ABAP Part II

Lesson 01: Reporting Events

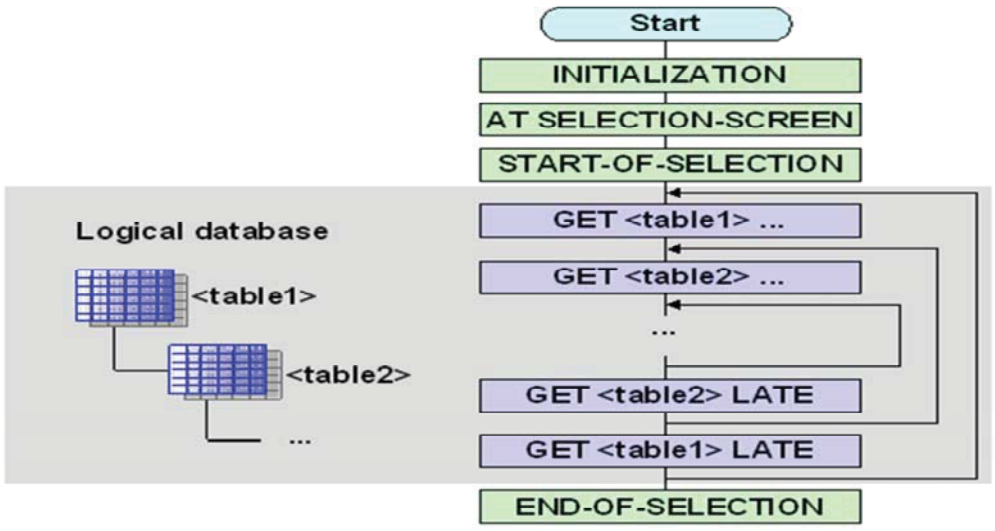
Lesson Objectives

- After completing this lesson, participants will be able to -
  - Identify the different events to trigger the Interactive report.
  - Purpose and usage of reporting events and its flow.
  - How to format the list output.
  - How to use the conditional and unconditional page breaks.
  - Illustrate the Interactive reporting events. AT LINE-SELECTION and AT USER-COMMAND.
  - Illustrate using the menu painter.

Reporting Events

- Event Blocks in Executable Programs
  - When you run an executable program, the program flow is controlled by the external events in the ABAP runtime environment. The following diagram (Next Slide) shows the sequence of the events.
  - The events in the gray box are only processed if you have entered a logical database in the program attributes. The AT SELECTION-SCREEN event is only processed if a selection screen is defined in the program or the logical database linked to the program.
  - The other events occur when any executable program is runs.

Reporting Events



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Event Blocks in Executable Programs

- The following events occur when you run a typical executable program
  - INITIALIZATION: Before the standard selection screen is displayed.
  - AT SELECTION-SCREEN: After user input on a selection screen has been processed, but while the selection screen is still active.
  - START-OF-SELECTION: After the standard selection screen has been processed, before data is read from the database.
  - END-OF-SELECTION: After all data has been read by the logical database.
  - TOP-OF-PAGE: In list processing when a new page starts.
  - END-OF-PAGE: In list processing when a page ends.
  - AT LINE-SELECTION: Event is triggered by either the user double clicking a particular line or using F2 to select it.
  - AT PF<NN>: When the user triggers the predefined function code PF<NN>.
  - AT USER-COMMAND: When the user triggers a function code defined in the program.
  - TOP-OF-PAGE DURING LINE SELECTION: Event called during list processing when a detailed list is called.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

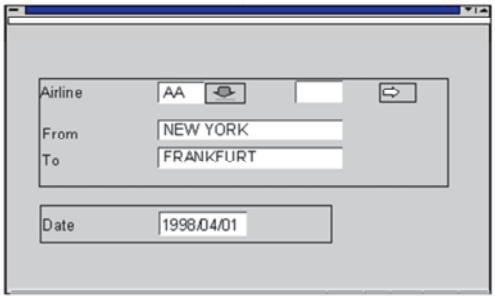
Initialization

- This event occurs before the standard selection screen is called.
- The input fields of the standard selection screen can only be initialized once the program has been started.
- Processed before the presentation of the selection screen can be used to initialize values in the selection screen or to assign values to any parameters or the select-options that appear on the selection screen.
- If an executable program declares a standard selection screen, the same program will be automatically called again by the system once the selection screen has been executed. This triggers the INITIALIZATION event again.

Initialization

```
REPORT  ZCAP_REPORT.
PARAMETERS DATUM TYPE SY-DATUM DEFAULT SY-DATUM.
NODES SPFLI.
INITIALIZATION.
CITY_FR = 'NEW YORK'. CITY_TO = 'FRANKFURT'.
CARRID-SIGN = 'I'. CARRID-OPTION = 'EQ'.
CARRID-LOW = 'AA'. APPEND CARRID.
DATUM+6(2) = '01'.
```

Output:



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

At Selection-Screen

- This is the basic form of events which occurs when the selection screen is being processed.
- It is used to validate the information which is entered in the selection screen.
- The standard selection screen is called automatically in the mid of the INITIALIZATION and START-OF-SELECTION events, either in an executable program or in the logical database which is linked to it.
- Occurs after all the input data is passed to the underlying ABAP program from selection screen.
- On passing the input data from Selection Screen to ABAP Program by the runtime environment, AT SELECTION-SCREEN event is triggered.

At Selection-Screen

- Processing block is started after the user has specified all the criteria in the selection screen.
- Selection Screen Processing:
  - Started after the INITIALIZATION event triggered.
  - Other events may be triggered for fields or for F4 help, depending upon user action on the selection screen.
- This event can also be called on a particular parameter or select-option using At Selection-Screen on <parameter or select-option>.
- To modify the Selection Screen elements before display, AT SELECTION-SCREEN OUTPUT event is used.

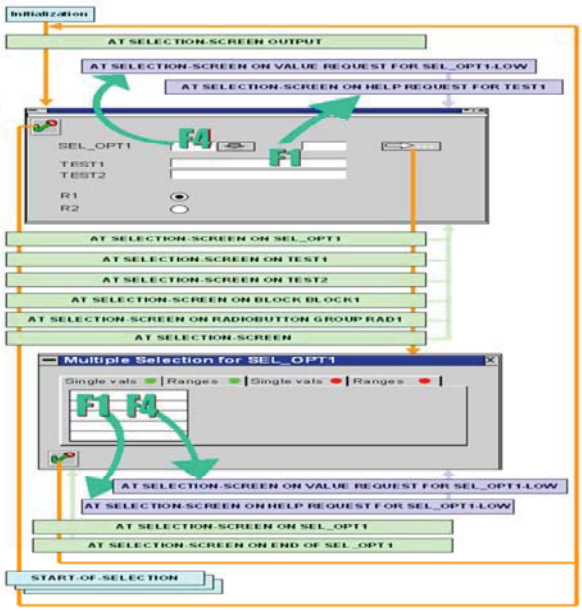
At Selection-Screen

```
DATA FIELD1(10).

SELECT-OPTIONS SEL_OPT1 FOR FIELD1.

SELECTION-SCREEN BEGIN OF BLOCK BLOCK1.
  PARAMETERS:  TEST1(10),
               TEST2(10).
SELECTION-SCREEN END OF BLOCK BLOCK1.

PARAMETERS: R1 RADIOBUTTON GROUP RAD1 DEFAULT 'X',
            R2 RADIOBUTTON GROUP RAD1.
```



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Start-of-Selection

- The associated event is raised by the ABAP Runtime environment during an executable program.
- It is used to prepare the required data for reading and also creating the list.
- Processing block is executed after processing the selection screen
- All the data is selected in this block.
- All the main processing on the data except for interactive reporting is handled in this block.
- Event Occurs at:
  - After the selection screen has been processed.
- Non-declarative statements which are written in between the report and first processing block are also processed in start of selection event.

End-of-Selection

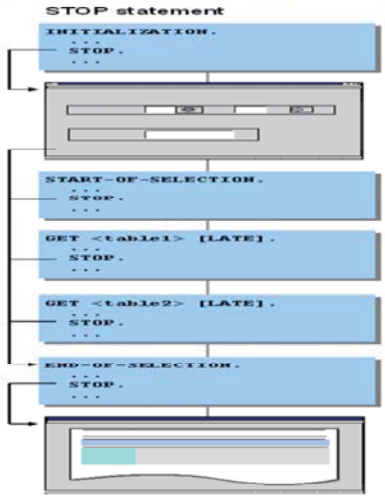
- This is the last event which is processed by the system.
- Data which is selected and has been processed is printed to the screen in this block
- This event is triggered after all the data has been read from the program/logical database before the list processor is being started.
- It is also used to process the data that has been stored in sequential datasets like internal tables or extracts by the program

Exiting Event Blocks

- Series of statements that allows to leave an event block in the program.
- The processing of further statements depends on the statements and the events in which it is used.
- Statements are used:
  - STOP
  - EXIT
  - CHECK

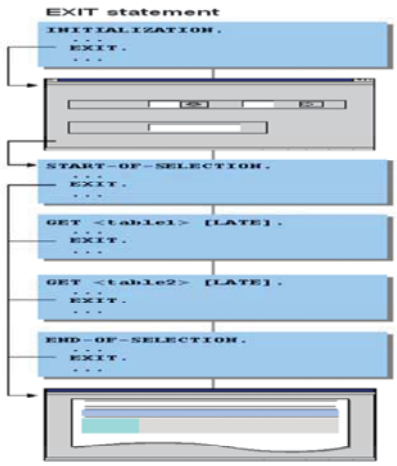
Leaving Event Blocks using STOP

- If you use the STOP statement within an event block, the system stops processing the block immediately. The ABAP runtime environment triggers the next event accordi



Leaving Event Blocks using EXIT

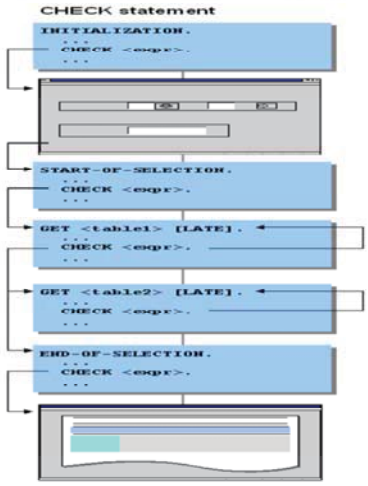
- From the START-OF-SELECTION event onwards, the system starts the list processor directly when the EXIT statement occurs, and displays the list.
- If the EXIT statement occurs in a loop using DO, WHILE, or LOOP, it is the loop that termina



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Leaving Event Blocks using CHECK

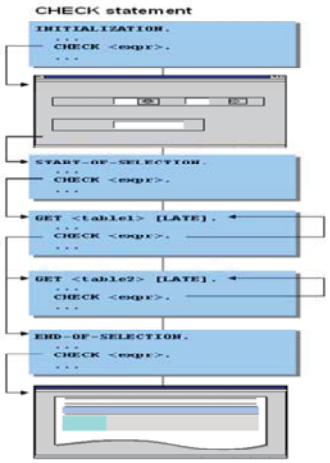
- If you use the CHECK <EXPR> statement within an event block but not within a loop, and the condition <EXPR> is not fulfilled, the system exits the processing block imr



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

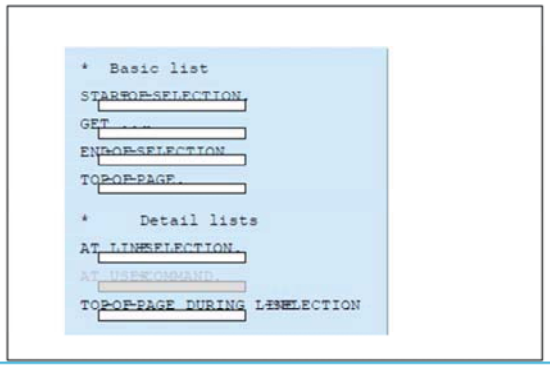
Leaving a GET event block using REJECT

- The REJECT statement was specially developed for leaving GET event blocks. Unlike CHECK , EXIT and REJECT always refers to the current GET event block. The REJECT statement allows you to exit a GET event block directly from a loop or a subrouti

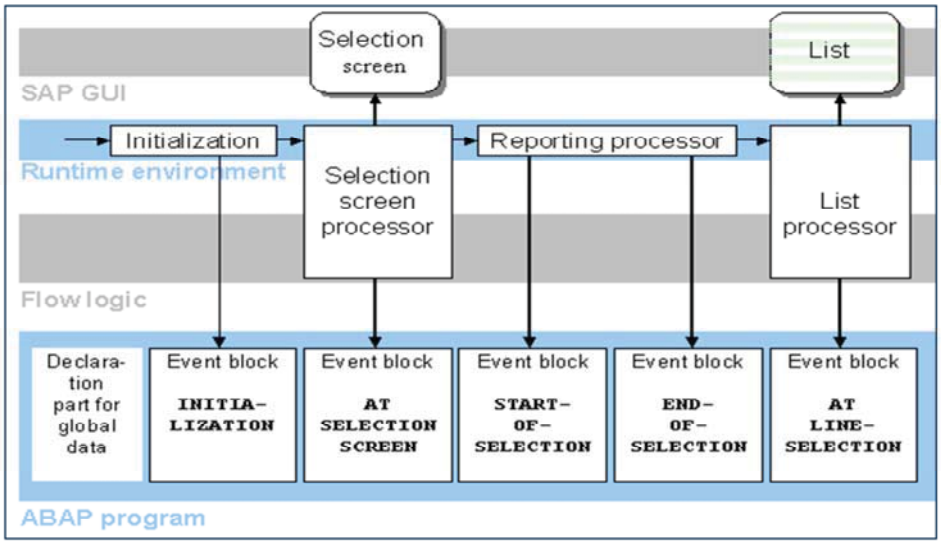


Event Blocks in Executable Programs

- The events START-OF-SELECTION, GET, END-OF-SELECTION, TOP-OF-PAGE and END-OF-PAGE can be used only to create basic lists. Once you leave a basic list, these events are no longer processed.
- Detail lists are created using two basic events: AT LINE-SELECTION and AT USER-COMMAND. The event TOP-OF-PAGE DURING LINE-SELECTION is used to create headers for all detail lists.



Interactive Process Flow



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Interactive Reporting Events

- Lists are the output medium for structured, formatted data from ABAP programs.
- In list processing the event will be intercepted by list processor and the list is processed.
- One of the following list events may be called, that depends on the function code which is triggered by the user.
  - AT LINE-SELECTION
  - AT USER-COMMAND

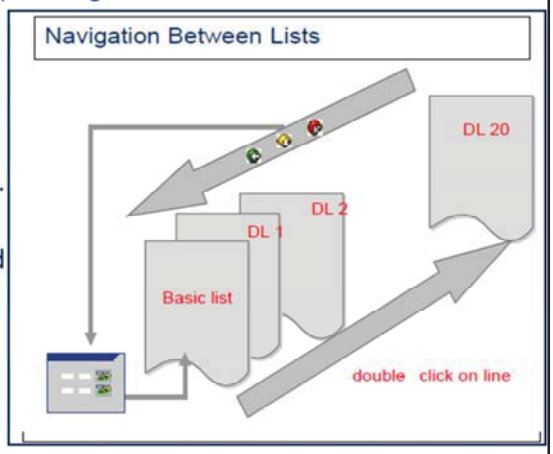
AT Line-Selection

- If the user double-clicks on a report line, the AT LINE-SELECTION event is triggered. The event can also be triggered by single clicking on a report line and either pressing the Detail icon in the application tool bar, pressing F2, or entering the word PICK in the OK code field and pressing Enter.

- A program can display up to 21 lists, out of which one is the basic list and 20 are the secondary lists. (Detailed List DL1, DL 2……and DL 20).

- The basic list is nothing but the standard screen of an executable program.

- Each list has in its own individual memory area called a list buffer.



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Creating Detail-Lists

- Detail lists are the lists created during an interactive list event
- Every time the user performs an action on a list, runtime environment will check whether there is an event block defined corresponding to the function code
- If an event block exists, the system field SY-LSIND is automatically increased by one and the relevant event block is then executed
- The list output that arises during this event block places its data into a new list (list level) with the index which is equal to SY-LSIND
- By default, the new list overwrites the previous list, however the list can also be displayed in the form of dialog box

Creating Detail-Lists

- Each interactive list event will create a new detail list
- If a list is created by the user on the next level , the system will store the previous list and displays the new one
- The user can interact with whichever list is currently displayed
- No standard page header can be defined for detail lists

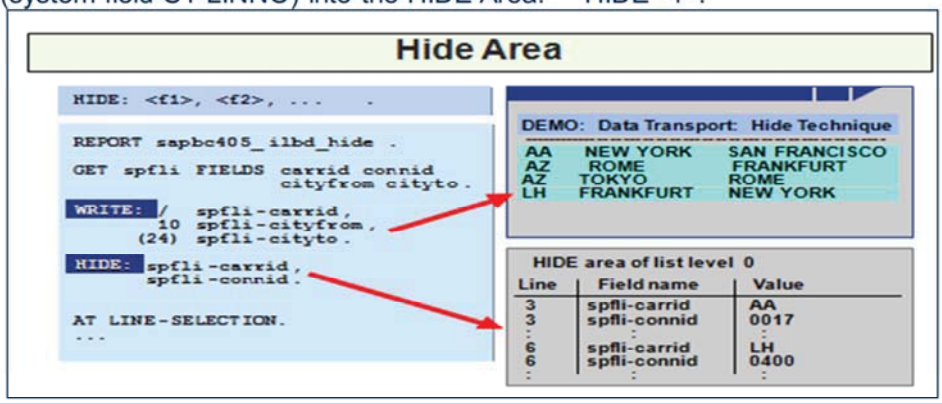
Consequences of Event Control

- It is not possible to nest the Processing blocks
- The event TOP-OF-PAGE cannot be used in secondary lists but TOP-OF-PAGE DURING LINE-SELECTION can be used
- The event END-OF-PAGE in secondary lists is not processed by the system
- The GET and GET LATE cannot be used to retrieve data for secondary lists

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

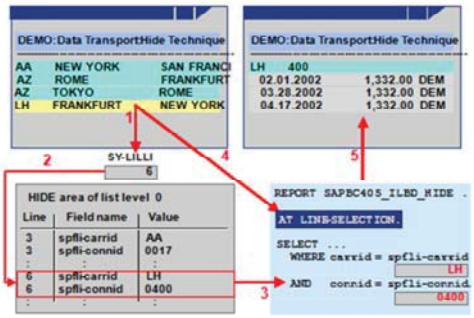
Data Transport by using HIDE

- The HIDE keyword is used to store data objects and their values so they can be made available when the User selects a report line. When a line is selected, the fields that were hidden are filled with the values that you hid for that line.
- It places the contents of the variable <f> for the current output line (system field SY-LINNO) into the HIDE Area. HIDE <f>.



Line Selection – Hide Statement

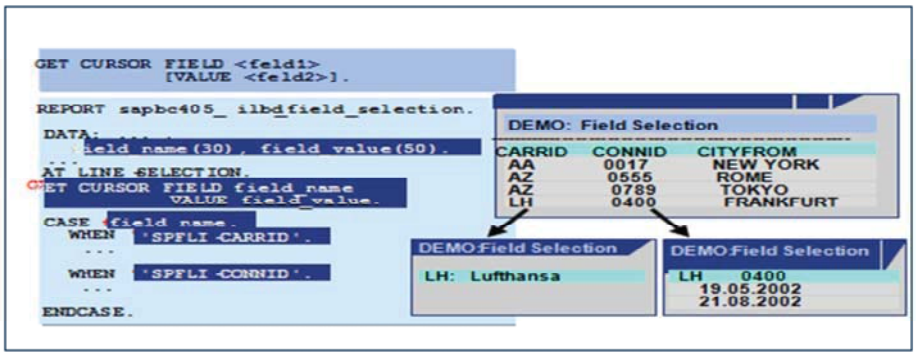
- The user selects a line for which data has been stored in the HIDE area.
- The runtime system evaluates field SY-LILLI to determine the selected line.
- The runtime system jumps to the point in the HIDE area where data for this line is stored.
- The runtime system then inserts all values stored for the selected line in the HIDE area into their corresponding fields.
- The runtime system processes the event AT corresponding program processing block.
- A detail list is created.



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Field Selection - GET CURSOR

- GET CURSOR statement is used to create detail lists according to the cursor position.
- The parameter FIELD provides the name of an output field. The parameter VALUE provide the output value.
- GET CURSOR FIELD <f> [OFFSET <off>] [LINE <lin>] [VALUE <val>] [LENGTH <len>].



Lists in Dialog Box

- List can be displayed in a dialog box instead of on the full screen using the following WINDOW statement

```
WINDOW STARTING AT <left> <upper> [ENDING AT <right> <lower>].
```

- The WINDOW statement takes effect only within the interactive event's processing block which is only for detail lists
- Dialog boxes do not have either menu bar or standard toolbar
- The application toolbar is seen at the bottom of the dialog box, this feature is common to all dialog boxes in the R/3 System

Reading and Modifying List Line

- READ - Reading a Line from a List
  - READS LINE NUMBER LINE OF THE LIST, USUALLY AFTER A LINE SELECTION
    - READ LINE LINE.
    - READ LINE LINE OF CURRENT PAGE.
    - READ LINE LINE OF PAGE PAG
    - READ CURRENT LINE
- MODIFY LINE n.
  - Change a List Line
    - INDEX IDX - Changes the corresponding line in the list at list level IDX
    - FIELD VALUE f1 FROM G1 ... FN FROM GN
- WINDOW STARTING AT X1 Y1 ENDING AT X2 Y2.
  - Displays the current secondary list as a modal dialog box only up to 20 windows

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Reading Lines from the Lists

- Used to read a line from a list after an interactive list event.

