ABAP REPORTS

TABLE OF CONTENTS

DEBUGGING

Many times an error free program doesn't give desired output. Behavior of program is different in different situations, with different values of variable. Such program needs additional testing, by which you can test the program by stopping at each point where you feel program is behaving abnormally.

The ABAP/4 debugger is the development workbench tool, which allows you to stop a program during its execution when a particular condition is met. When the program is stopped, you can use the debugger to display the contents of the table and variable being used by the program. It allows you to execute the program step by step, reviewing exactly the real flow of the program execution.

There are many occasions during normal system operation during which the ABAP/4 debugger can be started. When executing program, the ABAP/4 debugger is automatically started when the system encounters a breakpoint.

Starting the ABAP/4 debugger

There are many ways to start debugger

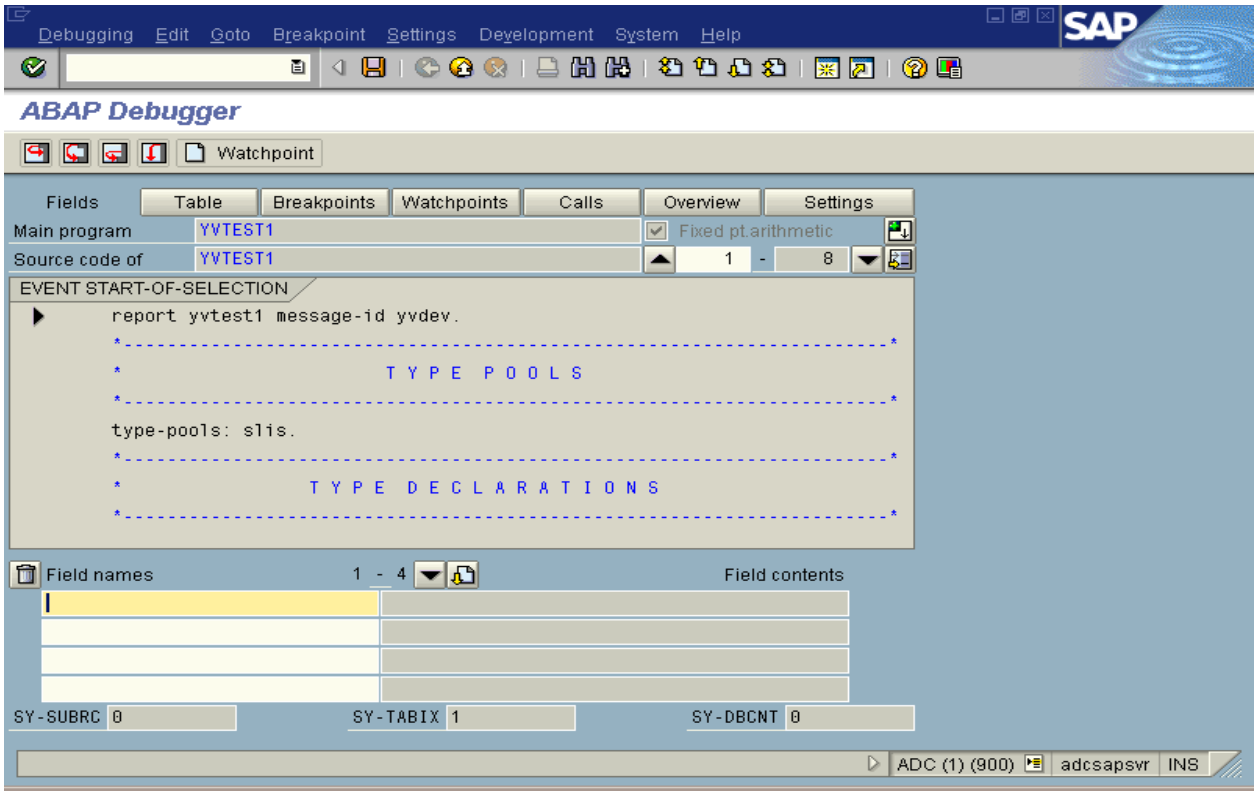
- By clicking the Execute button and selecting the debugging mode.
- From the ABAP/4 editor, by executing a program choosing Program → Execute → Debugging from the menu.
- Setting breakpoint in the program

Components of ABAP/4 debugger

The debugger shows the program information using six different views.

- **Fields:** Displays the field contents.
- **Table:** Allows modifying the contents of internal table.
- **Breakpoints:** Displays list of Breakpoint in the Program.
- **Watchpoints:** Allows dealing with Watchpoints.
- **Calls:** System call status like Event, Form etc.,
- **Overview:** Presents the program structure, events, subroutines, and modules.
- **Settings:** Displays the calling sequence within a particular event, up to the current breakpoint.

All these options are shown in the following screen.



Arrow indicates the breakpoint of the program i.e., where user has stopped the program.

Breakpoints

A breakpoint is the signal, which is specified in the program, tells the system to stop the program execution and to start the debugger. Following types of breakpoint are available with ABAP/4:

- Static are set up with the BREAKPOINT keyword inside the program, which you can directly display with the ABAP/4 source code editor. To set the breakpoint in the program enters the keyboard BREAKPOINT.
- Dynamic this breakpoint is not visible in the code. Position the cursor over the source code line to have the breakpoint and then select utilities - → breakpoint - → set. You can delete them or display them from breakpoint list. Or you can execute the program in the ABAP/4 debugger i.e., in debugging mode.
- Watchpoints are field specific. The program is stopped when the field reaches the value specified in the watchpoint. Execute the program in debugging mode. Position the cursor over the needed field. Press the F button to get the view of field. Select the checkbox for the needed watchpoint. Click on the continue button.
- Keywords/events The program stops just before executing a specific event or keyword. To set breakpoint at particular event, from initial screen of debugger, select Breakpoint → Breakpoint at → at event/at keyword. Enter the name of the keyword or event. Click on OK.

Navigating through the breakpoint

Following buttons are used to navigate through the program and debugger.

- **Single step:** Executes a single program command.
- **Execute:** Similar to the single step, but when a program calls a subroutine, it executes the whole subroutine unlike single step.
- **Continue:** Executes the program until it is finished or until it finds next breakpoint.
- **Return:** Allows for executing the program instruction up to the end of a routine and stops in the line of code where the subroutine gives back control to the main program.
- **Tables:** Switches the debugger to the table view.

Displaying and modifying values

Every time the program is stopped within a debugger, you can display and modify the contents of table field and fields.

To display the fields, click on V and you can view the contents of system field, program field, ABAP/4 dictionary fields, and external program fields.

Displaying and modifying internal tables

When you click on the Table button from the initial ABAP/4 debugger screen, the system will display the table debugger view. Here you need to enter the name of the internal table to be displayed. You can modify or delete or add i.e., insert the internal table Contents. These changes are applicable only for the debugging and do not affect the structure of internal table in the program.

Setting WATCHPOINTS

- Execute a program in debugging mode.
- Position the cursor over the needed field.
- Press the F button to get a view of the fields.
- Select the checkbox for the needed watchpoint.
- Click on the CONTINUE button.
- To display the active watchpoint select Goto - → Breakpoint

INTERNAL TABLES

Consider the following cases:

- You want to reorganize the contents of table.
- You want to modify few details of table and then display the contents of table to user.
- You want to perform table calculations on subset of database table.

In above cases you need to recognize or change the database table contents.

In ABAP/4 you work mainly with tables. Long life data is stored in database tables. You cannot afford to lose data from database table. In such cases where you can not work directly with database table (where you are modifying contents of table or reorganizing the contents of table or any other case where you are altering contents of table and then displaying output to the user) hence need of intermediate table where you put in all the data from database table and work with this data thus avoiding accidental loss of data.

These intermediate tables are called **INTERNAL TABLES** in ABAP/4 and are created only during runtime i.e., **no memory is reserved for internal tables.**

When you use Internal Table in a program, three steps are associated with it.

1. Declaration of Internal Table
2. Populating Internal Table
3. Processing Internal Table

Declaration of Internal Table

Depending on how you create, internal tables are of two types.

- Internal tables with header line.
  When you create internal tables with header line, and then default workarea or header is created. And you can work with this header.

