AA** the default value of the *CARRID* field before the input view is displayed.

1. Implement the method *WDDOINIT* of the view controller. Use the Web Dynpro Code Wizard, to read the context node FLIGHTINFO.
2. Replace the last method call (*get_static_attributes* method) with a call of a context element method to set the value of the attribute *CARRID* to **AA**.

Continued on next page

Task 3: (Optional)

Set the default values for all the input fields before the view is displayed. Use a structure and a single method call instead of setting the default values one by one.

1. Turn the last method call into a comment. Replace it as follows:

   Fill the structure *stru_flightinfo* with default values and call a method of the context element, to set all attributes in one step.

Solution 18: Accessing the Context at Runtime

Task 1:

Copy your Web Dynpro component **ZNET310_COND_##** or the template **NET310_COND_S** to Web Dynpro component **ZNET310_CONR1_##**.

Change the layout of both views and make sure that the two input fields (with their labels) and the buttons are placed below each other.

1. Copy the template.
   a) Perform this step as in previous exercises.
2. Change the layout of the views.
   a) Change the *Layout* property of ROOTUIELEMENTCONTAINER to **`MatrixLayout`**.
   b) Change the *Layout* property of the group to **`MatrixLayout`**.
   c) Set the *Layoutdata* property of the group, the labels and the buttons to **`MatrixHeadData`**.

Task 2:

Make **AA** the default value of the *CARRID* field before the input view is displayed.

1. Implement the method *WDDOINIT* of the view controller. Use the Web Dynpro Code Wizard, to read the context node FLIGHTINFO.
   a) Select the *WebDynpro Code Wizard* button.
   b) Select *Read Context Node/Attribute* and use the input help to choose the context node.
2. Replace the last method call (*get_static_attributes* method) with a call of a context element method to set the value of the attribute *CARRID* to **AA**.
   a) See the source code sample from the model solution under **Result**.

Task 3: (Optional)

Set the default values for all the input fields before the view is displayed. Use a structure and a single method call instead of setting the default values one by one.

1. Turn the last method call into a comment. Replace it as follows:

Continued on next page

Fill the structure *stru_flightinfo* with default values and call a method of the context element, to set all attributes in one step.

a) See the source code sample from the model solution.

Result

Model Solution:NET310_CONR_S1.

```
METHOD wddoinit .

  DATA:
    node_flightinfo                    TYPE REF TO if_wd_context_node,
    elem_flightinfo                    TYPE REF TO if_wd_context_element.

* navigate from <CONTEXT> to <FLIGHTINFO> via lead selection
  node_flightinfo = wd_context->get_child_node(
                                  name = if_input_view=>wdctx_flightinfo ).

* get element via lead selection
  elem_flightinfo = node_flightinfo->get_element(  ).

* set single attribute
  elem_flightinfo->set_attribute(
    EXPORTING
      name  = 'CARRID'
      value = 'AA' ).

ENDMETHOD.
```

Model Solution: NET310_CONR_S1_OPT

```
METHOD wddoinit .

  DATA:
    node_flightinfo                    TYPE REF TO if_wd_context_node,
    elem_flightinfo                    TYPE REF TO if_wd_context_element,
    stru_flightinfo                    TYPE if_input_view=>element_flightinfo .
```

Continued on next page

```
* navigate from CONTEXT to FLIGHTINFO via lead selection
  node_flightinfo = wd_context->get_child_node( name = ‘FLIGHTINFO‘ ).

* get element via lead selection
  elem_flightinfo = node_flightinfo->get_element(  ).

* set all declared attributes
  stru_flightinfo-carrid = ’AA’.
  stru_flightinfo-connid = ’0400’.

  elem_flightinfo->set_static_attributes(
    exporting
      static_attributes = stru_flightinfo ).

ENDMETHOD.
```

 349

Exercise 19: Context at Runtime: Binding Internal Tables to Context Nodes

Exercise Duration: 30 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Bind internal tables to context nodes at runtime

Business Example

You want to develop a Web Dynpro application where the user can enter selection criteria in a first view. After the user selects a button, data is selected and navigation to a second view is triggered, in which the selected data is displayed as a table.

Template: NET310_CONR_S1_OPT

Solution: NET310_CONR_S2

Task 1:

Copy your Web Dynpro component **ZNET310_CONR1_##** or the template **NET310_CONR_S1_OPT** to the Web Dynpro component **ZNET310_CONR2_##**.

In the component context, create a new context node to store datasets that are read from the database table **SFLIGHT**.

1. Copy the template.
2. In the component context, create a new context node (suggested name: **FLIGHTTAB**), with a reference to the ABAP Dictionary structure **SFLIGHT** and the cardinality **0...n**.

   The node should contain the following attributes:CARRID, CONNID, FLDATE, PLANETYPE, SEATSMAX, and SEATSOCC.

Continued on next page

Task 2:

Copy the new context node to the context of the view OUTPUT_VIEW and map the context nodes of the different controllers. Extend the layout of the view to display the data in a table.

1. Copy component context node FLIGHTTAB to the context of the view OUTPUT_VIEW. Map the node of the view controller context to the node in the component controller context.
2. Use the Web Dynpro Code Wizard to create a table display with a binding to the context node FLIGHTTAB.

Task 3:

Create a method in the component controller in which you select flights from database table SFLIGHT and store them in an internal table. Use the static method CL_NET310_FLIGHTMODEL=>READ_FLIGHTS(), to read the data. Store the result in the context node FLIGHTTAB.