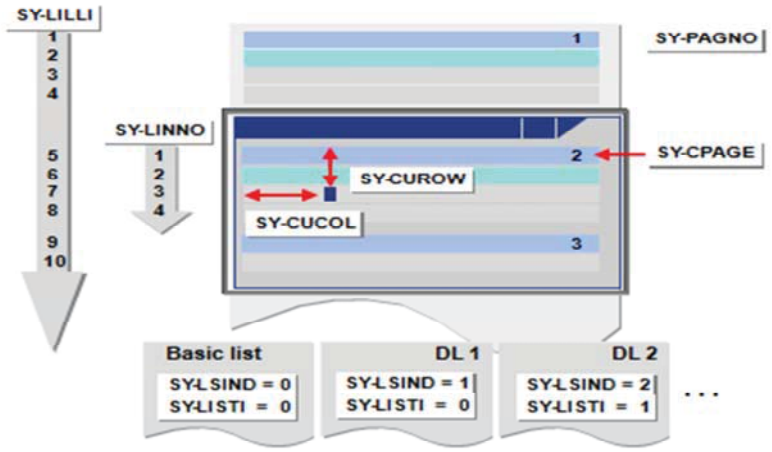
```
READ LINE <lin> [INDEX <idx>] [FIELD VALUE <f1> [INTO <g 1>] ... <f n> [INTO <g n>]]
  [OF CURRENT PAGE|OF PAGE <p>].
```

  - The statement stores the contents of line <lin> from the list on which the event was triggered (index SY-LILLI) in the SY-LISEL system field and will fill all HIDE information that is stored for this line back to the corresponding fields

```
READ CURRENT LINE [FIELD VALUE <f1> [INTO <g 1>] ...].
```

  - This statement reads a line read before by an interactive event or by READ LINE

System Fields for Interactive Lists



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

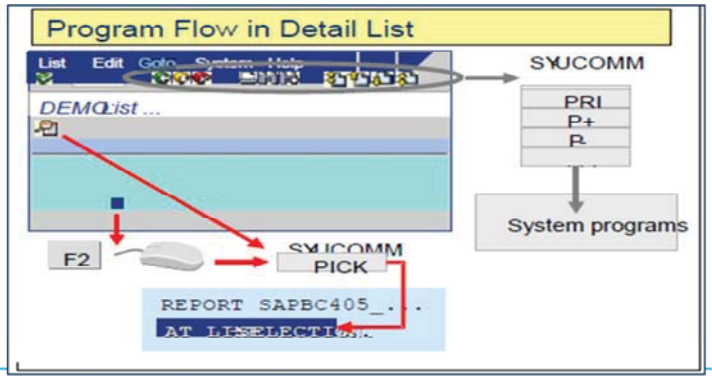
System Fields for Interactive Lists

- SY-LSIND Index for the current list.
- SY-LISTI Index of the last list displayed.
- SY-LILLI Absolute number of a selected line in the list displayed.
- SY-CPAGE Number of the upper-most displayed line in the list displayed.
- SY-LISEL Contents of the line from which the event was triggered.
- SY-CUCOL Number of the column in the window where the cursor was last positioned in the list displayed.
- SY-CUROW Number of the line in the window where the cursor was last positioned in the list displayed.
- SY-STACO Number of the first column displayed in the list displayed.
- SY-STARO Number of the first visible line in the top displayed page (SY- PAGE) in the list displayed (not including header lines).
- 10.SY-UCOMM Function code that triggered the interactive event in the list
- displayed.
- 11.SY-PFKEY Status of the list displayed.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

AT USER-COMMAND

- If the report has a custom GUI and the user selects a menu option, the event AT USER-COMMAND is triggered. The function code assigned to the menu option can be evaluated using a CASE construct to determine which menu option was selected.
- All other function codes are either intercepted by the runtime environment or trigger the event AT USER-COMMAND.



AT USER-COMMAND

- Function codes that trigger AT USER-COMMAND must be defined in the own GUI status.
- GUI status for lists can be defined and attached to list level using
  - SET PF-STATUS in executable program.
  - Function codes are used for this purpose and they will be maintained in the GUI status of the list screen.
- The GUI status will be maintained in Menu Painter tool in the ABAP Workbench.
- The system assigns function codes to user actions that are list-specific.

Dialog Status and Title in program

- Setting a Dialog-Status

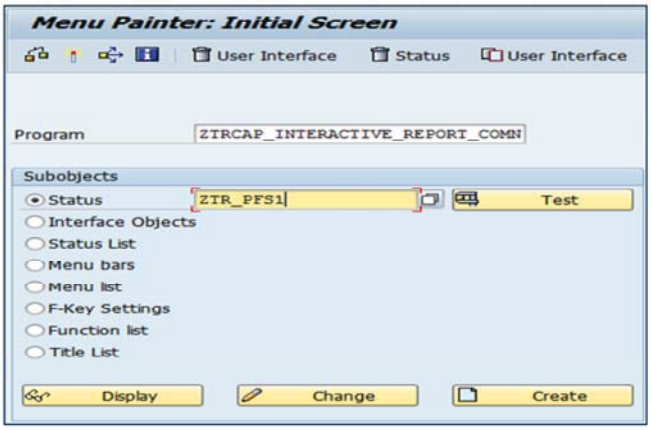
  SET PF-STATUS <stat> [EXCLUDING <f>|<itab>]
  [OF PROGRAM <prog>].
  - This statement sets the status <stat> for the current output list.
  - SET PF-STATUS SPACE is used to set the standard list status.
- Setting a Title for a List

  SET TITLEBAR <ttl> [WITH <g1> ... <g9>]
  [OF PROGRAM <prog>].
  - The above statement will set the title of the user interface for the output list.
  - This will remain active for all screens until another SET TITLEBAR is specified.

Menu Painter

- It is a tool used to design the user interfaces for ABAP Programs.
- The GUI status will be maintained in Menu Painter tool in the ABAP Workbench
- GUI status means, the instance of the interface consisting of menu bar, a standard toolbar, an application toolbar, and a function key setting.

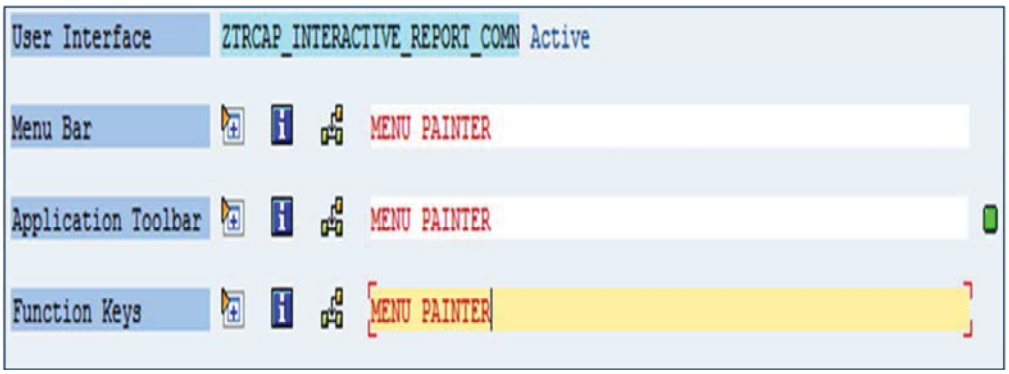
- The GUI status and GUI title defines how the user interface will look and behave in an ABAP program.
  - T-CODE SE41.



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

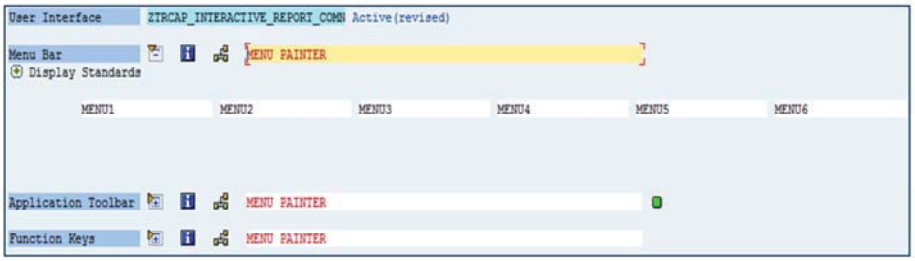
Menu Painter

- Menu Bar Consists of the following the options.
  - Menu Bar
  - Application Toolbar
  - Function Keys



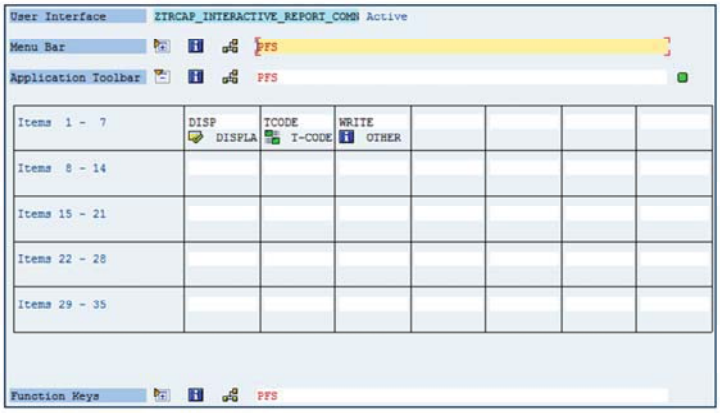
Menu Painter - Menu Bar

- A menu bar may contain six menus.
- Two standard menus are added by default
  - System
  - Help
- The two menus are displayed at the right-hand end of the menu bar



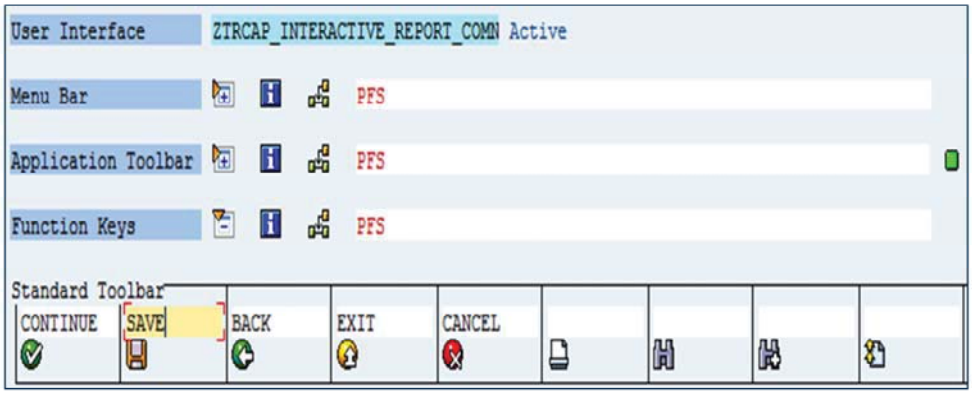
Menu Painter - Application Toolbar

- A function to a function key must be assigned before it can be included in the application toolbar.
- A maximum of 35 pushbuttons in the application toolbar can be included.
- Application toolbar functions contains
  - Icon
  - Text or
  - Both Icon and Text



Menu Painter - Function Keys

- A function keys contain the.
  - Standard Toolbar.
  - Recommended Function Key Settings.
  - Freely Assigned Function Keys.



Review Question

- Question 1. _____event occurs before the standard selection screen is called.
- Question 2: The ____ keyword is used to store data objects.
- Question 3: _____ is a tool used to design the user interfaces for ABAP Programs.

Summary

- In this lesson, you have learnt:
  - To Generate a list using simple reports
  - The Structure of Logical Database and use it in reports
  - To use different events in the report
  - To Illustrate events triggering interactive reports
  - To Develop programs on Interactive reports by using the At Line-Selection and At User-Command Events.
  - To Design Menus, Function Keys and application tool bar.

ABAP Part II

Lesson 02: Modularization Techniques

Lesson Objectives

- After completing this lesson, participants will be able to -
  - Function Modules
  - Subroutines

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 2

Modularization

- ABAP contains the following kinds of procedures:
  - Subroutines
    - 1.Subroutines are principally for local modularization, that is, they are generally called from the program 2.in which they are defined. You can use subroutines to write functions that are used repeatedly within a program. You can define subroutines in any ABAP program.
  - Function Modules
    - 1.Function modules are for global modularization, that is, they are always called from a different program.
    - 2.Function modules contain functions that are used in the same form by many different programs. They are important in the R/3 System for encapsulating processing logic and making it reusable.
    - 3.Function modules must be defined in a function group, and can be called from any program.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

Modularization - Subroutines

- A subroutine is a block of code introduced by FORM and concluded by ENDFORM.

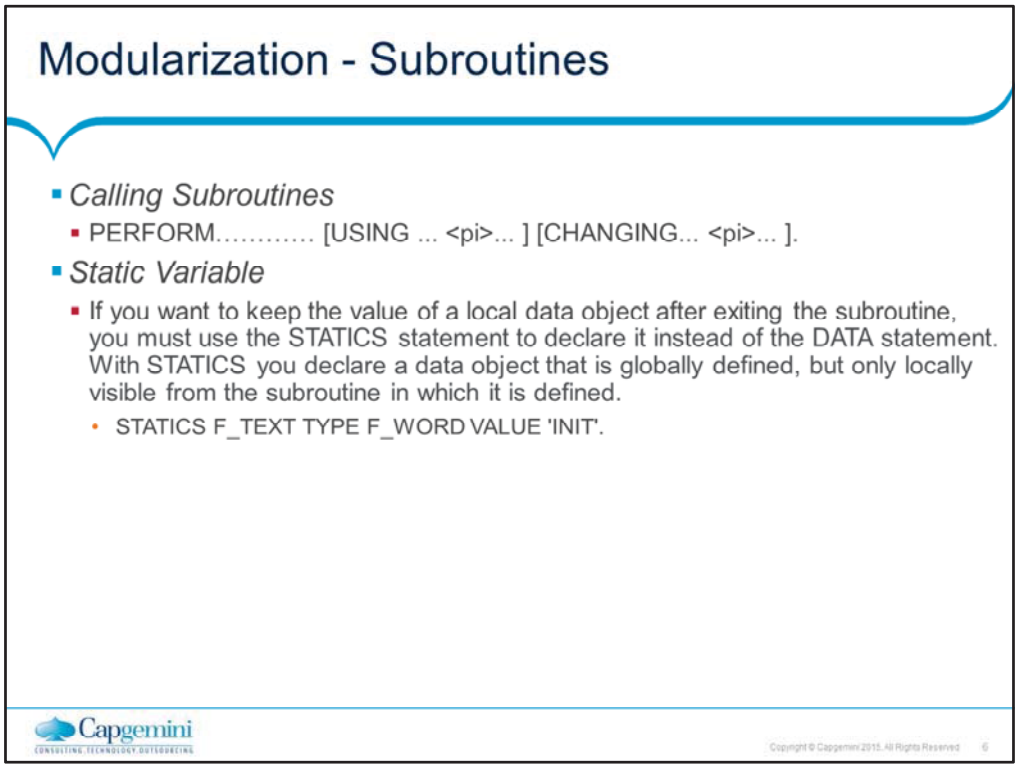
```
FORM <SUBR> [USING ... [VALUE()<pi>[)] [TYPE <t>|LIKE <f>]... ] [CHANGING...
[VALUE()<pi>[)] [TYPE <t>|LIKE <f>]... ].
...
ENDFORM.
```

  - <SUBR> is the name of the subroutine.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

Modularization - Subroutines

- *The Parameter Interface*
  - 1.Parameters Passed by Reference - list these parameters after USING without the VALUE addition
  - 2.Parameters Passed by Value - list these parameters after USING with the VALUE addition
  - 3.Passing by Value and Result - If you want to return a changed output parameter from a subroutine to the calling program only after the subroutine has run successfully, use CHANGING for the <pass> option of the FORM and PERFORM



Variables defined by the data statement at the top of the program are global.
Data definitions within a subroutine are local to the subroutine.
Memory is allocated for these variables when the subroutine is called, and freed when it returns.
Use the statics statement to create local variables that are not freed when the subroutine ends.
The syntax for statics is the same as the syntax for the data statement.
The memory for a static variable is allocated the first time the subroutine is called, and retained when the subroutine ends.
The memory of a static variable belongs to the subroutine that allocated it; that variable is not visible from other subroutines

Demo: Create Simple Subroutine

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

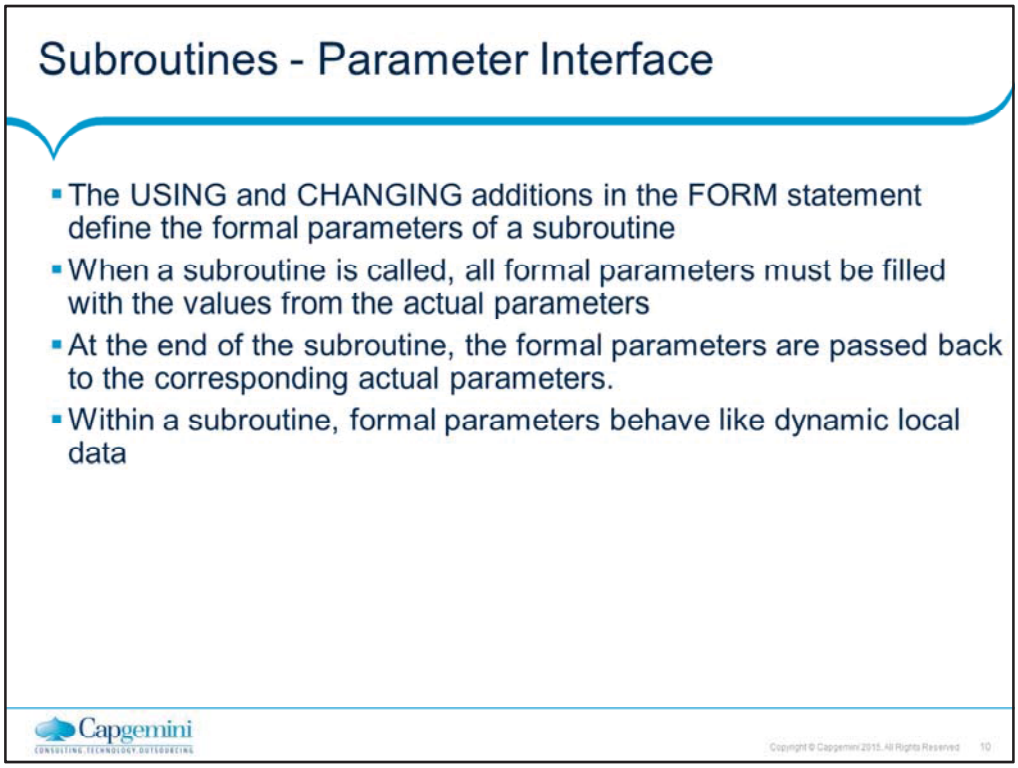
Global and Local Variables

- Global Variable
  - A global variable is one that is defined outside of a subroutine by using tables or data statement.
  - It can be accessed from any point in the program, be it inside an event or inside a subroutine.
- Local Variable
  - A local variable is a variable that is defined inside a subroutine using the local, data, or statics statement.
  - It is said to be local to the subroutine.
  - Variables defined by using local are accessible from outside the subroutine; variables defined by using data or statics are not.

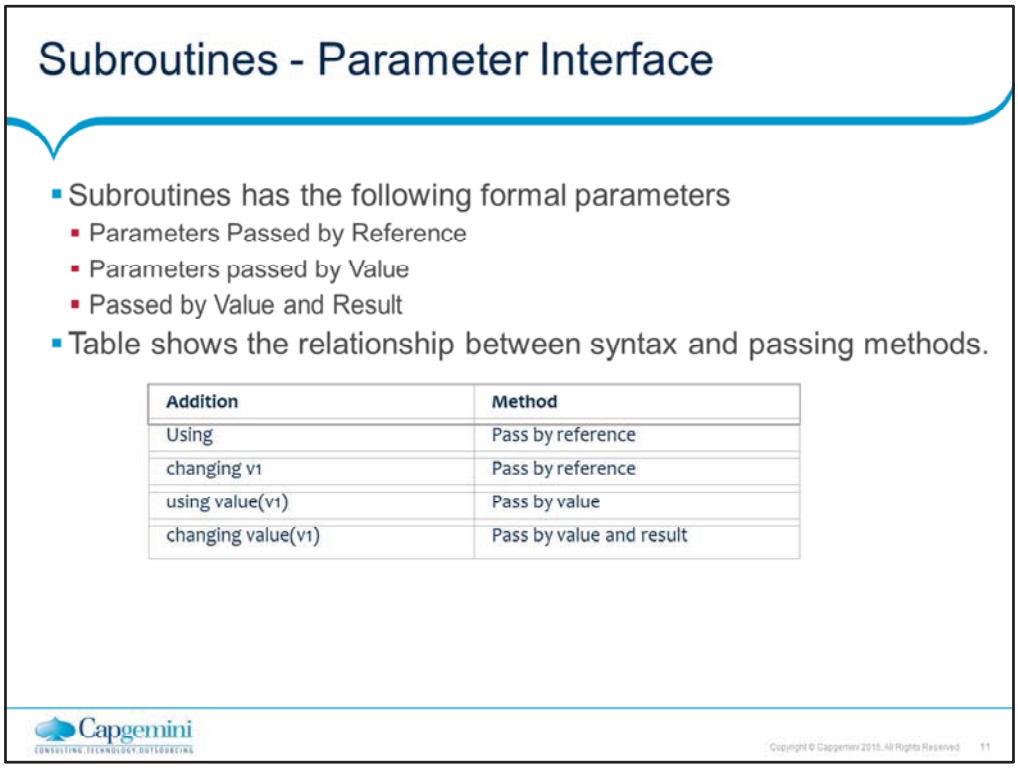
Demo: Create Subroutine with Local and Static Variable

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 9



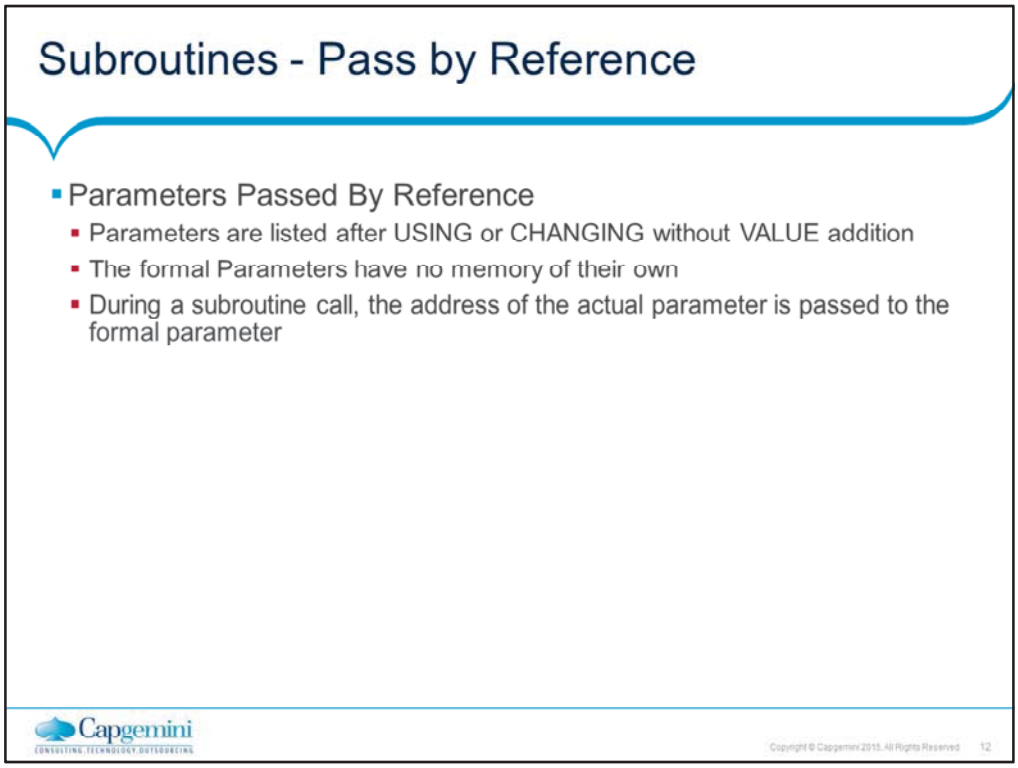
From within your subroutine, you can access global data of the report and of the system.
This data consists of globally defined fields, field strings, internal database fields, table fields, dynpro fields and system fields.
In addition to defining variables by using tables, local, data, and statics, variables can also be defined on the form statement itself.
These are known as parameters.
Parameter names that appear on the form statement are called formal parameters.
The term is easy to remember because “formal” starts with “form”.
Parameter names that appear on the perform statement are called actual parameters.
For example, in the statement perform s1 using f1 changing f2 f3, the parameters f1, f2, and f3 are called actual parameters.