- Internal Table without header line.
  In this case you need to create output explicit workarea to work with table. Only advantage of internal tables without header line is that you can nest them i.e., you can have table within table, which is not possible on internal tables with header line.

Internal table can be declared in the following way:

- Data : Itab like sflight occurs 0 with header line
  Here you are declaring internal table, which is similar to sflight i.e., itab like sflight i.e., you are referring to existing table (like).
  (By default internal table created with this declaration is without header line).

- Data : Begin of itab occurs 0.
  Include structure sflight
  Data : End of itab
  (Internal Table created with this type of declaration is similar to declaration done in 'a' type the only difference is by default internal table created by this type is with header line)

- Data : Begin of itab occurs 0
  carrid like sflight-carrid,
  connid like sflight-connid,
  fldate like sflight-f1date
  End of itab.
  By default internal table created by this type of declaration is with header line. In this type of declaration, you are using only those fields from database table, which you require for processing.

- Data : Begin of itab occurs 0
  carrid like sflight-carrid,
  connid like sflight-connid,
  bookid like sbook-bookid
  id like scustom-id,
  End of itab.
  Here you are combining fields from three different tables in one internal table.

- Data : Begin of itab occurs 0
  Carrid1 like sflight-carrid,
  End of itab.
  Here you are specifying different field names.

Populating Internal Table

- Itab-name = 'ABCD'.
  Append Itab.
  (In this case itab is filled with one name i.e., 'ABCD'.)

- Do 5 times.
  Itab-number = sy-index.
  Append itab.
  Enddo.
  (In this case itab is filled with sy-index for 5 times)

- Select * from sflight into itab.
  Append itab.
  Endselect.

- Select * from sflight into table itab.
  Note: Addition of *Table* in INTO clause, you omit append itab and Endselect

- Select * from sflight
  Move-corresponding sflight to itab.
  Append itab.
  Endselect.

  Note: While using *Move-corresponding*, field names of DB table & Int' table should be same.

- Select * from sflight.
  Move sflight to itab.
  Append itab.
  Endselect.
  Note: In this case structure of sflight and itab should be similar

- Select carrid1 connid1 from sflight into (itab-carr1, itab-connid1)
  Append itab.
  Clear itab.
  Endselect.

Processing Internal Table

Processing of internal table includes:

- Writing: write : / itab carrid. (will write only one field)
  Loop at itab. (Will write whole internal table)
  Write itab
  Endloop.

- Reading: You can read internal table by:
  Read table itab with key carrid = 'LH' (Here you are reading table with key)
  Or Read table itab with index 3. (Here you read table with index 3)
  (Note: Reading of internal table can be done inside the loop or outside the loop)

  Modifying: You can modify contents of internal table by specifying key or index.
  Itab-carrid = 'LM'
  Modify table itab index 3.

- Delete: delete table itab index 3.
  (Will delete record with index 3)

Commands associated with clearing of internal tables are as follows:

- Clear itab : Will clear header of internal table
- Clear itab [ ] : Will clear body of table.
- Refresh itab : Will remove contents of internal table.
- Fee itab : Will de-allocate memory associated with internal table.

Sorting of Internal Tables

To sort contents of internal table you can use

SORT ITAB

This command will sort internal table on all **non-numeric primary keys** in ascending order.

To specify internal table for given key the syntax is,

SORT ITAB BY CARRID ASCENDING.

In this case the table itab is sorted with carrid key in ascending order.

You can sort table in either ASCENDING or DESCENDING order. The default order is ASCENDING.

You can sort table with multiple keys also. The number of SORT key fields is restricted to 250. If you specify more than one field then the system sorts the record first by f1 then by f2 and so on (Here f1, f2 are fields).

Control Break Statements

Control break statements are used to create statement blocks which process only specific table lines the LOOP – ENDLOOP block.

You open such a statement block with the control level statement AT and close it with the control level statement ENDAT. The syntax is as follows:

Table should be sorted when you use control-break statements
You can break the sequential access of internal tables by using these statements.
Syntax:

```
At first.
        <Statement block>
Endat.
```

This is the first statement to get executed inside the loop (remember control break statements are applicable only inside the loop)

So in this block you can write or process those statements which you want to get executed when the loop starts.

```
At New carrid.
        Write:/ carrid.
Endat.
```

In this case whenever the new carrid is reached, carrid will be written.

```
At End of carrid.
        Uline.
Endat.
```

In this case whenever the end of carrid is reached, a line will be drawn.

```
At Last.
        Write:/ 'Last Record is reached'.
Endat.
```

Processing of statements within this block is done when entire processing of entire internal table is over. Usually used to display grand totals.

You can use either all or one of the above control break statements with in the loop for processing internal table.

```
At end of carrid.
        Sum.
Endat.
```

In above case the statement SUM (applicable only within AT-ENDAT) will sum up all the numeric fields in internal table and result is stored in same internal table variable.

SUBROUTINES

The process of breaking down a large program into smaller modules is supported by ABAP/4 through subroutine, also called forms.

Subroutines are programs modules, which can be called from ABAP/4 programs. Frequently used parts of program can be put into subroutines and these subroutines can be called explicitly from the program. **You use subroutines mainly to modularize and structure your program.**

Defining Subroutines

A subroutine is block of code introduced by FORM and concluded by ENDFORM. Following syntax is used to define a form or subroutine:

FORM <name> <parameters>
. . …..
…
ENDFORM.

Here parameters is optional. You can have plain subroutine without the parameters for example.

FORM SUB1.
… . .
… .
ENDFORM.

Calling Subroutines

You can call subroutines from program by following statement:

PERFORM <subr> [<para>].

Parameters are optional. i.e., you can call subroutine without passing any parameter

Perform SUB1.

Passing Data to Subroutines

When you work with global data in subroutines, you can put a copy of the global data on a local data stack and use it to avoid accidental loss of data (In this way you protect global data.)

You can pass data between calling program and subroutines by using parameters.

- Parameters, which are defined during definition of a subroutine with FORM statement are called 'formal parameter'.
- Parameters which are specified during the call of a subroutine with the PERFORM statement are called 'actual parameter'.

Parameters are passed to the FORM either:

- By value
- By Reference
- By value and return.

By Value

```
Data : a type I value 20.
Perform sub1 using a.
Write a.
FORM sub1 using value (p_a)
        P – a = 15
ENDORM.
```

In this case during subroutine call, the formal parameter are created as copies of actual parameter. The formal parameters have the memory of their own. Changes made to formal parameter have no effect on the actual parameter.

Like in this case, though value of p_a is changed to 15, it has no effect on 'a' which remains as 20.

By Reference

```
Data: a type I value 20.
Perform sub1 using a.
Write a.
FORM sub1 using value (p_a)
        P – a = 15.
ENDORM.
```

By default system calls all the forms by reference.

In this case, only the address of the actual parameter is transferred to the formal parameters. The formal parameter has no memory of its own. If you change the formal parameter, change is visible in actual parameter also.

By Value and Return

```
Data : a type I value 20.
        Perform sub1 changing a.
```

```
        FORM sub1 changing value (p_a)
        P – a = 15.
    ENDORM.
```

In this case if you change formal parameter, then the value of actual parameter is changed when the control is transferred back to the main program.

Assuming A is declared by DATA statement and has value 20 and subroutine SUB1 is called by passing A.

CALLING FORM	VALUE OF A IN PROGRAM	VALUE OF A IN FORM (p_a = 15)
BY VALUE (USING)	BEFORE CALLING FORM A = 20 AFTER RETURNING FROM FORM A = 20	A = 20 P_A = 100 (changing value of p_a) A = 20.
BY REFERENCE (USING)	BEFORE CALLING FORM A = 20 AFTER RETURNING FROM FORM A = 20	A = 20 P_A = 100 (changing value of p_a) A = 100
BE VALUE AND RETURN (CHANGING)	BEFORE CALLING FORM A = 20 AFTER RETURNING FROM FORM A = 100	A = 20 P_A = 100 (changing value of p_a) A = 100.

Passing Table to a Subroutine

You can pass internal tables as parameters USING or CHANGING in the FORM and PERFORM statements. If you want to access the components of the internal table, you must specify the type of the corresponding formal parameter.