1. Create a new method in the component controller (suggested name: **FLIGHTTAB_FILL**).
2. Use the Web Dynpro Code Wizard to read the context node FLIGHTINFO.
3. Create an internal table (suggested name: **IT_FLIGHTTAB**) of table type if_componentcontroller=>elements_flighttab. Use the static method **CL_NET310_FLIGHTMODEL=>READ_FLIGHTS()** to fill the internal table (export parameter e_flights). Use CARRID and CONNID of the context node FLIGHTINFO, to restrict the data selection.
4. Use the Web Dynpro Code Wizard to read the context node FLIGHTTAB after having called the static method **CL_NET310_FLIGHT-MODEL=>READ_FLIGHTS()**. Since you do not want to read data from the context, but instead wish to store data in the context, remove all method calls created by the wizard except for the first one (method GET_CHILD_NODE).
5. Call the method BIND_TABLE for the context node FLIGHTTAB, to store the content of the internal table in the context node.

Task 4:

Make sure your new component controller method is executed after the navigation, immediately before the output view is displayed.

1. Edit the method **HANDLEFROM_INPUT_VIEW** of the OUTPUT_VIEW view controller and use the Web Dynpro Code Wizard to implement a call of the component controller method.

Solution 19: Context at Runtime: Binding Internal Tables to Context Nodes

Task 1:

Copy your Web Dynpro component **ZNET310_CONR1_##** or the template **NET310_CONR_S1_OPT** to the Web Dynpro component **ZNET310_CONR2_##**.

In the component context, create a new context node to store datasets that are read from the database table **SFLIGHT**.

1. Copy the template.

   a) Perform this step as in previous exercises.

2. In the component context, create a new context node (suggested name: **FLIGHTTAB**), with a reference to the ABAP Dictionary structure **SFLIGHT** and the cardinality **0...n**.

   The node should contain the following attributes:CARRID, CONNID, FLDATE, PLANETYPE, SEATSMAX, and SEATSOCC.

   a) Perform this step as in previous exercises.

Task 2:

Copy the new context node to the context of the view OUTPUT_VIEW and map the context nodes of the different controllers. Extend the layout of the view to display the data in a table.

1. Copy component context node FLIGHTTAB to the context of the view OUTPUT_VIEW. Map the node of the view controller context to the node in the component controller context.

   a) Perform this step as in previous exercises.

2. Use the Web Dynpro Code Wizard to create a table display with a binding to the context node FLIGHTTAB.

   a) Perform this step as in previous exercises.

Continued on next page

Task 3:

Create a method in the component controller in which you select flights from database table SFLIGHT and store them in an internal table. Use the static method CL_NET310_FLIGHTMODEL=>READ_FLIGHTS(), to read the data. Store the result in the context node FLIGHTTAB.

1. Create a new method in the component controller (suggested name: **FLIGHTTAB_FILL**).

   a) In the component controller, choose the *Methods* tab, enter the name of the method, and double-click this.

2. Use the Web Dynpro Code Wizard to read the context node FLIGHTINFO.

   a) Perform this step as in previous exercises.

3. Create an internal table (suggested name: **IT_FLIGHTTAB**) of table type if_componentcontroller=>elements_flighttab. Use the static method **CL_NET310_FLIGHTMODEL=>READ_FLIGHTS()** to fill the internal table (export parameter e_flights). Use CARRID and CONNID of the context node FLIGHTINFO, to restrict the data selection.

   a) See the source code of the model solution.

4. Use the Web Dynpro Code Wizard to read the context node FLIGHTTAB after having called the static method **CL_NET310_FLIGHT-MODEL=>READ_FLIGHTS()**. Since you do not want to read data from the context, but instead wish to store data in the context, remove all method calls created by the wizard except for the first one (method GET_CHILD_NODE).

   a) See the source code of the model solution.

5. Call the method BIND_TABLE for the context node FLIGHTTAB, to store the content of the internal table in the context node.

   a) See the source code of the model solution.

Task 4:

Make sure your new component controller method is executed after the navigation, immediately before the output view is displayed.

1. Edit the method **HANDLEFROM_INPUT_VIEW** of the OUTPUT_VIEW view controller and use the Web Dynpro Code Wizard to implement a call of the component controller method.

   a) Open the method HANDLEFROM_INPUT_VIEW.

Continued on next page

b) Choose *WebDynpro Code Wizard*.

c) Select *Method Call in Used Controller*.

d) Select the component controller and enter the name of the method.

Result

Model Solution: NET310_CONR_S2, Method FLIGTHTAB_FILL

```
METHOD flighttab_fill .

  DATA:
    node_flightinfo   TYPE REF TO if_wd_context_node,
    elem_flightinfo   TYPE REF TO if_wd_context_element,
    stru_flightinfo   TYPE        if_componentcontroller=>element_flightinfo.

* navigate from <CONTEXT> to <FLIGHTINFO> via lead selection
  node_flightinfo = wd_context->get_child_node( name = 'FLIGHTINFO' ).

* get element via lead selection
  elem_flightinfo = node_flightinfo->get_element( ).

* get all declared attributes
  elem_flightinfo->get_static_attributes(
    IMPORTING
      static_attributes = stru_flightinfo ).


  DATA:
    node_flighttab    TYPE REF TO if_wd_context_node,
    it_flighttab      TYPE        if_componentcontroller=>elements_flighttab.

* read all flights related to CARRID and CONNID entered by user
* if carrid or/and connid are initial, all carriers/connection numbers
* are read
  CALL METHOD cl_net310_flightmodel=>read_flights
    EXPORTING
      i_carrid  = stru_flightinfo-carrid
      i_connid  = stru_flightinfo-connid
    IMPORTING
```

Continued on next page


```
      e_flights = it_flighttab.

* navigate from <CONTEXT> to <FLIGHTTAB> via lead selection
  node_flighttab = wd_context->get_child_node( name = 'FLIGHTTAB' ).

* bind table to context node <FLIGHTTAB>
  CALL METHOD node_flighttab->bind_table
    EXPORTING
      new_items = it_flighttab.

ENDMETHOD.
```

 355

Exercise 20: Context at Runtime: Lead Selection and Supply Functions

Exercise Duration: 30 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Use supply functions to populate context nodes

Business Example

You want to develop a Web Dynpro application with a view that displays two tables: A list of flights and a list of bookings. The data displayed in the second table should depend on the selected row in the first table. You want to use a supply function to make sure that the data in the second table is changed whenever the user selects a different row in the first table.