Although syntax on form and perform can differ, for the sake of program clarity they should be same. When calling a subroutine (perform), you can also distinguish between using and changing parameter. However, the distinction is used for documentation purposes only. What counts is the declaration when you define the subroutine (FORM).

Remember to keep the following thing in mind:

- perform and form statement must contain the same number of parameters
- The syntax on the perform and form statements can differ
- The syntax on the form statement alone determines the method by which a parameter is passed
- The value() addition can be used on the perform statement
- using must come before changing
- The addition using can only occur once in a statement. The same rule applies to changing



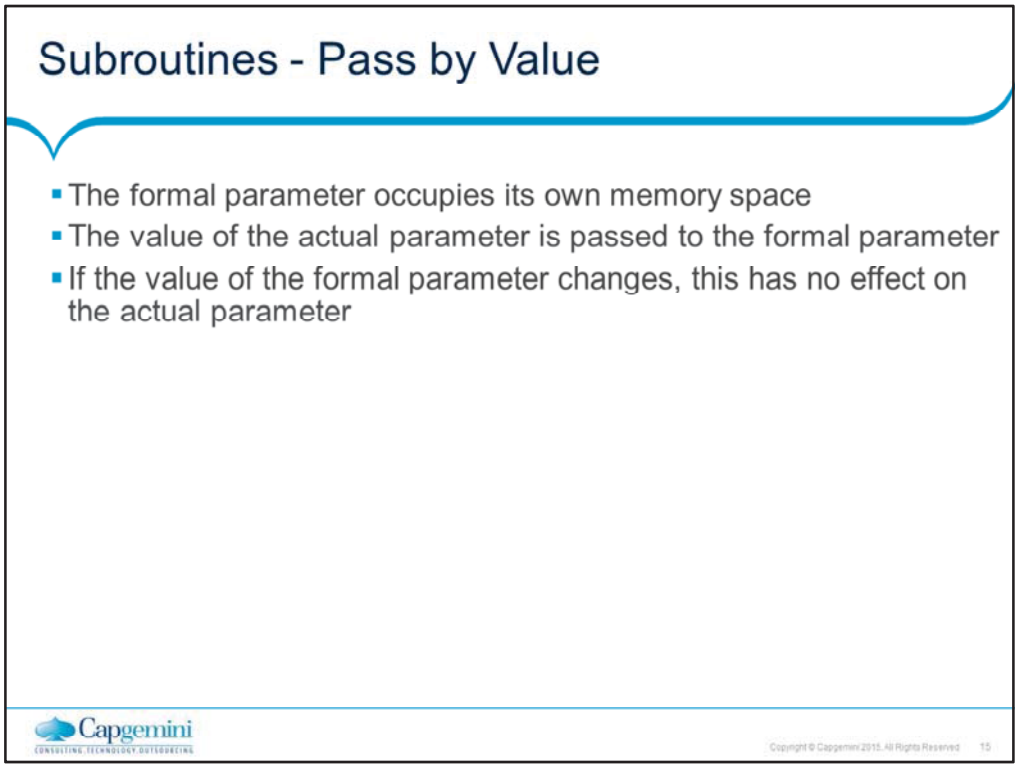
When you pass a parameter by reference, new memory is not allocated for the value.
Instead, a pointer to the original memory location is passed.
All references to the parameter are references to the original memory location.
Changes to the variable within the subroutine, update the original memory location immediately

Using .. Changing

- The additions using f1 and changing f1 both pass f1 by reference-they are identical in function.
- The reason they both exist is that-used properly-they can document whether the subroutine will change a parameter or not.
- Code changing with parameters, the subroutine changes.
- You code using with the parameters that are not changed by the subroutine

Demo: Subroutine – Pass By Reference

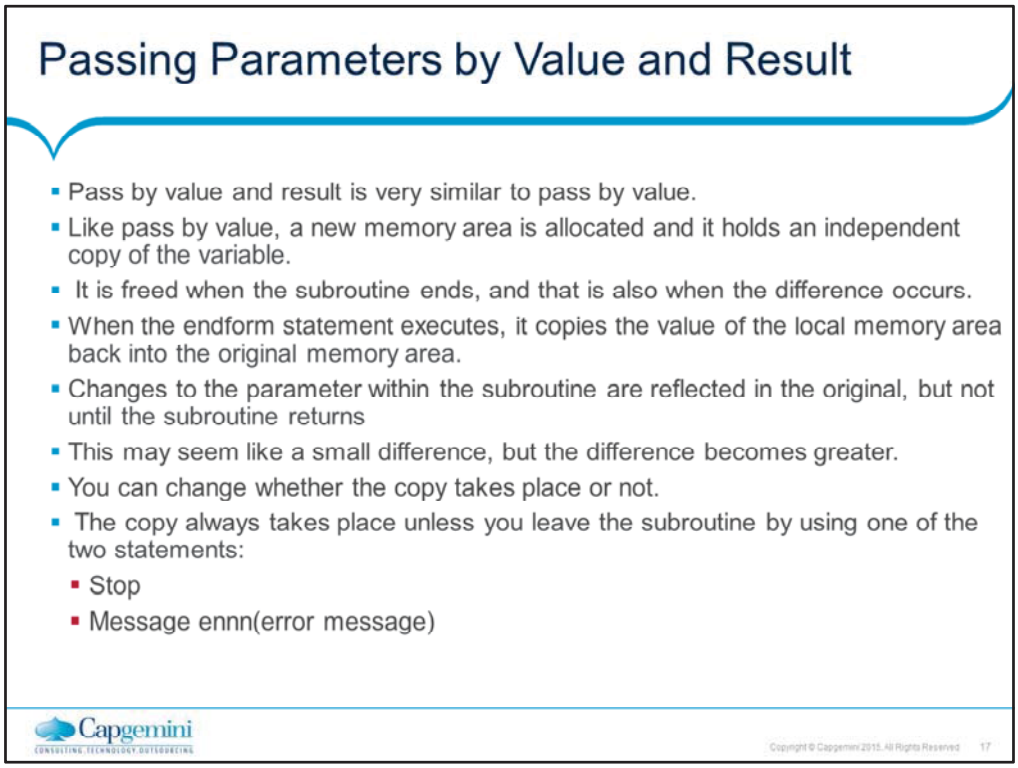
Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING



When you pass a parameter by value, new memory is allocated for the value.
This memory is allocated when the subroutine is allocated when the subroutine is called and is freed when the subroutine returns.
Therefore, references to the parameter are thus references to a unique memory area that is known only within the subroutine; the original memory location is separate.
The original is unchanged if you change the value of the parameter

Demo: Subroutine – Pass By Value

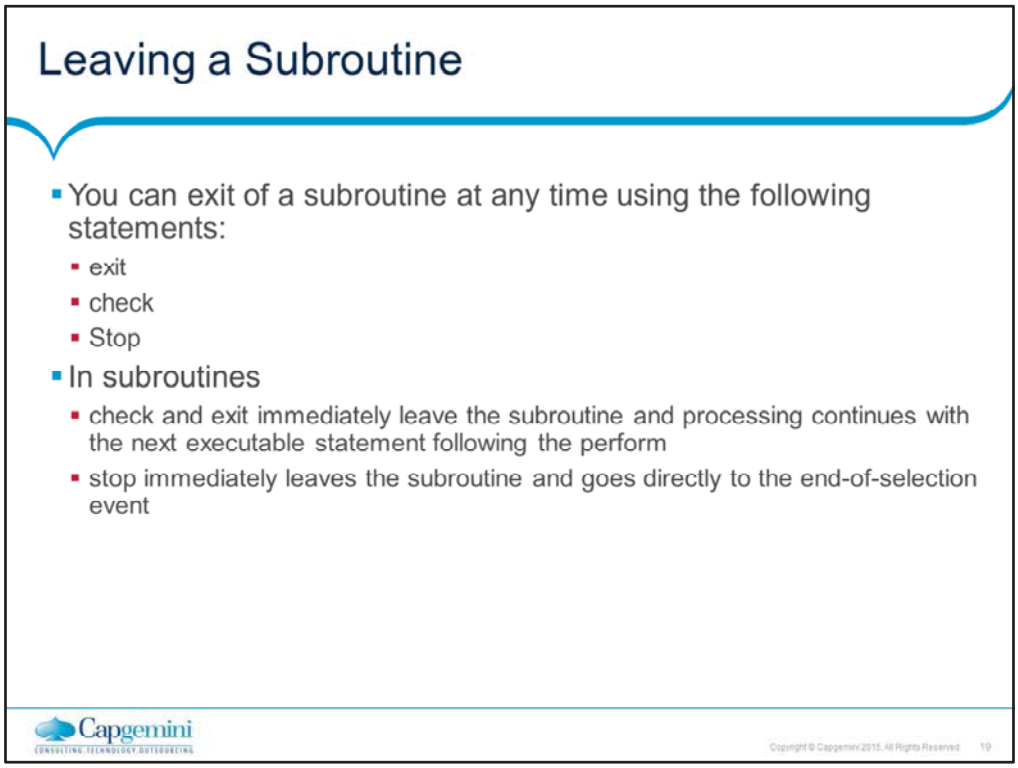
Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING



The stop statement terminates the subroutine and goes directly to the end-of-selection event.
If p1 was passed by value and result, changes to p1 are discarded before end-of-selection is triggered.
In a sense, stop behaves like a mini-rollback for value and result parameters.
When it is used inside of a subroutine, the stop statement is usually preceded by a test for an abnormal condition within a program.
If the abnormal condition arises, stop is executed.
It discards the changes to value and result variables, and triggers end-of-selection, where cleanup procedures are then executed.

Demo: Subroutine – Pass By Value and Result

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING



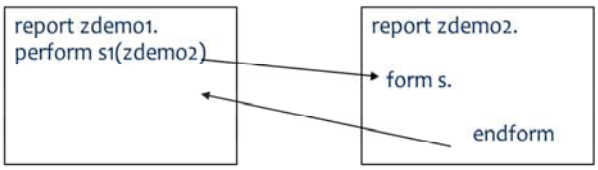
check, exit, and stop do not set the value of sy-subrc. If you want to set it, assign a numeric value to it before leaving.

Demo: Leaving subroutine using Exit, Check and Continue

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

Defining and Calling External Subroutines

- An external subroutine is one that resides in a different program than the perform statement that calls it. Figure A illustrates an external subroutine.



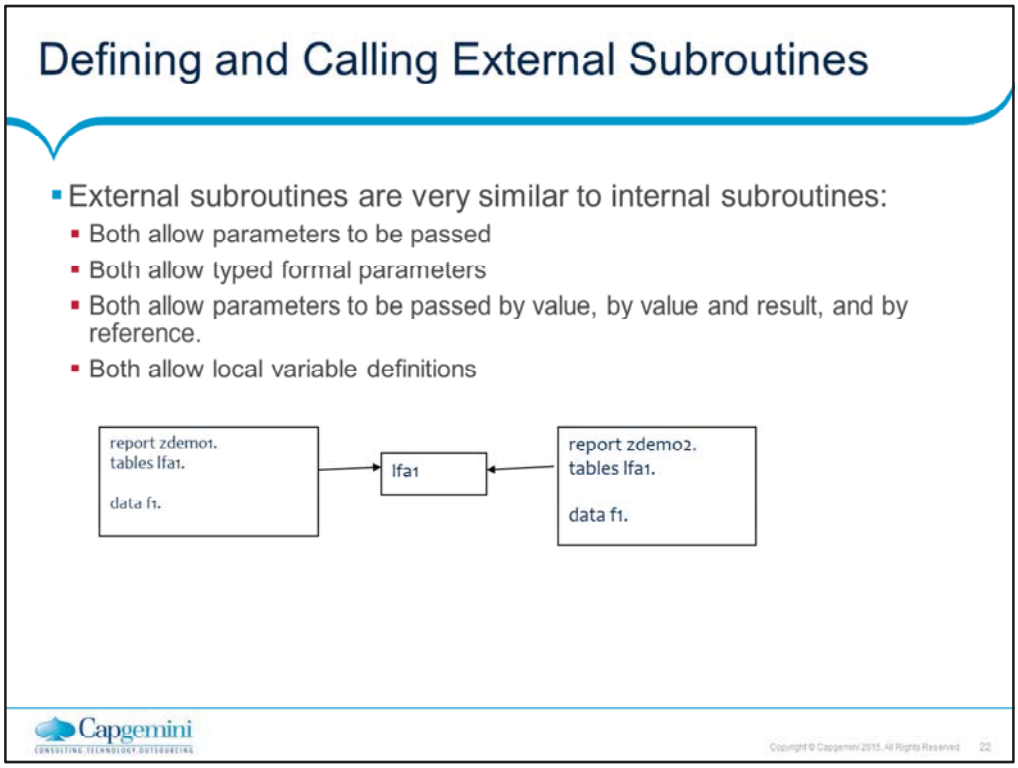
When a perform calls an external subroutine

- The external program containing the subroutine is loaded
- The entire external program is checked for syntax
- Control transfers to the form in the external program
- The statements within the external subroutine are executed
- The endform transfers control back to the statement following the perform

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 21

Ways of passing data
You can pass the field (passing by reference)
In this case, the formal parameter points to the field.
Within the routine, you work with the field itself.
If you change the formal parameter, they automatically change the field.
Passing data by reference is the default.
If you pass data with USING and make no further specifies, the system passes the data by reference.
You can pass the field content (passing by value)
If you pass the content of a field, the subroutine works with a copy of the field.
Changes to the formal parameter have no effect on the current parameter.



The following are the differences between external and internal subroutines

- A global variable defined by using data is known only within the program that defines it . For example in the above example the data f1 statement appears in both programs. This defines two memory areas named f1 in zdemo1 and zdemo2 are distinct i.e. different memory areas.
- A global variable that has the same name in both programs and is defined using the tables statement in both programs and is common to both programs. A change to this variable in one program affects the other

Defining and Calling External Subroutines

Demo: Call subroutine defined in other program

The LOCAL statement

- If you use the LOCAL statement when defining a subroutine, the system stores the current value of the field, field string, or field symbol you specify as soon as it enters the subroutine.
- When you leave the routine, it restores the original value.
- You can use LOCAL only after a FORM statement

Demo: Use LOCAL statement in subroutine

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

Local data

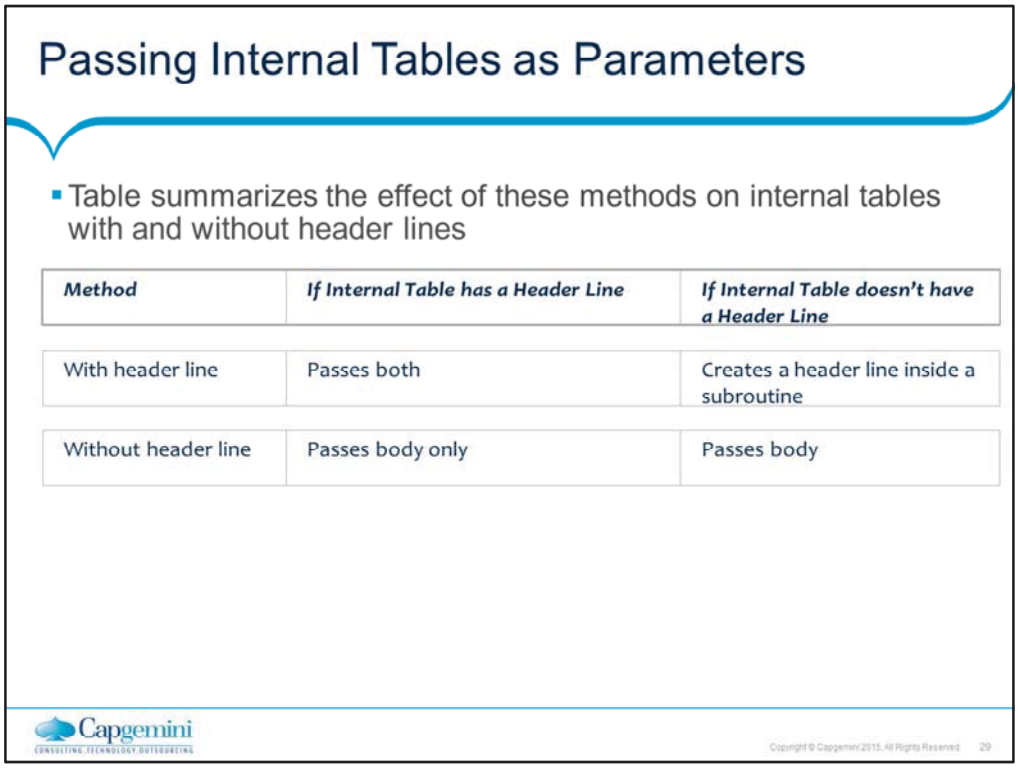
- Fields, field strings and field symbols you declare within a subroutine are initialized every time you enter the routine.
- However, their names have to be unique within report.
- You can use the PERFORM statement to call subroutine which are defined in other reports.
- After the subroutine name you specify in parenthesis the name of the report in which the subroutine is defined.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 27

Passing Internal Tables as Parameters to Subroutines

- You can use one of the two methods to pass an internal table to a subroutine:
  - Pass with header line
  - Pass body only



- If the Internal table has a header line, the first method in the above table passes both the header line and body to the subroutines. The rest of the method passes the body only
- If the Internal table does not has a header line, the first method in the above table passes the body and creates a header line within subroutines. The rest pass the body only

- If the internal table has a header line, method 1 passes both the header line and the body to the subroutine.
- Method 2 passes only the body to the subroutine.
- If the internal table doesn't have a header line, you can also use both methods.
- However, method 1 will behave a little differently-it will automatically create a header line for the internal table within the subroutine

Passing an Internal Table with Header Line

- If an internal table has a header line and you want to pass both the header line and the body to a subroutine, use the syntax
  - form s1 tables it.
- This passes both the header line and body, and they are both passed by reference.
- Therefore, changes made to either the header line or body of the internal table within the subroutine is immediately reflected in the original.

Demo: Passing body of an Internal Table with Header Line

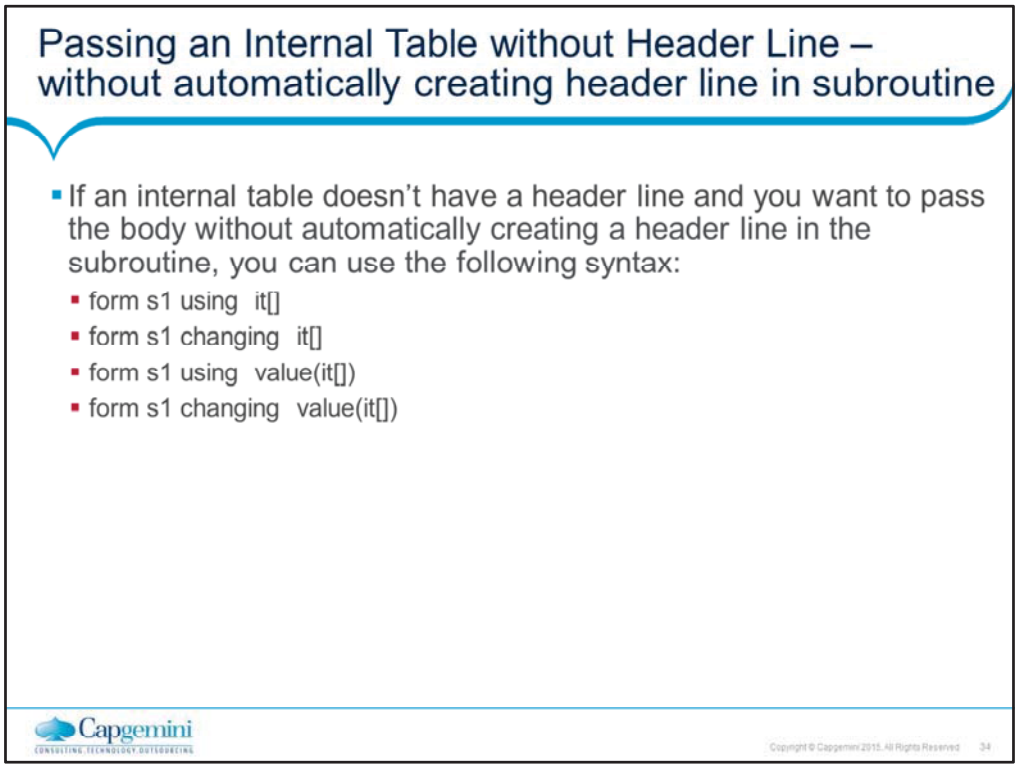
Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

Passing an Internal Table without Header Line and automatically creating a header line in Subroutine

- If the internal table doesn't have a header line and you want to pass the body and create a header line in the subroutine, you can also use the syntax
  - *form s1 tables it.*
- This passes the body by reference, and creates a header line locally in the subroutine.
- Changes made to the body of the internal table within the subroutine are immediately reflected in the original.

