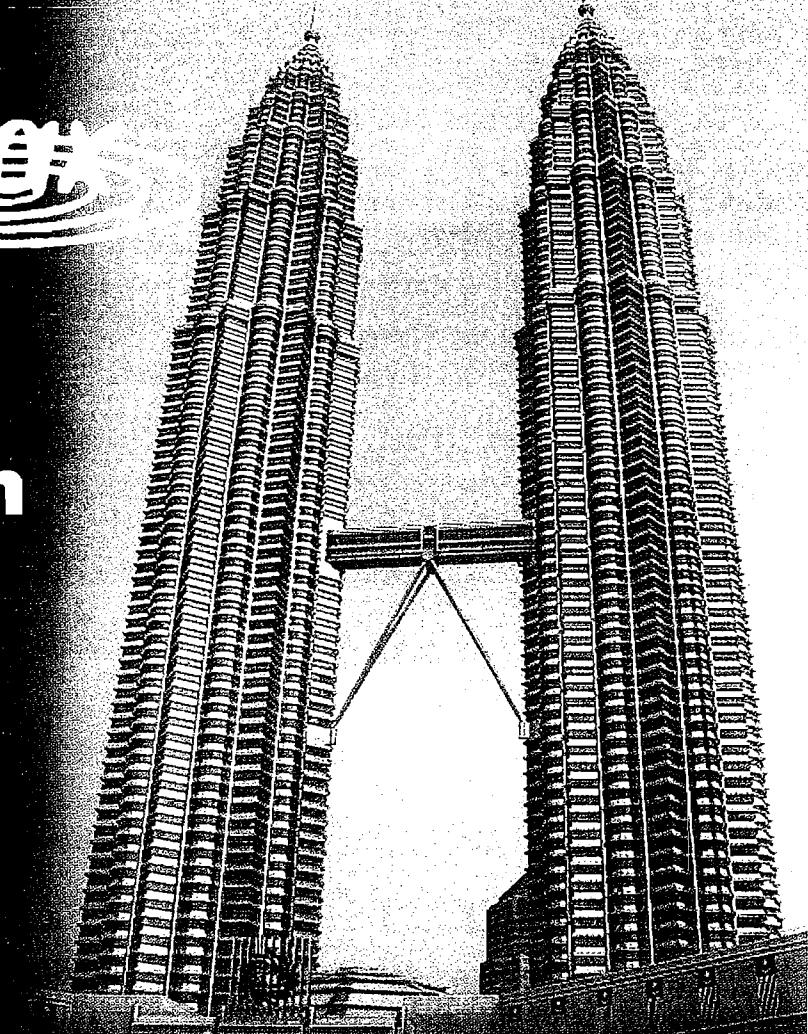


SAP@e *Meg*

**Helps to Reach
The Heights
in SAP**



ABAP/4

The Complete Reference

VOLUME -II

Ganapati Adimulam

About the Author

Ganapati Adimulam is an SAP Technical Consultant. Having nearly four years of Experience in the SAP Domain, several successful implementation, and a background in training and technical documentation has afforded him many opportunities to observe all aspects of SAP implementation.

Mr. Ganapati did his M.C.A in 2002. He started his SAP career in 2003 March. He has worked for corporate giants like CGEY (CapGemini Ernest & Young), Infosys and Satyam Computers and also for many other international clients as both onsite and off-shore Technical consultant.

As his basic interest is in SAP training domain, he has turned into SAP training services after relieving from the corporate biggies. He is presently offering excellent training services in SAP through **eMAX Technologies**. He is also a cynosure for imparting SAP training to the corporate consultants. He is a big talk among the corporate as a very good SAP corporate trainer.

From eMAX Technologies.

ACKNOWLEDGEMENTS

Doing a book of this magnitude requires a team effort.

Every task is completely successful if and only if we acknowledge the efforts contributed by the individual personnel.

We acknowledge our sincere thanks to our Honorable Director, **eMax Technologies, Mr. Rama Rao** for his continuous guidance, support and co-operation through this project.