Template: NET310_CONR_S2

Solution: NET310_CONR_S3

Task 1:

Copy your Web Dynpro component **ZNET310_CONR2_#** or the template **NET310_CONR_S2** to the Web Dynpro component **ZNET310_CONR3_##**.

In the component context, create a new context node as sub-node of the node FLIGHTTAB, to store data for a set of bookings read from the database table SBOOK.

1. Copy the template.
2. In the component context, create a subnode of the context node FLIGHTTAB (suggested name: **BOOKINGTAB**) with reference to the ABAP Dictionary structure **SBOOK**, and set the cardinality to **0...n**.

   The node should contain the following attributes: BOOKID, CUSTOMID, CUSTTYPE, LUGGWEIGHT, WUNIT, CLASS, and PASSNAME.

Continued on next page

Task 2:

Make sure the context node BOOKINGTAB also exists in the view OUTPUT_VIEW and that this node is mapped to the BOOKINGTAB node of the component controller context. Extend the layout of the view to display the bookings in a second table below the flights table.

1. Update the mapping for context node FLIGHTTAB in the OUTPUT_VIEW view context.

Hint: You have to use the update functions here. It is not possible to map subnodes directly by dragging and dropping them.

2. Use the Web Dynpro Code Wizard to create a table display with a binding to the context node BOOKINGTAB.

Task 3:

Create and implement a supply function to fill sub-node BOOKINGTAB according to the lead selection of node FLIGHTTAB.

1. In the component context, assign a supply function to the subnode BOOKINGTAB (suggested name: **BOOKINGS_READ**) and create it using forward navigation.
2. Remove the comment signs for the generated coding (data declaration and call of method BIND_TABLE for the sub-node BOOKINGTAB).
3. Declare a structure where the components correspond to the attributes of the parent node (suggested name: **STRU_FLIGHTTAB**).

Hint: You may want to copy the declaration of such a structure from method FLIGHTTAB_FILL.

4. Call the static method **CL_NET310_FLIGHTMODEL=>READ_BOOK-INGS()** to fill the internal table ITAB_BOOKINGTAB before it is bound to the subnode BOOKINGTAB. Use the attributes CARRID, CONNID, and FLDATE of the parent node to restrict the data selection.

Solution 20: Context at Runtime: Lead Selection and Supply Functions

Task 1:

Copy your Web Dynpro component **ZNET310_CONR2_#** or the template **NET310_CONR_S2** to the Web Dynpro component **ZNET310_CONR3_##**.

In the component context, create a new context node as sub-node of the node FLIGHTTAB, to store data for a set of bookings read from the database table SBOOK.

1. Copy the template.

   a) Perform this step as in previous exercises.

2. In the component context, create a subnode of the context node FLIGHTTAB (suggested name: **BOOKINGTAB**) with reference to the ABAP Dictionary structure **SBOOK**, and set the cardinality to **0...n**.

   The node should contain the following attributes: BOOKID, CUSTOMID, CUSTTYPE, LUGGWEIGHT, WUNIT, CLASS, and PASSNAME.

   a) Perform this step as in previous exercises.

Task 2:

Make sure the context node BOOKINGTAB also exists in the view OUTPUT_VIEW and that this node is mapped to the BOOKINGTAB node of the component controller context. Extend the layout of the view to display the bookings in a second table below the flights table.

1. Update the mapping for context node FLIGHTTAB in the OUTPUT_VIEW view context.

   Hint: You have to use the update functions here. It is not possible to map subnodes directly by dragging and dropping them.

   a) Open the context menu for the context node FLIGHTTAB and choose *Update Mapping*.

   b) Confirm the following two dialog boxes by choosing *Yes*.

2. Use the Web Dynpro Code Wizard to create a table display with a binding to the context node BOOKINGTAB.

   a) Perform this step as in previous exercises.

Continued on next page

Task 3:

Create and implement a supply function to fill sub-node BOOKINGTAB according to the lead selection of node FLIGHTTAB.

1. In the component context, assign a supply function to the subnode BOOKINGTAB (suggested name: **BOOKINGS_READ**) and create it using forward navigation.

   a) In the component context, double click the subnode.

   b) Enter the name for the supply function as an attribute of the property *Supply Function* and double-click the name.

2. Remove the comment signs for the generated coding (data declaration and call of method BIND_TABLE for the sub-node BOOKINGTAB).

   a) See the source code of the model solution.

3. Declare a structure where the components correspond to the attributes of the parent node (suggested name: **STRU_FLIGHTTAB**).

   Hint: You may want to copy the declaration of such a structure from method FLIGHTTAB_FILL.

   a) See the source code of the model solution.

4. Call the static method **CL_NET310_FLIGHTMODEL=>READ_BOOKINGS()** to fill the internal table ITAB_BOOKINGTAB before it is bound to the subnode BOOKINGTAB. Use the attributes CARRID, CONNID, and FLDATE of the parent node to restrict the data selection.

   a) See the source code of the model solution.

