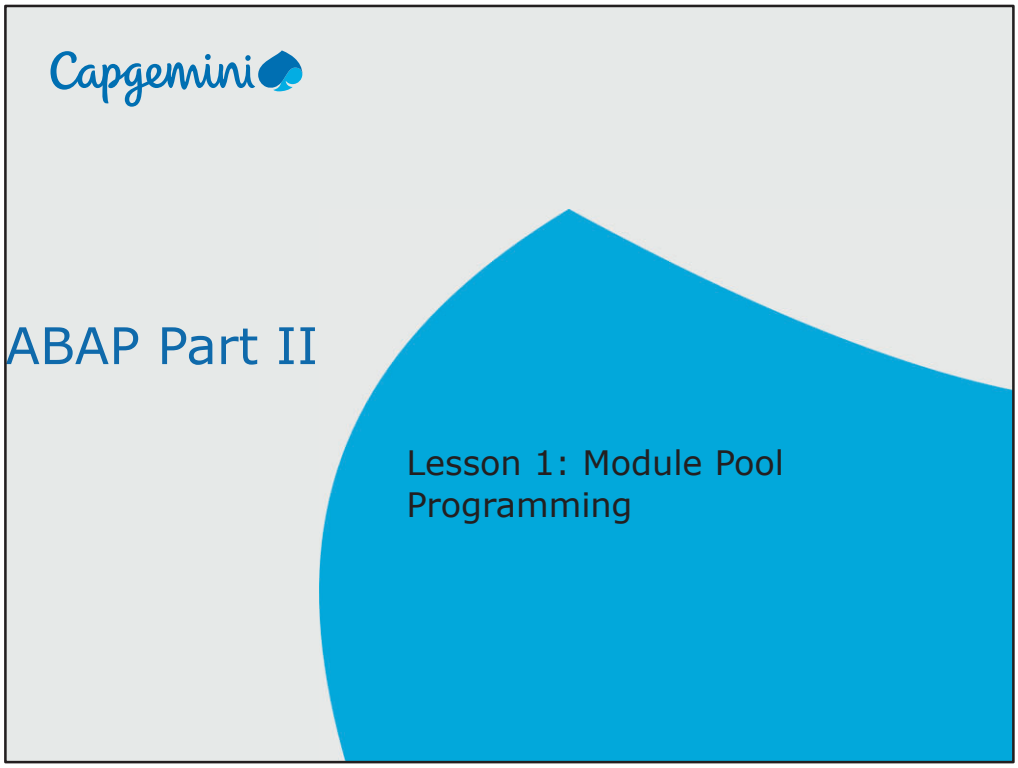

Capgemini
ABAP Part II
Lesson 1: Module Pool Programming

Lesson Objectives

After completing this lesson, participants will be able to –

- Work on Module pool programming
- Understand Tools for developing Module pool programming
- Work with Screen Painter
- Work with Flow Logic
- Know the Types of Events
- Know the GUI status & Messages
- Work on Screen commands & LUW
- Work on Table controls and Tab strips

Introduction to Module Pool Programming

Module Pool Program (also called online programs, dialog programs, or transactions) do not produces lists. These programs are a collection of screens, their Flow Logic, and the code within the main ABAP program.

Dialog Programming: Dialog-driven programs, or any program started using a transaction code, are known as SAP transactions, or just transactions. The term "transaction" stands for a user request.

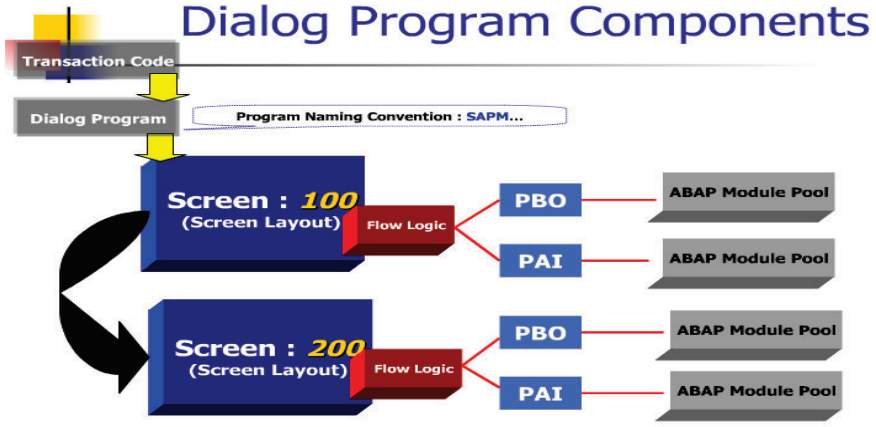
Programs with type M can only be started using a transaction code, in which an initial screen is defined.

Screens call dialog modules in the associated ABAP program from their flow logic.

Type M programs serve principally as containers for dialog modules, and are therefore known as Module Pools**.**

Introduction to Module Pool Programming

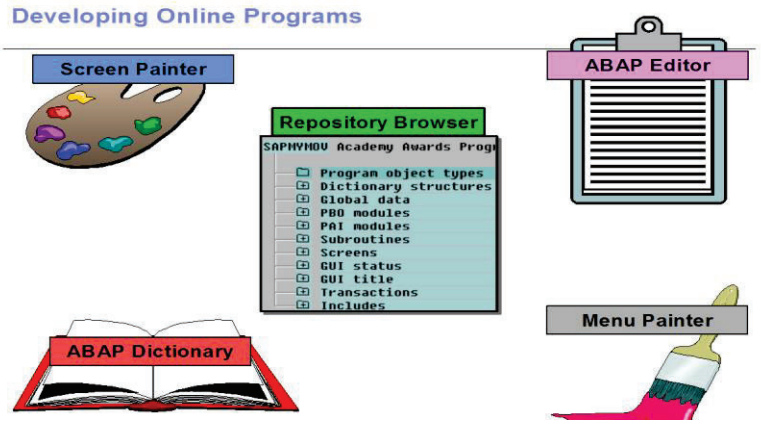
Dialog driven program consist of the following basic components.



Capgemini Public

Tools for Developing Module Pool Programming

When creating an module pool program, you will use many tools within the ABAP Development Workbench: Screen Painter, ABAP Editor, Menu Painter, ABAP Dictionary and Repository Browser.



Tools for Developing Module Pool Programming

ABAP Editor (SE38) – To maintain main ABAP program.

- Program contains data and modules declarations.

Screen Painter (SE51) – Used to maintain components of screen.

Menu Painter (SE41) – Used to design the GUI.

Maintain Transaction (SE93) – To create user defined transaction code for the program.

Object Navigator (Repository Browser) (SE80) - You should always use the object navigator for online programs because you will be able to see the hierarchy. From this hierarchy list, you will be able to branch to the Screen Painter, ABAP Editor, Menu Painter, and ABAP Dictionary.

Screen Painter

The Screen Painter is a ABAP Workbench tool that allows you to create screens for your transactions.

Also called Dynpro which is Dynamic Programing.

It will allow you to create the screen itself, with fields and other graphical elements, and to write the flow logic behind the screen.

You define screen in the screen painter.

Screens are the most general type of user dialog that you can use in ABAP programs.

Screen Painter

Consists of input/output and the screen flow logic

Screens can be defined for

- Executable Program
- Module Pool
- Function Group

Data passed between screen fields and data objects in ABAP program.

Data is copied between identically-named fields.

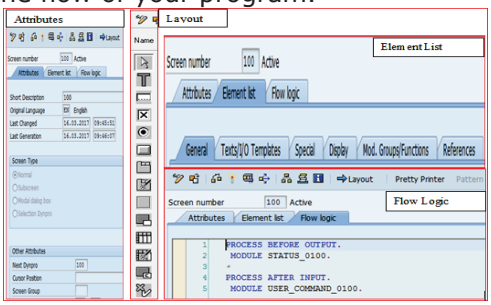
Screen Painter

Screen Attributes Screen attributes include the program the screen belongs to and the screen type.

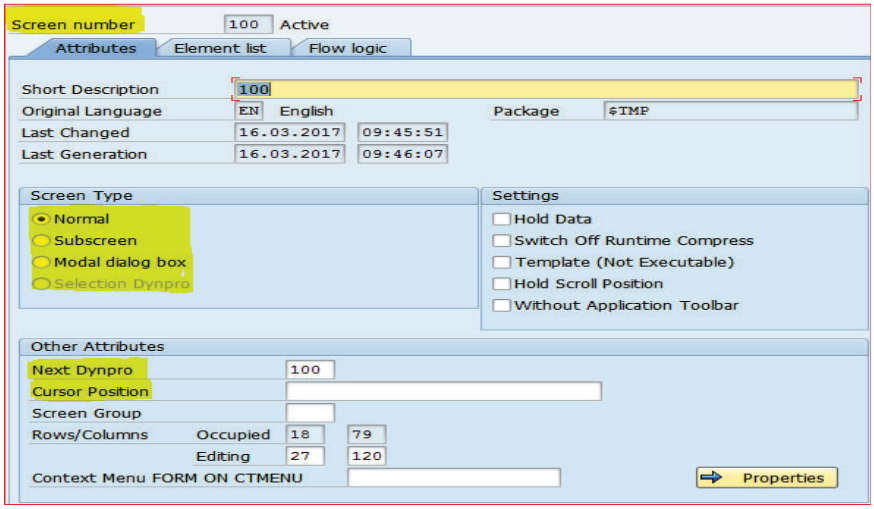
Screen Layout Screen elements are the parts of a screen with which the user interacts.

Elements Correspond to screen elements. Fields are defined in the ABAP Dictionary or in your program

Flow Logic Controls the flow of your program.



Screen Attributes



Capgemini Public

Screens have attributes that both describe them and determine how they behave at runtime.
1.Program: The name of the ABAP program (type 1, M, or F) to which the screen belongs.

2.Screen Number: A four-digit number, unique within the ABAP program, that identifies the screen within the program.

Note: a screen with screen number 1000 should not be created. It is reserved for standard selection screen.

3.Screen Type:
Normal Screen (occupies a whole GUI window)
Modal Dialog Box (occupies a whole popup window)
Sub screen (to display a screen within another screen (Normal Screen))

4.Next Screen: Statically-defined screen number, specifying the next screen in the sequence.

5.Cursor Position: Static definition of the screen element on which the cursor is positioned when the screen is displayed. You can overwrite the static cursor position dynamically in your ABAP program.

6.Screen Group: Four-character ID, placed in the system field SY-DYNGR while the screen is being processed.

7.This allows you to assign several screens to a common screen group. You can use this for example to modify all of the screens in the group in a uniform way.

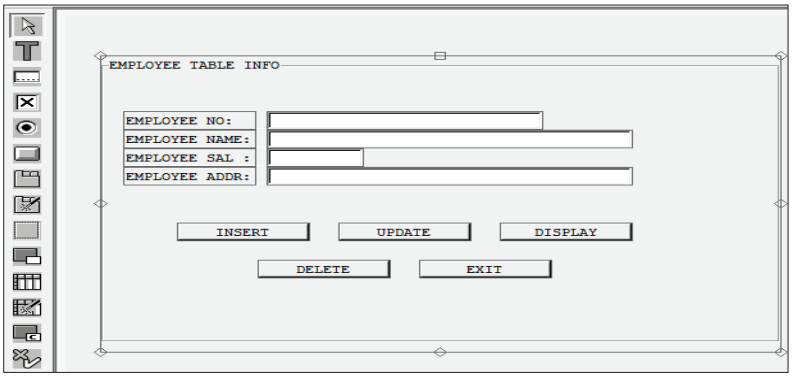
8.Hold Data: If the user calls the screen more than once during a terminal session, he or she can retain changed data as default values by choosing System -> User profile -> Hold data.

Demo: Screen Interface to show screen Attributes

Screen Modes

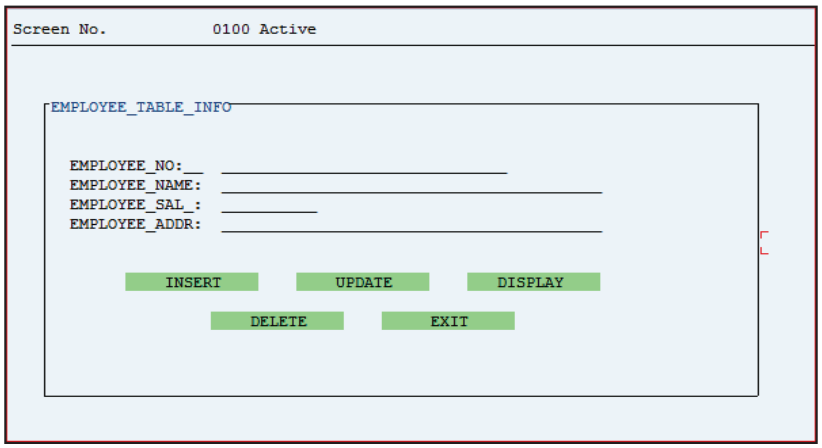
The Screen Painter has a layout editor that you use to design your screen layout. It works in two modes:

- Graphical Mode: The graphical layout editor provides a user-friendly environment for designing the screens.



Screen Modes

- Alphanumeric Mode: The graphical full screen editor provides a user-friendly environment for designing the screens on all platforms.



Screen Elements

Screen can contain a variety of elements, either for displaying field contents or for allowing the user to interact with the program.

- Example: Filling out input fields or choosing pushbutton functions.

Screen elements have a set of attributes, some of which are set automatically, others of which have to be specified in the Screen Painter.

Screen Elements:

1.Text Fields: Display elements, which cannot be changed either by the user or by the ABAP program.

2.Input/Output Fields: Used to display data from the ABAP program or for entering data on the screen and linked to screen fields.

3.Dropdown List Boxes: Special input/output fields that allow users to choose one entry from a fixed list of possible entries.

Screen Elements

4.Check Boxes: Special input/output fields which the user can select (value 'X') or deselect (value SPACE).

5.Radio Buttons: Special input/output fields that are combined into groups. Within a radio button group, only a single button can be selected at any one time. When the user selects one button, all of the others are automatically deselected.

6.Push Buttons: It triggers the PAI event of the screen flow logic when chosen by the user. There is a function code attached to each pushbutton.

7.Box: Display element, used for visual grouping of related elements on the screen, such as radio button groups.

8.Sub Screens**:** Area on the screen in which you can place another screen.

Screen Elements

9.Table Controls: Tabular input/output fields.

10.Tab Strip Controls: Areas on the screen in which you can switch between various pages.

11.Custom Control: Container on the screen in which you can display another control.

12.Status Icon: Display elements, indicating the status of the application program.

13.OK_CODE Field: Every screen has a twenty-character OK_CODE field (also known as the function code field), which is not displayed on the screen.

Flow Logic

Flow logic contains the procedural part of a screen.

The language used to program screen flow logic has a similar syntax to ABAP, but is not part of ABAP itself. It is sometimes referred to as screen language.

The screen flow logic is like an ABAP program in that it serves as a container for processing blocks.

The event block is introduced by the corresponding keyword statement. The first two statements are created automatically by the screen painter when you create a new screen.

There are four event blocks:

- PROCESS BEFORE OUTPUT (PBO).
- PROCESS AFTER INPUT (PAI).
- PROCESS ON HELP-REQUEST (POH).
- PROCESS ON VALUE-REQUEST (POV).

Flow Logic

PROCESS BEFORE OUTPUT (PBO) is automatically triggered after the PAI processing of the previous screen and before the current screen is displayed. At the end of the PBO processing the screen is displayed.