I want to take this opportunity to thank the entire management, beloved students and friends and several other individuals of **eMax Technologies** in reviewing and assuring the quality checks of the accomplishments at every stage of this project.

Finally I would like to extend my sincere thanks to my family members from the bottom of my heart.

Ganapati Adimulam.

INDEX

1. BDC Programming 22, 72

2. SAP SCRIPT 81

3. MODULE Programming 60

4. LSMW 27

5. working with Files 29

1. BDC Programming

1. Introduction to BDC Programming

2. Call Transaction Method

a. Processing Errors

3. Session Method

a. For Single transaction

b. For Multiple Transactions in One Session

4. Handling Table Control in BDC

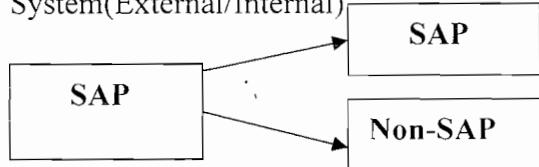
5. Direct Input

BDC (Batch Data Communication)

It is One of the Data Transfer Method.

Data transfer methods are divided as Outbound and Inbound.

Outbound: Transfer the data from the Source system to Target System(External/Internal)



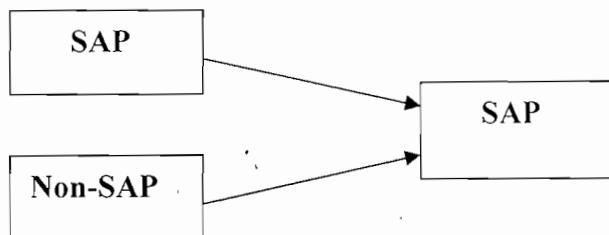
Outbound is a data transfer from SAP to SAP / Non SAP.

Note: Source System is always SAP.

Inbound: Data transfer from External System into SAP.

i.e. Inbound is a technique to transfer the Data from SAP/Non-SAP into SAP .

Note: Target System is always SAP.



Note: BDC is an Inbound Data transfer Technique.

About Data Transfer in R/3 System

When a company decides to implement the SAP R/3 to manage business-critical data, it

Usually does not start from a **no-data situation**. Normally, a SAP R/3 project comes in to replace or complement to an existing application.

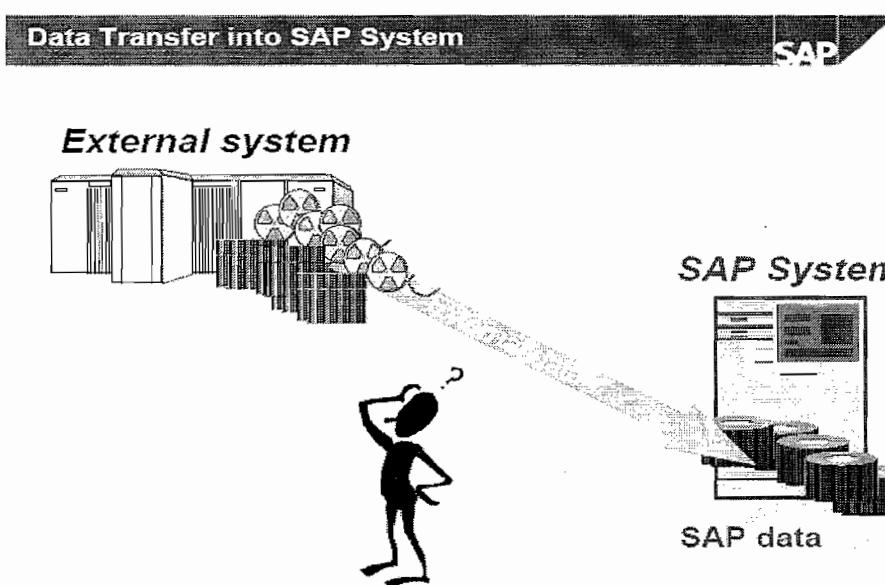
In the process of replacing current application and transferring application data, two

Situations might occur:

1. The first is when application data to be replaced is transferred at once and only once.
2. The second situation is to transfer data periodically from external system to SAP and

Vice versa.