Demo: Pass an Internal table without Header Line to a Subroutine

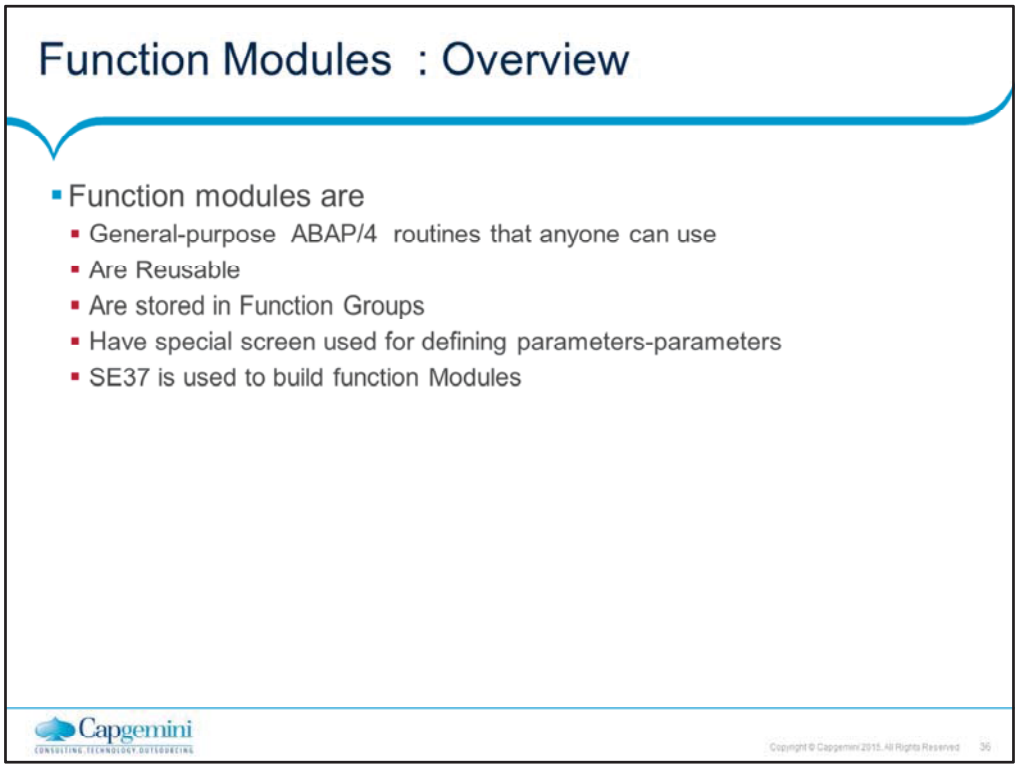
Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING



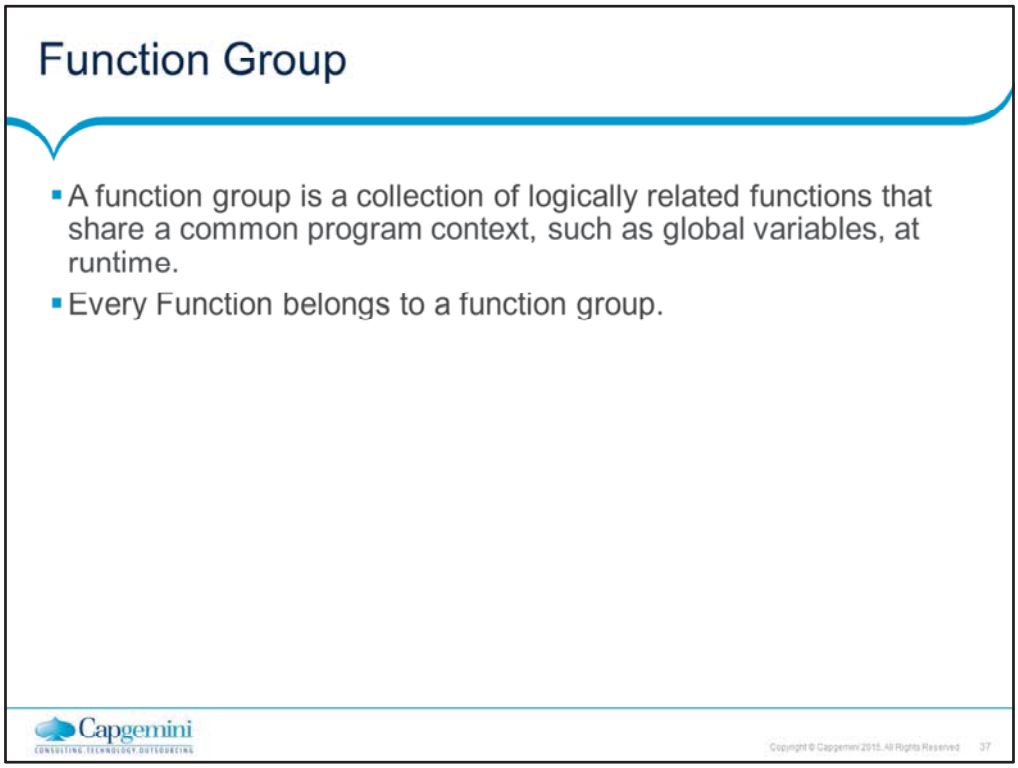
You can pass the body by reference, by value, or by value and result.
If you pass it by reference, changes made to the body of the internal table the subroutine are immediately reflected in the original.
If you pass it by value, a local copy of it is created- changes are discarded at the end of the subroutine when the local memory for it is freed.
If you pass it by value and result, changes are copied back into the original when endform is executed.
A stop statement within the subroutine will discard all changes to it and transfer control directly to end-of-selection.

Demo: Pass an Internal table without Header Line to a Subroutine

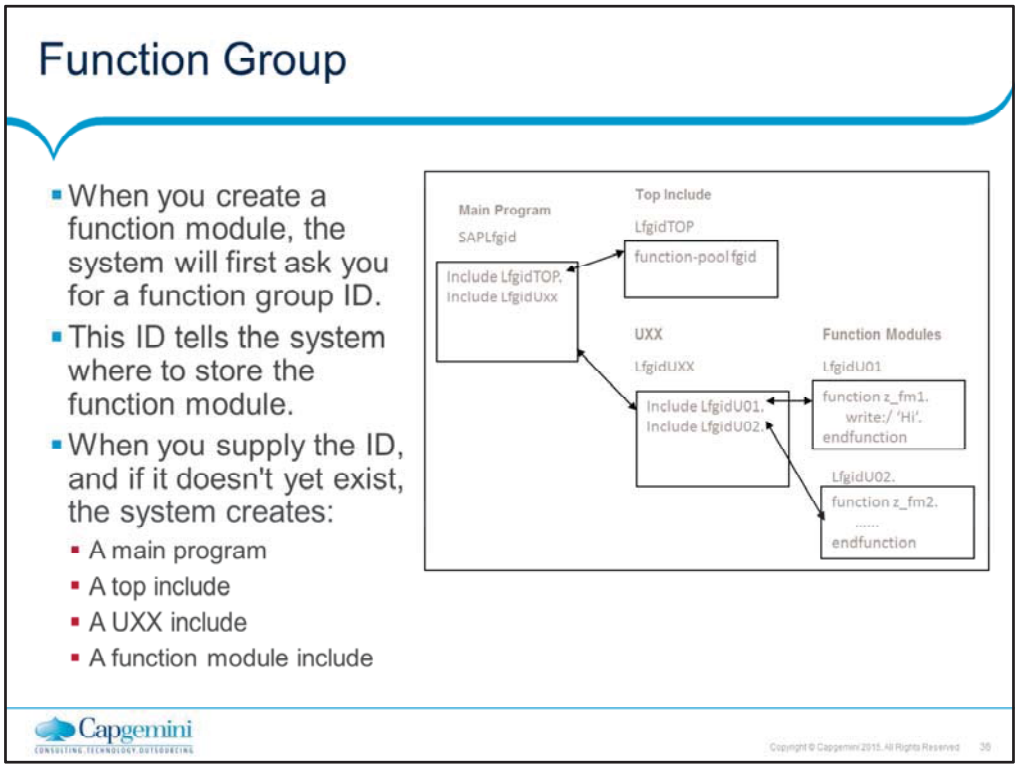
Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING



Function modules are general-purpose ABAP/4 routines that anyone can use.
The benefit of function modules lies in their reusability.
Hence you can reduce redundant coding and increase programming efficiency.
The ABAP/4 Development Workbench comes with a large number of standard function modules.
The Function Library stores these standard functions.
Function modules are procedures that are defined in function groups (special ABAP programs with type F) and can be called from any ABAP program.
Function modules allow you to encapsulate and reuse global functions in the R/3 System.
They are stored in a central library.
The R/3 System contains a wide range of predefined function modules that you can call from any ABAP program



As a norm all related functions should be created under the same group.



The name of the main program will be saplfgid, where fgid is your four-character function group ID.
The system automatically places two include statements into it:

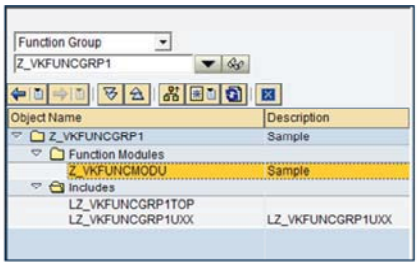
include lfgidtop.
include lfgiduxx.

The first include program-lfgidtop-is known as the top include.

Global data definitions can be placed within it
These are data definitions that are global to all function modules within the group

Function Group

- The name of the main program will be saplfgid, where fgid is your four-character function group ID.
- The system automatically places two include statements into it:
  - include lfgidtop.
  - include lfgiduxx.



The first include program-lfgidtop-is known as the top include.

Global data definitions can be placed within it

These are data definitions that are global to all function modules within the group

The second include program-lfgiduxx-is known as the UXX.

You are not allowed to modify the UXX.

The system will automatically place an include statement in the UXX for each function module you create in this function group.

For the first function module, the statement include lfgidu01 will be inserted into the UXX.

When you create a second function module in this group, the system will add a second statement: include lfgidu02

Each time you add a new function module to the group, the system will add a new include statement to the UXX.

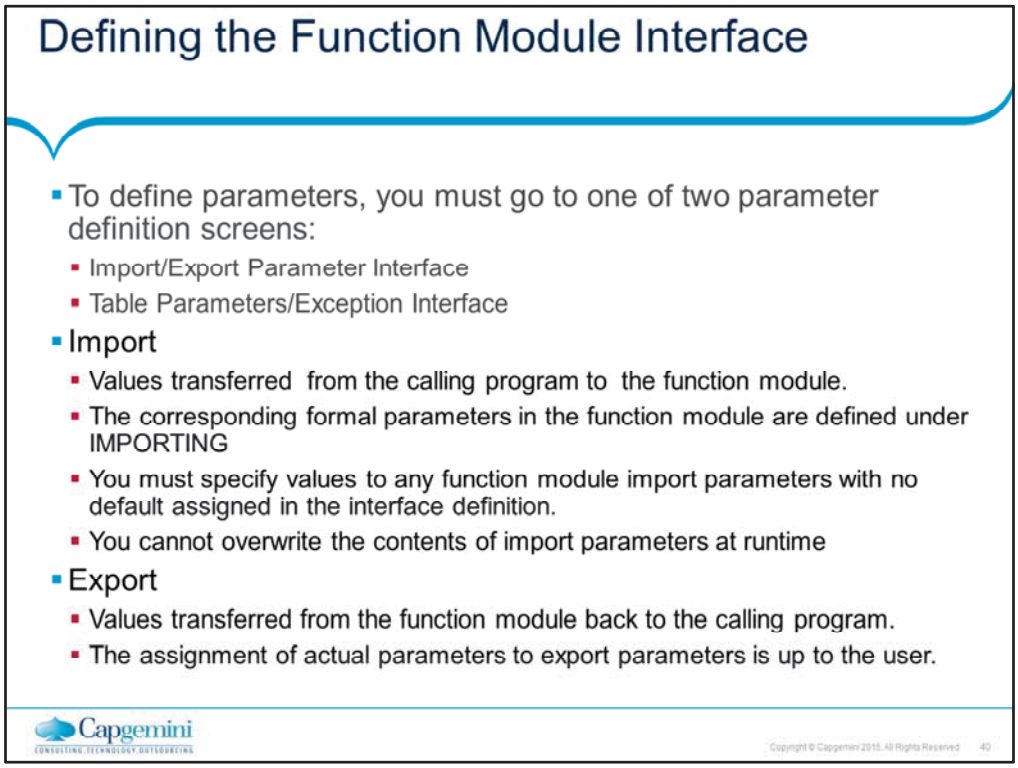
The number of the include will be 01 for the first function module created within the group, 02 for the second, 03 for the third, and so on.

Within each include mentioned in the UXX is the source code for a function module.

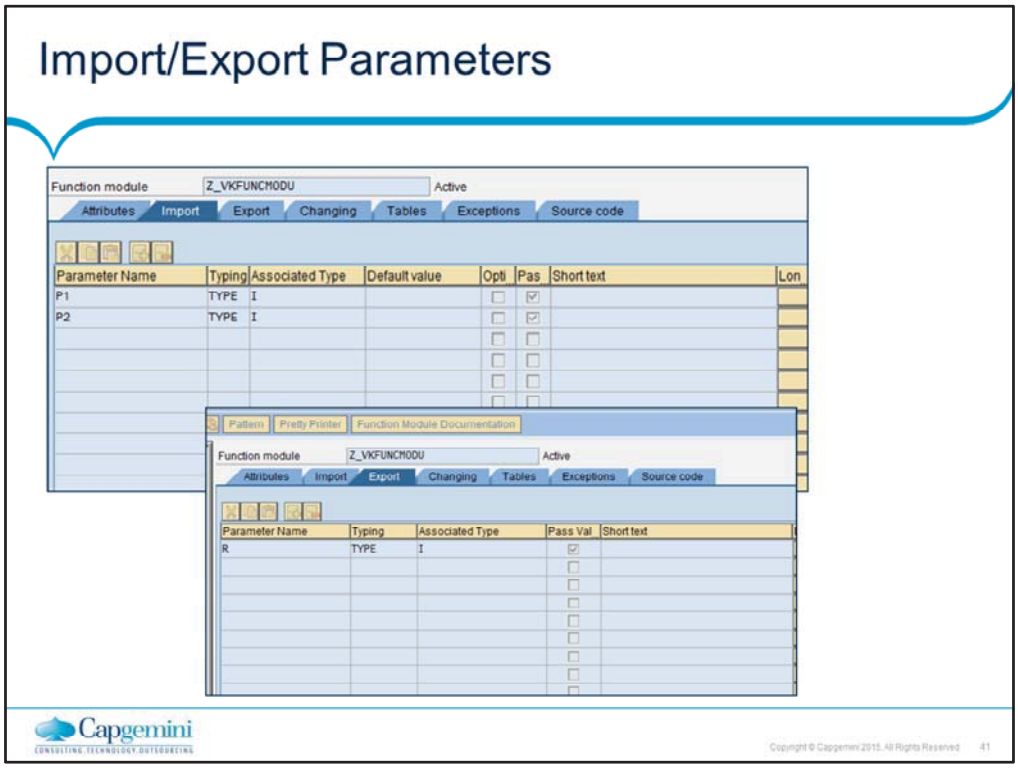
A function module must be activated before it can be called

When the function module is first created, the include statement for it within the UXX is commented out. Activating the function module causes the system to remove the comment from the include statement.

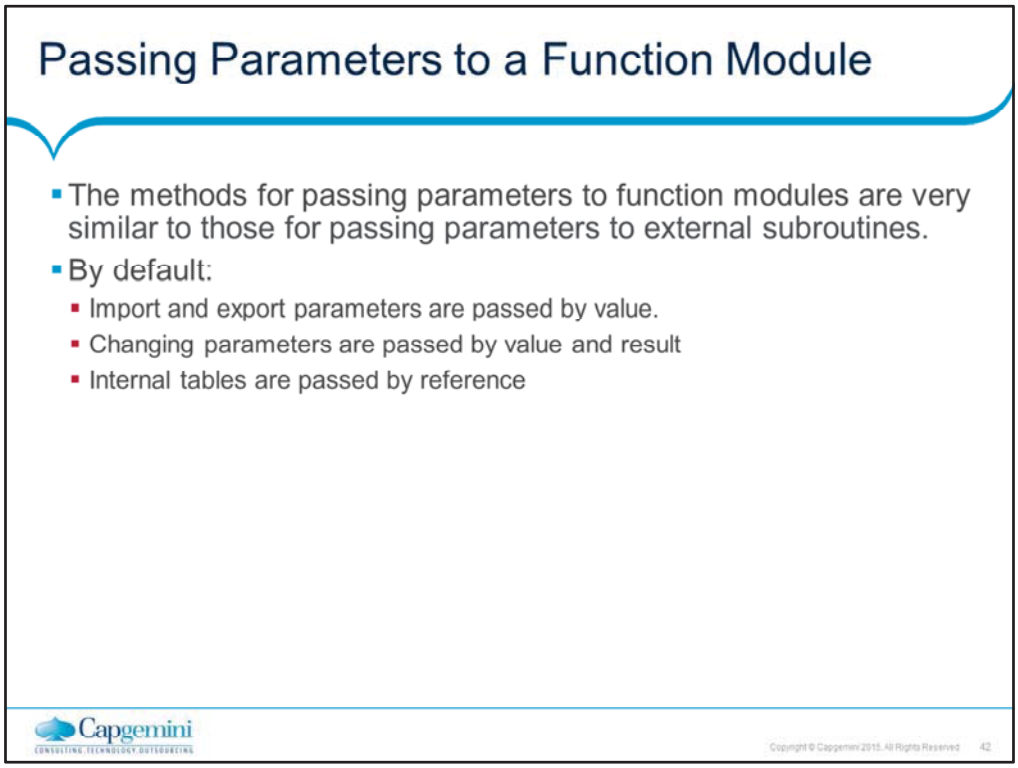
The function module is then available to be called from other programs



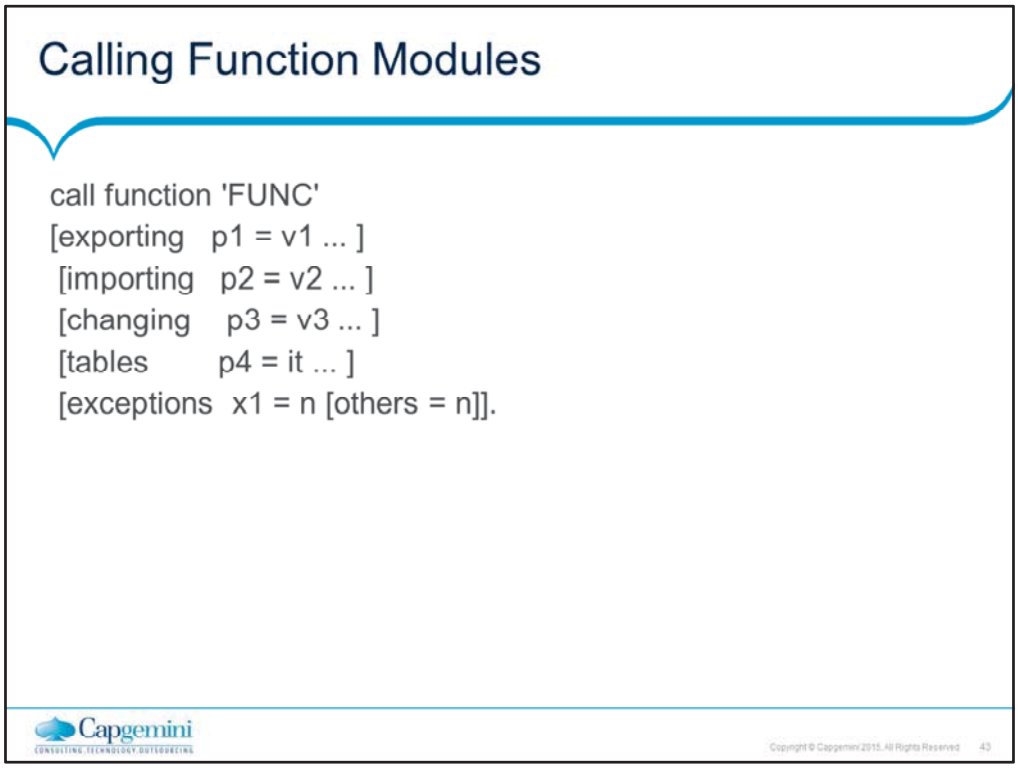
Data definitions within function modules are similar to those of subroutines.
Within a function module, data statement is used to define local variables that are reinitialized each time the function module is called.
Parameters are defined within the function module interface to create local definitions of variables that are passed into the function module and returned from it.
To pass parameters to a function module, you must define a function module interface.
The function module interface is the description of the parameters that are passed to and received from the function module.
It is also simply known as the interface.