Result

Model Solution: NET310_CONR_S3, Method BOOKINGS_READ

```
METHOD bookings_read .
* General Notes
* ==============
* A common scenario for a supply method is to aquire key
* informations from the parameter <parent_element> and then
```

Continued on next page


```
* to invoke a data provider.
* A free navigation thru the context, especially to nodes on
* the same or deeper hierachical level is strongly discouraged,
* because such a strategy may easily lead to unresolvable
* situations!!

*
* data declaration
  DATA:
    itab_bookingtab    TYPE if_componentcontroller=>elements_bookingtab.

  DATA:
    stru_flighttab    TYPE  if_componentcontroller=>element_flighttab.

* get all declared attributes
  parent_element->get_static_attributes(
    IMPORTING
      static_attributes = stru_flighttab ).

* read bookings related to selected flight
  CALL METHOD cl_net310_flightmodel=>read_bookings
    EXPORTING
      i_carrid   = stru_flighttab-carrid
      i_connid   = stru_flighttab-connid
      i_fldate   = stru_flighttab-fldate
    IMPORTING
      e_bookings = itab_bookingtab.

* bind all the elements
  node->bind_table(
    new_items =  itab_bookingtab
    set_initial_elements = abap_true ).

ENDMETHOD.
```

Lesson Summary

You should now be able to:

- List the hook methods that exist for the different controller types
- Explain in which order these hook methods are processed
- Create and call your own controller methods
- Use the standard controller attributes to access the controller functions, especially the controller context
- Define your own controller attributes and use them in the controller methods
- Access controller context and read, delete, change, or add collection elements

Unit Summary

You should now be able to:

- List the hook methods that exist for the different controller types
- Explain in which order these hook methods are processed
- Create and call your own controller methods
- Use the standard controller attributes to access the controller functions, especially the controller context
- Define your own controller attributes and use them in the controller methods
- Access controller context and read, delete, change, or add collection elements

Unit 12

363

Internationalization and Messages

Demos are available for all aspects explained in this unit.

Unit Overview

Internationalization of Web Dynpro applications is required, to be able offer the application in different target languages. Each text (for example, defined as a UI element property, text literal in source code) can be defined so that it can be translated. This is also true for texts that are used as messages on the client's screen. This unit focuses on how to define translatable texts and translatable messages and how to send the messages.

Unit Objectives

After completing this unit, you will be able to:

- Define translatable texts in the OTR or as text elements in an ABAP class
- Use OTR texts and text elements from ABAP classes to define translatable literals in the Web Dynpro environment
- Report messages based on T100 texts, OTR texts, and text elements from ABAP classes
- Report messages with and without connection to UI elements
- Define where messages are to be displayed on the screen

Unit Contents

Lesson: Internationalization and Messages

Lesson Duration: 70 Minutes

Lesson Overview

Offering a Web Dynpro application in different languages requires that all literals used in the application be defined so that they can be translated. At runtime, the literal can then be loaded in the relevant logon language.

There are several ways to make literals translatable. Text can be defined in the Online Text Repository (OTR), as a text element in an ABAP class, or in the ABAP Dictionary. All techniques will be discussed in this lesson.

When talking about translatable texts, messages have to be taken into account as well. Each text defined in the OTR, in the database table T100, or as a text element in an ABAP class can be sent as a message. This lesson will cover in detail what kind of messages exist and how these messages can be sent.

Lesson Objectives

After completing this lesson, you will be able to:

- Define translatable texts in the OTR or as text elements in an ABAP class
- Use OTR texts and text elements from ABAP classes to define translatable literals in the Web Dynpro environment
- Report messages based on T100 texts, OTR texts, and text elements from ABAP classes
- Report messages with and without connection to UI elements
- Define where messages are to be displayed on the screen

Use the demos from the package **NET310** to explain the content.

Business Example

You have created a Web Dynpro application. However, if the user provides wrong field values, your application crashes or does not work correctly. You want to check the user input and inform the user by displaying messages. These messages and all text displayed to the user should be translatable, since your application has to be offered in multiple languages.

The demo NET310_I18N_D1 can be used for all the techniques to define translatable texts. Analyze the coding of the DOINIT method of the view, to find out how assistance class texts can be accessed.

However, it is more instructive if you create and use your own OTR texts in your component. It is also better to define your own assistance class, text symbols, constants, and an alias for the interface method get_text() by yourself. Demonstrate how to use texts from the ABAP Dictionary as well.

Internationalization

Internationalization means that all language-dependent parts of applications have to be defined in a way that a language-specific version of the entity can be defined and that the correct version is used at runtime, based on the language the user used to log on.

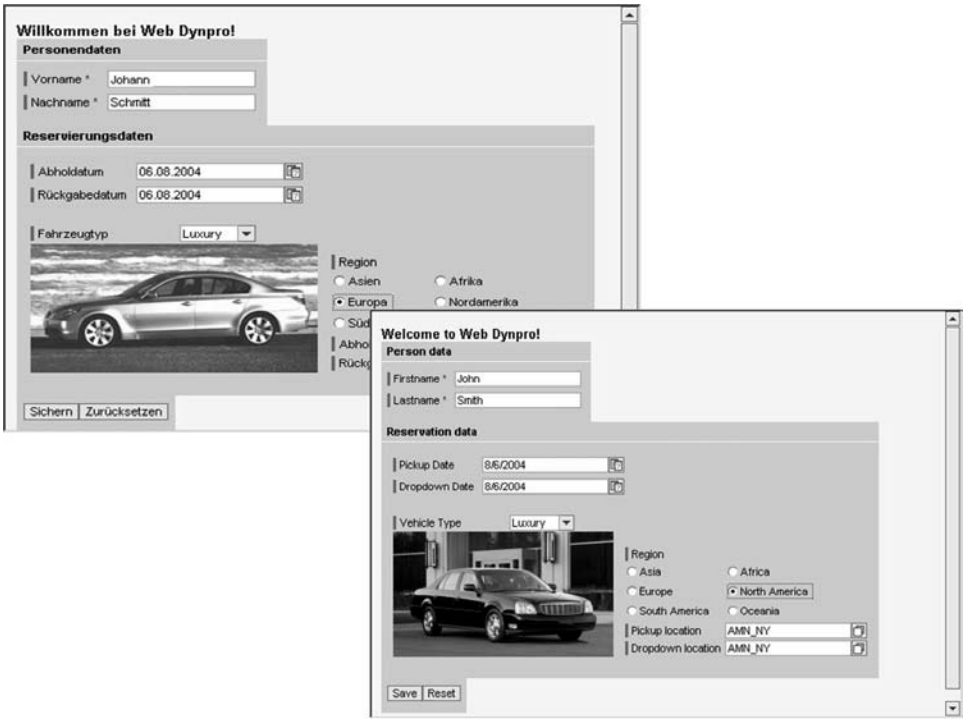


Figure 211: Internationalization

Note: Due to the fact that **internationalization** is such a long word, it is often abbreviated to **I18N**. That is, the first letter *I*, the last letter *N*, and don't bother about the 18 other letters in between.