There is a period of time when information has to be transferred from existing Application, to SAP R/3, and **often this process will be repetitive.**



For reasons of efficiency, large volumes of data cannot be transferred manually from an external system into the R/3 System. A data transfer is required that transfers the data automatically in the Background.

The SAP system offers Three methods for transferring data into SAP Systems

From non-SAP Systems or legacy system.

1. SESSION METHOD
2. CALL TRANSACTION
3. DIRECT INPUT.

Note: Both Session Method and Call Transaction, We call BACH INPUT Methods.

Where as Direct Inputs are standard programs to post the data into SAP.

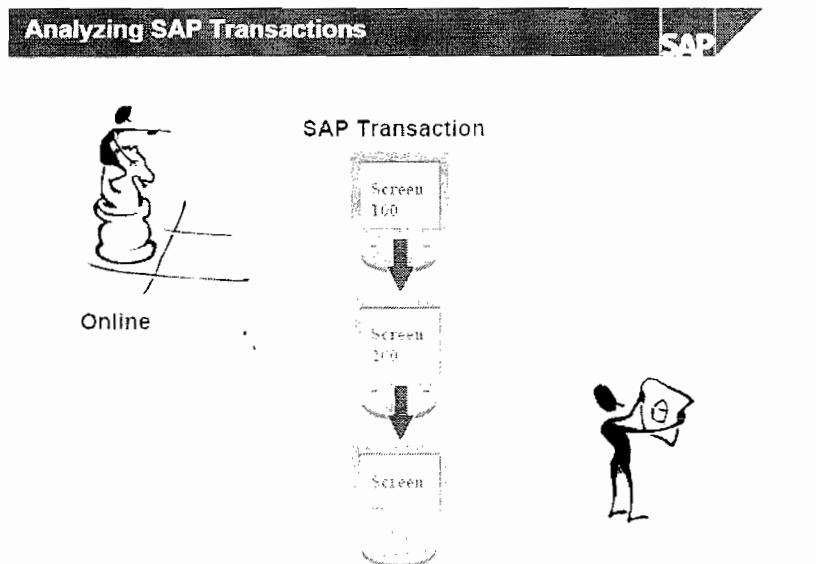
Advantages offered by BATCH INPUT method:

1. Can process large data volumes in batch
2. Can be planned and submitted in the background
3. No manual interaction is required when data is transferred.

Note : To implement one of the supported data transfers, **you must often write the program** that Exports the data from your non-SAP/SAP System. This program, known as a “**data transfer**” program must map the data from the external system into the data structure required by the SAP batch input program.

Writing a Data Transfer Program involves following prerequisites:

1. Analyzing transaction



- Analyzing transaction Involves following steps.
- Which fields require input i.e., mandatory
- Which fields you can allow defaulting to standard values.
- The names. Types and lengths of the **fields** that are used by a transaction.
- Check whether some fields will need conversion of their data types and/or data lengths. Most of the data from the external system must be converted into SAP format. We call this “**formatting**” the data.
- **Screen number** and name of **module pool program** behind that particular transaction

To analyze a transaction, do the following

- Start the transaction by menu or by entering the transaction code in the command box
- Step through the transaction, by entering the data will be required for processing your batch input data.
- On each screen, note the **program name** and **screen (dynpro) number**. (dynpro = dyn + pro. Dyn = screen. Pro = number)

- you can get the program name and screen number by **pressing F1** on any field or button on the screen. the **Technical info** pop-up shows not only the field information but also the **program and screen**.

Note : Repeat the Procedure of finding the Program Name, Screen no, Field Name and their data type and length for each field on each screen for all the fields.

Note: At the end of analyzing the transaction code, we should have all the screen and fields Details So that we can write the **DATA TRANSFER program**.