You also must distinguish between internal tables with or without header lines. For internal tables with header lies, you must specify the table body by using square brackets [ ], after the table name to distinguish it from the header line.

With internal subroutines, you can use TYPE or LIKE to refer to the internal table you want to pass directly.

You can pass all internal tables as parameters in the list after TABLES in the FORM and PERFORM statements. Internal tables passed with TABLES are always called by reference.

If you pass all internal table with a header line, the table body and the table work area are passed to the subroutine. If you pass an internal table without a header line, a header line is created automatically as a local data object in the subroutine.

```
PROGRAM ZDEMO
DATA: Begin of itab occurs 0,
            Number type I,
  end of itab

PERFORM SUB1 TABLES ITAB.

LOOP AT ITAB.
WRITE: / itab-number.
 ENDLOOP.

FORM SUB1 TABLES F_ITAB LIKE ITAB [ ].
DO 3 TIMES.
       F_itab-number = SY-INDEX.
       APPEND F_ITAB.
   ENDDO.
ENDFORM.
```

After starting ZDEMO the output appears as follows:

1
2
3

In this example, an internal table ITAB is declared with a header line. ITAB is passed to the subroutine SUB1, where it is filled using the table work area F_ITAB. And itab is written in main program.

FUNCTIONS

Function modules are special external subroutines stored in a central library. The R/3 system provides numerous predefined function modules that you can call from your ABAP/4 programs, and plus you can create your own function modules.

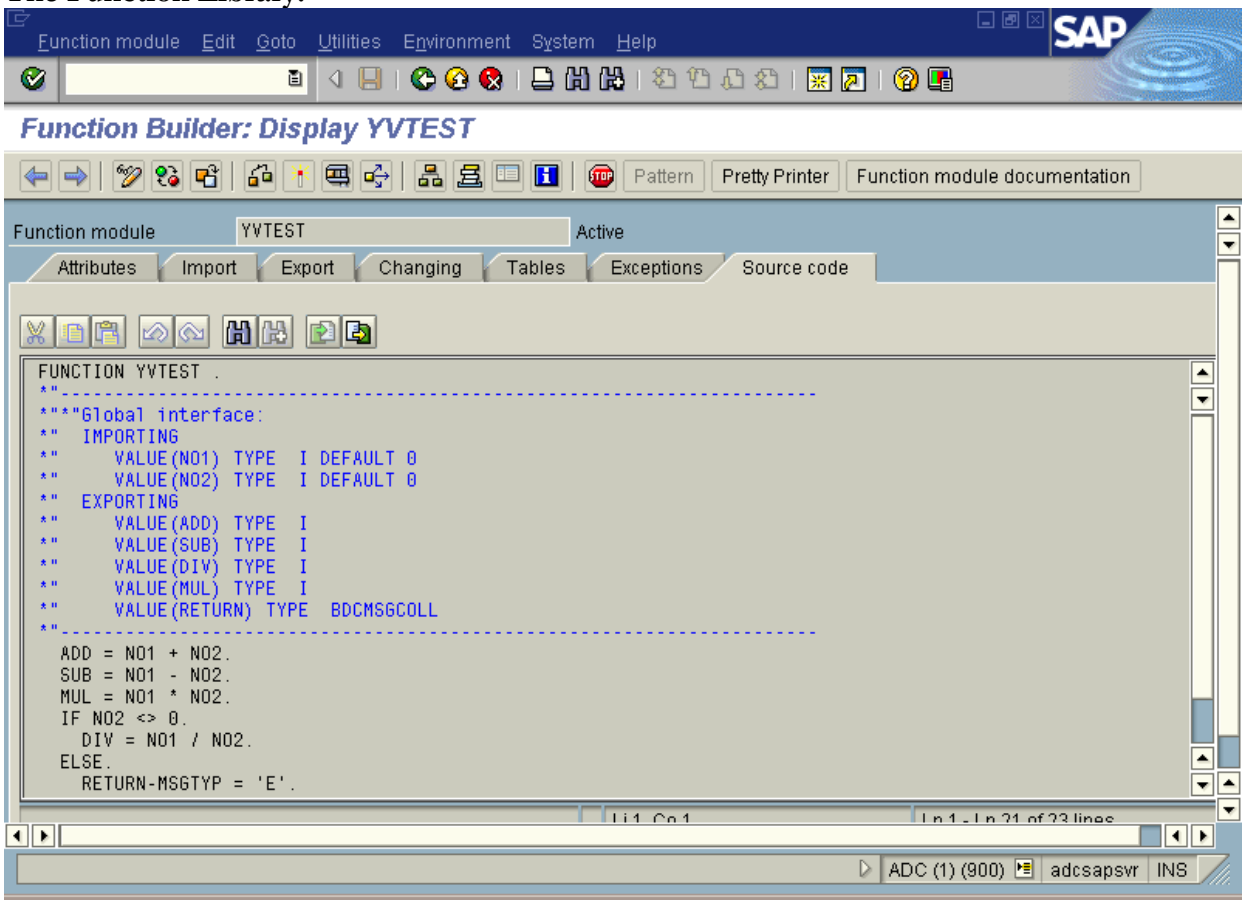
The main difference between a function module and a normal ABAP/4 subroutine is as follows: Function is stored in central library and has global presence while subroutines are confined to a particular program. Subroutine cannot return values while functions can return values. Unlike functions, subroutine cannot handle exceptions. And last but not least, the difference in the way the parameters are passed to functions.

Declaring data as common parts is not possible for function modules. The calling program and the called function module have separate work in ABAP/4 Dictionary tables.
You use the ABAP/4 Workbench to call, create, and maintain function modules.

You can **combine several function modules to form a function group in the function library.** Since you can define global data that can be accessed from all function modules of one function group, it is reasonable to include function modules that operate with the same data, for example internal table for sales module can be grouped, in one function group.

Via the ABAP/4 Development Workbench screen, choose Development → Function Library or select Function Library in the application toolbar or use se37 transaction code.

The Function Library:



The function library maintains Function Modules, the screen displays following components:
All function names should start with Z_ or Y_ followed by any name.

- Source code
- Documentation
- Administrative info
- Import-export parameters
- Table parameters and exception interface
- Global data
- Main program

  (Not necessarily in the same order)

Documentation

The documentation describes the purpose of the function module, lists the parameters for passing data to and from the module, and the exceptions. The parameters of the parameter type I are import parameters, which are used to pass data to the function module. Parameters of the parameter type E are export parameters, which are used to pass data from, the function module to the calling program. Exceptions describe error scenarios, which can occur in function modules.

Import-export parameters

Import parameters correspond to the formal input parameter of subroutines. They pass data from the **calling program to the function module.**

Export parameters correspond to the formal output parameters of subroutines. They pass data from the function module back to the calling program (which his not possible in subroutines)

Table parameters

Table parameters are internal tables. Internal tables are treated like changing parameters and are always passed by reference.

Exceptions

Exceptions are used to handle error scenarios, which can occur in function modules. The function module checks for any type of error and raise exception and returns SY-SUBRC to the calling program. Main program checks this SY-SUBRC for any errors that have occurred and then takes action accordingly.

Source Code

The ABAP/4 Edit screen displays the ABAP/4 source code of the function module. You can work with the source code in the same way as you would work with normal ABAP/4 programs.

Import parameters, changing parameters, and table parameters can be Optional. This means that you can omit the corresponding actual parameter when you call the function in the calling program. If the parameter is optional and the actual parameter is not specified, you can specify a default value for use in the function module. Export parameters are always optional.

As with subroutines, you can specify the data types of the formal parameters in the field Reference type. In the field Ref. structure, you can specify ABAP/4 Dictionary reference structures or fields. Then, the system checks the current parameter against the structure or field at runtime.

Testing of function module

You can test a function module without calling it from an ABAP/4 program via the Function Library: Maintain Function Modules screen by choosing Single test. You can assign values to the import parameters on the Test Function Modules screen.