Language-dependent entities include label texts, tool tips, button texts, message texts, and images.

Using Texts Defined in the ABAP Dictionary

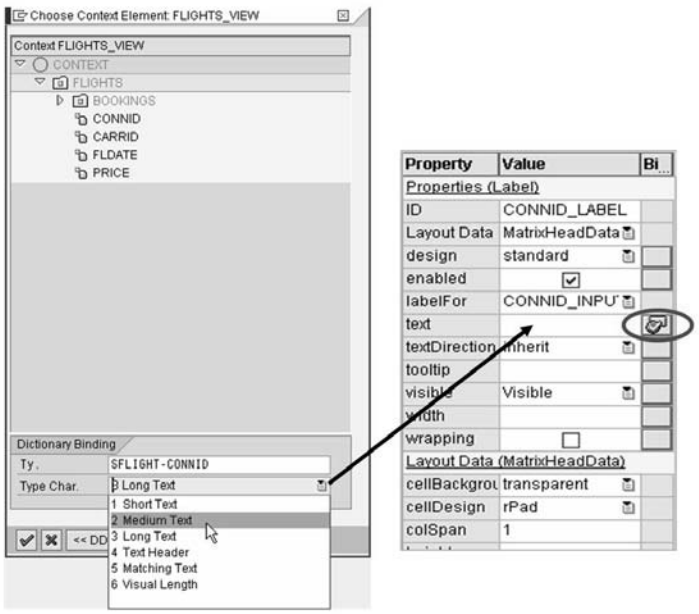


Figure 212: Referring to ABAP Dictionary Texts

Texts defined in the ABAP Dictionary can be used in the following ways:

- The UI element *Label* belongs to a UI element that allows user input (for example, *InputField*). This related UI element, in turn, must be bound to a context attribute in respect to the property that holds the field value. If the context attribute is typed with an ABAP Dictionary type, the corresponding data element text (middle text) will automatically be used as the label text.

  The same mechanism applies to *Captions* of *TableColumns*.

- Each data element text can also be referred directly. This is done by selecting the button to the right of a properties value field. On the dialog box that appears, select the button *DDIC binding on/off*, to choose a DDIC type and the text type (short, medium or long).
- From the source code, the method *get_available_texts( )* of the class *CL_TEXT_IDENTIFIER* can be used to read texts from the ABAP Dictionary.

Your component: Bind some dictionary texts to text properties of some UI elements of your views (for example, *Label* texts).

Defining and Using Online Text Repository (OTR) Texts

The Online Text Repository (OTR) is a central storage area for texts that can be used not only in a Web Dynpro context but also in BSPs, in classes, and in normal ABAP programs (type 1/type M). Different kinds of texts can be defined in the OTR: OTR long texts, OTR short texts, or alias texts.

The length of OTR long texts is not limited. However, the disadvantage is that they can only be used once. If you want to use the same long text a second time, it has to be rewritten in the original language again and (even worse) it has to be translated again. In the Web Dynpro context, OTR long texts are automatically created when a text is entered in a value field of a UI element's property.

OTR short texts are limited to 255 characters. However these texts can be reused, and they only have to be translated once. In the Web Dynpro context, only OTR short texts should be used.

The OTR provides services for accessing these texts at runtime, and it supports the entry and translation of texts. To create a new OTR short text, the Online Text Repository browser can be used. This tool can be found in the *Goto* menu when editing a view. The name of an OTR short text consists of the package name followed by a slash and an arbitrary identifier (<*package*>/<*alias*>). OTR texts can also be created using transaction SOTR_EDIT. This transaction also allows you to search for existing texts, translate texts, and find where OTR texts are used.

To use an OTR short text as the value of a UI element property, carry out the following steps:

- Select input help in the *Properties* value field. The OTR browser will appear, showing all texts of your package and of the *SOTR_VOCABULARY_BASIC* package, which contains standard texts.
- Select a text from the list and choose *Continue.*
- The OTR directive for using this text will be automatically entered in the *Properties* value field.

The OTR directive is a combination of the text's alias name, the package name, and the prefix OTR: *$OTR:*<*package*>/<*alias*>.