Note : Since it is very difficult to collect all the screen and field details by pressing F1 -> Technical Information on each field when we have multiple screens and more fields , So SAP provided one transaction SHDB to record all the Screen and Field Details along with the Function Codes

The SHDB transaction ,records all the screen, field and OK_Code details for any transaction.

2.Declaring internal tables.

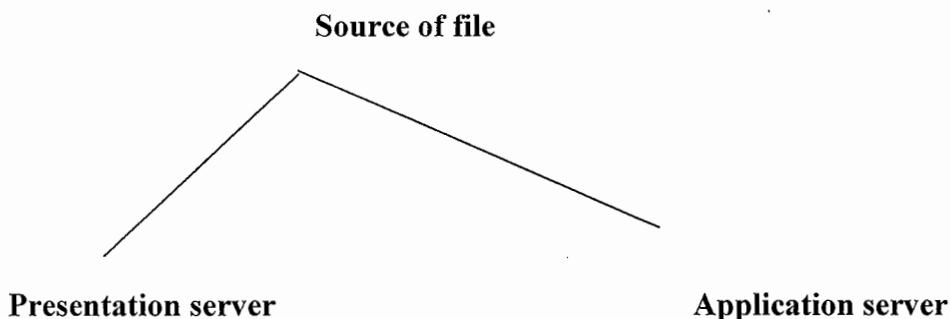
First internal table similar to structure like Input Flat file(**IT_DATA**)

Second internal table(**IT_BDCDATA**) like DDIC Structure BDCDATA.
(to maintain the screen and field details).

Third internal table(**IT_BDCMSGCOLL**) like DDIC Structure BDCMSGCOLL.

(This Internal table is required Only we work with Call Transaction to process the Errors in call transaction).

3. Transferring data from flat file to internal table(IT_DATA).
This depends on the source of the File .



| | |
|---|---|
| Call function module: 'GUI_UPLOAD' | Reading from file into <itab> |
| Input | Step 1 : open file. |
| Filename = 'file name + path' record | Step 2 : Read record by |
| Output | from file append to <itab> = 'internal table to store the data from file' <itab>. |

Step 3 : close file.

4. Population of BDCDATA into the Internal Table(IT_BDCDATA)
And Process the transaction using the IT_BDCDATA For Each record in IT_DATA.

i.e For each record in IT_DATA(Loop at IT_DATA).

A) At the beginning of each new screen, we must maintain the module pool name <program>, the screen number <dynpro> and a flag <dynbegin> :

```
WA_BDCDATA-PROGRAM = <program>.  
WA_BDCDATA -DYNPRO = <dynpro>.  
WA_BDCDATA -DYNBEGIN = 'X'.  
APPEND WA_BDCDATA TO IT_BDCDATA.  
CLEAR WA_BDCDATA.
```

B) For each field to which you want to assign values, insert an entry in the internal table **IT_BDCDATA**. Specify the technical field name <fnam> and the field content <fval> :

```
WA_BDCDATA-FNAM = <fnam>.  
WA_BDCDATA-FVAL = <fval>.  
APPEND WA_BDCDATA TO IT_BDCDATA.  
CLEAR WA_BDCDATA.
```

Now specify which action is to be executed in this screen. You must determine the triggered function code <fcode> and assign this with the field **FVAL**. Note that the character '/' should always be placed before the function key number. The character '=' must be placed before all other function codes.
Assign value **BDC_OKCODE** to the field **FNAM**:

WA_BDCDATA-FNAM = 'BDC_OKCODE'.

```
WA_BDCDATA -FVAL = <fcode>.  
APPEND WA_BDCDATA TO IT_BDCDATA.  
CLEAR WA_BDCDATA.
```

NOTE : Execute steps **A and B** for each additional screen in the transaction.

After the last screen in the transaction, internal table **IT_BDCDATA** is filled with all of the values required to process One record Completely .

IF Session Method ,

Insert the **IT_BDCDATA** into the session by calling the Function Module **BDC_INSERT**.