Calling Function Modules

To call a function module from an ABAP/4 program, use the CALL statement as follows:

Syntax:

```
CALL FUNCTION <module>
 [EXPORTING
        f1   = s1
        f2  = s2
        fn  = sn (parameters which you pass from program to function are
        s1, s2, sn)]
 [IMPORTING
        f1   = r1
        f2  = r2
        fn  = rn (parameters which program receives you pass from function in
        r1, r2, rn)]
 [TABLES      f1 = a1 … fn = an]
        EXCEPTIONS notvalid = 1
        not correct = 2
        OTHERS = 5].
```

You can specify the name of the function module <module> as a literal or as a variable. Parameters are passed to and from the function module by exactly assigning the actual parameters to the formal parameters in the lists after the EXPORTING, IMPORTING, TABLES.

If in your function if you have raised exception not valid then this exception can be handled in main program. Functions return different sy-subrc for different exceptions.

REPORTS

About reports

Reports, in the R/3 system are online programs whose function is to retrieve data from database and display it or print it for the user. An end user often needs some information to look up, depending upon which various management decisions are taken, or to just see business results or simply to continue work. As R/3 is collection of all business applications, it has provided a very powerful feature to satisfy this crucial business need i.e., reports are involved at each and every step of business. This type of extracting, collecting and formatting data that is stored in database is done by REPORT program.

The program that is written to retrieve and display data is REPORT program and the data that is displayed on the screen when you execute the program is called as LIST (output of the report).

SAP has provided thousands of preprogrammed reports. User just selects a menu option or just one click here and there, displays the report. Often user is unaware that by clicking one button he is executing a complicated report program, which is actually accessing database and then displaying the result. An end user might not feel the necessity of writing a REPORT program but a developer has to develop a report manually using the functions and facilities provided by the R/3 system. How to develop a report using these facilities, is the purpose of this section.

When you display data, you need to display the data, user needs. For example, user wants to see the all the employee, who has joined after 12th December 1997. In this case user has to pass this information, to the system, that he needs only those employee records where joining data is greater than 12th December 1997. For user, it is passing information to the system but for the system it is input from the user. System takes input from the user before it retrieves the data from the database. This is very common requirement of any report as the need of any business is to display data, which is required by user.

Selection criteria

System accepts inputs from user through **SELECTION CRITERIA.**

Selection criteria are nothing but input fields which allows the user to restrict information given to program for processing further data. If you don't specify any criteria for selection, your report program produces a long list of data, which might not be needed by the user. Basically, selection criteria are the fields where user enters some value for which he needs information. Through **selection criteria user can enter discrete value or ranges.** For example, user wants to see all the records of the employees, who have joined between 12th December 1997 and 12th May 1998. This range can be entered in selection criteria. As the user becomes more specific for mentioning the criteria, the list will be smaller and more specific.

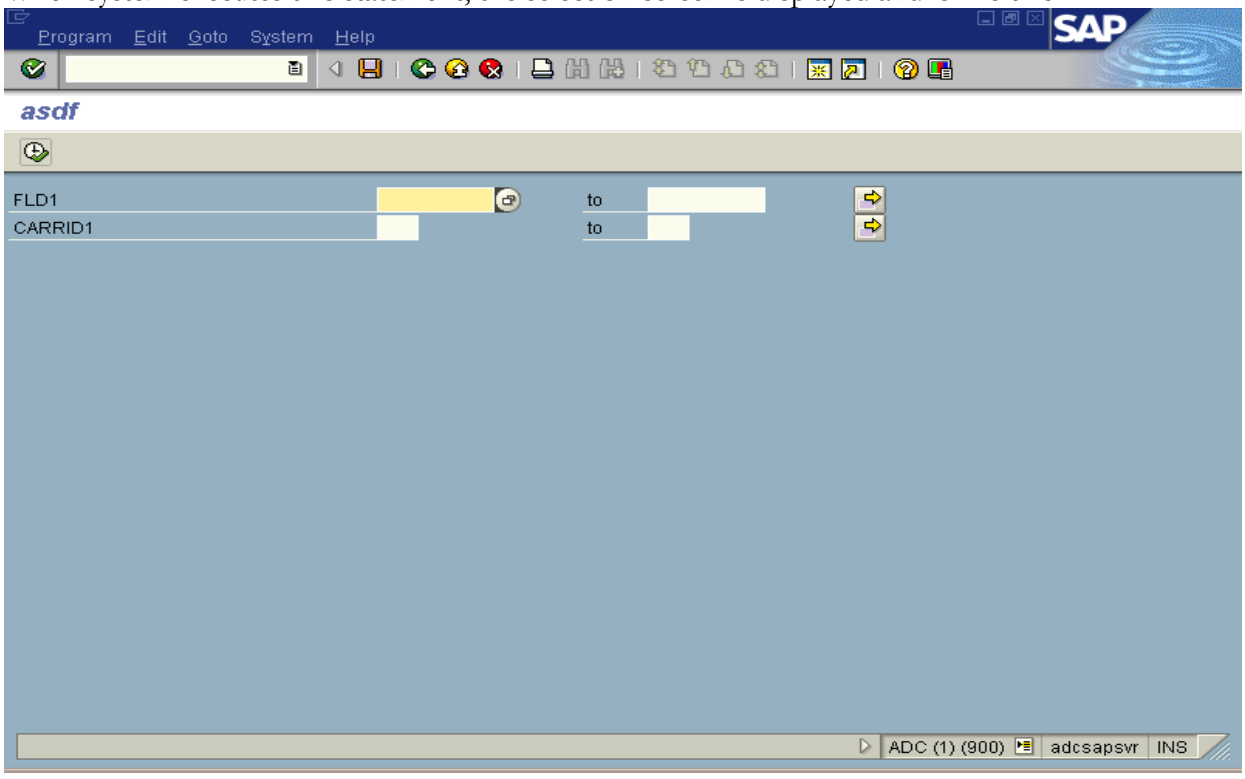
Syntax:

SELECT-OPTIONS <field> for <table field>.

Field is the variable, which you declare for accepting input from the user.

Table field is reference field.

SELECT-OPTIONS: fld1 for sflight-fldate,
carrid1 for sflight-carrid.
Maximum length of the name Select-Options variable is 8.
When system executes this statement, the selection screen is displayed and is like this.



When you enter the desired information and click on execute button, rest of the program is executed, that is retrieval of data from database, which matches this information and the list is displayed.

Behavior of **SELECT-OPTIONS**

When the Select-Options statement is executed the system creates the internal table with the same variable name (in this case it will be carrid1). This table is also called as **selection table.** The main purpose of selection table is to store selection criteria. The table has four standard fields, which are as follows:

- SIGN is a variable, which denotes the system whether the result should be included with those particular criteria. It can contain either I or E. I denotes Inclusion. The criteria are included. E denotes Exclusion. The criteria are excluded from the result.

- LOW the data type of LOW is the same as the field type of the database table, for which you are using selection criteria. This acts as lower limit of the selection.

- HIGH the data type of HIGH is the same field type of the database table, for which you are using the selection criteria. This acts as higher limit. If you don't enter HIGH value then the LOW value defines a single value selection.

- OPTION is two-character field, which contains operators like EQ, GT, LT, GE, and LE. When the user enters both the values i.e., high and low then this field acts as BT (between). If you don't enter high value then all other operators can be Applicable.

For each Select-Options statement system creates internal table.

Default values for select-options

If you want to display default values for the selection criteria before your screen is displayed, give default values for the selection table fields i.e., low or high.

SELECT-OPTIONS: CARRID1 FOR SFLIGHT-CARRID DEFAULT CARRID1-LOW = 'LH' AND CARRID1-HIGH = 'SQ'.

In this case selection screen is displayed with default values 'LH' for lower range and 'SQ' for higher range. User can use same values or overwrite these values with new values, whichever he needs.

Standard report

The normal format of any report is as follows:

HEADING FOR THE LIST i.e., header area

LIST HEADERS

Detailed data

At the end of page you can have sub total of page number i.e., footer area

---

Usually any report has some page headings at the top of page and then data is displayed with column headings, followed by data and at the end of page may be some grand total or page number. Any such report can be displayed by using report program. Such report is called as *Classical Report.*

Usually a standard report produced by system is one continuous page of 65k lines. The standard report can be declared as follows;

REPORT ZDEMO

By default the report produced by system is standard report. But when you need to have your report divided into pages of say 20 lines and you want to reserve some area for footer than you need to use following format of report statement:

```
REPORT ZDEMO line-count 20(3)
                    Line-size 75.
```

In this case the line count for one page is 20 lines, in which 3 lines are reserved for footer. You can display your data only in 17 lines. The width of page will be 75 characters.