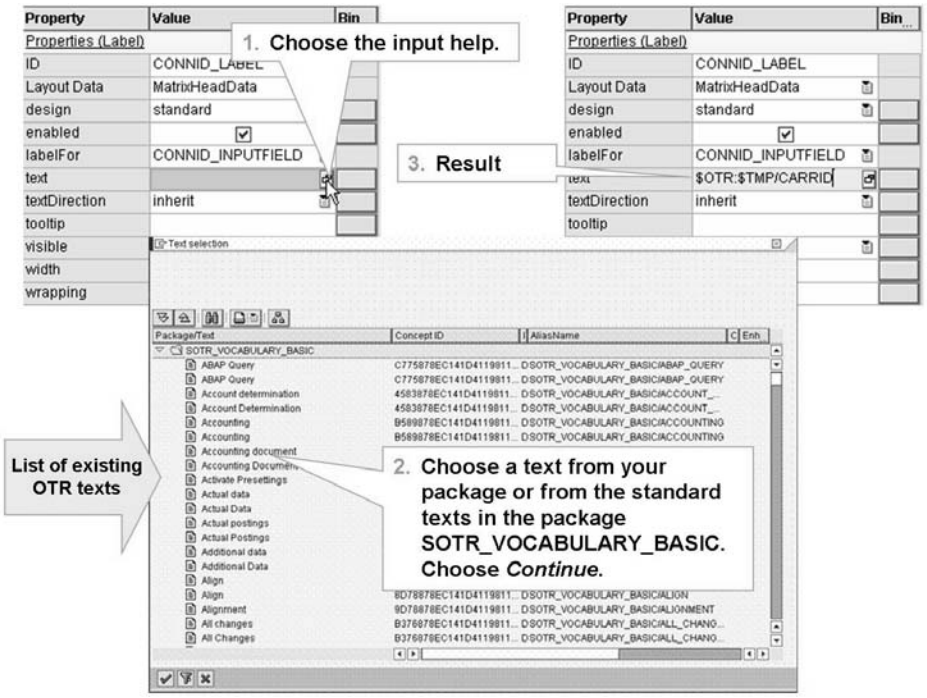


Figure 213: Using OTR Short Texts as UI Element Property Values

OTR short texts can also be accessed from a controller's source code. The class *CL_WD_UTILITIES* provides appropriate service methods. The method *get_otr_text_by_alias( )* allows you to access the value of an OTR short text on a language-dependent basis by providing the alias. The value returned is of the type *string*.

```
  DATA: lv_otr_text TYPE string.

  ...

* Get OTR short text in language sy-langu
  CALL METHOD cl_wd_utilities=>get_otr_text_by_alias
        EXPORTING
          alias = `NET310/CAPTION`
        RETURNING
          value = lv_otr_text.

 ...


* Set language-dependent value of single attribute
  CALL METHOD elem_lable_text->set_attribute
        EXPORTING
          name  = `GROUP1_CAPTION`
          value = lv_otr_text.
```

Figure 214: Using OTR Short Texts from the Controller Source Code

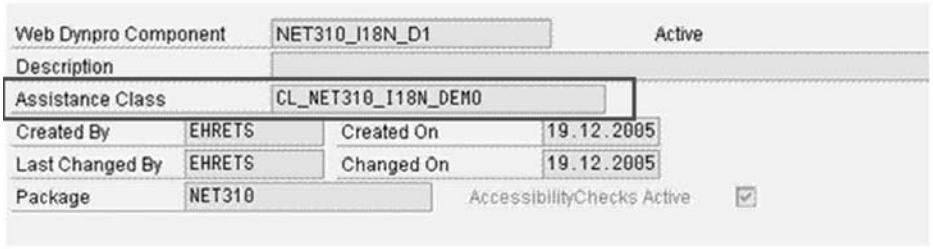
Your component: First, create some OTR texts using the OTR Browser from the menu. Then restart the ABAP Workbench (because it contains bugs) before you access the texts. Finally, use the input help for the value fields, to assign UI element properties to your OTR texts.

Using Texts Defined in an ABAP Class

Instead of using texts defined in the OTR, it is possible to define the texts as text elements in an ABAP class. To access the text element values, use a public method of the class. A class that provides methods to read the text elements is already available in each SAP NetWeaver AS 7.0. It is called *CL_WD_COMPONENT_ASSISTANCE*. Each class that inherits from this parent class will have the method *get_text( )*. Passing the identifier of the text symbol to the method will return the text in the language *sy-langu*.

To use this class in a Web Dynpro component, it has to be instantiated in each controller. However, an easier and more consistent way is to let the Web Dynpro runtime create the instance only once. This is done by writing the class name in the *Assistance Class* field, which can be found on the *Properties* tab for the Web Dynpro component. The individual steps are:

- Edit the Web Dynprpo component. Navigate to the *Properties* tab of the component.
- Enter a class name in the *Assistance Class* field.
- Double-click the class name. If the class does not exist, it will be created. The class will be derived from the parent class *CL_WD_COMPONENT_ASSISTANCE*.



If an assistance class is assigned, an instance of this class is automatically instantiated.

This instance can be addressed using the attribute `wd_Assist`, available in every controller of the component.

Figure 215: Using the Component Assistance Class

Text symbols can be added to the class by choosing *Goto → Text Symbols.* Be sure that the length of the text elements is long enough to support different languages.

- Optionally, you can define a public alias for the interface method *get_text( )* (just for convenience).
- If you want to refer to the text elements using arbitrary identifiers, you can define constants of the type WDR_TEXT_KEY for each text element. The constant value corresponds to the text element identifier in apostrophes.

Once you have assigned the assistance class to the component, the attribute *WD_ASSIST* is added to the attribute list of each controller. This attribute is set by the Web Dynpro runtime to the assistance class instance. Using *WD_ASSIST*, a text can be accessed as follows:

- Using the text element identifier *CAR*:

```
DATA: lv_text TYPE string.
lv_text = wd_assist->if_wd_component_assistance~get_text(
                          key = 'CAR' ).
```

- Using a constant *CARRID* that has the text element identifier *CAR* as its value:

```
DATA: lv_text TYPE string.
lv_text = wd_assist->if_wd_component_assistance~get_text(
                          key = wd_assist->carrid ).
```

Your component: Create an assistance class. Define a text symbol without any placeholder and another text symbol with one or more placeholders (names: &PARA1&, ...). Define class constants with better names. Define an alias get_text for the interface method if_wd_component_assistance~get_text().

From any method of any controller of your component: Use the ABAP Code Wizard, to generate the method call of the get_text() method. Use the text symbol's identifier (literal) or the class constant to access the text symbol. Use the alias or complete interface method name. For the method with the placeholders, hand over literals to replace the placeholders. Execute debugging.

Texts defined in the assistance class can also have **placeholders**. These are defined by using an ampersand (&), followed by the placeholder's name. In general, there are no restrictions regarding the number and names of the placeholders.