ELSEIF CALL TRANSACTION METHOD.

```
CALL TRANSACTION <tcode> USING IT_BDCDATA  
MESSAGES INTO IT_BDCMSGCOLL.
```

ENDLOOP. (FOR IT_DATA).

CALL TRANSACTION:

In the second method, the Conversion program uses the ABAP statement **CALL TRANSACTION USING** to run an SAP transaction. External data does not have to be deposited in a session for later processing. Instead, the entire batch input process takes place inline in your program.

Processing batch input data with **CALL TRANSACTION USING** is the faster of the two recommended data transfer methods. In this method, legacy data is processed inline in your data transfer program.

Syntax:

CALL TRANSACTION <tcode>

USING <bdc_tab>

MODE <mode>

UPDATE <update>

MESSAGES INTO <BDCMSGCOLL_TAB>.**Details :**

<tcode> : Transaction code

<bdc_tab> : Internal table of structure **BDCDATA**.

<mode> : Display mode:

| | |
|----------|----------------------------|
| A | Display all |
| E | Display errors only |
| N | No display |

DISPLAY MODE Parameter

Use the **MODE parameter** to specify whether data transfer processing should be displayed as it happens. You can choose between three modes:

A Display all. All screens and the data that goes in them appear when we run your program.

N No display. All screens are processed invisibly, regardless of whether there are errors or not. Control returns to your program as soon as transaction processing is finished.

E Display errors only. The transaction goes into display mode as soon as an error in one of the screens is detected. You can then correct the error.

<update> : Update mode:

| | |
|----------|--------------|
| S | Synchronous |
| A | Asynchronous |
| L | Local update |

The UPDATE Parameter

You use the **UPDATE parameter** to specify how updates produced by a transaction should be processed. You can select between these modes:

Asynchronous updating. In this mode, the called transaction does not wait for any updates it produces to be completed. It simply passes the updates to the SAP update service. Asynchronous processing therefore usually results in faster execution of your data transfer program.

Asynchronous processing is NOT recommended for processing any larger amount of data. This is because the called transaction receives no completion message from the update module in asynchronous updating. The calling data transfer program, in turn, cannot determine whether a called transaction ended with a successful update of the database or not.

Error analysis and recovery is less convenient than with synchronous updating.

Synchronous updating. In this mode, the called transaction waits for any updates that it produces to be completed. Execution is slower than with asynchronous updating because called transactions wait for updating to be completed. However, the called transaction is able to return any update error message that occurs to your program. It is much easier for you to analyze and recover from errors.

Local updating. If you update data locally, the update of the database will not be processed in a separate process, but in the process of the calling program.

The MESSAGES Parameter

The **MESSAGES** specification indicates that all system messages issued during a **CALL TRANSACTION USING** are written into the internal table **<MESSTAB>**. The internal table must have the structure **BDCMSGCOLL**.

Return codes:

| Value | Explanation |
|-------------|-------------------------|
| 0 | Successful |
| ≤ 1000 | Error in dialog program |
| > 1000 | Batch input error |

Error Handling in CALL TRANSACTION

When the records are uploaded in database table by Session Method error record is stored

In the log file. In Call transaction there is no such log file available and error record is

Lost unless handled. Usually you need to give report of all the error records i.e. records

Which are not inserted or updated in the database table and this can be done by following

Method.

Steps for the error handling in CALL TRANSACTION**Requirement :****LOOP AT IT_DATA.(i.e For each data record)**

Populate IT_BDCTAB table

Call transaction <tr.code> using IT_BDCDATA

Mode <A/N/E>

Update<S/A>

Messages IT_BDCMSGCOLL.

If sy-subrc ne 0. (Call transaction returns the sy-subrc if not successful
In updation).

Call function **FORMAT_MESSAGE**.

OR

Call Function **BAPI_MESSAGE_GETDETAIL**

OR

Querry the Message TEXT from the table T100 for the Messages Collected into IT_BDCMSGCOLL.

The Text in T100-TEXT would be maintained with Place Holders.