As all of us know ABAP/4 is event driven language, the ABAP/4 processor controls the execution of events. For example in above report at the end of the page, you want to write the page number, reaching end of page is an event. Or whenever at the top of a page you want to write list headers, then in this event you can write the code for displaying the list headers. Report program is nothing but a set of events either controlled externally or internally. All above events are external events as the top portion of your page is reached on your screen or on your printer and is no way connected to your program. **Internal events** are those events, which are controlled inside the program i.e., either **by if – endif** statement or any other decision-making statement.

A sample report program without any events is as follows:

```
Report zdemo1.
Tables: sflight.
Select-options carrid1 for sflight-carrid.
* For accepting input from user.
Write: 'This is my first report program'.
Write: / 10 'carrier id', 20 'connection id' 30 'flight date'.
* These two write statements are for writing heading and column headings.
Select * from sflight where carrid in carrid1.

Write: / 10 sflight-carrid, 20 sflight-connid, 30 sflight-fldate.
Endselect.
Write: / 'page number:', sy-pagno.
```

In this case we are writing list headings for report. But these list headings are applicable only for the first page. If your list is spilling over multiple pages, then these list headings are not applicable to other pages. Again in this program, we write page number after all the data is displayed. This is fine as long as you have one page, but the moment your list spills over multiple pages, the page number will be displayed only after all the records are displayed. So in order to have some data like company name or list headings or page number or total of some number field on each page, you need to make use of events.

CLASSICAL REPORTS

Events in Classical report

Events associated with classical report are as follows and each one will be discussed in detail.

- INITIALIZATION
- AT SELECTION-SCREEN
- AT SELECTION-SCREEN ON <field>
- START-OF-SELECTION
- TOP-OF-PAGE
- END-OF-PAGE
- END-OF-SELECTION

In this case first three events are associated with selection screen. Rest of the events are associated with your list.

♦ INITIALIZATION

We have already seen how to fill default values for the selection criteria. But in many cases you need to calculate the value and then put it in selection criteria. For example, say, you are accepting date from user and you need to fill in the default value for lower range as sy-datum – 30 days and sy-datum for higher range. In this case you are calculating lower range and then filling the criteria. This can be done in INITIALIZATION event. Piece of code to do the above task would look like the following:

```
Tables: Sflight.
Select-options: fldate1 for sflight-fldate.
INITIALIZATION.
Data: date1 like SY-DATUM.
Date1 = sy-datum – 30.
Fldate1-low = date1.
Fldate1-high = sy-datum.
Append fldate1.
* Here appending is required because fldate1 is int' table
```

This event is triggered when you execute your program for the first time i.e., before selection screen is displayed.

♦ AT SELECTION-SCREEN

When user enters the values in the fields of selection screen and clicks on execute button, this event gets triggered. This event is basically for checking the values entered by the user for the fields of the selection screen i.e., data validity checking. This event is for entire selection screen. For example:

You are accepting carrid, connid, fldate from user and you don't want to proceed if user enters no value for carrid and fldate. Using AT SELECTION-SCREEN can do this.

```
Select-options: carrid1 for sflight-carrid,
                Connid1 for sflight-connid,
                F1date1 for sflight-f1date.

AT SELECTION-SCREEN.
If carrid1-low ne ' ' and fldate1-low = ' '.
   Error message.
Endif.
```

In this case, if both the fields are entered blank, then the user gets error message.
Basically, this event is for many fields on selection screen. Usually, it is for the fields which are logically related.

- **AT SELECTION-SCREEN ON <field>**

When you want to check for specific value of a field. For example, carrid should be in the range of 'LH' and 'SQ'. This can be done in this event. Basically, this event is for checking individual fields. You can have many AT selection-screen events in your program (i.e., for each field specified in the Select-Options).

```
Select-Options carrid1 for sflight-carrid.
AT SELECTION-SCREEN.
If carrid1-low ne 'LH' and carrid1-high ne 'SQ'.
  Error message.
Endif.
```

Here the system will not proceed on entering wrong values.

- **START-OF-SELECTION**

This is the first event for your list. Once all the events are triggered for selection screen, the data is retrieved from database. Data declaration, select statements are done in this event. Consider the following example:

```
START-OF-SELECTION.
Data:  mtype i.
Tables: sflight.
Select * from sflight where carrid = 'LH'.
  Write: / sflight-carrid,sflight-connid.
Endselect.
```

- **TOP-OF-PAGE**

This event is triggered with first WRITE statement or whenever new page is triggered. Advantage of using this event is that, whatever you write under this event, is applicable to all the pages. If you don't have any write statement before TOP-OF-PAGE or in START-OF-SELECTION, this event is not triggered at all. For example, if you want the name of company and column headers for all the pages, it can be written in this event.

TOP-OF-PAGE
Write: / 'INTELLIGROUP ASIA PVT. LTD.'
Write : / 10 'carrid', 20 'connid', 30 'fldate'.

- **END-OF-PAGE**

This event is triggered at the end of page.

End-of-page.
Write : / 'page number', sy-pagno.

In this case page number will be written on every page.

Conditional triggering of EOP

Consider the following case.

REPORT ZDEMO1 line-count 15(3).
Top-of-page.
Write: 'this line is written by top-of-page event'.
Start-of-selection.
Write: 'this line is written by start-of-selection event'.
End-of-page.
Write : 'this line is written by end-of-page event'.

In this case EOP will never be triggered, as end of page is never reached. The total Line-count defined for page = 15 in which 3 lines are for footer area. The output of the above code will be

This line is written by top of page event.
This line is written by start of selection event.

In output screen, only two lines are written and cursor remains still on 3rd line, the end-of-page event is not triggered. To trigger end of page event, cursor should reach at the last position, in this case on 11th line.

Such cases are quite common, and could be overcome by conditional triggering of end of page.

Sy-linct is the system variable, which gives total line count of a list.
Sy-linno is the system variable, which gives the current line number where the cursor is placed on the list.

Consider the following case:

```
Report zdemo1 line count 20(1).
Start-of-selection.
Data: m type i.
Write: / 'this is first line'.
Do 5 times.
Write: / 'the number is', sy-index.
Enddo.
M = sy-linct, sy-linno – 1.
Skip x.
End-of-page.
Write: / 'page end'.
```

The output of above example is as follows :
This is first line.
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5

After skipping 10 lines
Page end

In this case, with all write statement, you don't reach to the end of page. After all write statement, m is calculated in this case:

M = 20 – 8 – 1, So m is 12. And 11 lines are skipped after write statement and end of page is reached. (In this case you have 6 write statement so 6 lines + 1 line for page number and 1 horizontal line which is displayed for any list. So cursor is on 8th line and is subtracted from total line count i.e, 20.)

Common errors that user commits

Stating of another event denotes the end of any event. If you forget to mention the next event then everything is included in the same event. Consider the following case:

```
AT SELECTION-SCREEN.
If carrid1-low ne ' '.
Err. or message.
Endif.
```

```
Write: / 'INTELLIGROUP ASIA P. LTD.'
WRITE: / 10 'CARRID', 20 'CONNID', 30 'FLDATE'.
START-OF-SELECTION.
....
....
....
```

In this case all the write statement are included in the `at selection screen' as top-of-page is not specified. The end of `at selection-screen' is denoted by the starting of start-of-selection.

Though the sequence of events in program is immaterial, it is a good programming practice to write all the events in the order specified above.

Using Variants with selection criteria

In many cases you need report to execute report at regular interval for certain fixed values of selection criteria. That means each times you execute the report you need to enter its values again and again. ABAP/4 provides the facility by which you can define the values for selection screen and store it. Using VARIANTS can do this. It can be defined as group of values used for selection criteria while executing report. For a particular report, you create a variant which means variant created for particular report cannot be used for another report. The group of values for the selection criteria is saved and assigned a variant name. So every time you call a report, you need not specify the values for selection criteria but instead call the variant thus avoiding extra typing. User can have many variants for a single report. Each of them can be used as different type of information. For example, if a manager wants to see how an employee in personnel department or admin department has performed. He need not enter the department, one has to just execute the report with variant. In case he doesn't know about the variant, which is available, he can display list of variants attached to the report and values assigned to each variant.