PROCESS AFTER INPUT (PAI) is triggered when the user chooses a function on the screen. At the end of the PAI processing, the system either calls the next screen or carries on processing at the point from which the screen was called.

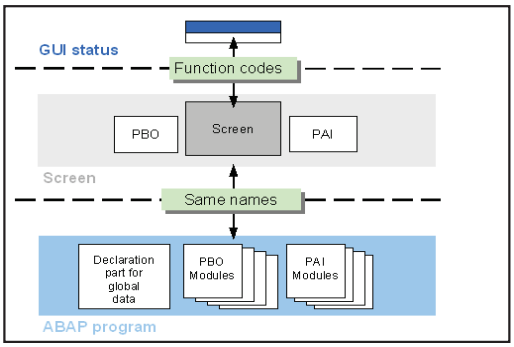
PROCESS ON HELP-REQUEST (POH) and PROCESS ON VALUE-REQUEST (POV) are triggered when the user requests field help (F1) or possible values help (F4) respectively. At the end of processing, the system carries on processing the current screen.

The screen flow logic must contain at least the two statements PROCESS BEFORE OUTPUT and PROCESS AFTER INPUT in the correct order.

Flow Logic

The screen flow logic calls dialog modules in the ABAP program, either to prepare the screen for display (PBO event) or to process the user's entries (PAI event). Screens are dynamic programs, and have their own data objects, called screen fields.

When the screen is displayed, and when it finishes processing, the system passes data between the screen fields and data objects in the ABAP program.



Flow Logic

Within the event blocks you can use the following keywords:

Keyword	Function
MODULE	Calls a dialog module in an ABAP Program.
FIELD	Specifies the point at which the contents of a screen field should be transported.
ON	Used in conjunction with FIELD.
VALUES	Used in conjunction with FIELD.
CHAIN	Starts a processing chain.
ENDCHAIN	Ends a processing chain.
CALL	Calls a sub screen.
LOOP	Starts processing a screen table.
ENDLOOP	Stops processing a screen table.

Flow Logic

The main task of the screen flow logic is to call dialog modules in an ABAP program.

This can be done using the MODULE statement, which can be programmed in the four possible event blocks of screen flow logic.

PBO Statement:

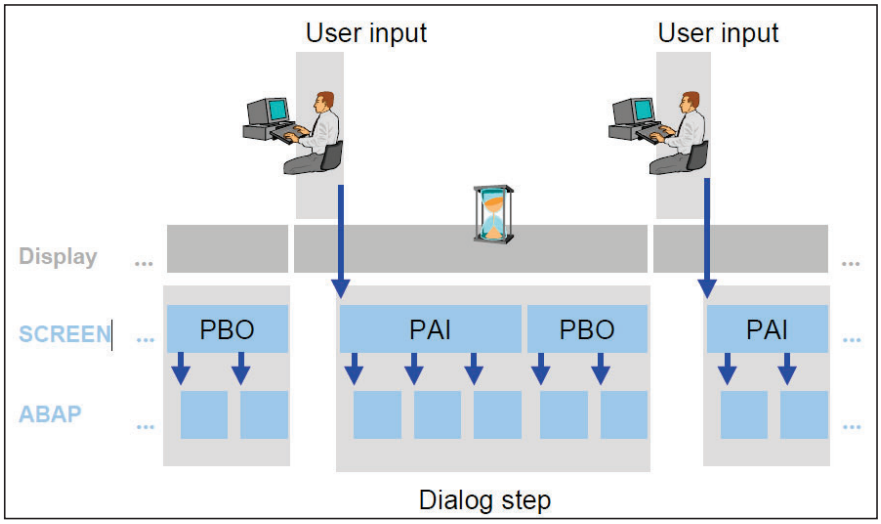
```
MODULE <mod> OUTPUT.
...
ENDMODULE
```

PAI Statement:

```
MODULE <mod> INPUT.
...
ENDMODULE
```

Flow Logic

Screen Sequence is as follows:



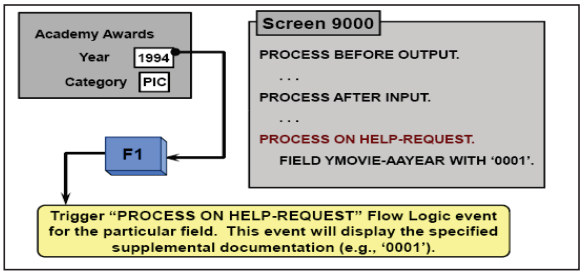
Types of Events

Process on Help Request (POH)

- This event is a user programmed help.
- If the user presses the 'F1' key with the cursor is positioned in <screen field>, the <supplemental documentation> will be displayed along with the data element's short text and documentation.
- The only other Flow Logic statement that can be used in the POH event is:

Syntax is: FIELD <screen field> MODULE <module>.

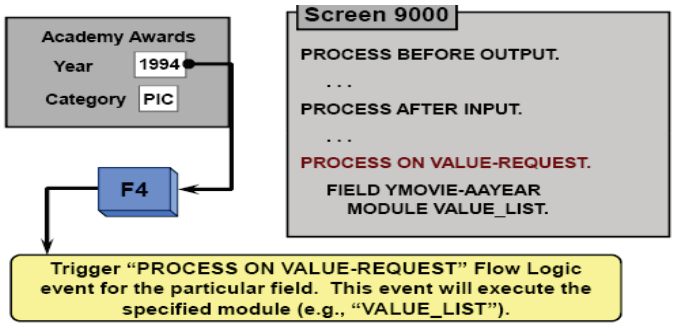
on Help Request (POH)



Types of Events

Process on Value Request (POV)

- This event is a user-programmed help that occurs when the user presses F4 with the cursor positioned on a screen field.
- The modules specified in the subsequent FIELD statement is called instead of the SAP help. **Syntax is: FIELD <screen field> MODULE <module>.**



Types of Events

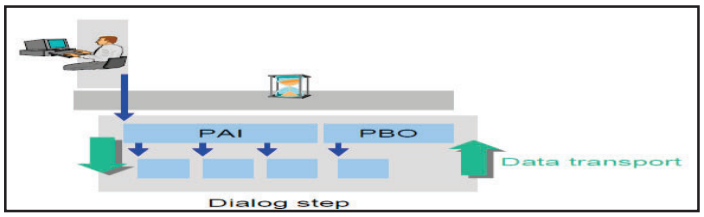
Calling Dialog Modules

- There are various ways of calling the dialog modules in the flow logic. The syntax of the flow logic allows you to call dialog modules conditionally and to control the transfer of data between the screen and the ABAP program.
  - Simple Module Calls
  - Controlling the Data Transfer
  - Calling Modules Unconditionally
  - Calling Modules Conditionally

Types of Events

Simple Module Calls

- 1. The system starts the module <mod>, which must have been defined for the same event block in which the call occurs.
- 2. Use simple modules in the screen flow logic, the data transport between the ABAP program and the screen is as follows:
- 3. In the PAI event, all of the data from the screen is transported to the ABAP program.
- 4. At the end of the last PBO module, and before the screen is displayed, all of the data is transported from the ABAP program to any identically-named fields in the screen.



Types of Events

Controlling the Data Transfer

- 1.The FIELD statement in the screen flow logic allows you to control the moment at which data is passed from screen fields to their corresponding ABAP fields.
- 2.To specify this point, use the following statement in the PAI flow logic: FIELD <f>.
- 3.Data is not transported from the screen field <f> into the ABAP field <f> until the FIELD statement is processed.
- 4.Do not use fields in PAI modules until they have been passed to the program from the screen, otherwise the ABAP field will contain the same value as at the end of the previous dialog step.
- 5.The basic syntax of these statements is: FIELD: <f1>, <f 2>,...MODULE <mod1>.

Types of Events

Calling Modules Unconditionally

- 1. AT EXIT-COMMAND - With the "AT EXIT-COMMAND" addition to the "MODULE" statement a module will be executed only if the user invokes a function code with the 'E' function type.
- 2. Regardless of where it occurs in the screen flow logic, this statement is executed immediately and before the automatic checks for the field contents on the screen.
- 3. Before the module <mod> is executed the contents of the OK-CODE field are transported to the ABAP field with the same name. However no other screen fields are transported to the program at this stage.
- 4. The basic syntax for this conditional execution Flow Logic command is: MODULE <module> AT EXIT-COMMAND.

Types of Events

Simple module calls are processed in the sequence in which they appear in the screen flow logic.

PAI module calls dependent on certain conditions by using the MODULE statement together with the FIELD statement. You can apply conditions to both single fields and groups of fields.

Conditional module calls can help you to reduce the runtime of your program particularly with modules that communicate with database tables.

Conditions for Single Screen Fields

- ON INPUT
- ON REQUEST
- FIELD <f> MODULE <mod> ON INPUT|REQUEST|*-INPUT.

Types of Events

On Input

- The ABAP module is called only if the field contains a value other than its initial value.
- This initial value is determined by the data type of the field: Space for character fields and zero for numeric fields.
- Even if the user enters the initial value of the screen as the initial value, the module is not called.
- The basic syntax for this conditional execution Flow Logic command is:
  - FIELD <screen field> MODULE <module> ON INPUT
  - The PAI <module> will be executed only if the value in <screen field> is not equal to its initial value.
  - The "ON INPUT" addition must be used with a "FIELD" statement because this condition depends on the value of a particular field.

Demo: Module Pool

Types of Events

On Request

- The module <mod> is only called if the user has entered something in the field.
- This includes cases when the user overwrites an existing value with the same value or explicitly enters the initial value.
- In general, the ON REQUEST condition is triggered through any form of "Manual Input".
- The basic syntax for this conditional execution Flow Logic command is:

  Syntax: FIELD <screen field> MODULE <module> ON REQUEST**.**

Types of Events

Conditions for Multiple Screen Fields

- 1.To ensure that one or more PAI modules are only called when several screen fields meet a particular condition you must combine the calls in the flow logic to form a processing chain. You define processing chains as follows:
- 2.All flow logic statements between CHAIN and ENDCHAIN belong to a processing chain.
- 3.The fields in the various FIELD statements are combined and can be used in shared conditions.

```
CHAIN.
  FIELD:  <f1>,  <f2>,...
  MODULE <mod1> ON CHAIN-INPUT|CHAIN-REQUEST.
  FIELD: <g1>, <g2>,...
  MODULE <mod2> ON CHAIN-INPUT|CHAIN-REQUEST.
  ...
ENDCHAIN.
```

Types of Events

Conditions Modules after Cursor Selection

- 1.You can specify that a module should only be called if the cursor is positioned on a particular screen element. To do this use the statement.
- MODULE <mod> AT CURSOR-SELECTION.
- 2.The module <mod> is called whenever the function code of the user action is CS with function type S.
- 3.If you use this statement, it is best to assign the function code CS to function key F2. This also assigns it to the mouse double-click.
- 4.The function code is empty, and neither SY-UCOMM nor the OK_CODE field is affected. You can also combine this MODULE statement with the FIELD statement: FIELD <f> MODULE <mod> AT CURSOR-SELECTION.

GUI Status

A GUI status is an independent component of an ABAP program.

The function of a GUI status is to provide the user with a range of functions on a screen.

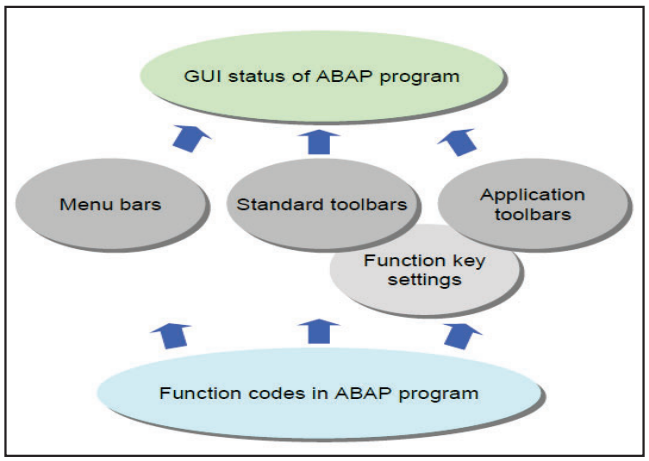
Each function has an associated function code, and when the user chooses a function, the PAI event is triggered.

Each PAI event, the function code, as long as it is not empty, is placed in the system field SYST-UCOMM (SY-UCOMM) and assigned to the OK_CODE field.

All function codes in an ABAP program, apart from those only assigned to pushbuttons on screens, are defined and administered in the Menu Painter (Transaction Code: SE41).

GUI Status

A GUI status consists of a menu bar, a standard toolbar, an application toolbar, and a function key setting.



GUI Status

Setting the GUI status: To assign a GUI status to a screen, use the ABAP statement SET PF-STATUS <stat>This statement defines the user interface for all subsequent screens of a screen sequence until another is set using a new SET PF-STATUS statement.

Setting the GUI Title: To assign a GUI status to a screen, use the ABAP statement SET TITLEBAR <title>.This statement defines the title of the user interface for all subsequent screens of a screen sequence until another is set using a new SET TITLEBAR statement

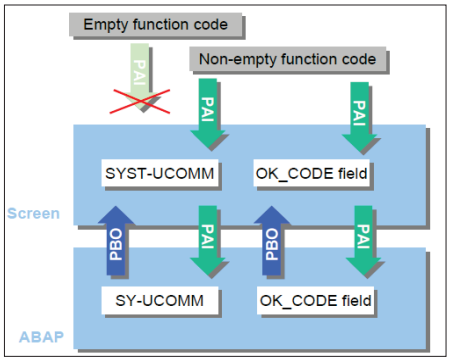
Example:

```
MODULE INIT_SCREEN_0100 OUTPUT.
SET PF-STATUS 'STATUS_100'.
SET TITLEBAR '100'.
ENDMODULE.
```

GUI Status

Reading Function Codes:

- 1.In each PAI event that a user triggers by choosing either a pushbutton on the screen or an element in a GUI status.
- 2.Function code is placed into the system field SYST-UCOMM or SY-UCOMM and placed in the OK_CODE field.

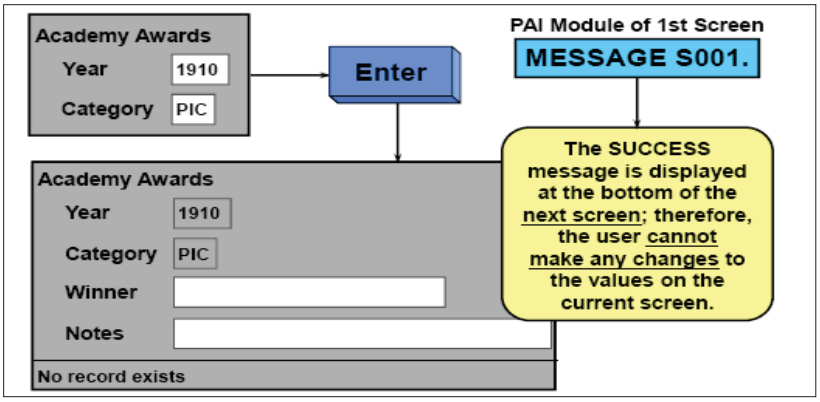


Message Types

S : Success
I : Information
A : Abend
W: Warning
E : Error
X : Exit

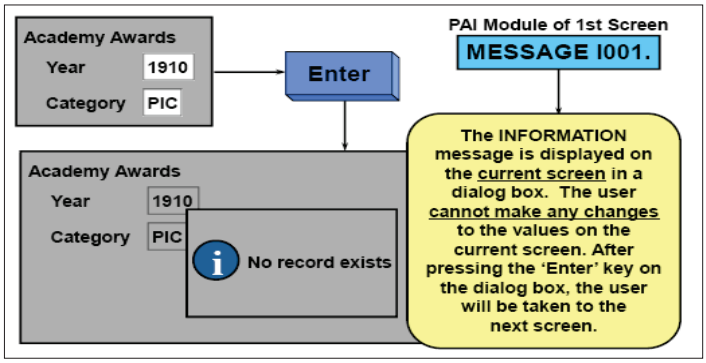
Message Types

Success Message Is displayed at the bottom of the screen. If you have next screen then the message will be displayed at the bottom of the next screen.