Replace the Place Holders with Variable Texts from IT_BDCMSGCOLL(MSGV1, MSGV2, MSGV3, MSGV4).

Either APPEND the error message to another Internal table IT_ERRO and Display it Finally

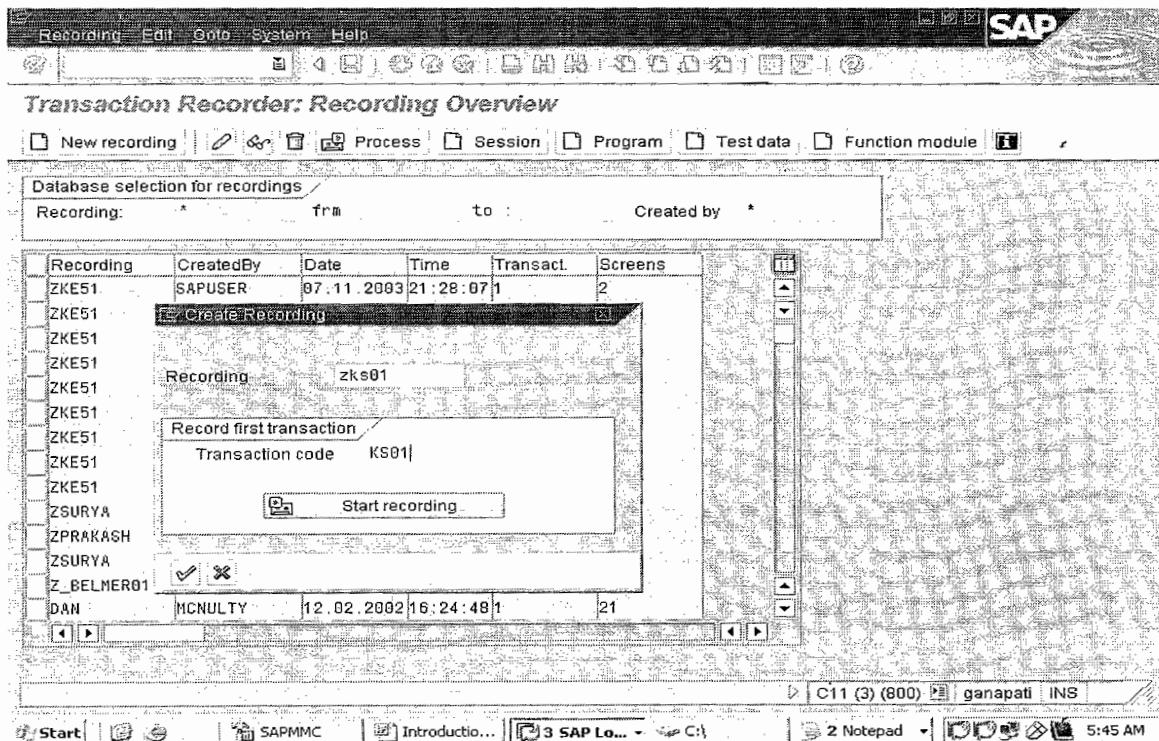
OR

Display the record and message immediately.

Endloop.