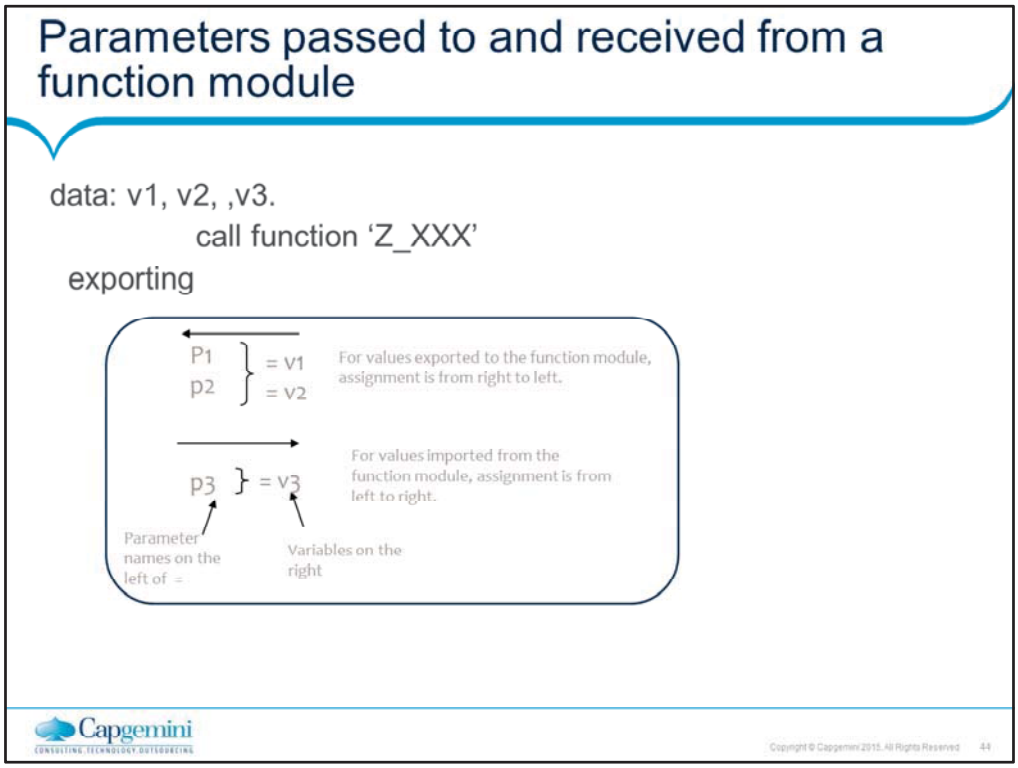
Import parameters are variables or field strings that contain values passed into the function module from the calling program. These values originate outside of the function module and they are imported into it.
Export parameters are variables or field strings that contain values returned from the function module. These values originate within the function module and they are exported out of it.
Changing parameters are variables or field strings that contain values that are passed into the function module, changed by the code within the function module, and then returned. These values originate outside the function module. They are passed into it, changed, and passed back.
Table parameters are internal tables that are passed to the function module, changed within it, and returned. The internal tables must be defined in the calling program.
An exception is a name for an error that occurs within a function module.
Exc



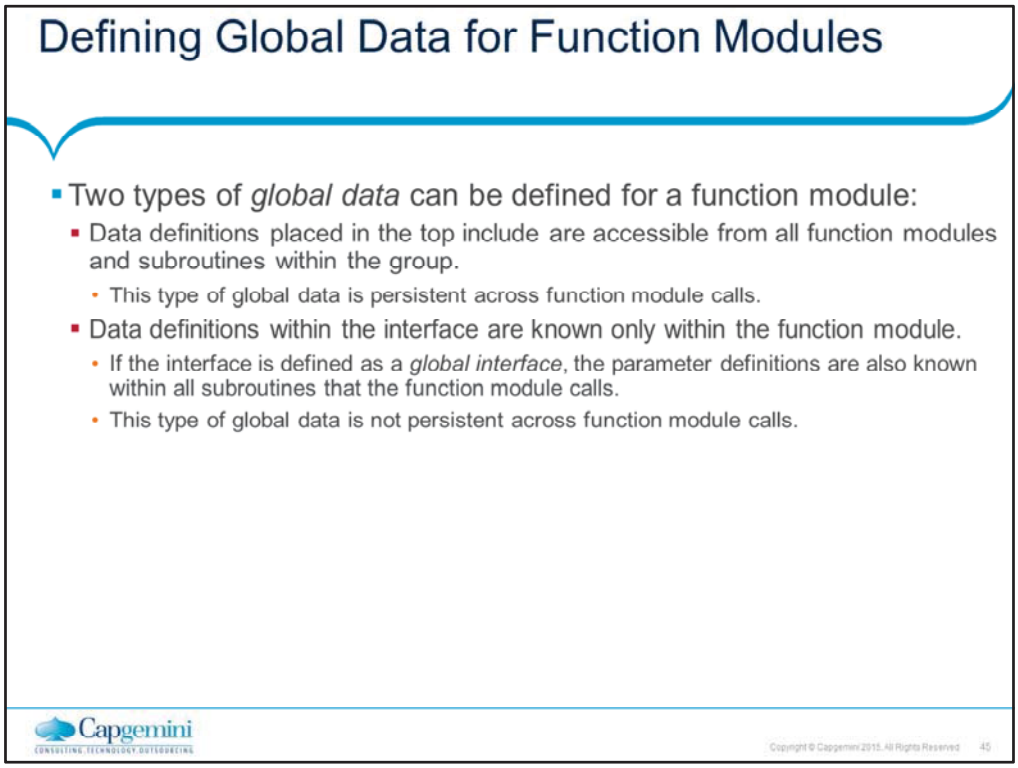
You can cause import, export, and changing parameters to be passed by value by placing a tickmark in the Pass Value box on the Import/Export Parameters screen. If the endfunction statement is not executed (for example, if you leave the function module abnormally via the raise statement), the value is not copied and the caller's variable remains unchanged.
Parameters can also be made optional and given default values. To make an import parameter optional, place a tick mark in the Optional column. If you make an import parameter optional, you do not have to name it on the call function statement when you call the function module. Export parameters are always optional i.e. you are not required to code any export parameters on the call function statement. For example, if function module z_xyz returns three export parameters, e1, e2, and e3, and you only want e2, then only code e2 on the call function statement. Simply omit the rest.
To give an import parameter a default value, enter a value in the default column, making sure to enclose character values within single quotes. sy variable names are also allowed.
All parameters that you define in the interface also appear at the top of the source code within special comments. Each special comment line begins with the characters *". Each time you change the interface the system automatically updates these special comments to reflect your changes.
It is recommended not to change or delete these comments; they are generated by the system. If you change them, the function module might not work.



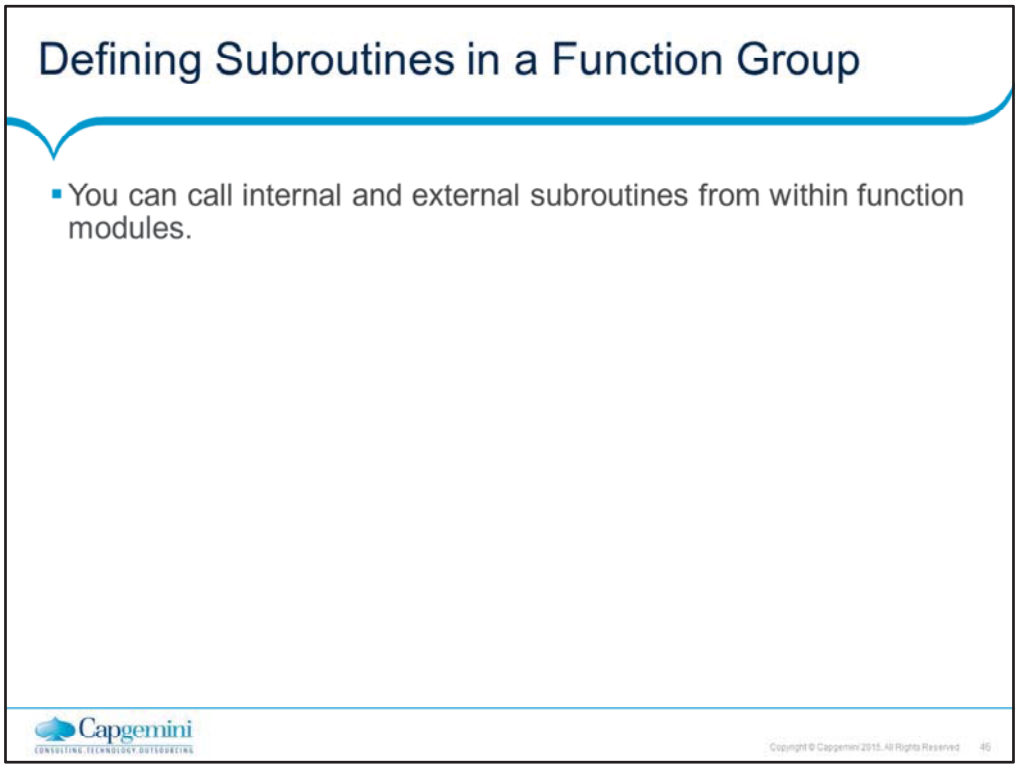
FUNC is the function module name and must be in uppercase
p1 through p4 are parameter names defined in the function module interface
v1 through v3 are variable or field string names defined within the calling program.
It is an internal table defined within the calling program.
n is any integer literal; n cannot be a variable
x1 is an exception name raised within the function module



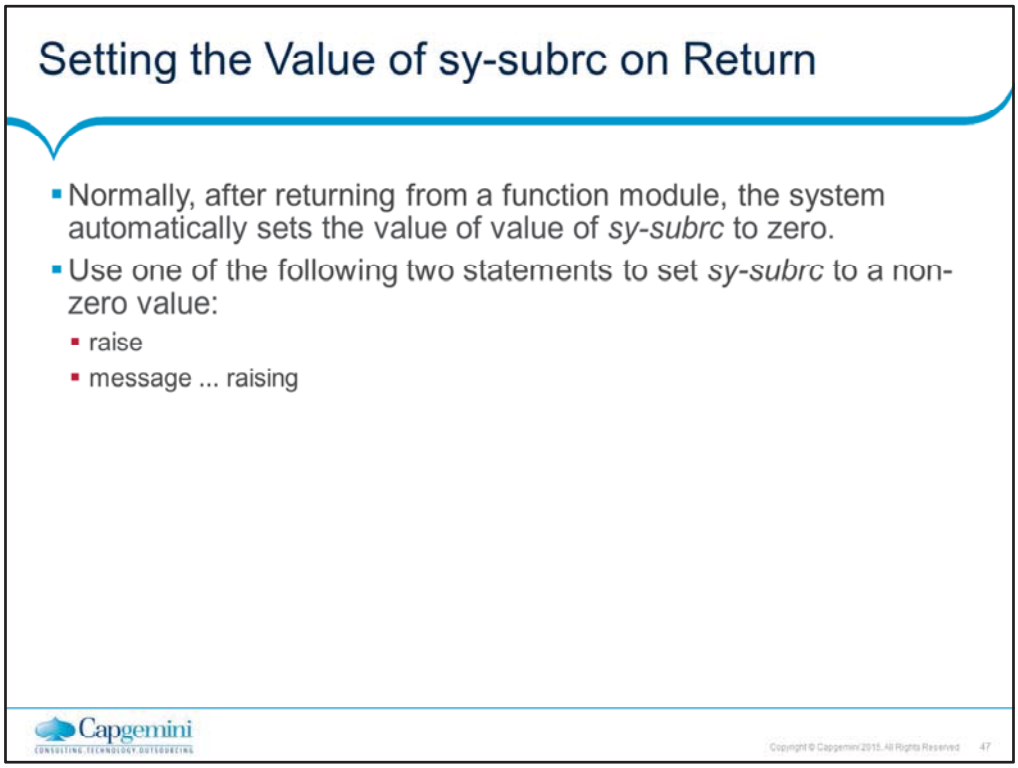
On the call function statement, exports and imports are from the point of view of the program.
Values that are exported by call function statement are imported by the function module.
Values exported from the function module are imported by the call function statement.
On the left of the assignment operator (=) is a list of the parameter names defined in the function module's interface.
On the right are variables defined within the calling program.
Assignments after exporting proceed from right-to-left.
Assignments after importing proceed from left-to-right.
Assignments after changing and tables are bi-directional.
Before the call, the assignment is from right-to-left.
On return, the return value is assigned from left-to-right



The global data defined within the top include is accessible from all function modules within the group. It is analogous to the global data definitions at the top of an ABAP/4 program.
The parameters defined within a global interface are accessible from all subroutines called from the function module. It is analogous to variables defined within a subroutine using the locals statement.
The parameters defined within a local interface are inaccessible from subroutines called from the function module. It is analogous to variables defined within a subroutine using the data statement. By default, interfaces are local.



Calls to external subroutines are the same for function modules as they are for reports.
Internal subroutines should be defined within a special include: the F01.
The F01 is included into the function group by editing the main program and inserting the statement include LfgidF01.
Then, double-clicking on the name of the include will create it automatically.
Within it, you can put subroutines accessible by all function modules within the group



Use the raise statement to exit the function module and set the value of sy-subrc on return.
Syntax:

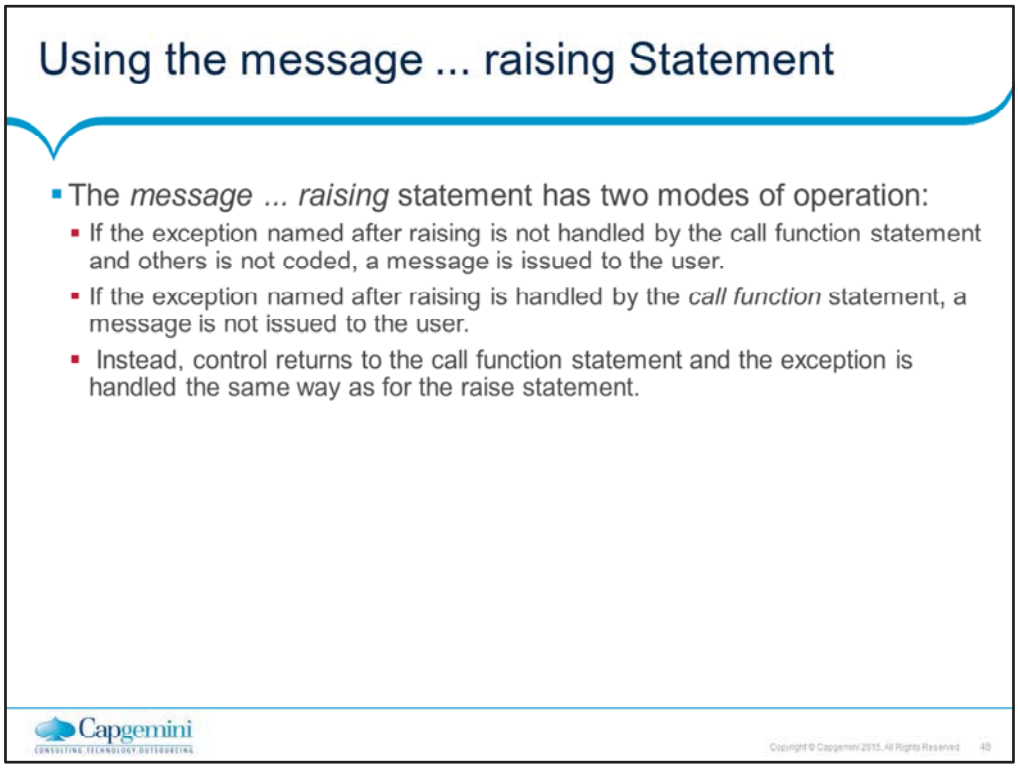
raise xname.

where:

xname is the name of the exception to be raised

The following points apply:

xname can be any name that you make up
It does not have to be previously defined anywhere
It can be up to 30 characters in length
All characters are allowed except " ' . , and :.
Do not enclose xname within quotes
xname cannot be a variable



Syntax:

message tnnn(cc) [with v1 v2 ...] raising xname
where:

t is the message type (e, w, i, s, a, or x).
nnn is the message number.
(cc) is the message class.
v1 and v2 are values to be inserted into the message text.
xname is the name of the exception to be raised.

The following points apply:

xname is an exception name as described for the raise statement.
If the message class is not specified here, it must be specified on the function-pool statement in the top include for the function group via the message-id addition.
The following sy variables are set: sy-msgid, sy-msgty, sy-msgno, and sy-msgv1 through sy-msgv4.
These can be examined within the calling program (after return).

Summary

- In this lesson, you have learnt:
  - Function Modules
  - Subroutines

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

Review Question

- Question 1. The _____ statement terminates the subroutine and goes directly to the end-of-selection event.
- Question 2: Parameter names that appear on the form statement are called _____ parameters.

Review Question

- Question 1. Function modules are organized into _____.
- Question 2: The _____ statement is used to exit the function module and set the value of sy-subrc on return.
- Question 3: *Tables* parameters are always passed by _____.

ABAP Part II

Lesson 03: 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

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

Introduction to Module Pool Programming

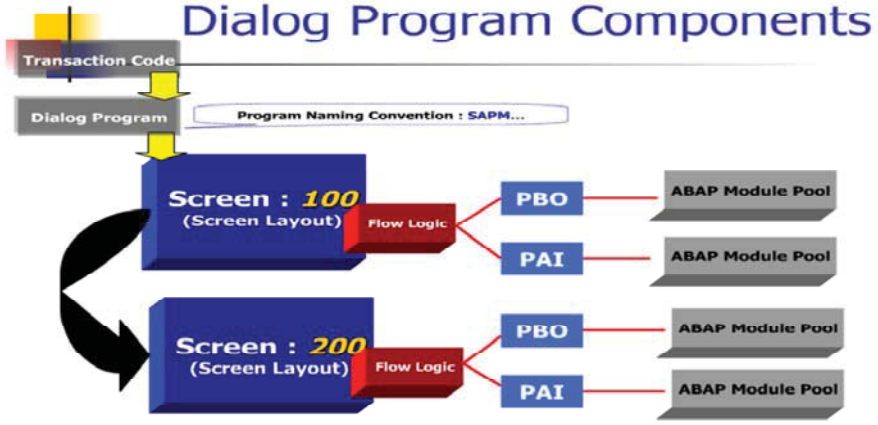
- Module Pool Program (also called online programs, dialog programs, or transactions) do not produces lists. These programs are a collection of screens, their Flow Logic, and the code within the main ABAP program.
- Dialog Programming: Dialog-driven programs, or any program started using a transaction 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.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 3

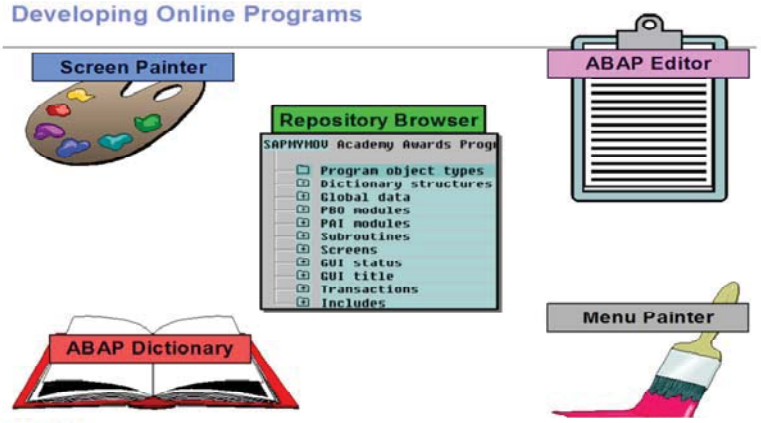
Introduction to Module Pool Programming

Dialog driven program consist of the following basic components.



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

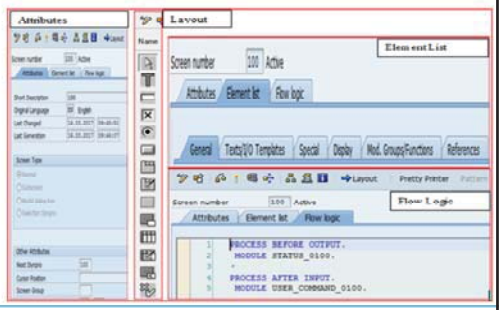
- The Screen Painter is a ABAP Workbench tool that allows you to create screens for your transactions.
- Also called Dynpro which is Dynamic Programing.
- It will allow you to create the screen itself, with fields and other graphical elements, and to write the flow logic behind the screen.
- You define screen in the screen painter.
- Screens are the most general type of user dialog that you can use in ABAP programs.