Message Types

Information Message is displayed in a dialog box in the same screen.
After pressing enter key on dialog box user will be taken to next screen.
The user is not restricted from going on to the next screen.

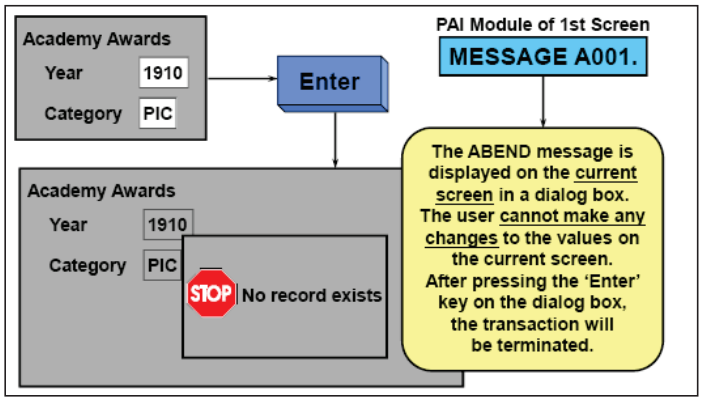


Message Types

Abend Message is displayed on the current screen in a dialog box.

The user cannot make any changes to the values on the current screen.

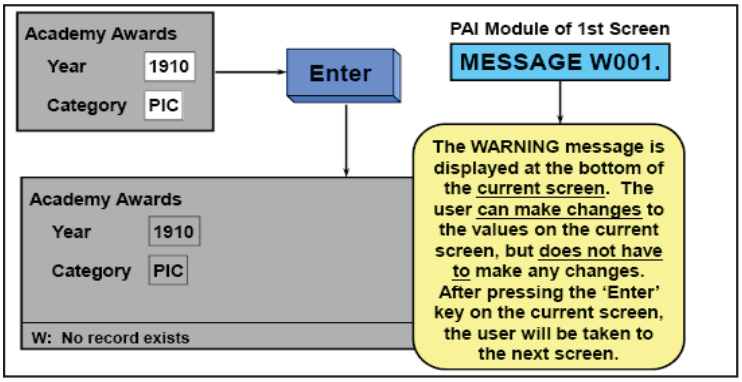
After pressing the 'Enter' key on the dialog box transaction will be terminated.



Message Types

Warning Message is displayed at the bottom of the current screen.

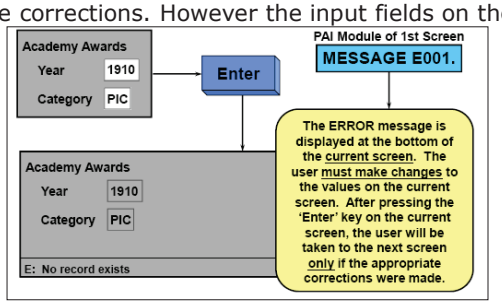
- On pressing the 'Enter' key on the current screen, the user will be taken to the next screen even if no changes were made on the current screen. Thus user is not restricted from going on to the next screen.



Message Types

Error Message is displayed at the bottom of the current screen.

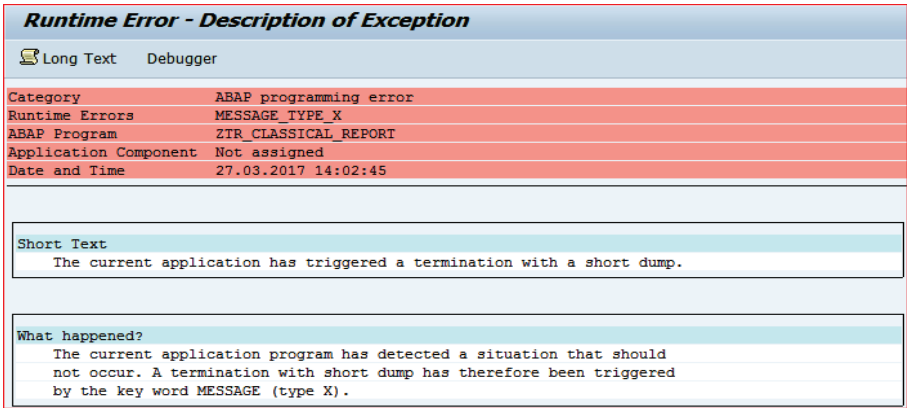
- The user must make changes to the values on the current screen.On pressing the 'Enter' key on the current screen, the user will be taken to the next screen only if the appropriate corrections were made on the current screen.
- If no corrections were made, the error message would be redisplayed at the bottom of the current screen.
- When a warning or error message is triggered the system will stop at current screen, prompting the user to make corrections. However the input fields on the screen will be disabled for input.



Message Types

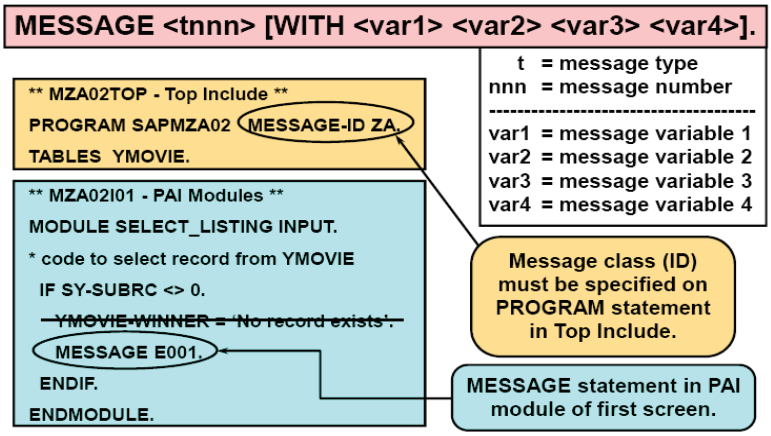
Exit Message is displayed on the next screen.

- It leads to Runtime Error. “the current application has triggered a termination with a short dump”.



Message Types

Message Statement



Message Types

Handling Error Messages

- There are 2 ways to issue error/warning messages.
  - 1. Issue an error or warning message with the Flow Logic SELECT statement.
  - 2. Define valid values for a screen field with the Flow Logic VALUES statement.

FIELD statement is a Flow Logic command, not an ABAP command.

The purpose of the "FIELD" statement is to keep a single screen field open for input after an error or warning message is issued.

FIELD statement is used in PAI event.

- Syntax is: FIELD <screen field> MODULE <module name>.

Message Types

FIELD with select statement

- Screen field can be validated against entry in database table.
- To validate Purchase order number in PAI module.

```
PROCESS AFTER INPUT.
  FIELD PO_NO MODULE PO_CHECK.
MODULE USER_COMMAND_0100.
```

Module PO_CHECK is as below.

```
MODULE PO_CHECK INPUT.
 DATA: LV_EBELN TYPE EKKO-EBELN.
   SELECT SINGLE EBELN INTO LV_EBELN
                FROM EKKO WHERE EBELN = PO_NO.
IF SY-SUBRC NE 0.
    MESSAGE E003(Message Class Name).
ENDIF.
ENDMODULE. " PO_CHECK INPUT
```

Screen Commands

Chain Statement

- To keep multiple screen fields open for input after an error or warning message is displayed, you need to use the "CHAIN" and "ENDCHAIN" Flow Logic commands.
- These statements group the "FIELD" statements and the "MODULE" statements together.
- The basic syntax of these statements is:

```
CHAIN.
  FIELD: <F1>, <F 2>,... MODULE <MOD1>. FIELD: <G1>, <G 2>,...
MODULE <MOD2>. ...
 ENDCHAIN.
```

- If an error or warning message is issued in <module name>, all the screen fields listed in the "FIELD" statements will be open for input.

Cursor Position

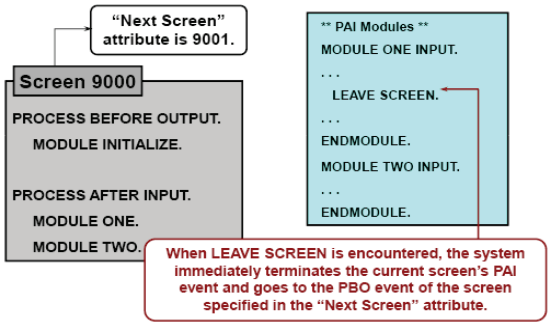
Cursor Position

- By default, the cursor will be positioned in the first field open for input on a screen.
- Cursor position can be changed in two ways.
  - By setting the cursor position in PBO event-
  - Syntax - SET CURSOR FIELD <field name>
  - Cursor will be placed on the screen field mentioned.
- Cursor position in screen attributes-
  - Mention the field name where cursor to be placed on screen in cursor position attribute .

Leave Screen

1.The LEAVE SCREEN statement ends the current screen and calls the subsequent screen.

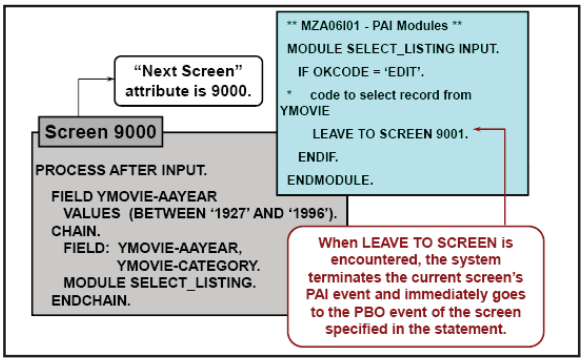
2.When the system encounters the "SET SCREEN <screen #>" ABAP statement, it temporarily overrides the "Next Screen" attribute with this <screen #> and the PAI processing continues. After all PAI modules are executed, the system goes to the PBO event of <screen #>



Leave to Screen

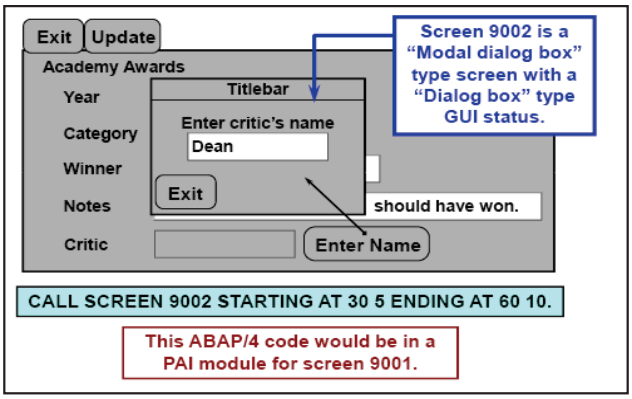
1."LEAVE TO SCREEN <screen #>" ABAP statement, terminates the screen's PAI event and immediately goes to the PBO of <screen #>.

2.The "LEAVE TO SCREEN <screen #> statement performs the functionality of two statements: "SET SCREEN <screen #>" and "LEAVE SCREEN".



Screen Commands

"CALL SCREEN <screen #>" ABAP statement, temporarily suspends the current screen's PAI processing and immediately goes to the PBO event of <screen #>. When control returns back to the "calling" screen, its PAI processing will resume.



Logical Unit of Work (LUW)

When external data is modified by application programs, it must be ensured that the data is consistent after the changes have been made.

This is particularly important when data is edited in the database. The time span in which a consistent data state is transferred to another consistent state is known as an LUW (Logical Unit of Work).

ABAP has two types of LUWs:

- Database LUWs realized by the database system.
- SAP LUWs realized using special ABAP programming techniques.

Accordingly, there are two lock types that are of significance:

- Database locks set by the system
- SAP Locks set using special ABAP programming techniques

Logical Unit of Work (LUW)

1.LUW refers to a collection of actions performed at the database level as a complete unit.

2.In this example: LUW is selecting A and B from the database, updating A, and deleting B.

3.This would be the desired LUW because we would want to rollback all changes if any of these actions failed.

4.Chanes will be saved (commit work) in database at the end of third screen.

5.An SAP LUW will end with either the COMMIT WORK or ROLLBACK WORK statement.

Logical Unit of Work (LUW)

6.A logical unit consisting of dialog steps, whose changes are written to the database in a single database LUW is called an SAP LUW.

7.If an SAP LUW contains database changes, you should either write all of them or none at all to the database.

8.Include a database commit when the transaction has ended successfully, and a database rollback in case the program detects an error.

9.Since database changes from a database LUW cannot be reversed in a subsequent database LUW, you must make all of the database changes for the SAP LUW in a single database LUW.

10.In our example, we need to perform UPDATE A and DELETE B at the end of third screen.

Table Control

A table control is an area on the screen where the system displays data in a tabular form. It is processed using a loop.

To create a table control, drag & drop table control from screen elements on screen painter. Give a name to table control.

Select the table definition and fields clicking on Dictionary/Program fields object button.

Each table control need to be declared in declaration part of the program as, CONTROLS <CTRL> TYPE TABLEVIEW USING SCREEN <SCREEN NO>. where <CTRL> is the name of the table control on a screen.

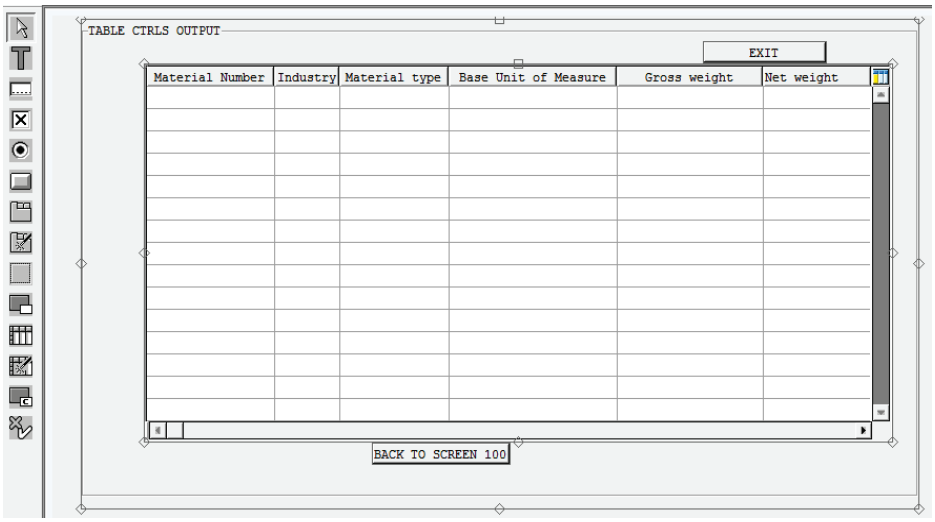
You must code a LOOP statement in both the PBO and PAI events for each table in your screen.