Creating variant

- Execute the report program. The selection screen is displayed.
- Enter the values for selection screen and click on saves.
  - System displays the variant screen
- Enter the variant name and description for it.
- Save it.

Usually the variants are useful when you need to execute the report in background, which will be discussed in background processing.

.

INTERACTIVE REPORTS

About interactive report

A classical report consists of one program that creates a single list. This means that when the list is displayed, it has to contain all the requested data, regardless of the number of details the user want to see. This procedure may result in extensive and cluttered lists from which the user has to pick the relevant data. The desired selections must be made before hand and the report must provide detailed information.

This is not possible using the classical report and for this ABAP/4 has provided reporting feature called **INTERACTIVE REPORT.** The list produced by classical report doesn't allow user to interact with the system but the list produced by interactive report allows the user to interact with the system i.e., user can tell the system, that he needs further information. Depending upon what the user tells the system, the action is taken. Interactive reporting thus reduces information retrieval to the data actually required.

Interactive reporting allows the user to participate in retrieving and presenting data at each level during the session. Instead of presenting one extensive and detailed list with cluttered information, with interactive reporting you can create a condensed basic list from which the user can call detailed information by positioning the cursor and entering commands.

Detailed information is presented in secondary lists. A secondary list may either overlay the basic list completely or appear in an additional dialog window on the same screen. The secondary list can itself be interactive again. The basic list is not deleted when secondary list is created.

User can interact with the system by:

- Double clicking or pressing F2
- Selecting menu option

Like classical report, the interactive report is also event driven. Both the action mentioned above trigger events and code is written to handle these events. The events triggered by this action are as follows:

- At line-selection
- At user-command

- Top-of-Page During Line-Selection for Secondary Page Header info

Interactive report consists of one BASIC list and 20 secondary list. Basic list is produced by START-OF-SELECTION event. When the user double clicks on the basic list or chooses the menu option, the secondary list is produced. All the events associated with classical report except end-of-page are applicable only to basic list.

AT LINE-SELECTION event

Double clicking is the way most users navigate through programs. Double clicking on basic list or any secondary list triggers the event AT LINE-SELECTION. SY-LSIND denotes the index of the list currently created. For BASIC list it is always 0. Following piece of code shows how to handle the event.

Start-of-selection.
Write: / 'this is basic list'.
At line-selection.
Write : 'this is first secondary list'.

In this case the output will be displayed on basic list i.e.
This is basic list.

When user double clicks on this line, the event at line-selection gets triggered and secondary list is produced, i.e.

This is first secondary list.

You can go back to basic list by clicking on F3 or back icon on the standard tool bar. For this list, the value of sy-lsind will be 1.

HIDE technique

In this case thins are much simpler. Consider the case, wherein you display fields from table sflight in basic list. When user double clicks on any sflight-carrid, you are displaying the detailed information related to that particular carrid on secondary list. Hence there is a need to store the clicked carrid in some variable. So that you can access this carrid for next list. ABAP/4 has facility; a statement called **HIDE**, which provides the above functionality.

HIDE command temporarily stores the content of clicked field in system area.

Syntax:

HIDE <FIELDS>.

This statement stores the contents of variable <f> in relation to the current output line (system field SY-LINNO) internally in the so-called HIDE area. The variable <f> must not necessarily appear on the current line.

You have to place the HIDE statement always directly after the output statement i.e., WRITE for the variable <f>. As when you hide the variable, control is passed to next record. While writing, **WRITE statement takes that record from header and writes it on to the list, but when writing onto your interactive list you will miss out 1st record.**

To hide several variables, use chain HIDE statement.

As soon as the user selects a line for which you stored HIDE fields, the system fills the variables in the program with the values stored. A line can be selected.

- By an interactive event.

For each interactive event, the HIDE fields of the line on which the cursor is positioned during the event are filled with the stored values.

The HIDE area is a table, in which the system stores the names and values of all HIDE fields for each list and line number. As soon as they are needed, the system reads the values from the table. (Please try to find the name of this table.)

Sy-lsind indicates the index of the list and can be used to handle all the secondary lists. When the user double clicks on the line or presses F2, sy-lsind is increased by one and this new sy-lsind can be handled. For example:

```
Write: / 'this is basic list'.
• Will create a basic list.
If sy-lsind = 1.
Write: / 'this is first secondary list'.
Elseif sy-lsind = 2.
Write: / 'This is second secondary list'.
Endif.
```

When this code is executed,

- Basic list is produced.
- When the user clicks on the basic list, sy-lsind becomes one.
- AT LINE-SELECTION event is triggered.
- Whatever is written under IF Sy-lsind = 1, gets executed.
- Secondary list is produced.
- Again if user clicks on this list, sy-lsind becomes two.
- AT LINE-SELECTION gets triggered.
- Code written under IF Sy-lsind = 2, gets executed.

A sample program for AT LINE-SELECTION.

User interface

An interactive report starts with basic list where condensed information is stored on basic list and detailed information is stored on secondary list. To implement this kind of reporting, you need to provide the user with things like menu, icons, function keys. You also need to write code, which must react, to the user's action.

For this, you need to create the interface, which interacts with the user.

The user interface is independent of the program or list or screen. However both interface and list can be associated by means of GUI status. A GUI status groups together the interface components.

- Menu bar
- Application tool bar
- Function keys
- Title bar

The last element of the user interface is independent of the GUI status i.e., titlebar.

To assign status to your list the statement SET PR-STATUS.

Some facts about GUI status are:

- A program can have multiple GUI status and titles for different lists.
- Multiple lists can be assigned to same GUI status.
- Normally both GUI status and title go together.

Function code

An important concept, when working with user interface. Function is a four character alphanumeric code, which the system stores in the system variable called SY-UCOMM for each function key, push button, or menu option. Whenever any push button is clicked or menu option is selected, the code attached to that, is stored in SY-UCOMM and can be handled in your program.

Menu painter

Menu painter is the ABAP/4 workbench tool for creating and maintaining user interface.

Starting Menu Painter

ABAP/4 development workbench → menu painter
Or
Transaction SE41 in the command field.
Or
Through program SET PF-STATUS <var>

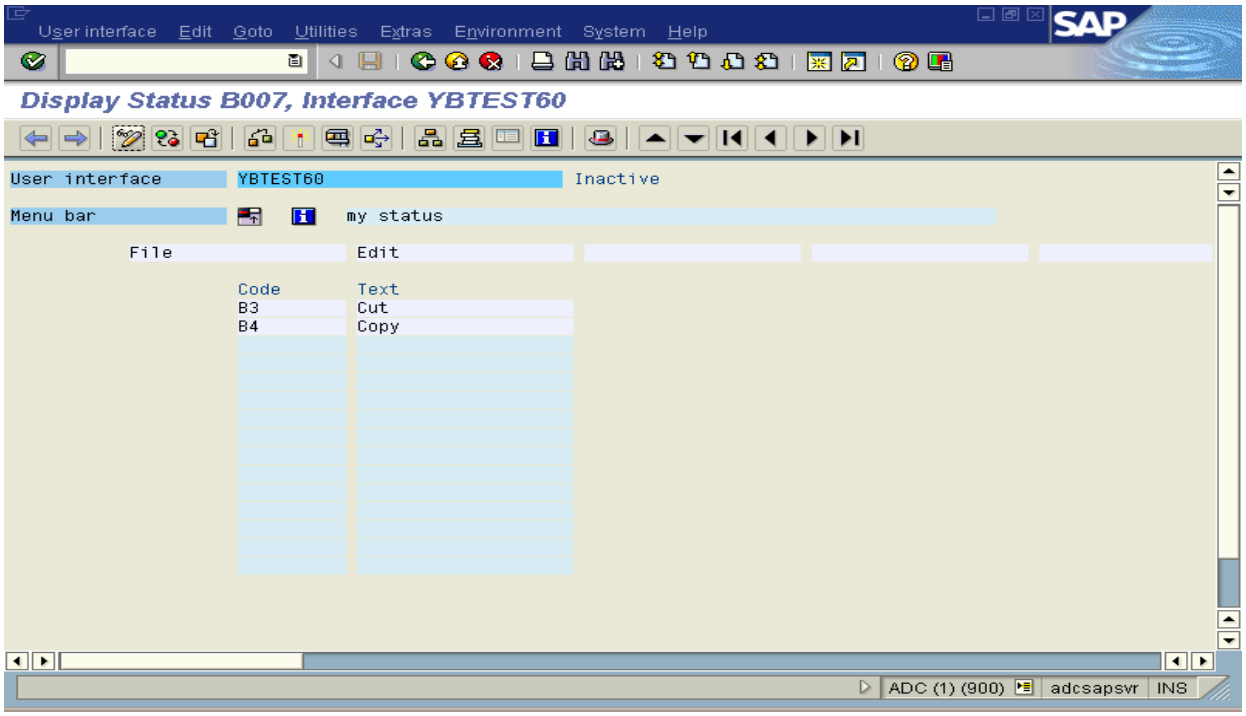
If you double click on the variable, the system takes you to the menu painter screen.