Screen Painter

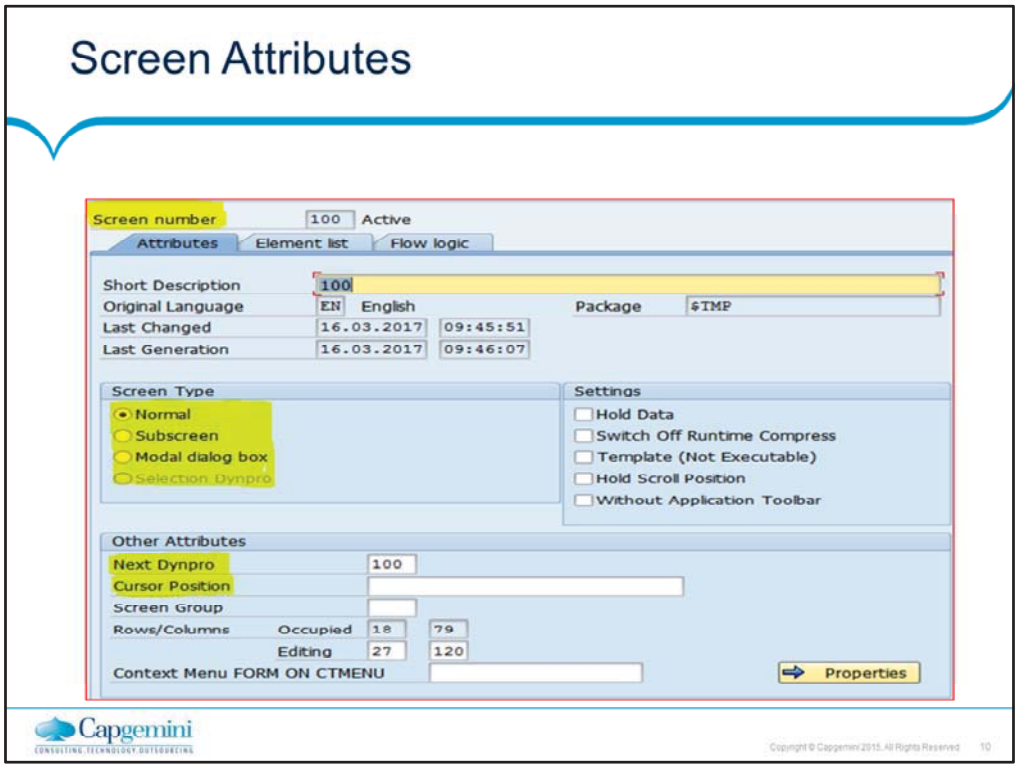
- Consists of input/output and the screen flow logic
- Screens can be defined for
  - Executable Program
  - Module Pool
  - Function Group
- Data passed between screen fields and data objects in ABAP program.
- Data is copied between identically-named fields.

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.



Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING



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.

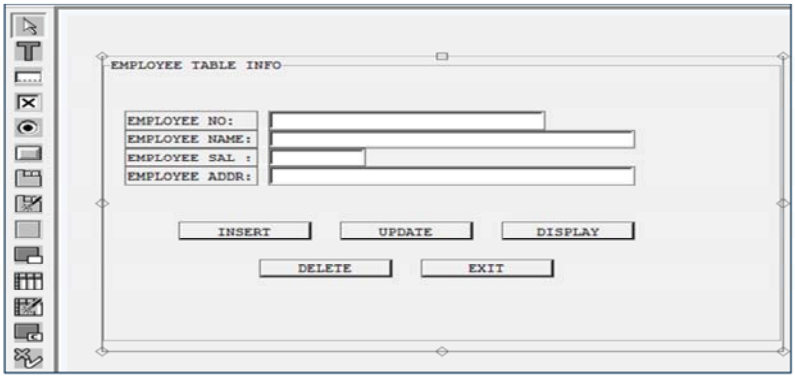
6.Screen Group: Four-character ID, placed in the system field SY-DYNGR while the screen is being processed.

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.

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.

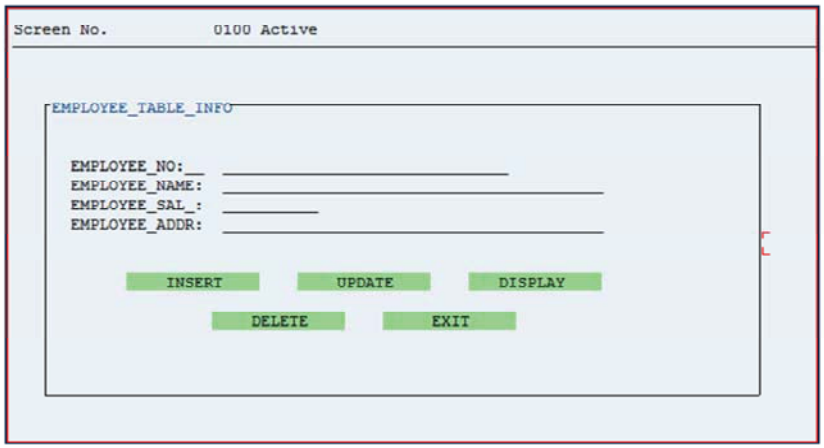


Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 11

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.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

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).

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 16

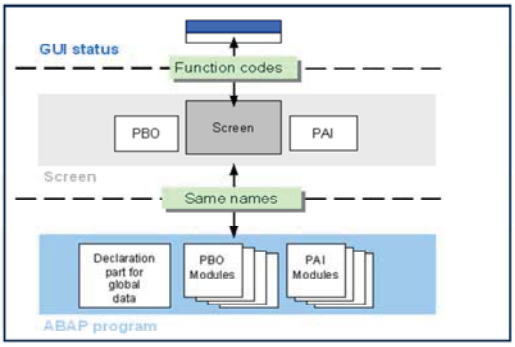
Flow Logic

- 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.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

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
```

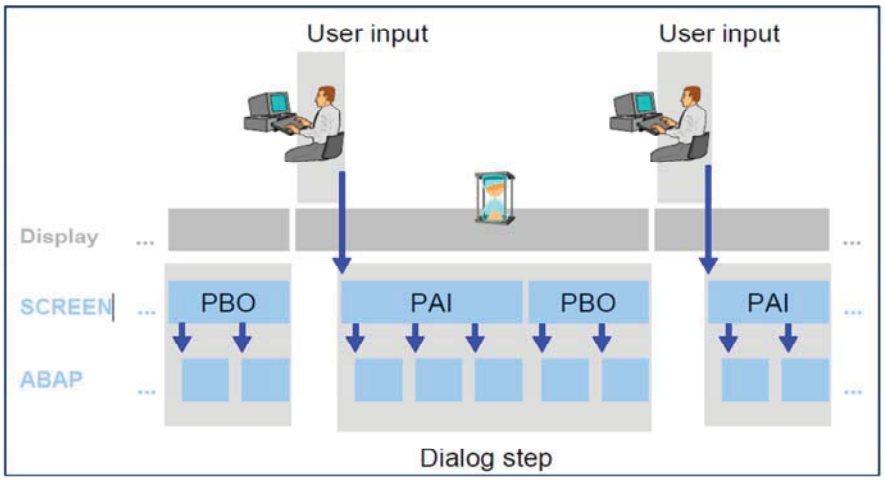
PAI Statement:

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

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

Flow Logic

- Screen Sequence is as follows:



Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

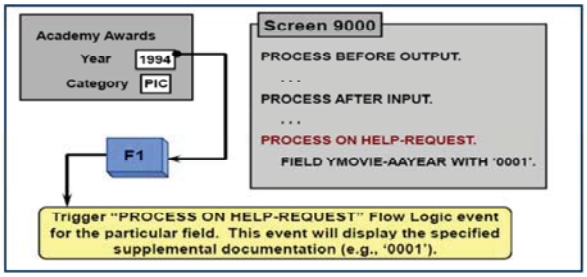
 21

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>.

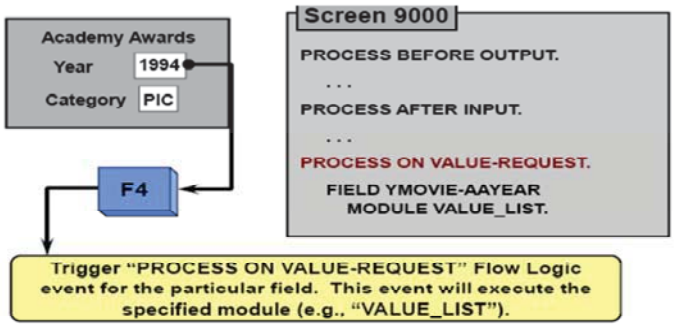
on Help Request (POH)



Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

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>.**



Trigger "PROCESS ON VALUE-REQUEST" Flow Logic event for the particular field. This event will execute the specified module (e.g., "VALUE_LIST").

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.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 25

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>.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 26

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.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 27

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.

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.
```

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING.

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.

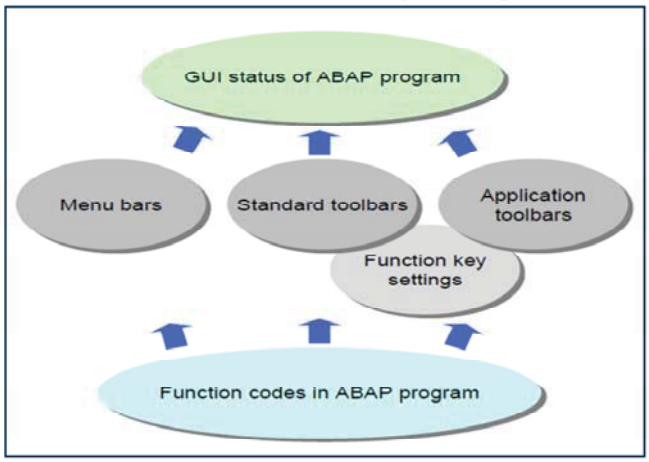
Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

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.



Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

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

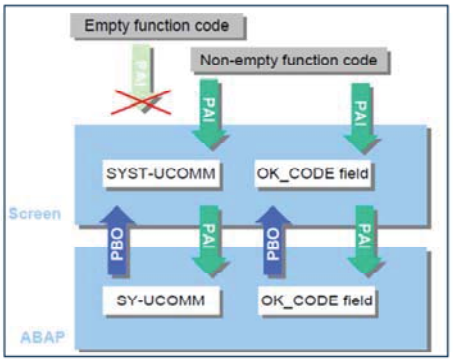
Example:

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

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

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.



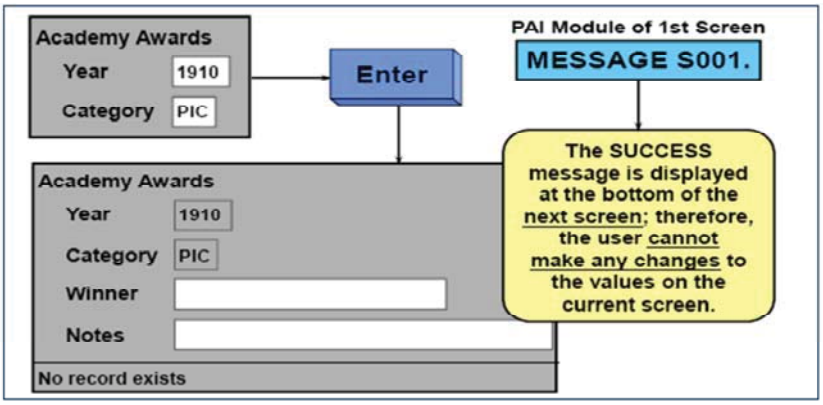
Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

Message Types

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

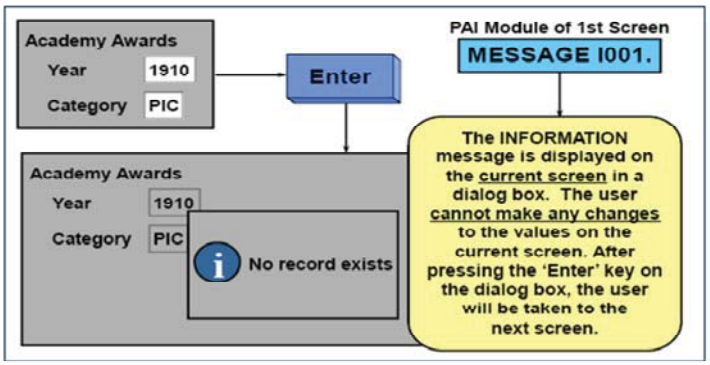
Message Types

- 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.



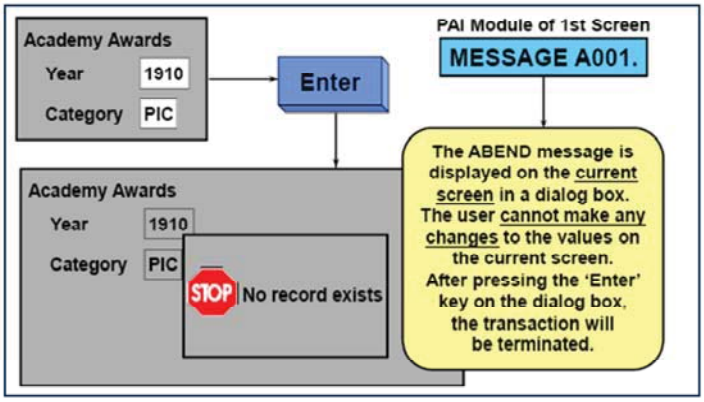
Message Types

- 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

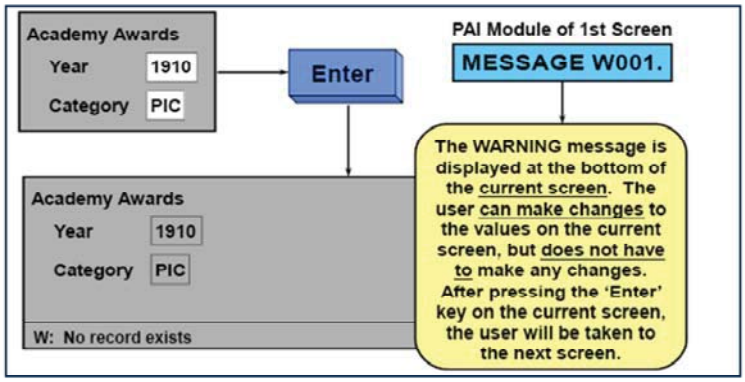
- Abend Message is displayed on the current screen in a dialog box.
- The user cannot make any changes to the values on the current screen.
- After pressing the 'Enter' key on the dialog box transaction will be terminated.



Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

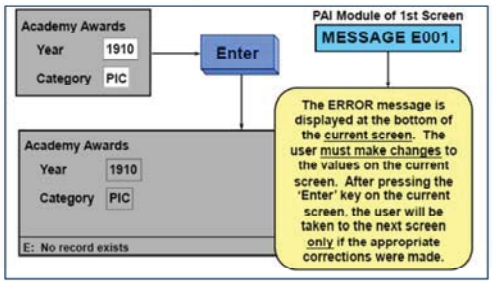
Message Types

- 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.



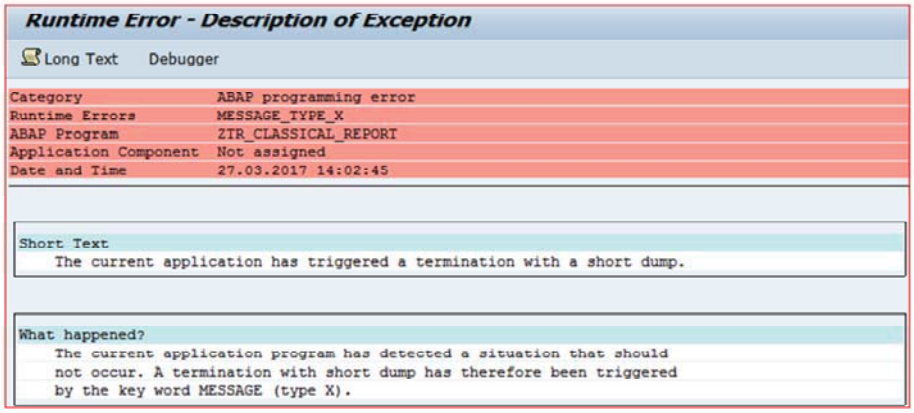
Message Types

- 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.



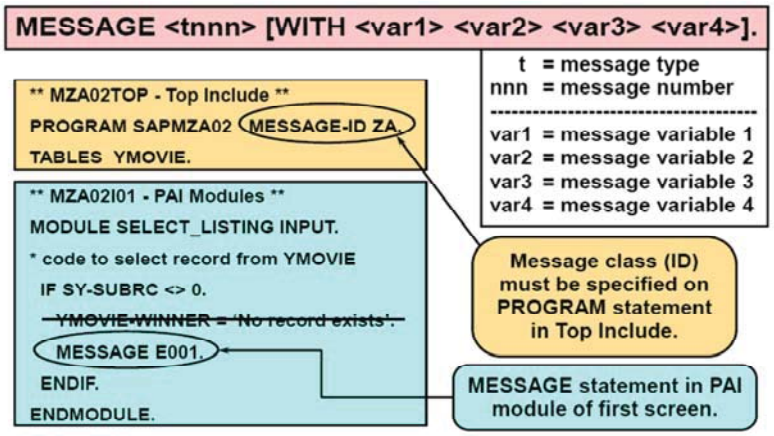
Message Types

- 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>.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 45

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
```

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

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.
```

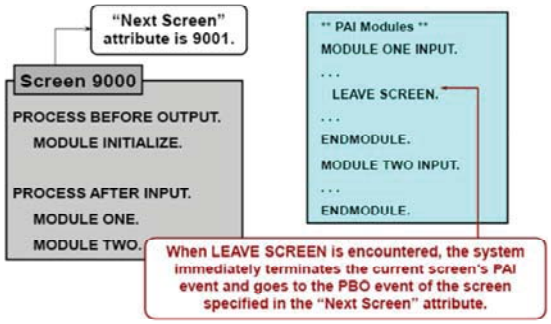
  - If an error or warning message is issued in <module name>, all the screen fields listed in the “FIELD” statements will be open for input.

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

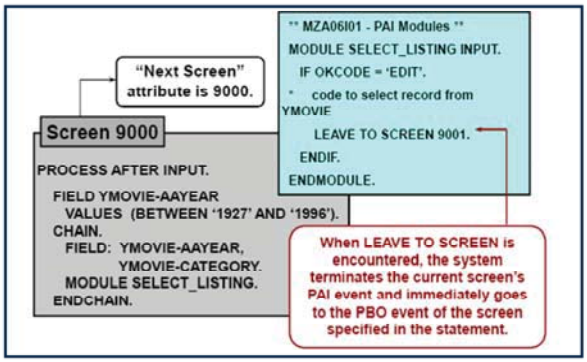
- 1.The LEAVE SCREEN statement ends the current screen and calls the subsequent screen.
- 2.When the system encounters the “SET SCREEN <screen #>” ABAP statement, it temporarily overrides the “Next Screen” attribute with this <screen #> and the PAI processing continues. After all PAI modules are executed, the system goes to the PBO event of <screen #>



Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

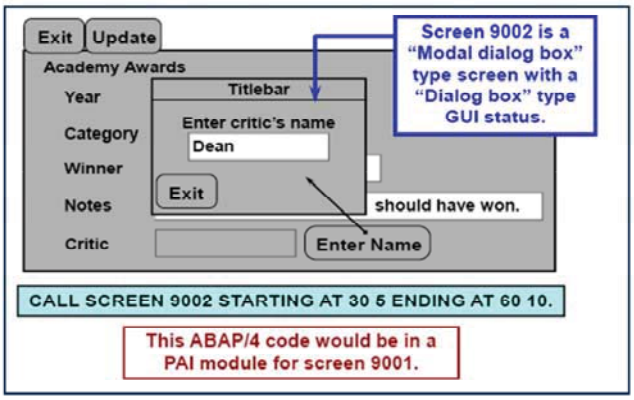
Leave to Screen

- 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

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 52

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

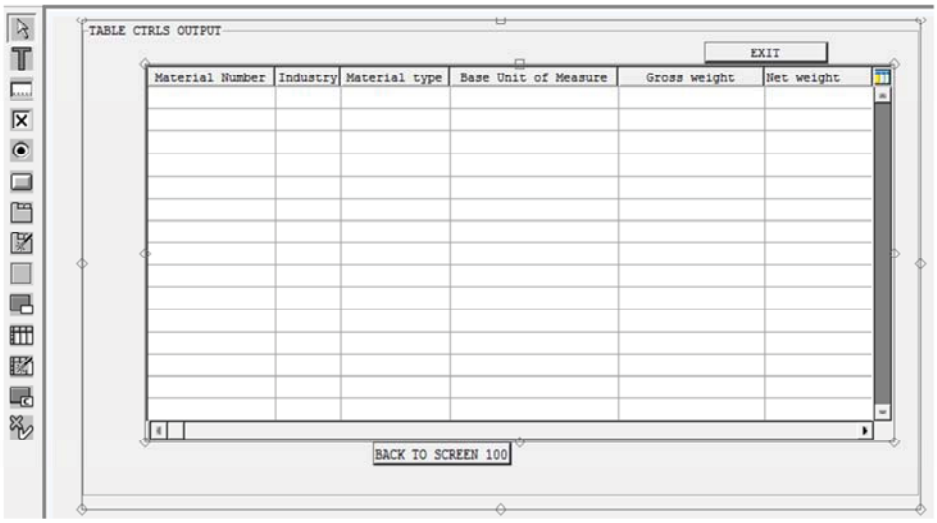
- This is because of the LOOP statement causes the screen fields to be copied back and forth between the ABAP program and the screen field. For this reason at least an empty LOOP AT ...ENDLOOP must be there.

```
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.
```

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

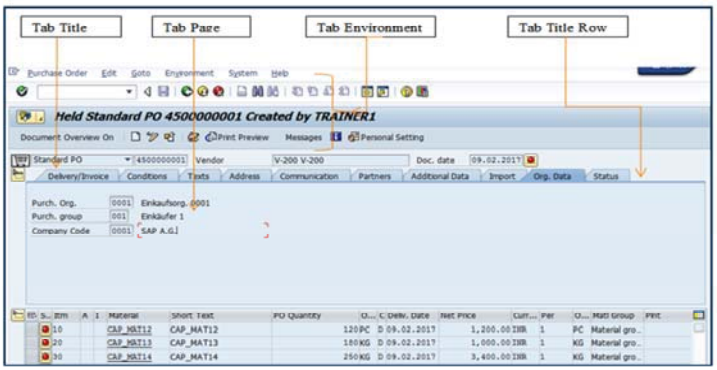
 56

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

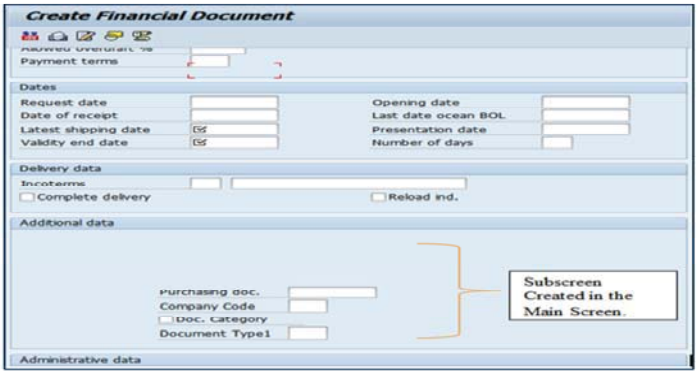
- Subscreen
  - Embedding one screen within another and a subscreen cannot be displayed by itself.
  - A subscreen is a screen that is displayed in a specified area of the main screen.
  - The subscreen displayed in the predefined area will depend on the user's request on the main screen.
  - Subscreen 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

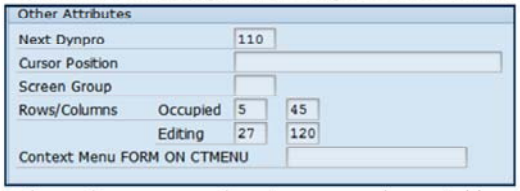
- Creating Subscreen In main screen
  - In the screen layout, place the screen element 'subscreen' on layout.
  - You can set the length and width of the subscreen by double clicking on subscreen. Provide unique name to subscreen, save and activate it.