Flow Logic Code

This is because of the LOOP statement causes the screen fields to be copied back and forth between the ABAP program and the screen field. For this reason at least an empty LOOP AT ...ENDLOOP must be there.

```
PROCESS BEFORE OUTPUT.
MODULE STATUS_0100.
        LOOP AT <internal table>  INTO <work are>
                [WITH CONTROL <CTRL> ]
                CURSOR  <CTRL>-CURRENT_LINE.
        ENDLOOP.
PROCESS AFTER INPUT.
MODULE USER_COMMAND_0100.
        LOOP AT <internal table>.
        ENDLOOP.
```

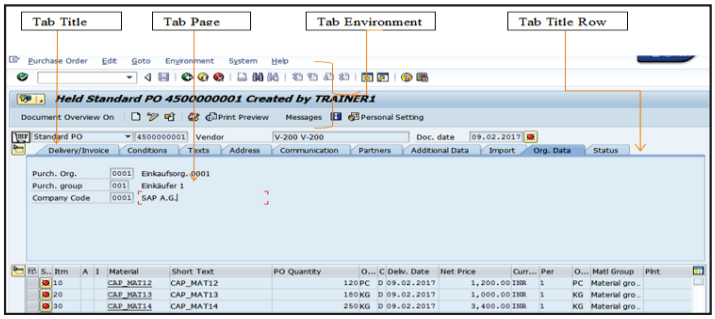
Table Control



Tab Strip Controls

Using tab strip control, we can place series of screens belonging to an application on a single screen and also we can navigate between them easily.

Use of tab strip control is to give complex applications a uniform structure and make it easier for users to navigate between their components and to make the structure of the application easier for users to learn and understand.



Tab Strip Controls

Components

- Tab Title:- Title of the component to which user can navigate. They are push buttons.
- Tab title row: - All tab titles will appear in a row.
- Tab Page: - A tab page contains a collection of fields that logically belong together. Tab pages are implemented using sub screens. A tab page is a Subscreen with a pushbutton assigned to it.
- Tab Environment: - The screen environment around the tabstrip must remain constant. When you change between tab pages, the menus, application toolbar, and other fields outside the tabstrip control must not change.
- For designing and using tab strip controls, see the transaction code: BIBS.

Tab Strip Controls

Subscreen

- Embedding one screen within another and a subscreen cannot be displayed by itself.
- A subscreen is a screen that is displayed in a specified area of the main screen.
- The subscreen displayed in the predefined area will depend on the user's request on the main screen.
- Subscreen area is an area in the main screen where a Subscreen is placed.
- The subscreen's flow logic is also embedded in the main program's flow logic.
- Subscreen allows to expand the content of a screen dynamically at runtime

Tab Strip Controls

Subscreen

- Subscreen is used to vary the fields displayed on a screen.

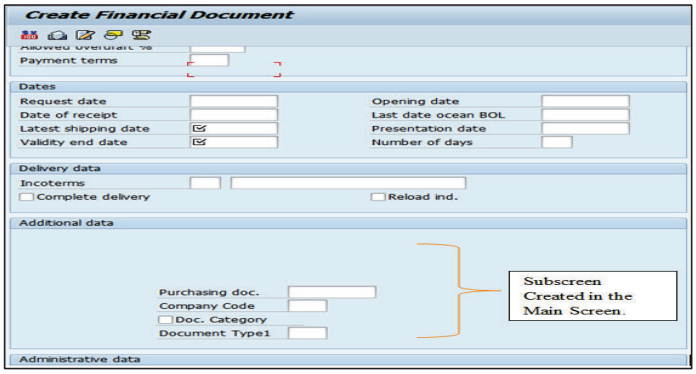
As an example:

- The “main” screen contains the customer number, name, and a predefined area for a subscreen.
- One subscreen contains customer address information.
- Another subscreen contains customer bank information.

Tab Strip Controls

Creating Subscreen In main screen

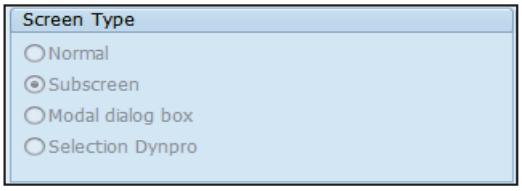
- In the screen layout, place the screen element 'subscreen' on layout.
- You can set the length and width of the subscreen by double clicking on subscreen. Provide unique name to subscreen, save and activate it.



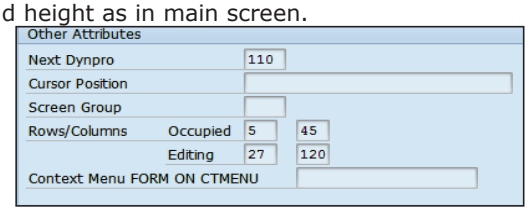
Tab Strip Controls

Create a subscreen

- Create a screen as given in earlier slides. Screen type should be subscreen.



- Specify the screen length and height as in main screen.

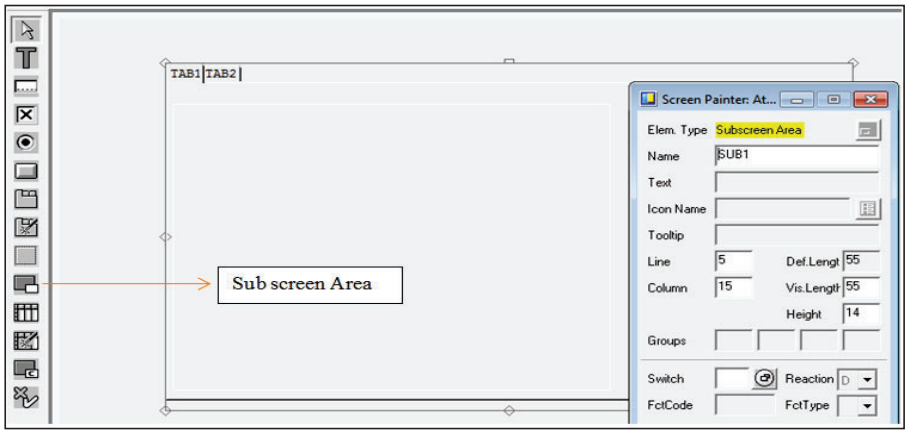


- Save the subscreen and go to screen layout. Here subscreen are will be displayed.

Tab Strip Controls

Creating Sub Screen Area

- User can design the subscreen area as showed in below screen.



Tab Strip Controls

Call Subscreen in PBO

- To include a subscreen in a predefined area on the “main” screen, use the “CALL SUBSCREEN” statement in the “main” screen’s PBO event.
  - Syntax : CALL SUBSCREEN <area> INCLUDING <program> <subscreen #>.
- <area> : Name of the subscreen area defined on the “main” screen. This subscreen area will be the location of the subscreen. This <area> cannot be enclosed in single quotes.
- <program> : Name of the program where the subscreen exists.
- <subscreen #> : Number of the subscreen to display in the subscreen area.
- The <program> and <subscreen #> can be literals (i.e. enclosed in single quotes) or variables.
- When the “CALL SUBSCREEN” statement is encountered in the “main” screen’s PBO event, the system executes the PBO event of the subscreen. Then, the system returns to finish the PBO event of the “main” screen.

Tab Strip Controls

Call subscreen in PAI

- If the subscreen contains any PAI code, use the "CALL SUBSCREEN" statement in the PAI event of the "main" screen.
  - Syntax : CALL SUBSCREEN <area>.
- The <area> is the name of the subscreen area defined on the "main" screen. This <area> cannot be enclosed in single quotes.
- The "CALL SUBSCREEN" statement must be used to invoke the PAI event of the subscreen.
- When the "CALL SUBSCREEN" statement is encountered in the "main" screen's PAI event, the system executes the PAI event of the subscreen.
- Then, the system returns to finish the PAI event of the "main" screen.
- Without this "CALL SUBSCREEN" statement in the PAI event of the "main" screen, the PAI event of the subscreen cannot be processed by the system.
- In both the PBO and PAI of the "main" screen, the "CALL SUBSCREEN" statement cannot be used inside a "LOOP" or a "CHAIN"

Creating Tab Strips

Program the ABAP Processing Logic

- The control is created in the declaration part of the program
  - CONTROLS <ctrl> TYPE TABSTRIP.

- The only component of the control which is used in the program is ACTIVETAB
  - In the PBO Event
    - To activate the tab page asssign the function codes to the component ACTIVE TAB
    - <CTRL>-ACTIVETAB = <'Function Code'>.
  - In the PAI Event
    - The function code of the last active tab title on the screen is contained in ACTIVETAB

Tab Strip Controls

Subscreen Restriction

- The following ABAP/4 statements cannot be used in a sub screen's PBO or PAI modules (instead, they must be used in the "main" screen):
  - SET PF-STATUS
  - SET TITLEBAR
  - SET SCREEN
  - LEAVE TO SCREEN
  - CALL SCREEN

Note: These ABAP/4 statements in a subscreen will pass a syntax check; however, they will result in runtime errors.

Review Question

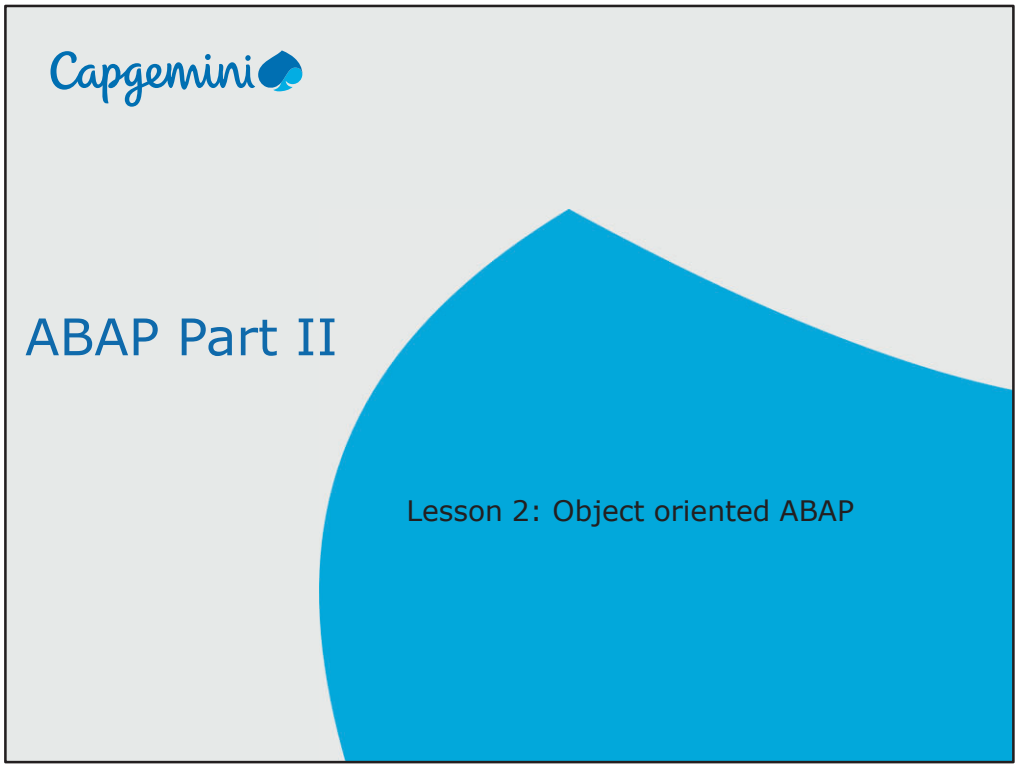
Question 1._____ Message Is displayed at the bottom of the screen.

Question 2: Parameter names that appear on the form statement are called _____ parameters.

Summary

In this lesson, you have learnt how to:

- Work on Module pool programming
- Understand Tools for developing Module pool programming
- Work with Screen Painter
- Work with Flow Logic
- Know the Types of Events
- Know the GUI status & Messages
- Work on Screen commands & LUW
- Work on Table controls and Tab strips


Capgemini
ABAP Part II
Lesson 2: Object oriented ABAP

Lesson Objectives

After completing this lesson, participants will be able to understand -

- OOPS Concepts
- ABAP Objects
- Creating & Accessing objects
- Constructor
- Inheritance
- Casting
- Interfaces
- Events
- Exceptions

OOPS Concept

Object-oriented programming, is a problem-solving method in which the software solution reflects objects in the real world.

Benefits of Object-oriented programming are :

- Multiple Instances
- Encapsulation
- Inheritance
- Polymorphism
- Compatibility
- Maintainability

Benefits of Object-oriented programming

Multiple Instances

- The ability to create multiple instances of a "class", such as a vehicle, is one of the central attributes of object-oriented languages.

Encapsulation

- Encapsulation means that the implementation of an object is hidden from other components in the system, so that they cannot make assumptions about the internal status of the object and therefore dependencies on specific implementations do not arise

Benefits of Object-oriented programming

Polymorphism

- Polymorphism (ability to have multiple forms) in the context of object technology signifies that objects in different classes react differently to the same messages.

Inheritance

- Inheritance defines the implementation relationship between classes, in which one class (the subclass) shares the structure and the behavior defined in one or more other classes (super classes).
- Note: ABAP Objects only allows single inheritance.

Compatibility

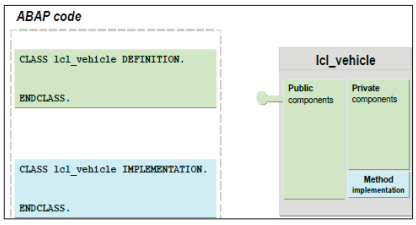
- ABAP object is true extension of ABAP language. ABAP OOPS statements can be used in procedural ABAP programs. Object themselves can contain classic ABAP statements, Only OOPS concepts that have been proved useful have been included. It has been kept the simplest. There is increased use of type checks.

ABAP Objects - Class

A class is a set of objects that have the same structure and the same behavior. A class is therefore like a blueprint, in accordance with which all objects in that class are created.

The components of the class are defined in the definition part. The components are attributes, methods, events, constants, types, and implemented interfaces. Only methods are implemented in the implementation part.

The CLASS statement cannot be nested, that is, you cannot define a class within a class.



Classes

Classes are the central element of object-orientation.
A Class is an abstract description of an object.
Classes are templates for objects.
Defines the state and behavior of the object.
Types of classes

- Local classes
  - Defined within an ABAP program
  - Can be used only within that program
- Global classes
  - Defined in the class builder SE24
  - Stored centrally in class library in the R/3 repository
  - Can be accessed from all the programs in the R/3 system
  - e.g. CL_GUI_ALV_GRID, CL_GUI_CUSTOM_CONTAINER

Defining Local Classes:

- A complete class definition consists of a declaration part and, if required, an implementation part.
- The **declaration part** of a class is a statement block that contains the declaration for all components (attributes, methods, events) of the class.

```
CLASS <class> DEFINITION.
  ...
ENDCLASS.
```

- If you declare methods in the declaration part of a class, you must also write an **implementation part** for it. The implementation part of a class contains the implementation of all methods of the class.

```
CLASS <class> IMPLEMENTATION.
  ...
ENDCLASS.
```

ABAP Objects - Attributes

Attributes describe the data that can be stored in the objects of a class.