Creating Menu bar

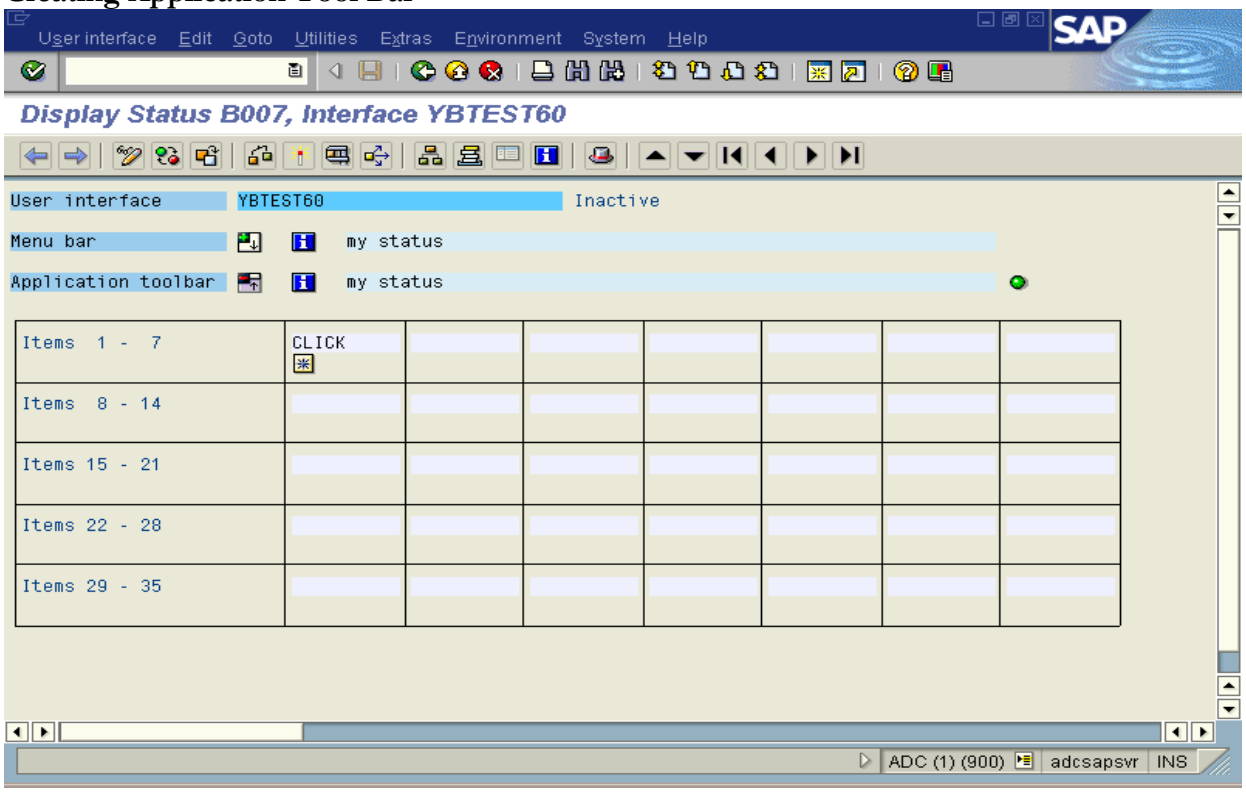
Steps involved are as follows:

- Enter the name in the first field. It is just a name given to the menu and is not displayed anywhere in the output.
- Enter the name of each menu item. You can create up to six menus (total eight menu items are available, out of which *system* and *help* are mandatory).
- Enter name of the menu items and function code. You can have fifteen menu items under one menu.

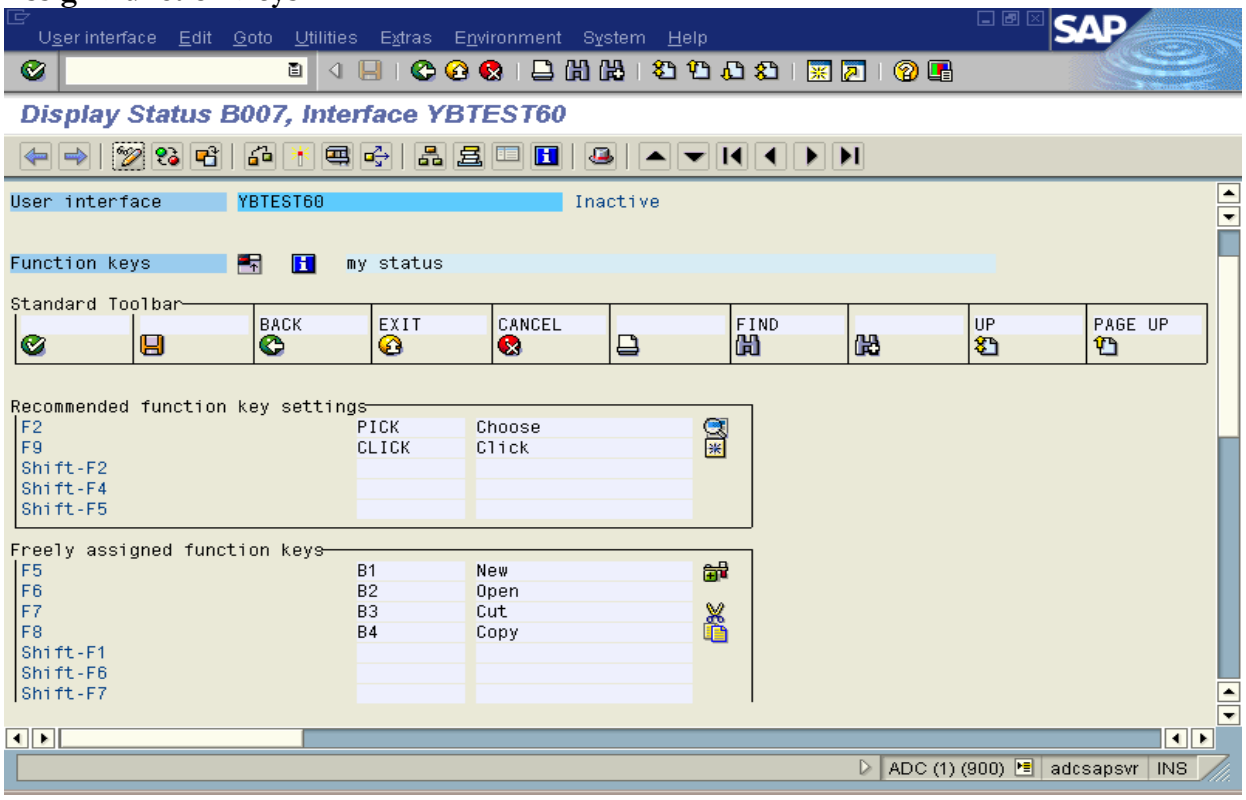
If you leave function blank, the system assumes that this particular menu item will have submenu. You can create the sub menu items under this menu item. User can go up to three levels.