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.

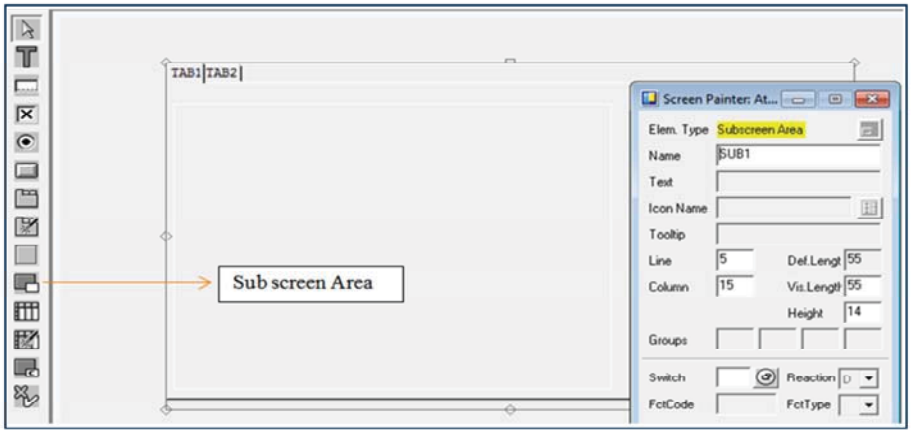


  - Save the subscreen and go to screen layout. Here subscreen are will be displayed.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

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.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 65

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

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 67

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
CONSULTING. TECHNOLOGY. OUTSOURCING

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

ABAP/4
Module - 3 LAB BOOK

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Table of Contents

Getting Started

1.1 Overview

This lab book is a guided tour for learning SAP ABAP. It comprises of assignments to be done. Refer the demos and work out the assignments given by referring the case studies which will expose you to work with Java applications.

1.2 Setup Checklist for SAP ABAP

Here is what is expected on your machine in order to work with lab assignment.

Minimum System Requirements

- Intel Pentium 90 or higher (P166 recommended)
- Microsoft Windows 2010 or higher.
- Memory: (8GB or more recommended)

Please ensure that the following is done:

- SAP GUI is installed
- Connection to the SAP Server is present

Lab 1-1 Modularization Techniques

Goals	• How to create the RFC connection and Call the RFC functions by using the RFC DESTINATION. • How to Create the Function Group and Function Modules. • How to Create and Call the Subroutines.
Time	120 Minutes
Lab Setup	• Connectivity to SAP server • Login details for connecting to SAP server

Function Modules:

Assignment # 1:

Spell out numbers based on the user input value.

SPELL_AMOUNT This function module converts an amount or number into words based on the Language.

- Convert a number into words.

  To do this, the transfer parameters LANGUAGE and AMOUNT have to be entered.

Create an executable program to call the predefined function module **SPELL_AMOUNT**.

Eg: If the user input is **1000** (One Thousand) as showed in the below screen.

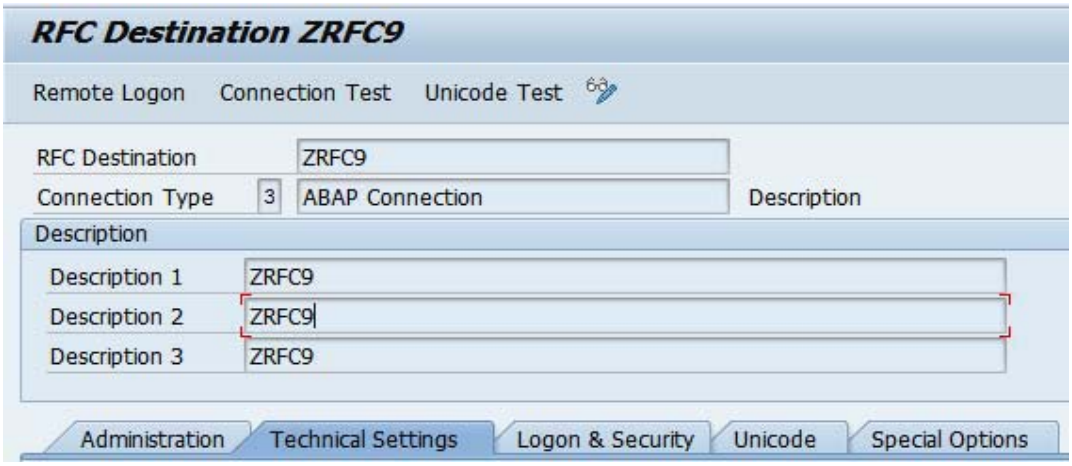
Expected output should be **ONE THOUSAND** in words for the Language key EN.

Assignment # 2:

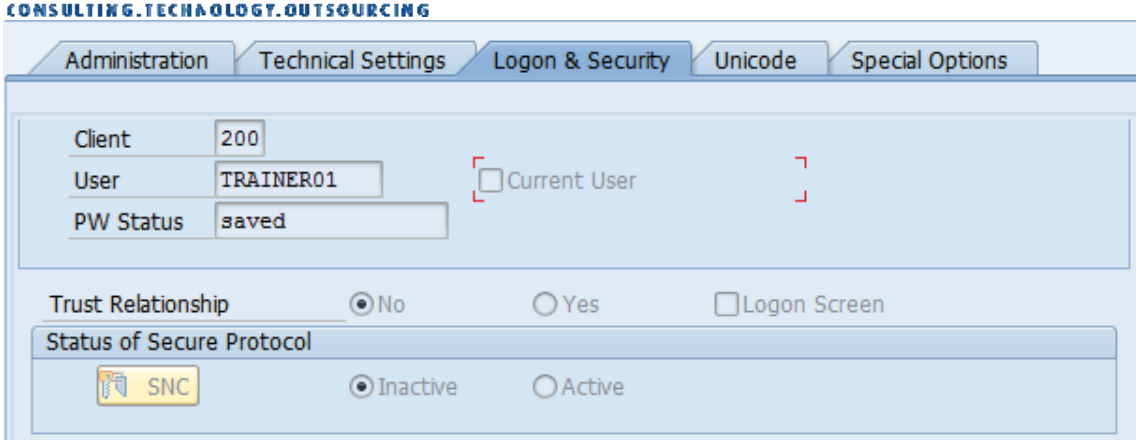
Create a RFC Connection between the two clients in the same or different application system.

Step # 1: Go to SM59 T-code and select the ABAP (Type- 3) connections and create the RFC.

Step # 2: Provide the RFC destination name, short description and click on the tab **Logon & Security.**



Step # 3: Provide the client number, user name and password for which client you want to connect the same or other application server.

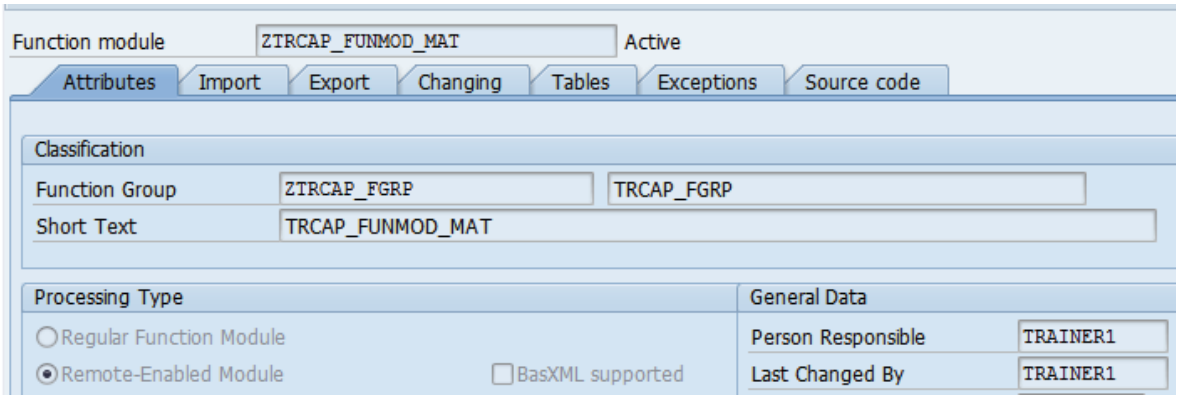


Step # 4: Save the details and Click on the Remote Logon. It will connect to other client.

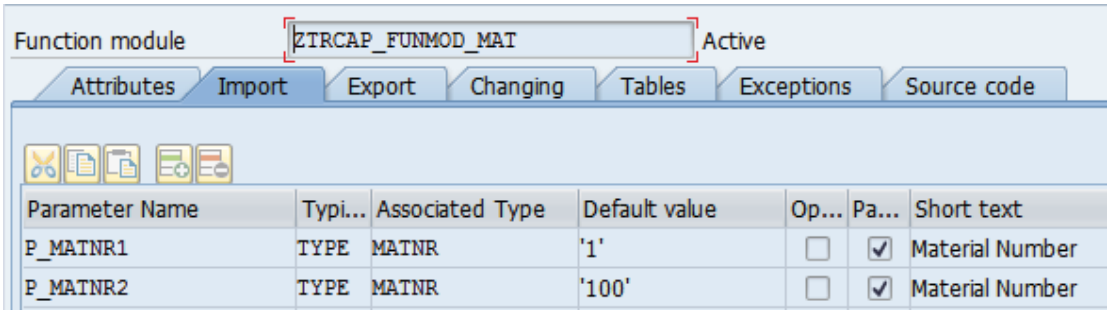
Assignment # 3:

1) Create a Function Group and RFC Function Module in Client 100 Outbound.
2) In Client 100 write an abap program to call the RFC function module to get the data remotely from other client 200.

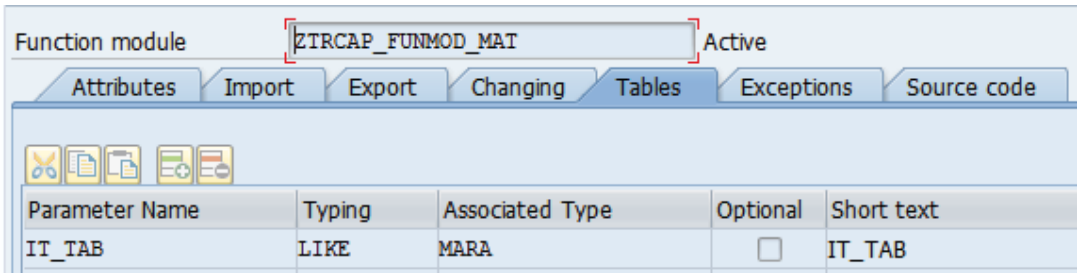
Step # 1: Go to SM37 T-Code and provide the function name, description and select the Remote-Enabled Module radio button in the **Attributes Tab.**



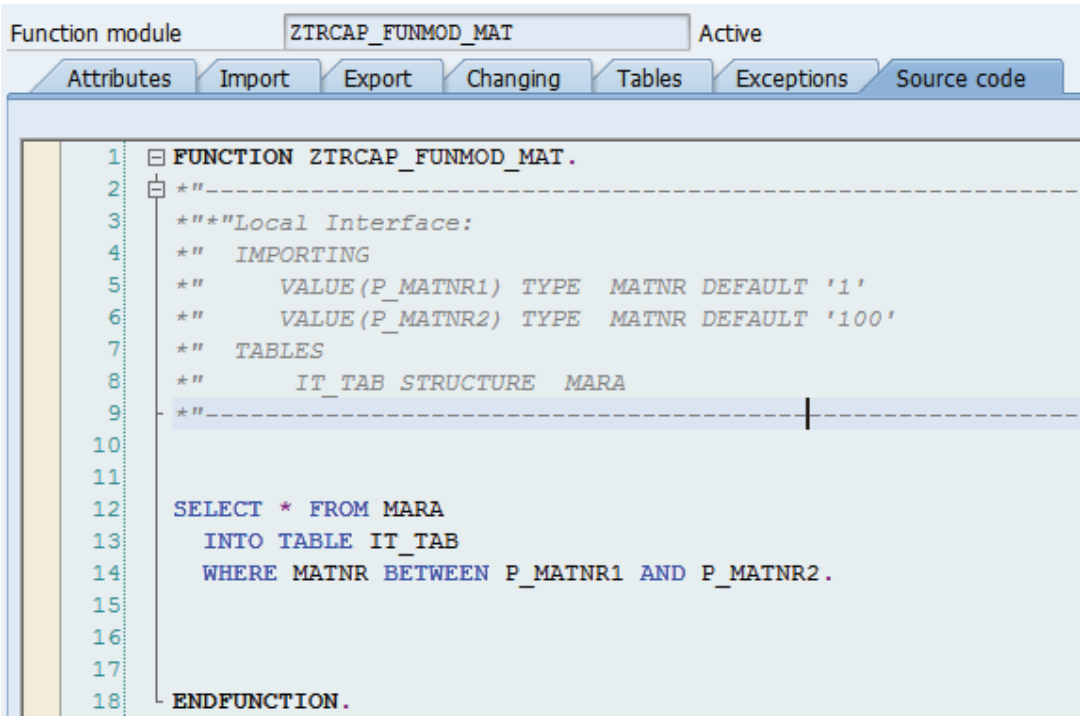
Step # 2: Create the two import parameters to accept the range of materials from the user.



Step # 3: Create an internal table under the tables parameter based on the requirement.



Step # 4: write the function module **source code** logic based on the import parameters and tables.



Step # 5: Execute the function module and provide the material value range in import parameters.

Test Function Module: Initial Screen

Debugging | Test data directory

Test for function group: ZTRCAP_FGRP
Function module: ZTRCAP_FUNMOD_MAT
Uppercase/Lowercase: ☐

RFC target sys:

Import parameters	Value
P_MATNR1	1
P_MATNR2	100

Tables	Value
IT_TAB	0 Entries

Step # 6: Click on the table entries icon.

Tables	Value
IT_TAB	0 Entries
Result:	14 Entries

Step # 7: you will get the data from the login client eg: client no 100.

14 Entries

MAN	MATNR	ERSDA	ERNAM	LAEDA	AENAM	VPSTA	PSTAT
100	3	08.02.2017	TRAINEE01			K	K
100	4	08.02.2017	TRAINEE01			K	K
100	5	08.02.2017	TRAINEE01			K	K
100	6	08.02.2017	TRAINEE01			K	K
100	7	08.02.2017	TRAINEE04			K	K
100	13	15.02.2017	TRAINEE06			K	K
100	31	01.03.2017	TRAINEE05			K	K
100	32	01.03.2017	TRAINEE05			K	K
100	33	01.03.2017	TRAINEE05			K	K
100	34	01.03.2017	TRAINEE05			K	K
100	35	01.03.2017	TRAINEE05			K	K
100	37	01.03.2017	TRAINEE05			K	K
100	38	01.03.2017	TRAINEE05			K	K
100	39	01.03.2017	TRAINEE05			K	K

Step # 8: Write an executable program in client 100 and execute it to get the data from the remote client 200 by using the RFC destination.

CALL FUNCTION 'ZTRCAP_FUNMOD_MAT' DESTINATION 'ZRFC9'(SM59 RFC Name)

Note: Afer call function with **DESTINATION <RFC NAME>** you will get the 200 client data, without that you will get the login clietn 100 data only.

```
*&---------------------------------------------------------------------*
*& Report ZTRCAP_FUNMOD1_MAT_PRG
*&---------------------------------------------------------------------*
*&
*&---------------------------------------------------------------------*
REPORT ZTRCAP_FUNMOD1_MAT_PRG.
TABLES MARA.

DATA IT_TAB2 TYPE MARA OCCURS 0 WITH HEADER LINE.

SELECT-OPTIONS S_MATNR FOR MARA-MATNR.

CALL FUNCTION 'ZTRCAP_FUNMOD_MAT'  DESTINATION 'ZRFC9'
 EXPORTING
   P_MATNR1       = S_MATNR-LOW
   P_MATNR2       = S_MATNR-HIGH
  TABLES
    IT_TAB         = IT_TAB2
          .

LOOP AT IT_TAB2 INTO IT_TAB2.
  WRITE : / IT_TAB2-MANDT,
            IT_TAB2-MATNR,
            IT_TAB2-MBRSH,
            IT_TAB2-MTART,
            IT_TAB2-MEINS.
  ENDLOOP.
```


SUBROUTINES:

Assignment # 1:

Passing Parameters by Vaule
Create a simple program which accepts a material number. Write a subroutine which passes the material number by value and displays the following details regarding in the subroutine:
Material Number
Industry Sector
Material Type
Base UOM
Gross weight
Net Weight

Assignment # 2:

Passing Parameters by Reference
Make a copy of the above program and pass the material number by reference. Change the material number in the subroutine and display the details in the main program.

Assignment # 3:

Passing Structures
Create a simple program which accepts a material number. Write down a select query in the program which retrieves details of the Material Number in the structure.
Create a subroutine which receives the structure and displays the data.
Hint: Use Select Single to retrieve a single record

Assignment # 4:

Passing Internal Tables
Create a simple program which accepts a material number. Write down a select query in the program which retrieves details of the Material Number in the internal table.
Create a subroutine which receives the internal table and displays the data.
Note: Do various options of declaring internal table with/without header line.

Lab 2-1 Interactive List Events

Goals	• How to use an interactive list events.
Time	2 Hours
Lab Setup	• Connectivity to SAP server • Login details for connecting to SAP server

Assignment # 1:

Create an Interactive list event report with Hide statement for maximum 2 secondary list levels.

Create an executable program to prepare the range of materials in the basic list from the MARA table based on the user selection of materials, prepare at least two secondary list reports accordingly.

Program Logic Hints:

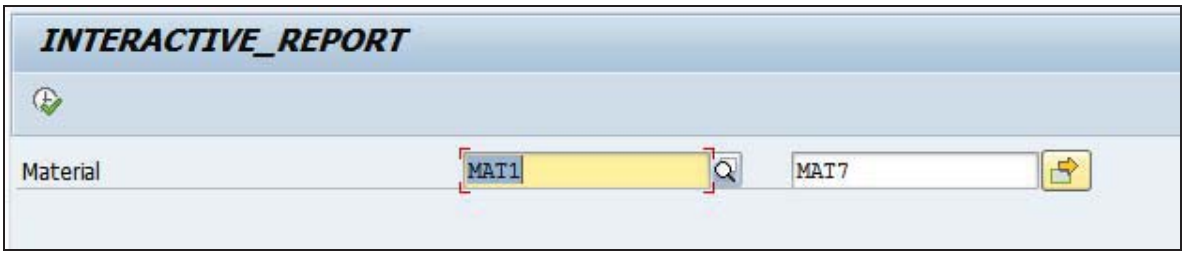
- Declare the events in the report.
  START-OF-SELECTION. END-OF-SELECTION.
  TOP-OF-PAGE. END-OF-PAGE. AT LINE-SELECTION.
  TOP-OF-PAGE DURING LINE-SELECTION.

- Use the HIDE Statement Inside the loop for each list.

Reference T-Codes and Tables:

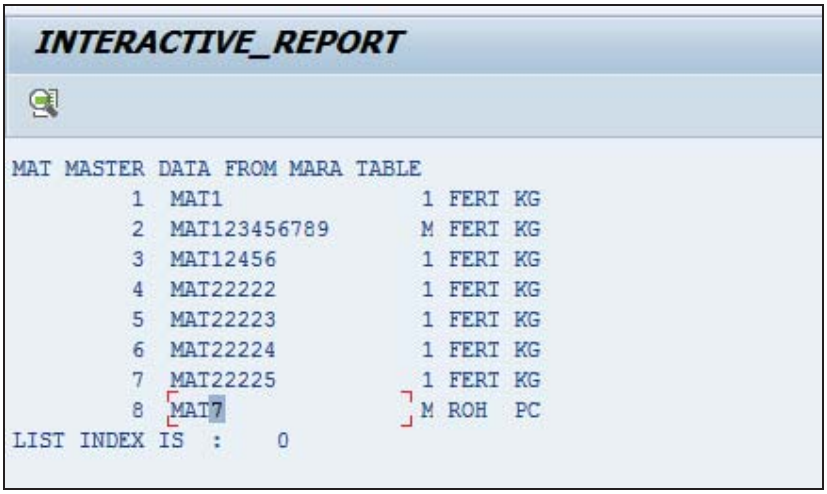
T-Codes: SE38 and MM03. **Tables:** MARA, MARC and MAKT.

Step # 1. Go SE38 T-Code to create an executable program and the Input should be Materials range and it should be an obligatory.

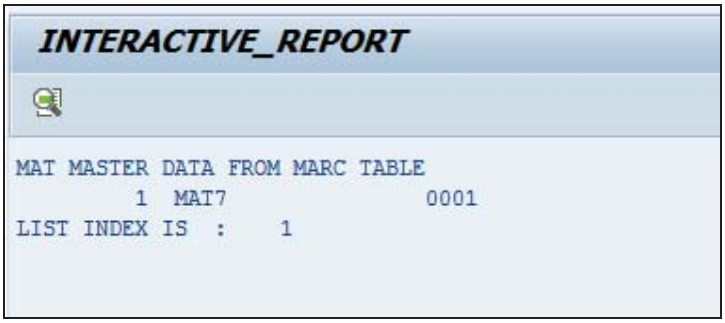


Step # 2. Prepare the Basic List (INDEX 0) fields from the MARA table.