Attributes can have any kind of data type:

- C, N, I, P, ..., STRING
- Dictionary types
- User-defined types
- TYPE REF TO defines a reference to an object, in this case "r_car"

Public attributes

- Can be viewed and changed by all users and in all methods
- Direct access

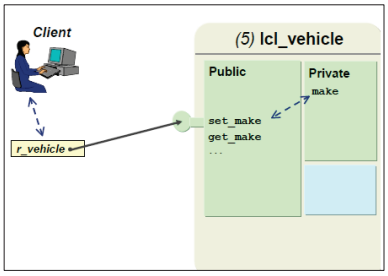
Private attributes

- Can only be viewed and changed from within the class
- No direct access from outside the class

- Attributes are internal data fields within a class that can have any data type.
- The state of an object is determined by the contents of its attributes.
- Two types of attributes:
    1. Instance Attributes
        - Define the instance specific state of an object.
        - Declared using DATA statement.
    2. Static Attributes
        - Define the state of the class that is valid for all instances of the class.
        - Exist once for each class.
        - Declared using CLASS-DATA statement.

ABAP Objects - Attributes

Accessing private attributes :You can access an object's private attributes using public methods, which in turn output this attribute or change it.

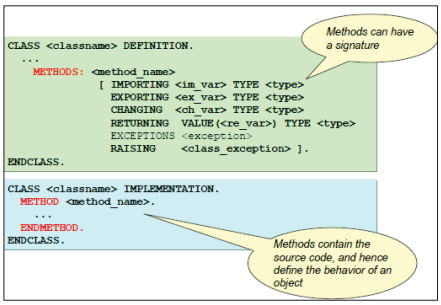



```
CLASS lcl_vehicle DEFINITION.

  PUBLIC SECTION.
    DATA: make TYPE string.

  PRIVATE SECTION.

ENDCLASS.
```

```
CLASS lcl_vehicle DEFINITION.

  PUBLIC SECTION.
    ...

  PRIVATE SECTION.
    DATA: make      TYPE string.

ENDCLASS.
```

ABAP Objects - Methods

1. Methods are internal procedures in classes that determine the behavior of an object. They can access all attributes in their class and can therefore change the state of an object.
2. Methods have a parameter interface (called signature) that enables them to receive values when they are called and pass values back to the calling program.



- Methods are internal procedures in a class that define the behavior of an object.
- Methods have a parameter interface with which user can supply them with values when calling them and receive values back from them.
- Two types of methods:
    1. Instance Methods
        - Declared using METHODS statement.
        - Can access all the attributes and methods of the class.
    2. Static Methods
        - Declared using CLASS-METHODS statement.
        - Can access only the static attributes and methods of the class.

Methods and Visibility

Public methods

- Can be called from anywhere

Private methods

- Can only be called within the class

- You can divide the declaration part of the class definition into visibility sections.
- Each of the class component must be assigned to one of the visibility sections.
- Three visibility sections are:
    1. Public Section
    – All the components declared here are accessible to all users and the methods of that class and classes that inherit from it.
    2. Protected Section
    – All of the components declared here are accessible to all methods of that class and classes that inherit from it.
    3. Private Section
    – All of the components declared here are only visible in the methods of that class.

Demo: Create a class with instance attributes and instance methods

Instance attributes and Static attributes

Instance attributes
- One per instance
- Statement: DATA

Static attributes
- Only one per class
- Statement: CLASS-DATA
- Also known as class attributes

```
CLASS lcl_vehicle DEFINITION.

  PUBLIC SECTION.
        ...
  PRIVATE SECTION.
    DATA: make      TYPE string,
          ...


    CLASS-DATA: n_o_vehicles TYPE i.

ENDCLASS.
```

Instance method and Static method

Instance methods

- Can use both static and instance components in their implementation part
- Can be called using an instance

Static methods

- Can only use static components in their implementation part
- Can be called using the class

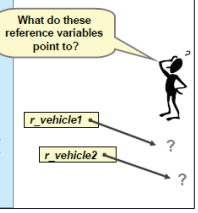
Demo: Create a class with static attributes and static method

Reference Variable

A reference variable acts as a pointer to an object.

- DATA: R_VEHICLE1 TYPE REF TO LCL_VEHICLE.
- Declares a reference variable that acts as a pointer to an object

```
...

CLASS lcl_vehicle DEFINITION.
  PUBLIC SECTION.
    ...
  PRIVATE SECTION.
    ...
ENDCLASS.


CLASS lcl_vehicle IMPLEMENTATION.
  ...
ENDCLASS.


DATA: r_vehicle1 TYPE REF TO lcl_vehicle,
      r_vehicle2 TYPE REF TO lcl_vehiloe.


START-OF-SELECTION.
  ...
```

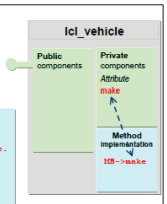


Reference Variable ME

You can address the object itself within instance methods using the implicitly available reference variable me.

Description of example: In the constructor, the instance attribute make is covered by the locally defined variable make. In order to still be able to address the instance attribute, you need to use me.

```
CLASS lcl_vehicle DEFINITION.
  PUBLIC SECTION.
    METHODS constructor
        IMPORTING im_make  TYPE string
                  im_model1 TYPE string.
  PRIVATE SECTION.
    DATA make TYPE string. ...
ENDCLASS.

CLASS lcl_vehicle IMPLEMENTATION.
  METHOD constructor.  "just a demo
    DATA make TYPE string VALUE 'Benz'.
    CONCATENATE im_make make INTO ME->make.
  ENDMETHOD.

  METHOD DUMMY.
* calling own methods from inside class
* call method display_attributes( )
* with the short syntax variant:
    display_attributes( ).
  ENDMETHOD.
ENDCLASS.
```



Demo: Create a class with Reference variable Me

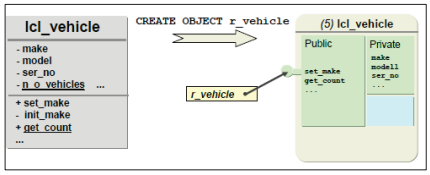
Creating and Accessing Objects

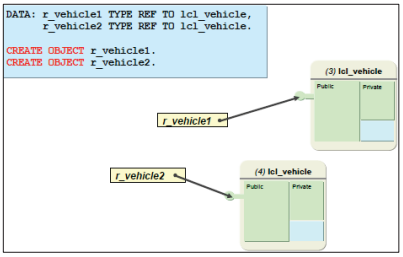
An object is a section of source code that contains data and provides services

Objects are created using the CREATE OBJECT statement

Objects can only be created and addressed using reference variables

An object is a section of source code that contains data and provides services.





- Objects are instances of classes.
- Each object has a unique identity and its own attributes.
- A class can have any number of objects (instances).
- To access an object from an ABAP program, we use object references.

  Defined using **<ref> TYPE REF TO <class>.**

  Created using **CREATE OBJECT <ref>**.

- Accessing object attributes **<ref>–><attribute>**.
- Accessing class (static) attributes **<class>=><attributes>**.
- Accessing object methods **CALL METHOD <ref>-><method>**.
- Calling class (static) methods **CALL METHOD <class>=><method>**.

Accessing Attributes and Method

Instance methods are called using

CALL METHOD <reference>-><instance_method>.

Static methods (also referred to as class methods) are called using

CALL METHOD <classname>=><class_method>.

If you are calling a static method from within the class, you can omit the class name.

Static attributes are accessed using

<classname>=><class_attribute>

instance attributes are accessed using

<instance>-><instance_attribute>

=> and -> are the component selectors

Functional Method

Methods that have a RETURNING parameter are described as functional methods. These methods cannot have EXPORTING or CHANGING parameters, but has many (or as few) IMPORTING parameters and exceptions as required. You can only do this for a single parameter, which additionally must be passed as a value.

```
CLASS lcl_vehicle DEFINITION.
  PUBLIC SECTION.
    METHODS: get_average_fuel
                      IMPORTING im_distance    TYPE s_distance,
                                im_fuel        TYPE ty_fuel
                      RETURNING VALUE(re_fuel) TYPE ty_fuel,
ENDCLASS.
```

```
DATA: r_vehicle1 TYPE REF TO lcl_vehicle,
      r_vehicle2 TYPE REF TO lcl_vehicle,
      avg_fuel TYPE ty_fuel.

...
* example for short syntax in aritmet. operation
avg_fuel =
    r_vehicle1->get_average_fuel( im_distance = 500 im_fuel = 50 )
  + r_vehicle2->get_average_fuel( im_distance = 600 im_fuel = 60 ).
```

Demo: Create a class with Functional Method

Constructor

The constructor is a special instance method in a class with the name constructor.

Each class can have one constructor.

The constructor is automatically called at runtime within the CREATE OBJECT statement.

If you need to implement the constructor, then you must define and implement it in the PUBLIC SECTION.

You cannot normally call the constructor explicitly

Special method for creating objects with defined initial state

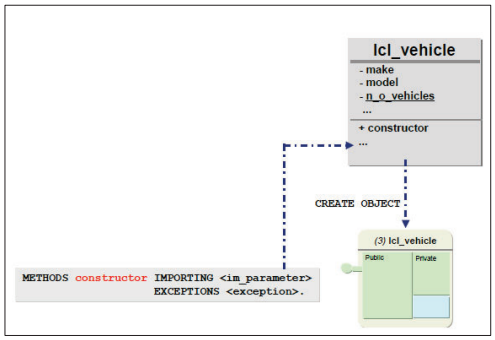
Only has IMPORTING parameters and EXCEPTIONS

Is executed only once per instance

- Constructors are special methods called automatically by the system to set the starting state of a new object or class.
- Constructors are called when the class is instantiated.
- Two types of constructors:
  1. Instance Constructors
     - Declared using **METHODS CONSTRUCTOR.**
     - Used to initialize instance attributes.
  2. Static Constructors
     - Declared using **CLASS-METHODS CLASS_CONSTRUCTOR.**
     - Used to initialize static attributes.
- Constructors are implemented in the implementation part using

```
METHOD <method>.
 ...
ENDMETHOD
```

Constructor



Demo: Create a class with Constructor

Static Constructor

The static constructor is a special static method in a class with the name class_constructor.

It is executed precisely once per program.

The static constructor of a class <classname> is called automatically when the class is first accessed, but before any of the following actions are executed:

Creating an instance in the class using CREATE OBJECT <obj>, where <obj> has the data type REF TO <classname>

Addressing a static attribute using <classname>=><attribute>

Calling a static attribute using CALL METHOD <classname>=><classmethod>

Registering a static event handler method using SET HANDLER <classname>=><handler_method> FOR <obj>

Registering an event handler method for a static event in class <classname>.

The static constructor cannot be called explicitly.

Demo: Create a class with Static Constructor

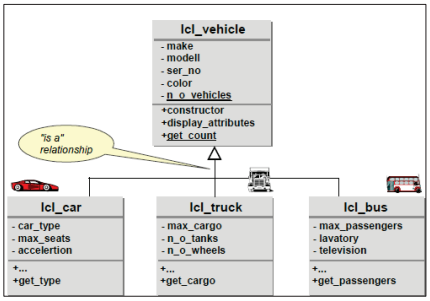
Demo: Global Class creation through SE24

Inheritance

Inheritance is a relationship, in which one class (the subclass) inherits all the main characteristics of another class (the superclass). The subclass can also add new components (attributes, methods, and so on) and replace inherited methods with its own implementations.

The inheritance relationship is often described as an "is a" relationship:

- A truck is a vehicle



Inheritance

Single Inheritance

- ABAP Objects only has single inheritance.
- A class may only have one direct superclass, but it can have more than one direct subclass. The empty class OBJECT is the root node of every inheritance tree in ABAP Objects.

Inheritance

Relationship between Super class and Sub class

- Common components only exist once in the superclass
  - New components in the superclass are automatically available in subclasses
  - Amount of new coding is reduced ("programming by difference")
- Subclasses are extremely dependent on superclasses
- "White Box Reuse":
  - Subclass must possess detailed knowledge of the implementation of the superclass

Inheritance

Normally the only other entry required for subclasses is what has changed in relation to the direct superclass. Only additions are permitted in ABAP Objects, that is, in a subclass you can "never take something away from a superclass". All components from the superclass are automatically present in the subclass.

Inheritance

The REDEFINITION statement for the inherited method must be in the same SECTION as the definition of the original method.

If you redefine a method, you do not need to enter its interface again in the subclass, but only the name of the method.

In the case of redefined methods, changing the interface (overloading) is not permitted; exception: Overloading is possible with the constructor.

Within the redefined method, you can access components of the direct superclass using the SUPER reference.

The pseudo-reference super can only be used in redefined methods.

Demo: Inheritance

Casting

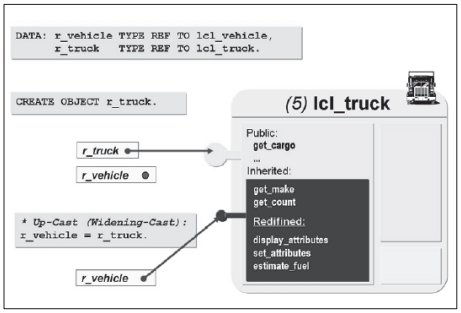
Up-Cast (Widening Cast)

- The assignment of a subclass instance to a reference variable of the type "reference to superclass" is described as a Widening cast. As the target variable can accept more dynamic types in comparison to the source variable, this assignment is also called widening cast.

What is a Widening cast used for?

- A user who is not interested in the finer points of cars, trucks, and busses (but only, for example, in the fuel consumption and tank gauge) does not need to know about them. This user only wants and needs to work with (references to) the lcl_vehicle class. However, in order to allow the user to work with cars, busses, or trucks, you generally need a narrowing cast.

Widening Cast



Demo: Inheritance – Widening cast

Casting

Down-cast (Narrowing Cast)

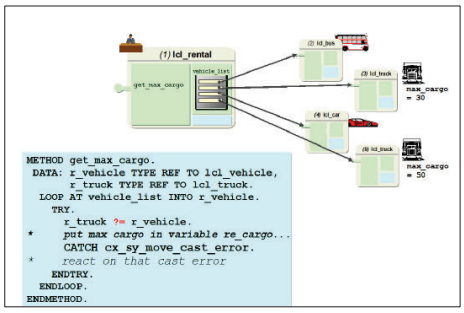
- The Narrowing cast logically represents the opposite of the widening cast. The narrowing cast cannot be checked statically, only at runtime. The Cast Operator ?= (or the equivalent MOVE ... ?TO…) must be used to make this visible.
- As the target variable can accept less dynamic types after the assignment, this assignment is also called narrowing cast.

What is a Narrowing *cast used for?*

- The client, the car rental company wants to execute a function for specific vehicles form the list (vehicle_list). For example, the client wants to ascertain the truck with the largest cargo capacity. However, not all vehicles are in the trucks list, it also includes references to cars and busses.

Casting

With this kind of cast, a check is carried out at runtime to ensure that the current content of the source variable corresponds to the type requirements of the target variables. If it is, the assignment is carried out. Otherwise, an exception of the error class CX_SY_MOVE_CAST_ERROR is raised.




```
METHOD get_max_cargo.
 DATA: r_vehicle TYPE REF TO lcl_vehicle,
       r_truck TYPE REF TO lcl_truck.
  LOOP AT vehicle_list INTO r_vehicle.
    TRY.
      r_truck ?= r_vehicle.
*     put max cargo in variable re_cargo...
      CATCH cx_sy_move_cast_error.
*     react on that cast error
    ENDTRY.
  ENDLOOP.
ENDMETHOD.
```

Demo: Inheritance – Narrowing cast

Interfaces

Interfaces only describe the external point of contact of a class (protocols), they do not contain any implementation.