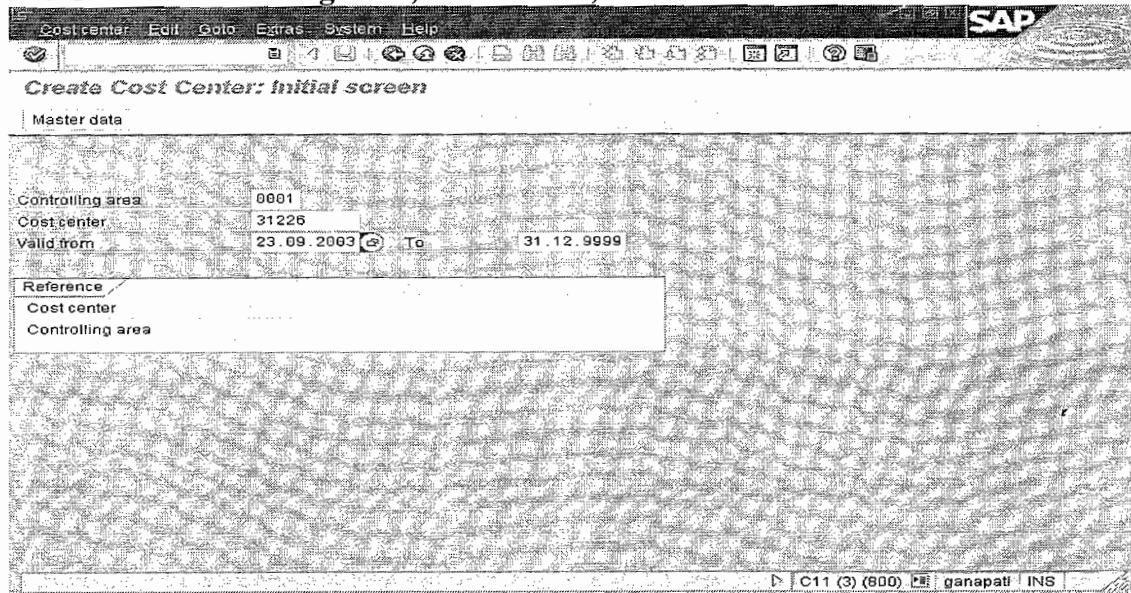
BDC (Batch Data Communication)

We Never Compromise in Quality, Would You?



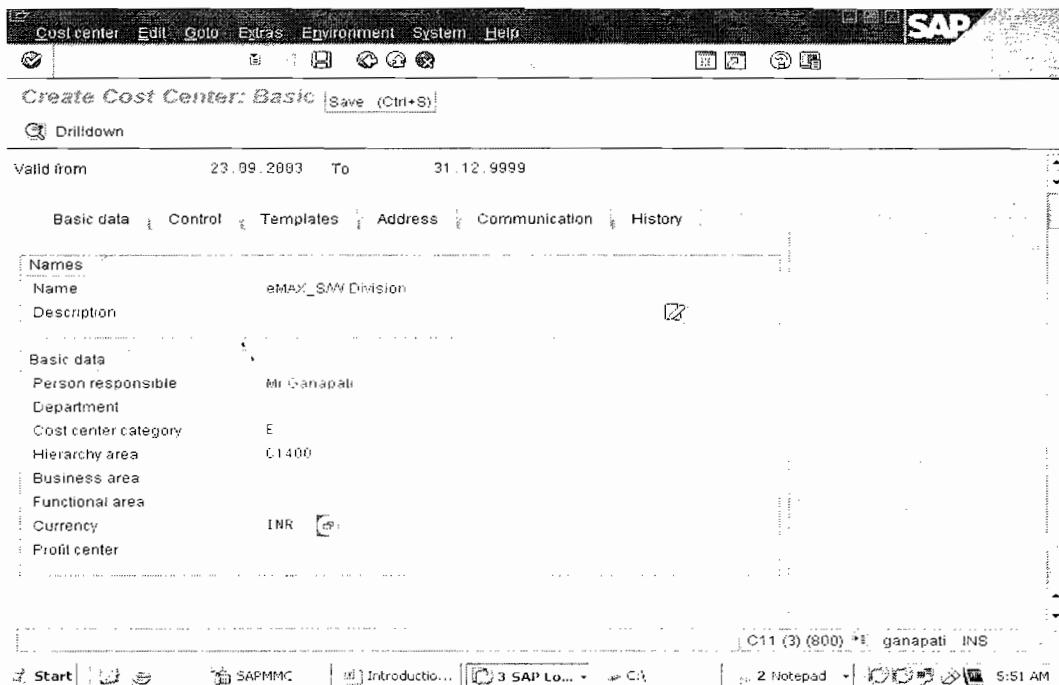
ENTER

Provide the Controlling Area, Cost Center, valid from and Valid to.