Creating Application Tool Bar



Assign Function Keys



In Application tool bar you can include icon assigned for function keys.
Procedure:

- Select function key.
- From the menu, more utilities - → change text type. The system displays a dialog box, click on icon and presses ENTER.
- Select icon from list of icons displayed

Creating GUI title

From your program, you can set title for your list and SET TITLEBAR is used.

Syntax:

SET TITLEBAR <var>.

Here var can be any three-character name. When developer double clicks on the var, system displays the dialog box in which you enter the title number, the description, and the actual text for title.

Similar to dictionary objects, the GUI status must be generated to be accessible by program.

AT USER-COMMAND

When the user selects the menu item or presses any function key, the event that is triggered is AT USER-COMMAND, and can be handled in the program by writing code for the same. The system variable SY-UCOMM stores the function code for the clicked menu item or for the function key and the same can be checked in the program. Sample code would look like

AT USER-COMMAND.

Case sy-ucomm.
When 'DISP'.
Select * from sflight.
Write sflight-carrid, sflight-connid.
Endselect.

When 'EXIT'.
LEAVE.

If GUI status, suppose you have set menu bar for two items and the function code is 'DISP' and 'EXIT' respectively. If the user clicks the menu item 'DISPLAY', then function code 'DISP' is stored in the sy-ucomm and whatever is written under the when 'DISP', gets executed. This is applicable for EXIT as well.

Sy-lsind for the screen increases when the user clicks the menu item.

Usually you have combination of all the three navigations in your user interface, i.e., you have to create menu bar, assign function code for the function keys and write code to handle all this in addition to handling double clicking.

Things to remember while using all the combinations:

- Sy-lsind increases even if you select menu-item.
- When the user double clicks on particular line, value of sy-ucomm is 'PICK.
- If you set sy-lsind = 2 for your 4th secondary list, when control is transferred to the 2nd secondary list, all the other lists after 2nd are lost or memory allocated to them is lost.
- Sy-lisel also gives you the value of clicked line but in this case you cannot differentiate between field. To retrieve the exact field, you have to know the field length of each field.
- If you use statement SY-LSIND = 1.

The system reacts to a manipulation of SY-LSIND only at the end of an event, directly before displaying the secondary list. So, if within the processing block, you use statements whose INDEX options access the list with the index SY-LSIND, make sure that you manipulate the SY-LSIND field only after processing these statements. The best way is to have it always at the `as the last statement' of the processing block.

Important system fields for reports

Sy-linct - Gives total line numbers for a page.
Sy-linno - Gives current line number on the list
Sy-lsind - Index of the lists created in interactive report
Sy-listi - Index of the list from where event was triggered, usually previous list
Sy-lilli - Line number of a list from where event was triggered
Sy-lisel - Contents of a line from where event was triggered
Sy-ucomm - Holds the function code of clicked menu item or function key

EXERCISES

INTERNAL TABLES and REPORTS

1 Create an internal table taking all the fields from BKPF and display fields Company code, Document number, Document type and date of document.

2 Create an internal table taking fields' Company code, Document number, Account type and Tax code from table BSEG and display the same with column headings.

3 Create an internal table with following fields:
Sales Document and Material from table VBAP.
Date, Name of the user and sales document type from table VBAK and
Price group and customer group from table VBKD.
Sort the table according to Material number and display the contents.

4 Create an internal table called T_BSIS having a similar structure as table BSIS. Explore all possible methods to create the internal table with header line / without header line. (use data types, data … begin of ….end of, data …. data …. Include structure … etc.)
Also create a field string F_BSIS. Populate the internal code and display contents of BSIS. Sort the table according to company code and display contents.

5 Determine for each material type (MTART) the 5 table entries with the highest gross weight (as a ranked list).
To do this, read the table MARA and store the material type (MTART), material number (MATNR), unit of measure (MEINS) and gross weight (BRGEW) into an internal table.
Allow the user to specify the Material type as a parameter on the selection screen.

6 Create a list of the maximum number of available seats for each carrier. To do this, read the table SFLIGHT and store the airline carrier id (CARRID) and maximum number of seats (SEATSMAX) in an internal table. Determine the total number of seats for each airline carrier when filing the int' table.

7 Read the table SFLIGHT into an internal table and then output the internal table with the fields CARRID, FLDATE and PRICE.
Delete all the internal table entries where the airline carrier (CARRID) is not equal to LH. Read the internal table with entry with the key CARRID = LH and CONNID = 0400, multiply the price by 3 and write the modified entry back to the internal table. Then Output the internal table.

8 Read table TABNA into internal table and output the fields **Country, Id, Name1, and Sales. Sort the table with Country**.
Delete all internal table lines with sales lower than 50,000.

Read internal table with key 'GB' and '00000003' and multiply the sales by 3 and change table entry.
Insert any one record of your choice.
Find out how many lines are there in the internal table.
Remove all the contents of the table.
De-allocate the memory associated with table.

9 Use tables LFA1, LFB1 and LFM1.
Define an internal table with the following:
Lifnr like Lfa1-lifnr,
Bukrs like Lfb1-bukrs,
Ekorg like Lfm1-ekorg.
Add data from these table into the internal tables.
Sort the internal table Lifnr.
Read the internal table with Lifnr = 'A5' and change name to trainee.
Put back the record into the table.
Delete first three records of internal table.
Clear header for internal table each time you access a record.

10 Create a report, which will give the existing stock for a material. The report should have subtotal of the stock for each storage location and Grand total of the stock at the end of the plant.

Plant data should start at new page.
Input: Selection screen which will allow one to select a range of materials.
Materials:
Output Report format as follows:

Plant	Storage location	Material number	Description	Stock (unrestricted)

Grand total ******

Tables and fields:

Material Number	MARA-MATNR
Description	MAKT-MAKTX
Plant	MARC-WERKS
Storage location	MARD-LGORT
Stock	MARD-LABST

Use Standard Formatting colors.
Exception handling:
Error message " Material not found " – if Material not present.

11 Generate a report to list the following details from the tables LFA1 & LFB1 vendor no., vendor name, city, state, telephone no., fax no., company code and terms of payment. User shall have options to select the vendor number range for which the report is generated.

.

Report title: Vendor Master Listing 2	
Vendor Name: _____	Vendor no. : _____
Address: _____	
Telephone no. _____	Fax no : _____
Company code	Terms of payments
_____	_____

All text used in the report shall be generated using text elements only.
The output list shall have a footer showing page no., date and Intelligroup'(left corner)

12 Generate a report for displaying material description, plant & storage data.
Input: Material number (MARA-MATNR)
Data to be displayed for the following three materials only.

1. BP770M15
2. FP56790031
3. FP28652011

Output format:

11/11/1998 Report to analyze material stocks

Material No.	Description		
FP96412101	Meditech Oatmeal 60'	Patterned	Weld Rod
Plant	Storage Loc.	Unrestricted stock	
xxxx	xxxxxxxxxxx	xxxxxxxxxxxxxxx	

All texts are to be generated using Text Elements only.
Use tables: MARA, MAKT, MARC and MARD.

SUBROUTINES

1. Read a number between 0 and 100 and another digit between 0 and 9. Write a subroutine that will calculate the sum of all numbers (below the limit) that end with the digit. The parameters to be passed are limit and digit both by value and sum by reference.

   Ex. If limit = 67 and digit = 4 then sum should be the sum of 4,14,24,34, ..64.

2. Write a subroutine CENTRE-STRING, which will output a string on the center of a line. The subroutine will accept parameters STRING pass by value.

3. Write a program extensively using subroutines to print the equivalent number in words.
   For example: if the number is 66, the output should be SIXTY SIX.
   Limit the number range from 0 – 99.
   Accept the input number as a parameter.

4. Accept a date from the user.
   Write a date as dd-mm-yyyy where mm is month written as JAN/FEB/Mar…etc.
   Make use of subroutines.

5 For each flight connection, calculate the sales for all flights of an airline carrier. Use internal table for calculating the sales. Use a subroutine for the output by passing the internal table as the parameter.