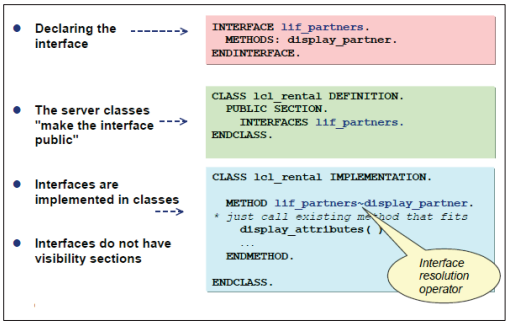
Interfaces are usually defined by a user. The user describes in the interface which services (technical and semantic) it needs in order to carry out a task.

The user never actually knows the providers of these services, but communicates with them through the interface.

In this way the user is protected from actual implementations and can work in the same way with different classes/objects, as long as they provide the services required. This is known as polymorphism with interfaces.

- Interfaces exclusively describe the external point of contact of a class, but they do not contain their own implementation part.
- Interface has only declaration part, implemented in the public section of classes.
- Interfaces do not have visibility section.
- A class can implement any number of interfaces and an interface can be implemented by any number of classes.
- Interface resolution operator enables to access interface components using an object reference belonging to the class implementing the interface.

Interfaces



Demo: Interfaces

Exceptions

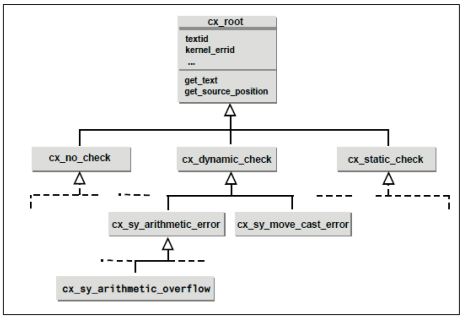
Exception refer to a situation that arises while an ABAP program is being executed, where there is no point in continuing to run the program in the normal way.

Class-based exceptions are raised either using the ABAP statement RAISE EXCEPTION or by the ABAP runtime environment.

If a class-based exception occurs, the system interrupts the normal program flow and tries to navigate to a suitable handler. If it cannot find a handler, a runtime error occurs.

The use of class-based exceptions is not limited to object-oriented contexts. Class-based exceptions can be raised and handled in all ABAP processing blocks.

Exceptions Classes: Inheritance hierarchy



Exceptions Classes: Inheritance hierarchy

The root class CX_ROOT contains two predefined methods that are inherited by the other classes.

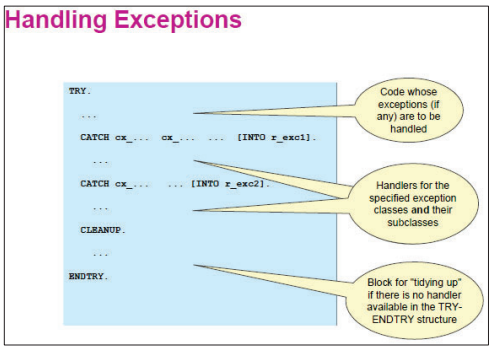
The GET_SOURCE_POSITION method returns the program name, include name (if relevant), and line number in the source code where the exception occurred.

The GET_TEXT method returns an exception text of a class in the form of a string.

You can assign several texts to each class. You can then specify which text is to used when an exception is raised by passing an identifier to the IMPORTING parameter TEXTID of the instance constructor.

All exception classes inherit the KERNEL_ERRID attribute from CX_ROOT. This attribute contains the name of the appropriate runtime error if the exception was raised by the runtime environment - such as COMPUTE_INT_ZERODIVIDE if the program tries to divide by zero

Handling Exceptions



Handling Exceptions

The TRY block contains the application code that is to handle the exceptions.

If an exception occurs in the TRY block the system searches first for a CATCH statement (which will handle the exception) in the same TRY-ENDTRY structure and then step by step outwards in all the enclosing TRY-ENDTRY structures.

A CATCH block contains the exception handler that is executed if a specified exception has occurred in the TRY block in the same TRY-ENDTRY structure.

In some cases, the system cannot find a handler for an exception within a specific TRY-ENDTRY structure but the exception is handled in a surrounding TRY-ENDTRY structure or passed along to a calling program. If this occurs, a CLEANUP block is executed before leaving the TRY-ENDTRY structure.

Events

By triggering an event, an object or class announces a change of state, or that a certain state has been achieved.

- **Triggering events**
  1. **Class defines event**
     (`EVENTS, CLASS-EVENTS`)
  2. **Object or class triggers event**
     (`RAISE EVENT`)
- **Handling events**
  3. **Event handler class defines and implements event handler method**
     (`[CLASS-]METHODS... FOR EVENT ... OF ...`)
  4. **Event handler object or handler class registers itself to specific events at runtime**
     (`SET HANDLER`)

Events

At the moment of implementation, a class defines its :Instance events (using the EVENTS statement) Static events (using the CLASS-EVENTS statement)

Classes or their instances that receive a message when an event is triggered at runtime and want to react to this event define event handler methods. Statement: [CLASS-]METHODS <handler_method> FOR EVENT <event> OF <classname>.

These classes or their instances are registered to one or more events at runtime. Statement: SET HANDLER <handler_method> FOR <reference>. (for instance events) SET HANDLER <handler_method>. (for static events)

A class or instance can trigger an event at runtime using the RAISE EVENT statement.

Events

Both instance and static events can be triggered in instance methods. Only static events can be triggered in static methods.

Events can only have EXPORTING parameters which must be passed by value.

Triggering an event using the statement RAISE EVENT has the following effect:

- 1.The program flow is interrupted at that point
- 2.The event handler methods registered to this event are called and processed
- 3.Once all event handler methods have been executed, the program flow continues

Events

Registering for an event

- When an event is triggered, only those event handler methods are executed that have, by this point, registered themselves using SET HANDLER.
- You can register an event using ACTIVATION 'X', and deregister it using ACTIVATION space. If you do not specify ACTIVATION, then the event registers (default behavior).
- You can register several methods in one SET HANDLER statement: SET HANDLER <ref_handle1>-><handler_method1> ... <ref_handleN>-><handler_methodN> FOR <ref_sender> | FOR ALL INSTANCES

Events

Registration/De registration: Handler Tables

- Every object that has defined events has an internal table, the handler table. All objects that have registered for events are entered in this table together with their event handler methods.

Demo: Events

Summary

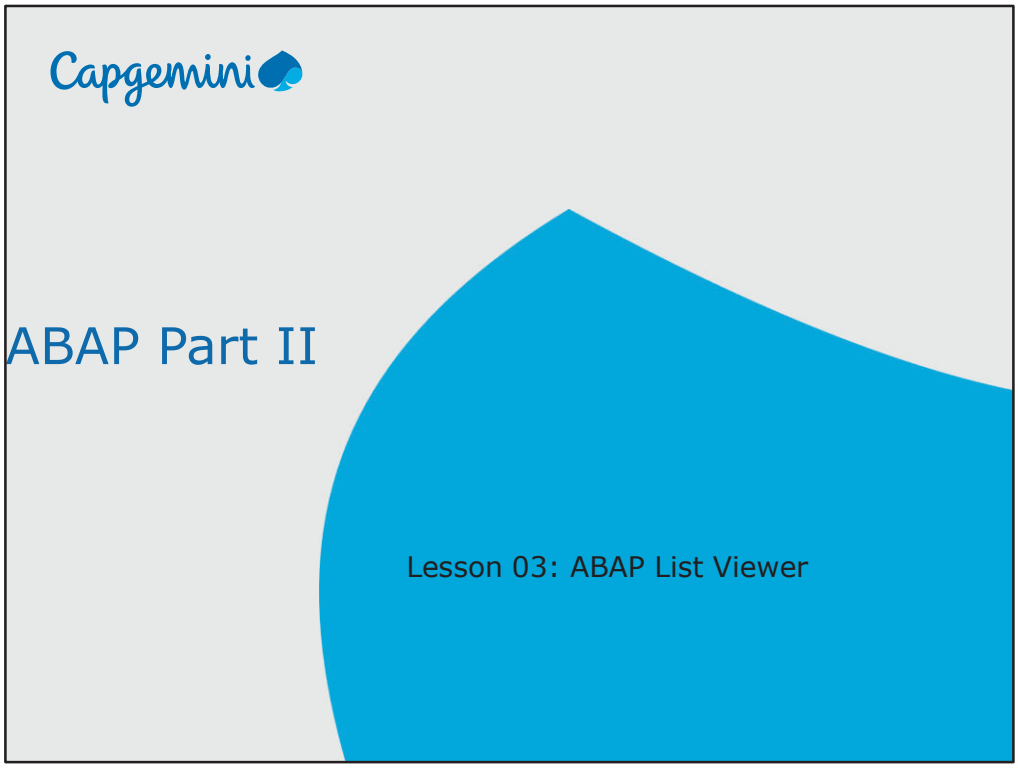
In this lesson, you have learnt:

- OOPS Concepts
- ABAP Objects
- Creating & Accessing objects
- Constructer
- Inheritance
- Casting
- Interfaces
- Events
- Exceptions

Review Question

Question 1: _____ do not contain implementation.

Question 2: _____means that the implementation of an object is hidden from other components in the system.


Capgemini
ABAP Part II
Lesson 03: ABAP List Viewer

Lesson Objectives

After completing this lesson, participants will be able to understand the following -

- Control Framework
- ALV Grid
- Non Event Based Functionality
- Event Based Functionality

Introduction

The common features of report are column alignment, sorting, filtering, subtotals, totals etc.

To implement these, a lot of coding and logic is to be put.

To avoid that we can use a concept called ABAP List Viewer (ALV).

ALV (ABAP List Viewer) is a grid control, used for displaying lists.

The tool provides common list operations as generic functions and enhanced by user-defined options.

The grid control itself consists of a toolbar, a title and the output table displayed in a grid control. The user has control over the look of the grid (to certain degree) by the use of layout variants

Functions Provided by ALV Grid Control

Display non-hierarchical lists consistently with a modern design

Use typical list functions -such as sorting and filtering without extra programming effort

Adapt predefined list functions and their enhancements

Program responses to user actions (such as double-clicking a line) individually

Features

Navigating Within the List
Sorting in Ascending/Descending Order
Selecting and Deselecting Rows
Defining Exceptions
Setting and Deleting Filters
Displaying and Deleting Sums
Creating Subtotals
Optimizing the Column Width

Types of ALV Reports

Using ALV, we can have three types of reports:

- Simple Report
- Block Report
- Hierarchical Sequential Report

Simple Report

There are some function modules which will enable to produce the above reports without much effort.

All the definitions of internal tables, structures and constants are declared in a type-pool called SLIS.

The important function modules are:

- Reuse_alv_fieldcatalog_merge
- Reuse_alv_list_display
- Reuse_alv_grid_display

Simple Report

Reuse_ALV_fieldcatalog_merge:

- This function module is used to populate a fieldcatalog which is essential to display the data in ALV.
- If the output data is from a single dictionary table and all the columns are selected, then we need not exclusively create the field catalog.
- Its enough to mention the table name as a parameter(I_structure_name) in the REUSE_ALV_LIST_DISPLAY.
- In other cases, it has to be created.

1. Export:
   a. I_program_name : report id
   b. I_internal_tabname : the internal output table
   c. I_inclname : include or the report name where all the dynamic forms are handled.
2. Changing:
   ct_fieldcat : an internal table with the type SLIS_T_FIELDCAT_ALV which is declared in the type poolSLIS.

Simple Report

REUSE_ALV_LIST_DISPLAY: This is the function module which prints the data.

The important parameters are:
1. Export:
a. I_callback_program: report id
b. I_callback_pf_status_set: routine where a user can set his own pf status or change the functionality of the existing pf status.
c. I_callback_user_command: routine where the function codes are handled.
d. I_structure name: name of the dictionary table
e. Is_Layout : structure to set the layout of the report
f. It_fieldcat : internal table with the list of all fields and their attributes which are to be printed (this table can be populated automatically by the function module REUSE_ALV_FIELDCATALOG_MERGE)
g. It_events : internal table with a list of all possible events of ALV and their corresponding routine names.
2. Tables:
a. t_outtab : internal table with the data to be output

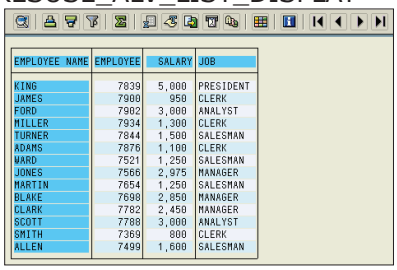
Simple Report

REUSE_ALV_GRID_DISPLAY: To display the results in grid rather than as a list.

- Parameters : same as reuse_alv_list_display

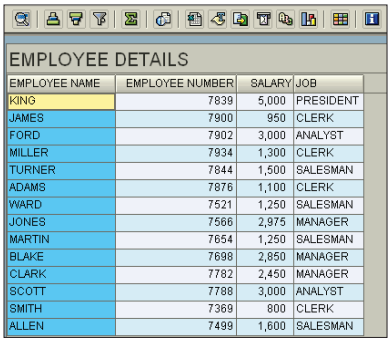
Display Output Internal Table

RESUSE_ALV_LIST_DISPLAY



EMPLOYEE NAME	EMPLOYEE	SALARY	JOB
KING	7839	5,000	PRESIDENT
JAMES	7900	950	CLERK
FORD	7902	3,000	ANALYST
MILLER	7934	1,300	CLERK
TURNER	7844	1,500	SALESMAN
ADAMS	7876	1,100	CLERK
WARD	7521	1,250	SALESMAN
JONES	7566	2,975	MANAGER
MARTIN	7654	1,250	SALESMAN
BLAKE	7698	2,850	MANAGER
CLARK	7782	2,450	MANAGER
SCOTT	7788	3,000	ANALYST
SMITH	7369	800	CLERK
ALLEN	7499	1,600	SALESMAN

REUSE_ALV_GRID_DISPLAY



EMPLOYEE DETAILS

EMPLOYEE NAME	EMPLOYEE NUMBER	SALARY	JOB
KING	7839	5,000	PRESIDENT
JAMES	7900	950	CLERK
FORD	7902	3,000	ANALYST
MILLER	7934	1,300	CLERK
TURNER	7844	1,500	SALESMAN
ADAMS	7876	1,100	CLERK
WARD	7521	1,250	SALESMAN
JONES	7566	2,975	MANAGER
MARTIN	7654	1,250	SALESMAN
BLAKE	7698	2,850	MANAGER
CLARK	7782	2,450	MANAGER
SCOTT	7788	3,000	ANALYST
SMITH	7369	800	CLERK
ALLEN	7499	1,600	SALESMAN

Demo

ALV List Display and ALV Grid Display

Field Catalog

A field catalog is prepared using the internal table (I_FIELDCAT) of type SLIS_T_FIELDCAT_ALV

Field catalog containing descriptions of the list output fields (usually a subset of the internal output table fields)

A field catalog is required for every ALV list output to add desired functionality (i.e. Key, Hotspot, Specific headings, Justify, Col. position etc) to certain fields of the output.

If not mentioned specifically, then the defaults are taken

No of ways to build Field Catalog

Preparing the data dictionary structure

Build field catalog using function module

- Reuse_alv_fieldcatlog_merge

Prepare the field Catalog manually

Demo

Create a Field Catalog Manually and use it in list display

OO ALV

The Control Framework is required for OO ALV as it provides global classes for various functionalities.