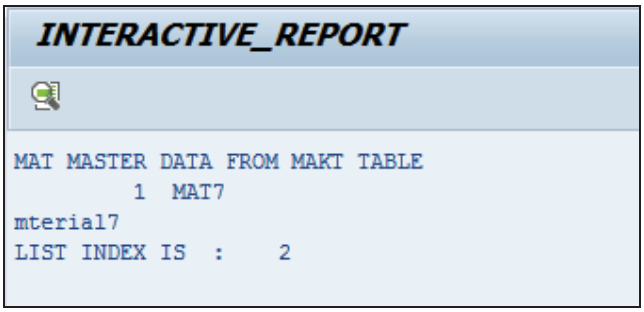
Field names: MATNR, MBRSH, MTART and MEINS.



Step # 2. Prepare the First Secondary list (INDEX 1) fields from the MARC Table.
Field Names: MATNR and WERKS.



Step # 2. Prepare the Second Secondary list (INDEX 2) fields from the MAKT Table.
Field Names: MATNR, MAKTX and SPRAS.



Assignment # 2:

Create an At User-Command Interactive report.

Create an executable program to prepare the basic list and user command by using the menu painter.

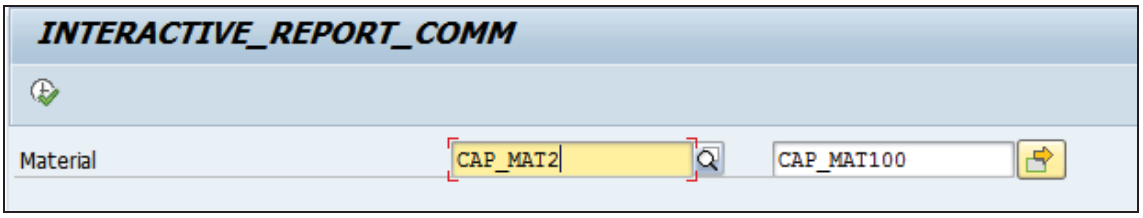
Program Logic Hints:

- Declare the events in the report.
  START-OF-SELECTION. END-OF-SELECTION. TOP-OF-PAGE. AT USER-COMMAND.
- Use the SET PF-STATUS to design the menu painter.

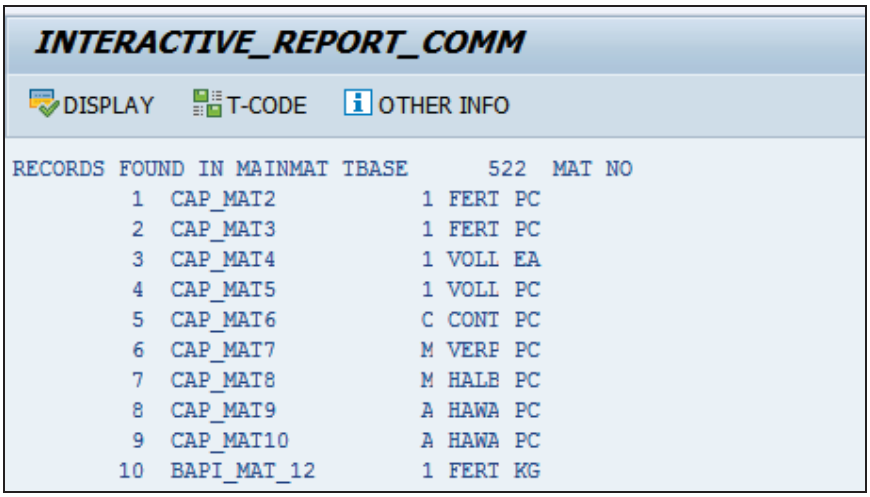
Reference T-Codes and Tables:

T-Codes: SE38 and MM03 **Tables:** MARA, MARC and MAKT.

Step # 1. Go SE38 T-Code and create an executable program and the Input should be Materials range and it should be an obligatory.



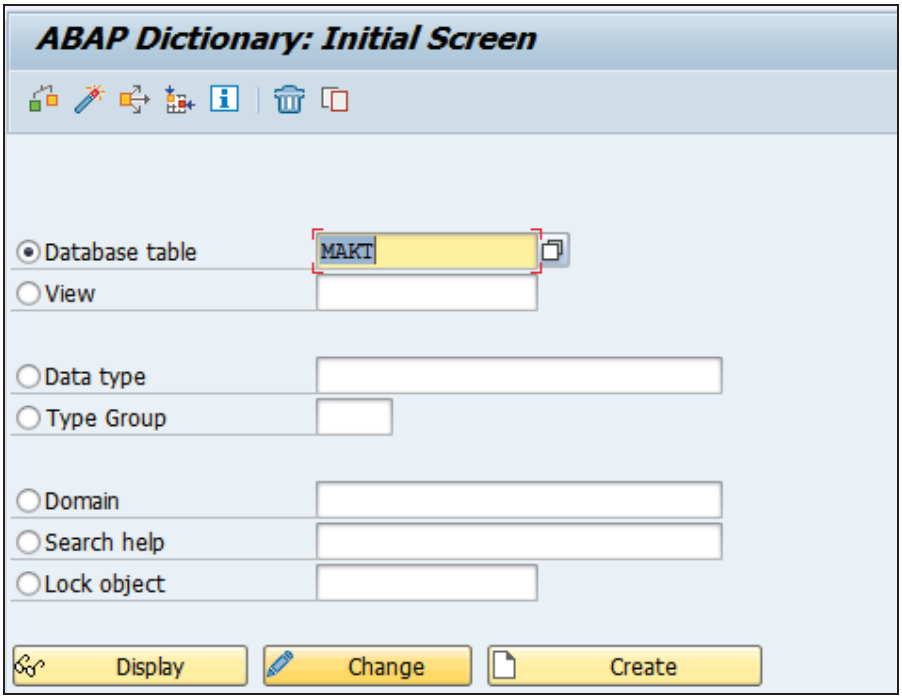
Step # 2. Display the PUSHBUTTONS , T-CODE and OTHER INFO by using the Application Tool bar (T-Code SE41 Menu Painter) in the report output and prepare the basic list data from MARA Table for MATNR, MBRSH , MTART and MEINS fields based on the select options.



Step # 3. When user selects the **DISPLAY Pushbutton**, Get the data from MARC and MAKT tables based on the select options range by using the for all Entries Concept.

Display the following fields output:
MARC Table: MATNR, WERKS, PSTAT, LVORM, BSTMI and BSTMA.
MAKT Table: MATNR, SPRAS, MAKTX and MAKTG.

Step # 4. When user selects the **T-Code Pushbutton**, Call the transaction SE11.



Assignment # 3:

Create an Interactive report by using the GET CURSOR Technique.

Reference T-Codes and Tables:

T-Codes: SE38 and ME23N and MK03 **Tables:** EKKO, EKPO and LFA1.

Go SE38 T-Code and create an executable program with simple interactive report by using the

GET CURSOR FIELD FNAM VALUE FVAL.

Display the basic list report with Vendor no(LFA1-LIFNR) and Name (LFA1-NAME1) and Purchase order (EKKO-EBELN).

Vendor No	Vendor Name	Purchase Order No
Eg:		
V-101	V-100ABC	4500000001
V-102	V-101PQR	4500000002
V-103	V-102XYZ	4500000003

```
IF FNAM = 'WA_TAB-VENDOR'. " If user clicked on the VENDOR NO.
```

Display the Vendor Master details by using the LFA1 table and display the fields LIFNR AND NAME1 and any other 5 Fields output.

```
ELSEIF FNAM = 'WA_TAB-EBELN'. " If user clicked on the PURCHASE ORDER NO.
```

Display the Purchase order details by using the EKKO table and display the Fields EBELN and any other 5 Fields output.

```
ENDIF.
```

Assignment # 4:

Create an Interactive report by using the model dialog box.

Go SE38 T-Code and create an executable program with model dialog box for both the basic list and secondary lists.

Consider the sales order header and line items tables i.e VBAK and VBAP. Display the output for any 5 Fields from each tables.

Lab 3-1 Module Pool Programming

Goals	• Screen designing by using subscreens, tab strips, table controls and user defined transaction codes to update the data in the ztables.
Time	4 Hrs.
Lab Setup	• Connectivity to SAP server • Login details for connecting SAP server

Assignment # 1:

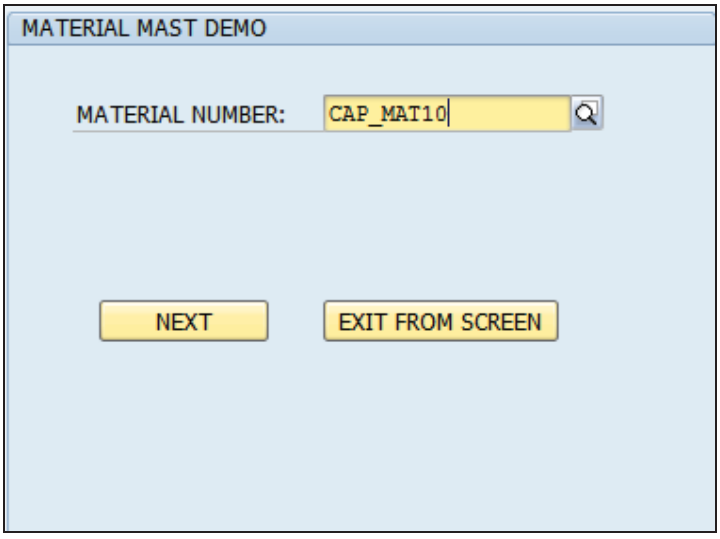
Create a simple module pool transaction to display the single material output.

Reference T-Codes and Tables:

T-Codes: SE38, SE51, SE93 and MM03 **Tables:** MARA.

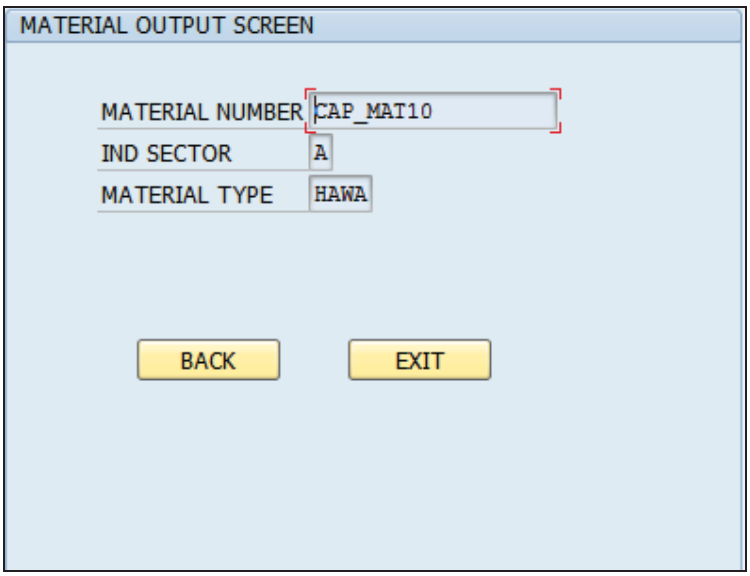
Go to SE38 and SE51 T-Codes to create module pool program.

Step # 1: In the first screen accept the single material number from the user and when you click on the **NEXT** button the output should be displayed in the second screen and when you click on the **EXIT FROM SCREEN** button **L**eave from the screen.



Step # 2: In the second screen display the single material output based on the input provided in the first screen and when you click on the **BACK** Button control should be back to first screen to modify the input of the material number and when you click on **EXIT** Button leave from the program.

MATERIAL NUMBER, IND SECTOR and MATERIAL TYPE Fields should be in display mode (Output fields) and user cannot edit the fields at runtime.



Assignment # 2:

Create a module pool program to display the table control output.

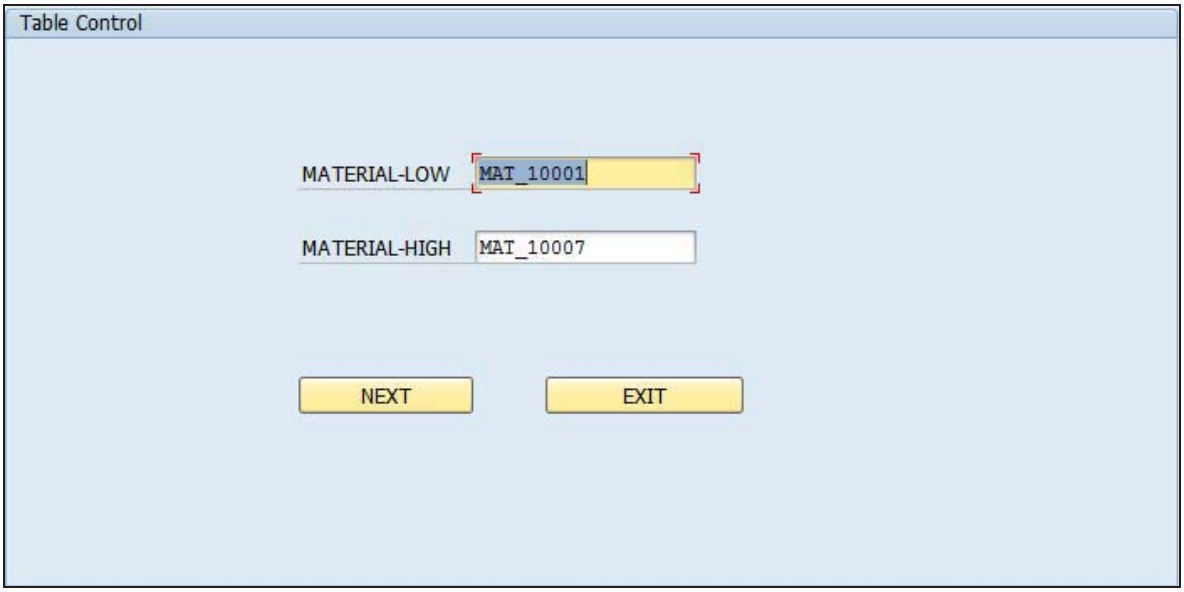
Reference T-Codes and Tables:

T-Codes: SE38, SE51, SE93 and MM03 **Tables:** MARA, MARC And MAKT.

Go to SE38 and SE51 T-Codes to create module pool program.

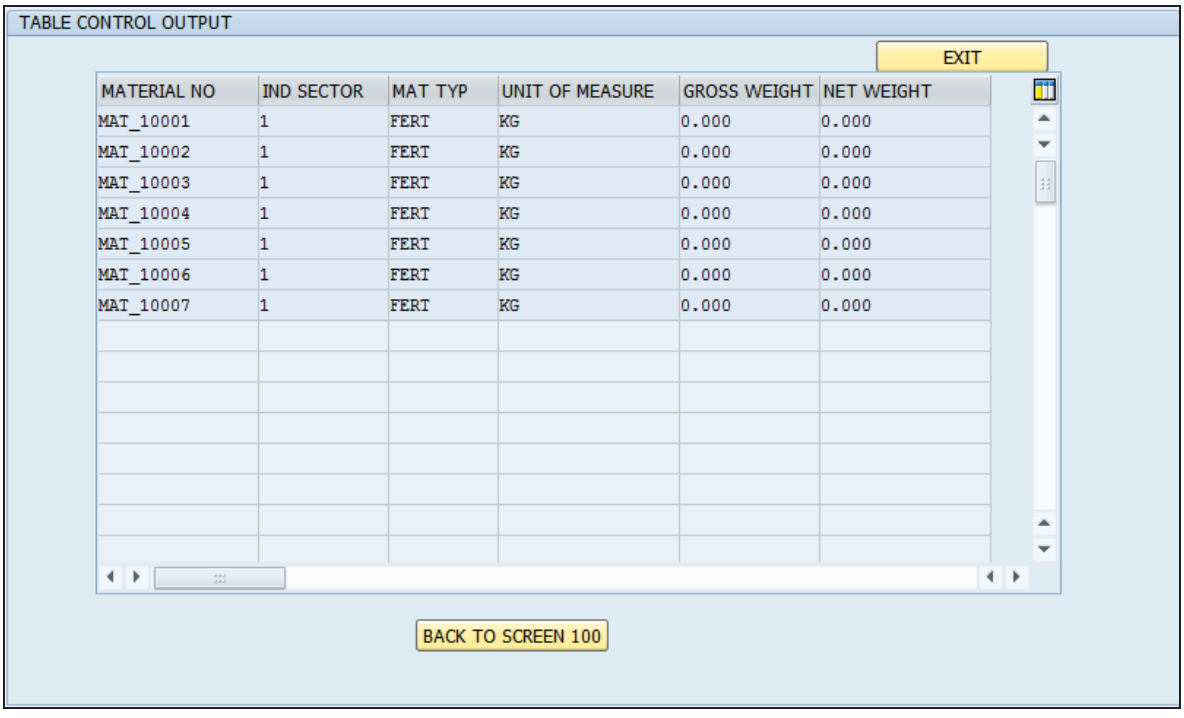
In the First Screen accept material range from the user and when you click on the NEXT pushbutton the output should be displayed in the second screen as a table control.

When you click on the EXIT button Leave from the screen.



In the Second Screen When you click on the **BACK TO SCREEN 100** push button, go back to the initial screen to modify the input of the material range and when you click on the **EXIT** push button leave from the program.

Table Control output should be in display mode user cannot be edit the fields at runtime.



MATERIAL NO	IND SECTOR	MAT TYP	UNIT OF MEASURE	GROSS WEIGHT	NET WEIGHT
MAT_10001	1	FERT	KG	0.000	0.000
MAT_10002	1	FERT	KG	0.000	0.000
MAT_10003	1	FERT	KG	0.000	0.000
MAT_10004	1	FERT	KG	0.000	0.000
MAT_10005	1	FERT	KG	0.000	0.000
MAT_10006	1	FERT	KG	0.000	0.000
MAT_10007	1	FERT	KG	0.000	0.000

Assignment # 3:

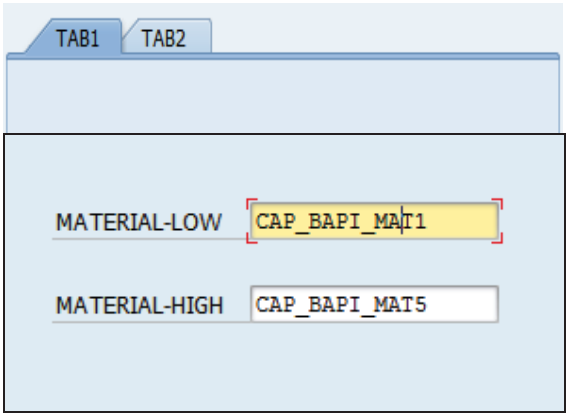
Create a module pool program to display the tabstrip control output.

Go to SE38 and SE51 T-Codes to create module pool program.

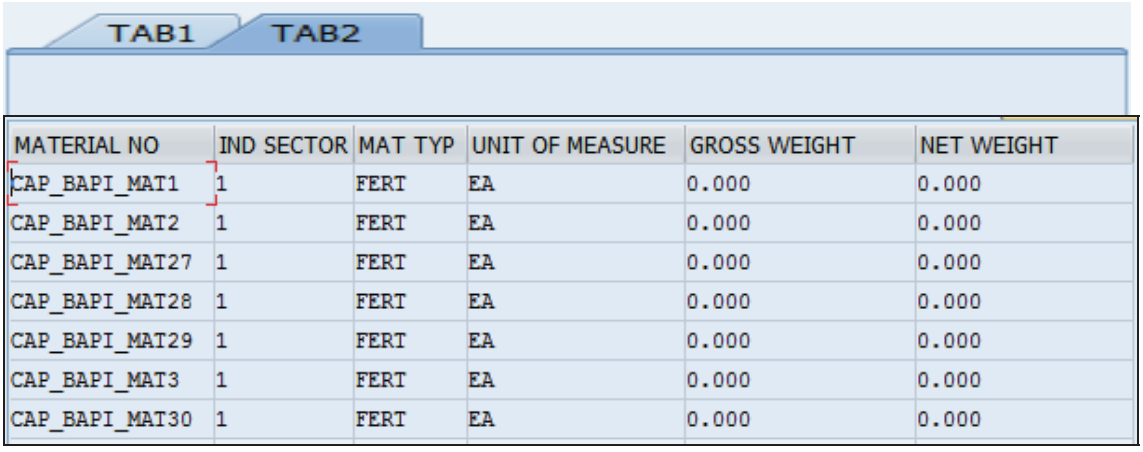
Reference T-Codes and Tables:

T-Codes: SE38, SE51, SE93 and MM03 **Tables:** MARA.

Step # 1: In the first screen create the tabstrip control in **TAB1** to accept the material range from the user and when you click on the **TAB2** pushbutton the output should be displayed in the second screen as a table control format.



Step # 2: In the second screen create table control and get the data from mara table based on the material range provided in TAB1.



MATERIAL NO	IND SECTOR	MAT TYP	UNIT OF MEASURE	GROSS WEIGHT	NET WEIGHT
CAP_BAPI_MAT1	1	FERT	EA	0.000	0.000
CAP_BAPI_MAT2	1	FERT	EA	0.000	0.000
CAP_BAPI_MAT27	1	FERT	EA	0.000	0.000
CAP_BAPI_MAT28	1	FERT	EA	0.000	0.000
CAP_BAPI_MAT29	1	FERT	EA	0.000	0.000
CAP_BAPI_MAT3	1	FERT	EA	0.000	0.000
CAP_BAPI_MAT30	1	FERT	EA	0.000	0.000

Assignment # 4:

Create a module pool program to work with ztable DML operations.

Go to SE38 and SE51 T-codes to create module pool program.

Reference T-Codes and Tables:

T-Codes: SE38, SE51 and SE93. **Tables:** zemp.

Design the module pool screen to update the zemployee table information.

Note: Select the **Display/Maintenance Allowed with restrictions** option under **Delivery and Maintenance** tab in the table (SE11) to prevent the data Load/Insert directly on to the table**.**