The method *get_text( )* allows you to export values for the placeholders. It returns the corresponding text in the logon language, with the placeholder substituted by the provided values.

Caution: Only a maximum of four placeholders with the names (including the starting ampersand) *&PARA1& ...&PARA4&* are replaced by the method *get_text( )*. Here, the placeholder names must be defined in capital letters.

If the text is obtained without providing values for the placeholders, the text is returned as defined. The placeholders can then later be substituted by another algorithm (for example, if the text is used as the text of a message).

Not shown here: You can translate the OTR texts or the text symbols in the SE63 - if you would like to show that.

Exercise “Translatable Texts in the UI”

Reporting Messages

The source code for reporting messages can be generated using the Web Dynpro Code Wizard. The generated code consists of a common part, containing the source code for obtaining the reference to the message manager, and a part that is dependent on the method to be called. The message manager is responsible for collecting and sending the messages reported in the actual component and all subcomponents.

Since obtaining the reference to the message manager is obligatory before a message can be reported, the corresponding source code should be moved to the standard hook method *wddoinit()* and the reference should be stored in a user-defined controller attribute.

Use the Web Dynpro Code Wizard to create the coding required to report a message.


```
DATA: l_current_controller TYPE REF TO if_wd_controller,
      l_message_manager    TYPE REF TO if_wd_message_manager.


* get reference to actual component controller
l_current_controller ?= wd_this->wd_get_api( ).


* get reference to message manager
CALL METHOD l_current_controller->get_message_manager
  RECEIVING
    message_manager = l_message_manager

* report message
. . .
```

Figure 216: Reporting a Message: Common Part

Your component: Report any message using the WD Code Wizard. Restrict your discussion to the general code first.

Category TEXT

All methods for reporting messages of the category *TEXT* have a common interface. At least the message text has to be exported using the parameter *message_text*. If the message text contains placeholders, the parameter list must be defined before calling the method. Placeholders in message texts can have arbitrary names (in the example below, *X1*). If a method from the category *TEXT* belongs to a UI element, a reference to the context element and the name of the attribute in this context element, which contains the faulty value, have to be provided. As this attribute is bound to the property of a UI element, the Web Dynpro runtime can relate the message to this UI element.

```
DATA: lv_text    TYPE string,
      lt_params  TYPE wdr_name_value_list,
      ls_param   TYPE wdr_name_value.

* get translatable text defined as text symbol in assistance class
lv_text = wd_assist->if_wd_component_assistance~get_text( … ).

* fill parameter table
* (text contains placeholder
ls_param-name  = 'X1'.
ls_param-value = 'LH'.
APPEND ls_param TO lt_params.


* report text message related to UI element
* wd_this->gr_element_flight is reference to context element
* FLIGHT at lead selection
CALL METHOD l_message_manager->report_attribute_error_message
  EXPORTING
    message_text   = lv_text
    params         = lt_params
    element        = wd_this->gr_element_flight
    attribute_name = 'CARRID'.
```

lv_text = 'Flight carrier &X1 not found'

Figure 217: Example: Reporting messages:Category TEXT

You can show all kinds of TEXT messages using demo NET310_I18N_D3. Error messages, warnings, and success messages are reported. Some of them are related to context attributes, some have placeholders. Discuss in detail.

Category T100

To use texts already defined in database table T100, two methods exist. You must enter the message number, the message class, and the message type. If the message text contains placeholders, appropriate values can be exported. Depending on the method, all message components are exported by using scalar parameters or by using an export structure At least the message text has to be exported using the parameter *message_text*.

If a method from the category *T100* belongs to a UI element, a reference to the context element and the name of the attribute in this context element, which contains the faulty value, have to be provided. As this attribute is bound to the property of a UI element, the Web Dynpro runtime can then relate the message to this UI element.

```
DATA: ls_message TYPE symsg.

* report T100 message using placeholder P1
CALL METHOD l_message_manager->report_t100_message
  EXPORTING
    msgid = 'NET310'
    msgno = '001'
    msgty = 'E'
    P1    = 'LH'.

* define message fragments
ls_message-msgid = 'NET310'.
ls_message-msgno = '001'.
ls_message-msgty = 'E'
ls_message-msgv1 = 'LH'.

* report T100 message related to UI element using placeholder P1
CALL METHOD l_message_manager->report_attribute_t100_message
  EXPORTING
    msg            = ls_message
    element        = wd_this->gr_element_flight
    attribute_name = 'CARRID'.
```

Message 001(NET310):
'Flight carrier &1 not found'

Figure 218: Reporting messages: Category T100

Demo NET310_I18N_D2 shows all kinds of T100 messages. Discuss in detail.

Category EXCEPTIONS

Runtime exceptions can be caught in an exception object, which then contains the text for the error. This text is defined in the exception class and therefore cannot be influenced. The message object can be used to create a Web Dynpro message by calling methods of the category *EXCEPTIONS*. The message object is exported using the parameter *message_object.* If a method from the category *EXCEPTIONS* belongs to a UI element, a reference to the context element and the name of the attribute in this context element, which contains the faulty value, have to be provided. Since this attribute is bound to the property of a UI element, the Web Dynpro runtime can then relate the message to this UI element.

```
DATA: lr_msg_obj TYPE REF TO cx_root
      lv_result  TYPE p DECIMALS 3 LENGTH 3.


* report exception message related to UI element
TRY.

   lv_result = wd_this->gv_integer1 / wd_this->gv_integer2.
 CATCH cx_root INTO lr_msg_obj.
```

Message text defined in exception class

```
   CALL METHOD l_message_manager->report_attribute_exception
    EXPORTING
      message_object = lr_msg_obj
     element    = wd_this->gr_elem_calculator
     attribute_name = 'INTEGER2'.

ENDTRY.
```

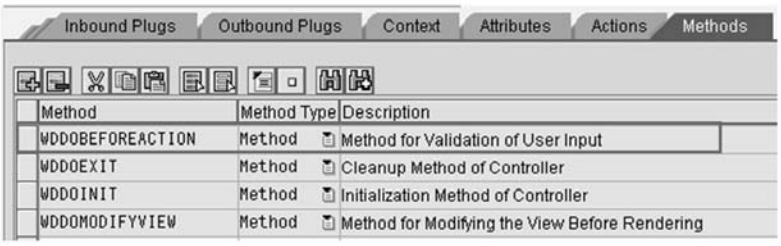
Figure 219: Reporting messages: Category EXCEPTIONS

Demo NET310_I18N_D4 shows how to report an exception message. Discuss in detail.