CL_GUI_ALV_GRID

- The wrapper class implemented to encapsulate ALV Grid functionality for list display.

Basic Components

While preparing a list to be displayed via an ALV grid control, we have some basic components to prepare. These are:

List data :

- Data in an internal table to be listed

Field Catalog :

- Define specifications on how the fields of our list will be displayed
- Has technical and additional information about display options for each column to be displayed.
- The internal table for the field catalog must be referenced to the dictionary type LVC_T_FCAT.

Container

- Storage area where the list will be displayed.
- It should be of type CL_GUI_CUSTOM_CONTAINER

Basic Components

Layout Structure :

- Fill a structure to specify general layout options for the grid
- To set
  - general display options
  - grid customizing
  - totals options
  - color adjustments etc...
- The layout structure must be of type LVC_S_LAYO.

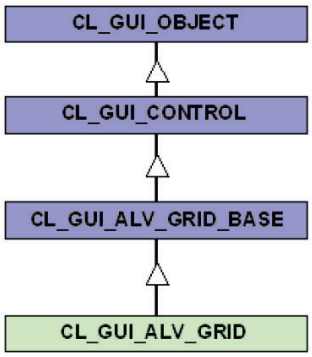
Basic Components

Event Handler

- To handle events triggered by the ALV Grid instance.
- Upon creating ALV Grid instance, register an instance of this event handler class to handle ALV Grid events
- Various Events are as follows
  - Print_Top_Of_Page:
    - Used for Headers. Handler is 'SET HANDLER'.
  - Print_End_Of_Page
    - Used for Footers. Handler is 'SET HANDLER'.
  - OnDropComplete
    - Event to change the state after a successful drag and drop operation.
  - OnDrag
    - To 'fetch' information from the drag source.
  - OnDrop
    - Used to use the dragged information in combination with drop source. Here, it should be checked whether the operation is successful

Grid Control – Inheritance Hierarchy

CL_GUI_ALV_GRID' class encapsulates communication with the instance on the presentation server, along with many other functions. For this reason, you should instantiate this class, not its super class.



Steps to work with OO ALV

Create an object of class CL_GUI_CUSTOM_CONTAINER.

Create an object of class CL_GUI_ALV_GRID.

Populate the internal table to display on the GRID.

Call the screen that contains the CUSTOM CONTAINER, in which the list has to be displayed.

Call the screen.

Call the method SET_TABLE_FOR_FIRST_DISPLAY of class CL_GUI_ALV_GRID and pass the required parameters

CL_GUI_ALV_GRID

'CL_GUI_ALV_GRID' class provides various methods and Events

Building Field Catalog

There are 3 methods for doing this:

- Automatic generation
- Semi-automatic generation
- Manual generation

Structure of Field Catalog (LVC_T_FCAT)

FIELDNAME	Assign a field name of your output table to a row of field catalog
REF_FIELD	Must specify this if field name in the output table is not identical to the field name of the field in Data Dictionary
REF_TABLE	Must fill this field only if the output table described by the current entry in the field catalog has a corresponding entry in the DDIC
COL_POS	Sequence of the fields
OUTPUTLEN	Desired width of the field in output
SCRTEXT_L/M/S	Field Labels

Building Field Catalog (Manually)

The work in this procedure is just filling the internal table for the field catalog. We have already seen the structure of a field catalog. To achieve filling the field catalog correctly, one must at least fill the above fields of the field catalog structure for each column of the list.

Output table fields *with* DDIC reference	Output table fields *without* DDIC reference	Explanation
FIELDNAME	FIELDNAME	Name of the field of the internal output table
REF_TABLE		Name of the DDIC reference structure
REF_FIELD		Name of the DDIC reference field (only needed if other than FIELDNAME)
	INTTYPE	ABAP data type of the field of the internal output table
	OUTPUTLEN	Column width
	COLTEXT	Column header
	SELTEXT	Column description in column selection for layout

Building Field Catalog (Manually)

```
DATA LS_FCAT TYPE LVC_S_FCAT .
LS_FCAT-FIELDNAME = 'CARRID`.
LS_FCAT-INTTYPE = 'C'.
LS_FCAT-OUTPUTLEN = '3'.
LS_FCAT-COLTEXT = 'CARRIER ID'.
LS_FCAT-SELTEXT = 'CARRIER ID'.
APPEND LS_FCAT TO PT_FIELDCAT.
CLEAR LS_FCAT.
LS_FCAT-FIELDNAME = 'CONNID'.
LS_FCAT-REF_TABLE = 'SFLIGHT'.
LS_FCAT-REF_TABLE = 'CONNID'.
LS_FCAT-OUTPUTLEN = '3'.
LS_FCAT-COLTEXT = 'CONNECTION ID'.
LS_FCAT-SELTEXT = 'CONNECTION ID'.
APPEND LS_FCAT TO PT_FIELDCAT .
…. AND SO ON FOR ALL THE FIELDS TO BE DISPLAYED IN THE List
```

Demo

Create a Field Catalog Manually and use it in list display

Building Field Catalog (Semi-automatically)

Layout Adjustment

It comes now painting our ALV Grid in a general aspect. To define general appearance of our ALV Grid we fill a structure of type "LVC_S_LAYO". This table contains fields and functionalities serviced by this adjustment. Some of the generally used options are as below:

ZEBRA	If this field is set, the list shows a striped pattern in the print preview and when it is printed (SPACE, 'X')
SMALLTITLE	If this field is set, the title size in the grid control is set to the font size of the column header. (SPACE, 'X')

```
FORM prepare_layout  CHANGING P_GS_LAYOUT TYPE lvc_s_layo.

P_GS_LAYOUT-zebra = 'X'.
P_GS_LAYOUT-grid_title = 'Flights'.
P_GS_LAYOUT-smalltitle = 'X'.

ENDFORM.                    " prepare_layout
```

ALV Display

Data transfer to the ALV control takes place during the call of method "SET_TABLE_FOR_FIRST_DISPLAY" of class "CL_GUI_ALV_GRID". The method call must be programmed at the PBO event of the screen with the SAP Grid Control container.

Remember to use Pattern > ABAP Objects > Method of a Class

ALV Display

If the ALV_GRID is initial (First Call) the method "SET_TABLE_FOR_FIRST_DISPLAY" is called as described in the previous slide. Else on subsequent calls; "REFRESH_TABLE_DISPLAY" is called. Reason being; there is no need to instantiate the Custom Container, Grid every time in the PBO of the Screen.

The parameters of this method:

- IS_STABLE: If the row or column field of this structure is set, the position of the scroll bar for the rows or columns remains stable.
- I_SOFT_REFRESH: If set, any totals created, any sort order defined and any filte set for the data displayed remain unchanged when the grid control is refreshed.

```
    CALL METHOD GR_ALVGRID->REFRESH_TABLE_DISPLAY
*      EXPORTING
*    IS_STABLE      =
*    I_SOFT_REFRESH =
     EXCEPTIONS
       FINISHED       = 1
       OTHERS         = 2.
    IF SY-SUBRC <> 0.
      MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
                 WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
    ENDIF.
```

Setting Sort Condition

It is possible to set sort conditions for the table data. This is achieved by filling an internal table of structure “LVC_T_SORT” which consists of the sort criteria. To have an initial sorting, pass it to the parameter “IT_SORT” of the method “SET_TABLE_FOR_FIRST_DISPLAY”.

```
FORM PREPARE_SORT_TABLE  CHANGING PT_SORT TYPE LVC_T_SORT.

  DATA LS_SORT TYPE LVC_S_SORT.

  LS_SORT-SPOS = '1'.
  LS_SORT-FIELDNAME = 'CARRID'.
  LS_SORT-UP = 'X'.  "A to Z
  LS_SORT-DOWN = SPACE.
  APPEND LS_SORT TO PT_SORT.

  LS_SORT-SPOS = '2'.
  LS_SORT-FIELDNAME = 'SEATSOCC'.
  LS_SORT-UP = SPACE.
  LS_SORT-DOWN = 'X'.  "Z to A
  APPEND LS_SORT TO PT_SORT.

ENDFORM.                    " PREPARE_SORT_TABLE
```

Setting Filter Condition

The procedure is like the one in sorting. Here, the type of the table you must fill is “LVC_T_FILT”.

Filling this table is similar to filling a RANGES variable.

```
FORM PREPARE_FILTER_TABLE  CHANGING PT_FILT TYPE LVC_T_FILT.

  DATA LS_FILT TYPE LVC_S_FILT.

  LS_FILT-FIELDNAME = 'FLDATE'.
  LS_FILT-SIGN = 'E'.
  LS_FILT-OPTION = 'BT'.
  LS_FILT-LOW = '20030101'.
  LS_FILT-HIGH = '20031231'.
  APPEND LS_FILT TO PT_FILT.

ENDFORM.                    " PREPARE_FILTER_TABLE
```

Event Based Functionality

Additional Event based Functionalities that the ALV Grid can handle:-

- GENERAL SCHEME FOR THE EVENT HANDLER CLASS
- HOTSPOT CLICKING
- DOUBLE CLICKING
- PUSHBUTTONS ON THE LIST
- ADDING YOUR OWN FUNCTIONS
- OVERRIDING STANDARD FUNCTIONS
- MAKING ALV GRID EDITABLE
- CONTROLLING DATA CHANGES
- LINKING F1 HELP TO FIELDS
- LINKING F4 HELP TO FIELDS

Summary

In this lesson, you have learnt:

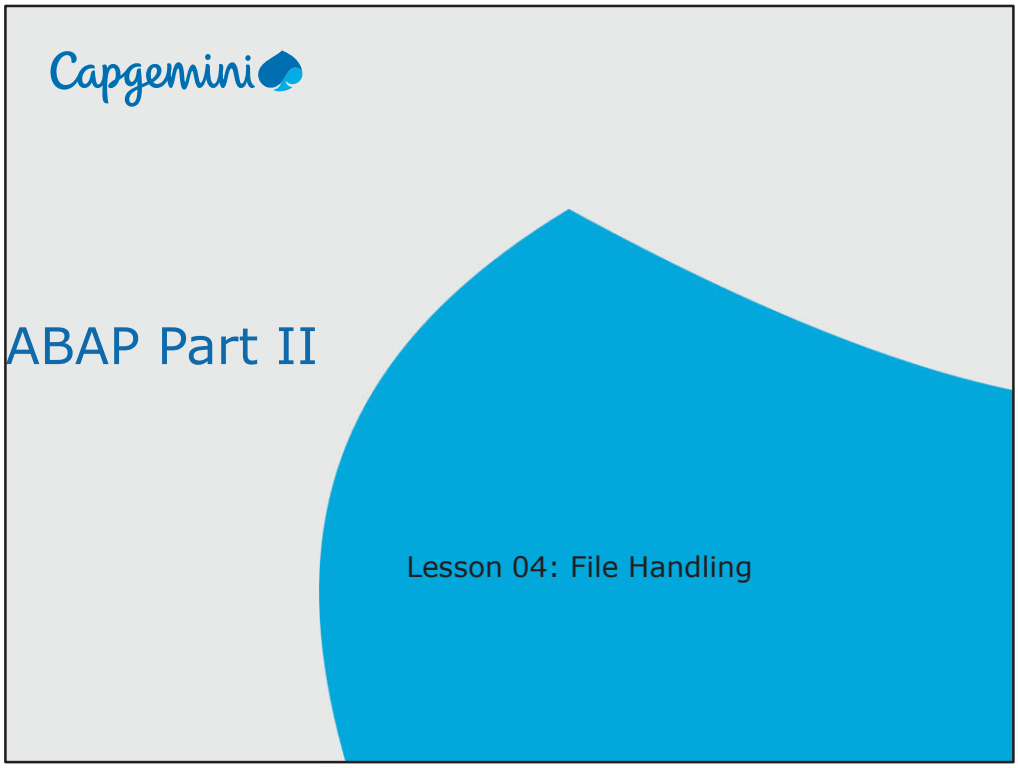
- Control Framework
- ALV Grid
- Non Event Based Functionality
- Event Based Functionality

Review Question

Question 1: _____ is used to find the length of the string.

Question 2: Condense and Concatenate command perform the same function.

- True/False

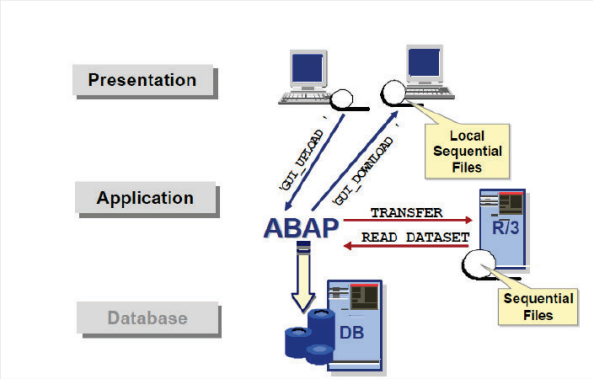

Capgemini
ABAP Part II
Lesson 04: File Handling

Lesson Objectives

After completing this lesson, participants will be able to know -

- File Handling Application Server
- File Handling Presentation Server

Introduction



Introduction

File is a place where information or data is stored. File handling in simple terms is opening, closing, reading, writing, deleting, copying, renaming the files.

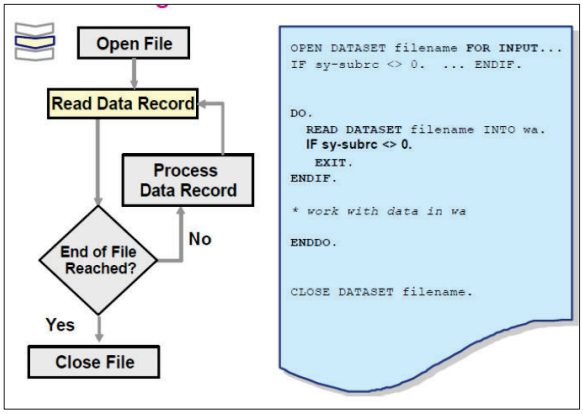
The runtime environment is implemented on the application server, which executes the ABAP programs. ABAP supports the file transfer (data transfer) technique to the application server and the front end hosts.

The interface to the file system on the application server is implemented in the form of ABAP language elements.

You can process sequential files using various file-handling methods and procedures to read data, process data and transfer it into SAP system. These files act as data source and these methods ensure consistency of the data in SAP R/3 system.

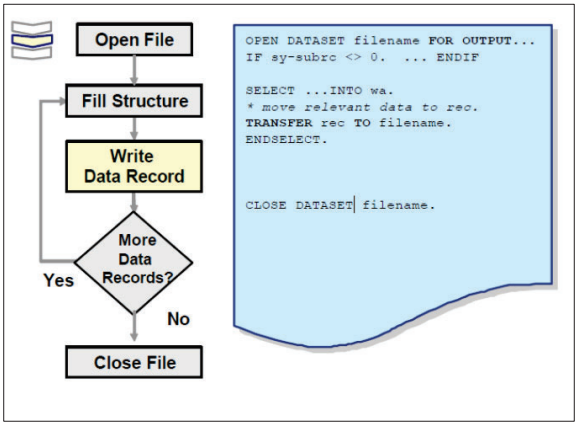
File Handling on Application Server

Overview Diagram – Read File



File Handling on Application Server

Overview Diagram – Write File



File Handling on Application Server

ABAP/4 provides five statements for handling files:

- OPEN DATASET
- CLOSE DATASET
- DELETE DATASET
- READ DATASET
- TRANSFER

File Handling on Application Server

Open Dataset

- Opens the specified file. If you do not use any additions, the file is opened for reading in binary mode. It returns SY-SUBRC = 0 if the file is opened successfully. Otherwise SY-SUBRC = 8.