Coding for Reporting Messages

If validation coding is only to be performed for a certain user action, it can be placed in the corresponding action handler method (*onaction<action>*). However, if multiple action handler methods exist, some validations may have to be performed in more than one of the action handler methods. In this case, it is a good idea to put the coding in an extra method in order to have it written only once.

For this reason, a special hook method exists for each view, which is processed before any of the action handler methods is called. The name of this hook method is *wddobeforeaction( )*. If a screen is composed of multiple views (nested views), then all *wddobeforeaction( )* methods are processed before any action handler method is called.



WDDOBEFOREACTION( )

- **Only a view controller has this hook method.**
- **Method is processed before any action handler method is processed.**
- **Method can therefore be used for data validation and reporting related messages.**

Figure 220: Reporting messages: Hook method WDDOBEFOREACTION()

Validation-Dependent Processing

The processing of the view's hook methods and the navigation may depend on the type of the triggered action and on the type of messages reported in the hook method *wddobeforeaction( )*:

If the action is validation-independent (*action type = validation-independent*), the messages reported in *wddobeforeaction( )* have NO influence on the processing of the hook methods or on the navigation.

If the action is validation-dependent (*action type = standard*), the following message types will cancel the navigation and prevent the processing of the action handler method and the hook method *wddomofifyview( )*:

Fatal error messages

Error messages related to context attributes

Reporting messages in the action handler method has no influence on the navigation or the processing of the hook methods.

Exercise "Value Checks and Messages"


377

Exercise 21: Internationalization: Translatable Text in the UI

Exercise Duration: 10 Minutes

Exercise Objectives

After completing this exercise, you will be able to:

- Create OTR texts
- Use OTR texts for UI elements

Business Example

You want to develop a Web Dynpro application where all the UI texts can be translated. For those texts that are not retrieved from ABAP Dictionary, you will use the OTR.

Template: NET310_CONR_S3

Solution: NET310_I18N_S1

Task:

Copy your Web Dynpro component **ZNET310_CONR3_##** or the template **NET310_CONR_S3** to the WebDynpro component **ZNET310_I18N1_##**.

Use OTR texts for group captions and button texts.

1. Copy the template.
2. Create the required OTR texts in your package (identifiers start with **ZNET310_##/**).
3. For all UI elements of the type GROUP or BUTTON, set the property *text*, with a reference to an OTR text.

Solution 21: Internationalization: Translatable Text in the UI

Task:

Copy your Web Dynpro component **ZNET310_CONR3_##** or the template **NET310_CONR_S3** to the WebDynpro component **ZNET310_I18N1_##**.

Use OTR texts for group captions and button texts.

1. Copy the template.
   a) Perform this step as in previous exercises.
2. Create the required OTR texts in your package (identifiers start with **ZNET310_##**/).
   a) Choose *Goto* → *Online Text Repository Browser*.
   b) Choose *Create* .
   c) Enter the identifier, the length, and the text. Choose *Save* .
3. For all UI elements of the type GROUP or BUTTON, set the property *text*, with a reference to an OTR text.
   a) Edit the respective UI element and open the value help for the property *text*.
   b) Double-click the appropriate OTR text in the list.

Lesson Summary

You should now be able to:

- Define translatable texts in the OTR or as text elements in an ABAP class
- Use OTR texts and text elements from ABAP classes to define translatable literals in the Web Dynpro environment
- Report messages based on T100 texts, OTR texts, and text elements from ABAP classes
- Report messages with and without connection to UI elements
- Define where messages are to be displayed on the screen

Unit Summary

You should now be able to:

- Define translatable texts in the OTR or as text elements in an ABAP class
- Use OTR texts and text elements from ABAP classes to define translatable literals in the Web Dynpro environment
- Report messages based on T100 texts, OTR texts, and text elements from ABAP classes
- Report messages with and without connection to UI elements
- Define where messages are to be displayed on the screen

Course Summary

You should now be able to:

- Use fundamental elements of object-oriented modeling in UML
- Create ABAP Objects programs that contain all useful object-oriented programming techniques
- Use the relevant tools to create object-oriented Repository objects
- Describe and exploit the range of applications of ABAP Objects
- Define, raise, and handle class-based exceptions
- Query type and class attributes at runtime
- Make specialized changes to the SAP standard system
- Evaluate the different methods for modification and choose the right one for any given situation
- Explain the architecture of a Web Dynpro component
- Describe the parts of a Web Dynpro controller
- Create context elements in the Web Dynpro controller
- Explain how navigation and data transfer in and between Web Dynpro components can be implemented
- Define the UI of a Web Dynpro component
- Internationalize a Web Dynpro application
- Define and send messages in a Web Dynpro component
- Define input help and semantic help for UI elements in a Web Dynpro component

Related Information

- Use a URL or a cross-reference tag to point out additional information that the participants may find useful,such as Web sites or White Papers. Delete this if it is not relevant.

Index

Feedback

SAP AG has made every effort in the preparation of this course to ensure the accuracy and completeness of the materials. If you have any corrections or suggestions for improvement, please record them in the appropriate place in the course evaluation.