Syntax

- OPEN DATASET <DSN> [Additions].

Additions:

1. FOR INPUT (Default)
2. FOR OUTPUT
3. FOR APPENDING
4. IN BINARY MODE
5. IN TEXT MODE
6. AT POSITION p
7. TYPE ctrl
8. MESSAGE mess
9. FILTER f

File Handling on Application Server

OPEN DATASET <DSN> FOR INPUT.

- This statement tries to open the field in 'read/update' mode (as long as the user has write authorization).If the user does not have write authorization, the system opens the file in 'read' mode. If this fails, an error occurs.

OPEN DATASET <DSN> FOR OUTPUT.

- This statement tries to open the file in 'write/update' mode as long as the user has read authorization. If the authorization is missing, the system opens the file in 'write' mode. If the file already exists, its existing content is deleted. If the file does not exist, the system creates it.

OPEN DATASET <DSN> FOR APPENDING.

- This statement tries to open the file in 'append' mode. If the file is already open, the system moves to the end of the file. When you open a file using FOR APPENDING, attempting to read the file sets SY-SUBRC to 4. The system display the end of the file.
- Note : You can only use one of the additions 1 to 3 in a single statement.

OPEN DATASET <DSN> IN BINARY MODE.
The contents of the file are not structured in lines in the READ DATASET or TRANSFER operations. Instead, they are input or output as a stream. You do not have to specify the IN BINARY MODE addition explicitly.

OPEN DATASET <DSN> IN TEXT MODE.
If you use this addition, the contents of the file are structured in lines. Each time you use the READ DATASET or TRANSFER statement, the system reads or writes a single line. If the data object to which you are transferring the data is too big, it is padded with spaces. If it is too small, the data record is truncated.

Note : You can only use one of additions 4 and 5 in a single statement

OPEN DATASET <DSN> AT POSITION p.
Use this addition to specify the explicit starting position p in the file (calculated in bytes from the start of the file). The next read or write operation will start at this position. You cannot position before the beginning of the file. Do not use this addition with the IN TEXT MODE addition, since the physical representation of a text file depends heavily on the underlying operating system.
If you use OPEN ... FOR OUTPUT AT POSITION ..., the contents of the file are destroyed if the file already existed. To avoid this, use OPEN ... FOR INPUT AT POSITION ... instead.

Note: OPEN ... AT POSITION p does not work for file positions where p >= 2 Gigabytes.

OPEN DATASET <DSN> TYPE ctrl .
You can use the ctrl field to specify further file attributes. The contents of this field are passed unchanged and unchecked to the operating system. The syntax for the attributes is dependent on the operating system.

OPEN DATASET <DSN> MESSAGE msg.
If an error occurs while the file is being opened, the corresponding operating system message is placed in field msg.
Example

```
DATA: DSN(20) VALUE '/USR/TEST.DAT', MSG(100). OPEN DATASET
DSN FOR INPUT MESSAGE MSG. IF SY-SUBRC <> 0. WRITE / MSG.
ENDIF.
```

OPEN DATASET <DSN> FILTER f.
If you are working under UNIX or Windows NT, you can specify an operating system command in the field f.
Example Under UNIX, the following statements opens the file DSN and writes the data to the file in compressed form because of the UNIX command 'compress' :

```
DATA DSN(20) VALUE '/USR/TEST.DAT'.
OPEN DATASET DSN FOR OUTPUT FILTER 'COMPRESS'.
```

File Handling on Application Server

CLOSE DATASET

- Closes the specified file.
  - Syntax CLOSE DATASET <DSN>.

DELETE DATASET

- Deletes the file specified file. If it deletes the file successfully it returns SY-SUBRC = 0. Otherwise returns SY-SUBRC = 4. The possible reasons for failing are:
  - The file does not exist
  - The file is a directory
  - The file is a program that is currently running

File Handling on Application Server

READ DATASET

- Used to read a record from a file.

Syntax

- READ DATASET DSN INTO F.

Addition : LENGTH LEN.

- The actual length of the data objet read is placed in the field LEN after the read access. LEN must be defined as a variable. A syntax error will occur if you define it as a constant. The following example displays 9.

Example

```
DATA: LEN TYPE I,
TEXT(30) TYPE C VALUE 'BEETHOVEN',
DIR(30) TYPE C VALUE '/USR/TEST.DAT'.
OPEN DATASET DIR IN TEXT MODE.
TRANSFER TEXT TO DIR.
CLOSE DATASET DIR.
OPEN DATASET DIR IN TEXT MODE.
READ DATASET DIR INTO TEXT LENGTH LEN.
CLOSE DATASET DIR.
WRITE / LEN.
```

File Handling on Application Server

TRANSFER

- Used to write a record into a file.

Syntax

- TRANSFER F TO DSN.
  - Transfers the data object f to a sequential file whose name is specified in DSN. DSN can be a field or a literal. You must already have opened the file. . If the specified file is not already open, TRANSFER attempts to open the file FOR OUTPUT IN BINARY MODE. If this is not possible, a runtime error occurs f can be a field, a string, or a structure.

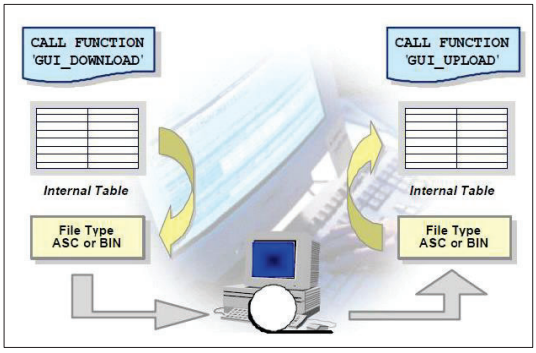
Addition : LENGTH LEN.

- The length of the data object to be written is defined by LEN, where LEN can be either a constant or a variable. If LEN is smaller than the length of the data object f, the system truncates character fields (C, N, D, T, X,P, STRING) on the right. With type I or F fields, unexpected results may occur if LEN is shorter than the default length for the field type

File Handling on Presentation Server

To work with files on the presentation server , SAP provides some special function modules GUI_UPLOAD, for reading from a file, and

GUI_DOWNLOAD, for writing into the file. An internal table must be used as an interface between the program and the function module



File Handling on Presentation Server

GUI Download - Signature

Writing data to a file on the presentation server:

To write data from an internal table to a file on the presentation server, use function module GUI_DOWNLOAD.

The most important parameters that are exported are as follows:

```
CALL FUNCTION 'GUI_DOWNLOAD'
  EXPORTING
*   BIN_FILESIZE
    filename                =
*   FILETYPE                = 'ASC'
*   APPEND                  = ' '
*   WRITE_FIELD_SEPARATOR   = ' '
*   HEADER                  = '00'
*   TRUNC_TRAILING_BLANKS   = ' '
*   WRITE_LF                = 'X'
*   COL_SELECT              = ' '
*   COL_SELECT_MASK         = ' '
*   DAT_MODE                = ' '
*   CONFIRM_OVERWRITE       = ' '
*   NO_AUTH_CHECK           = ' '
*   CODEPAGE                = ' '
*   IGNORE_CERR             = ABAP_TRUE
*   REPLACEMENT             = '#'
*   WRITE_BOM               = ' '
*   TRUNC_TRAILING_BLANKS_EOL = 'X'
*   WK1_N_FORMAT            = ' '
*   WK1_N_SIZE              = ' '
*   WK1_T_FORMAT            = ' '
*   WK1_T_SIZE              = ' '
*   WRITE_LF_AFTER_LAST_LINE = ABAP_TRUE
*   SHOW_TRANSFER_STATUS    = ABAP_TRUE
IMPORTING
*   FILELENGTH              =
TABLES
    data_tab                =
*   FIELDNAMES              =
*EXCEPTIONS
*   FILE_WRITE_ERROR        = 1
*   NO_BATCH                = 2
*   GUI_REFUSE_FILETRANSFER = 3
*   INVALID_TYPE            = 4
*   NO_AUTHORITY            = 5
*   UNKNOWN_ERROR           = 6
*   HEADER_NOT_ALLOWED      = 7
*   SEPARATOR_NOT_ALLOWED   = 8
*   FILESIZE_NOT_ALLOWED    = 9
*   HEADER_TOO_LONG         = 10
*   DP_ERROR_CREATE         = 11
*   DP_ERROR_SEND           = 12
*   DP_ERROR_WRITE          = 13
*   UNKNOWN_DP_ERROR        = 14
*   ACCESS_DENIED           = 15
*   DP_OUT_OF_MEMORY        = 16
*   DISK_FULL               = 17
*   DP_TIMEOUT              = 18
*   FILE_NOT_FOUND          = 19
*   DATAPROVIDER_EXCEPTION  = 20
*   CONTROL_FLUSH_ERROR     = 21
*   OTHERS                  = 22
          .
```

File Handling on Presentation Server

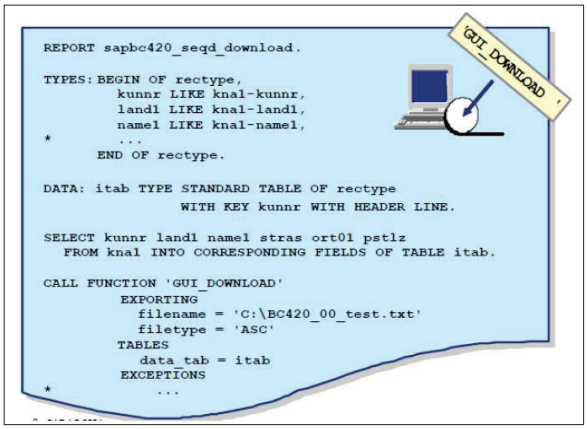
Some of the important parameters that are exported are as follows:

- BIN_FILESIZE - File Length for binary files. A length of zero or the length which is larger than the number of bytes in the internal table (width * number of lines) causes an exception.
- FILENAME - The name of the file that is to be generated on the presentation server(if necessary with predefined path name). If the path doesn't exist or the file cannot be opened, an exception will be raised.
- APPEND - By default, existing local files are overwritten by new versions. By setting APPEND to 'X', the downloaded data is appended to an existing file. If the file does not yet exist, it is created.
- CONFIRM_OVERWRITE - If this parameter is set, a file is overwritten only after a confirmation by the user
- FILELENGTH - Number of bytes transferred
- Tables Parameter - DATA_TAB
  - The source internal table whose contents are downloaded into a file.

FILETYPE - The target format of the file. Valid values are:
'ASC' : ASCII format, the table is stored with rows.
'DAT': ASCII format as in 'ASC', additional column separation with TABs.
'BIN' : Binary format (specification of BIN_FILESIZE required)
'DBF' : Stored as Dbase file (always with DOS code page).
'IBM' : ASCII format as in 'ASC' with IBM code page conversion (DOS)
'WK1' : Data is downloaded in Lotus 1-2-3 format

File Handling on Presentation Server

Example of GUI_DOWNLOAD




```
REPORT sapbc420_seqd_download.

TYPES: BEGIN OF rectype,
         kunnr LIKE kna1-kunnr,
         land1 LIKE kna1-land1,
         name1 LIKE kna1-name1,
*        ...
       END OF rectype.

DATA: itab TYPE STANDARD TABLE OF rectype
                WITH KEY kunnr WITH HEADER LINE.

SELECT kunnr land1 name1 stras ort01 pstlz
  FROM kna1 INTO CORRESPONDING FIELDS OF TABLE itab.

CALL FUNCTION 'GUI_DOWNLOAD'
          EXPORTING
            filename = 'C:\BC420_00_test.txt'
            filetype = 'ASC'
          TABLES
            data_tab = itab
          EXCEPTIONS
*           ...
```

File Handling on Presentation Server

GUI Upload - Signature

- Reading data from a file on the presentation server:
- To read data from the presentation server into an internal table we use the function module GUI_UPLOAD. The most important parameters that are exported are as follows

```
CALL FUNCTION 'GUI_UPLOAD'
    EXPORTING
      filename                  =
    *  FILETYPE                  = 'ASC'
    *  HAS_FIELD_SEPARATOR       = ' '
    *  HEADER_LENGTH             = 0
    *  READ_BY_LINE              = 'X'
    *  DAT_MODE                  = ' '
    *  CODEPAGE                  = ' '
    *  IGNORE_CERR               = ABAP_TRUE
    *  REPLACEMENT               = '#'
    *  CHECK_BOM                 = ' '
    *  VIRUS_SCAN_PROFILE        =
    *  NO_AUTH_CHECK             = ' '
  IMPORTING
    *  FILELENGTH                =
    *  HEADER                    =
  TABLES
      data_tab                  =
```

File Handling on Presentation Server

GUI Upload - Signature

Some of the exporting parameters

- FILENAME - Name of the file
- FILETYPE - The source file type. Valid values are:
  - 'BIN' : Binary files.
  - 'ASC': ASCII files, text files with end-of-line markers.
  - 'DAT': The file is loaded line by line into the transferred table. Tabs in the file mean a change of field.
- HAS_FIELD_SEPARATOR: Specifies if the fields in the file are separated by a tab. This is necessary if the structure passed contains several components. CR/LF occurs instead of a tab after the last field of a row

File Handling on Presentation Server

GUI Upload - Signature

- Importing Parameter
  - FILELENGTH : Number of bytes transferred.
- Table parameter
  - DATA_TAB : Internal target table, to which the data is loaded.
- Exceptions
  - CONVERSION_ERROR - Errors in the data conversion.
  - FILE_OPEN_ERROR - System cannot open file.
  - FILE-READ_ERROR - System cannot read from file
  - INVALID_TABLE_WIDTH - Invalid table structure
  - INVALID_TYPE - Invalid value for parameter FILETYPE

File Handling on Presentation Server

GUI Upload - Example

```
REPORT sapbc420_seqd_upload.

TYPES: BEGIN OF rectype,
*        ...
       END OF rectype.

DATA: itab TYPE STANDARD TABLE OF rectype
                 WITH KEY kunnr WITH HEADER LINE,
      wa LIKE LINE OF itab.

CALL   FUNCTION 'GUI_UPLOAD'
          EXPORTING
            filename = 'C:\BC420_00_test.txt'
            filetype = 'ASC'
          TABLES
            data_tab = itab
          EXCEPTIONS
*         ...

LOOP AT itab INTO wa.
  WRITE: / wa-kunnr, wa-land1, ...
ENDLOOP.
```

File Archiving

Need: Many a times file cannot be deleted after processing is complete. There are certain implications due to government regulations, taxation (IRS) regulations, FDA requirements, internal organizational requirements, and audit requirements.

Method: As against standard SAP archiving way, one can archive processed file by moving it to a pre-defined folder on the application server. In future if need arise one can retrieve the file from archive folder.

Summary

In this lesson, you have learnt:

- Reading files from Presentation Server
- Reading files from Application Server

Review Question

Question 1: If the dataset is opened in write mode and the If the file already exists, its existing content is deleted.

- True/False

Question 2 A File can be opened for Output and Read at the same time.

- True/False