ENTER

Provide Name of the Cost Center,
Person Responsible,
Cost Center Category,

Hierarchy Area, Currency**SAVE.****Opens the recording details.**

| Line | Program | Screen | St... | Field name | Field value |
|------|----------|--------|-------|------------------|-------------------|
| 1 | | | | CSK SZ | |
| 2 | SAPLIMAT | 30200 | X | EDC_CURSOR | CSKSZ-DATAB_ANFO |
| 3 | | | | EDC_OHCODE | 000 |
| 4 | | | | CSKSZ-KOKRS | 0001 |
| 5 | | | | CSKSZ-POSTL | 31226 |
| 6 | | | | CSKSZ-DATAB_ANFO | 23.09.2003 |
| 7 | | | | CSKSZ-DATBL_ANFO | 31.12.9999 |
| 8 | | | | | |
| 9 | SAPLIMAT | 40204 | X | EDC_OHCODE | =BII |
| 10 | | | | EDC_SUBSCR | SAPLIMAT |
| 11 | | | | EDC_CURSOR | CSKSZ-WAERS |
| 12 | | | | CSKSZ-KTEXT | eMAX_S/W Division |
| 13 | | | | CSKSZ-VERAK | Mr.Ganapati |
| 14 | | | | CSKSZ-KOSAR | E |
| 15 | | | | CSKSZ-KHINR | C1400 |
| 16 | | | | | |

Line 1 16 Fr. 17

Recording was saved C11 (3) (800) *B ganapati INS

SAVE it and use in the BDC program.

Program Details:

```
*****  
* PROGRAM : ZDEMO_UPLOAD_COST_CENTER_DATA *  
* AUTHOR : GANPATI . ADIMULAM *  
* PURPOSE : BDC PROGRAM TO UPLOAD ALL THE COST *  
* CENTER MASTER DATA INTO SAP USING *  
* CALL TRANSACTION METHOD *  
* REFERENCE : NA *  
* COPIED FROM : NA *  
* TRAPORT REQUEST NO : C11D2K9001 *  
*****
```

REPORT ZDEMO_UPLOAD_COST_CENTER_DATA .

```
DATA : BEGIN OF WA_DATA,  
        KOKRS TYPE KOKRS, "CONTROLLING AREA  
        KOSTL TYPE KOSTL, "COST CENTER  
        DATAB TYPE DATAB, "START DATE  
        DATBI TYPE DATBI, "END DATE  
        KTEXT TYPE KTEXT, "NAME  
        LTEXT TYPE LTEXT, "DESCRIPTION  
        VERAK TYPE VERAK, "PERSON RESPONSIBLE  
        KOSAR TYPE KOSAR, "COST CENTER CATEGORY  
        KHINR TYPE KHINR, "HIERARCHY AREA  
        END OF WA_DATA.
```

```
DATA : IT_DATA LIKE TABLE OF WA_DATA,  
      IT_BDCDATA LIKE TABLE OF BDCDATA,  
      WA_BDCDATA LIKE LINE OF IT_BDCDATA,  
      IT_BDCMSGCOLL LIKE TABLE OF BDCMSGCOLL,  
      WA_BDCMSGCOLL LIKE LINE OF IT_BDCMSGCOLL.  
      DATA V_FILE TYPE STRING.
```

```
*CONSTANTS  
CONSTANTS : C_KS01(4) TYPE C VALUE 'KS01',  
            C_X(1)  TYPE C VALUE 'X',  
            C_A(1)  TYPE C VALUE 'A'.
```

```
SELECTION-SCREEN BEGIN OF BLOCK B1 WITH FRAME TITLE TEXT-  
000.  
PARAMETER : PA_FILE LIKE FC03TAB-PL00_FILE OBLIGATORY .  
SELECTION-SCREEN END OF BLOCK B1.
```

* AT SELECTION-SCREEN ON VALUE-REQUEST *

*EVENT TO BE TRIGGERED WHEN WE PRESS F4.

AT SELECTION-SCREEN ON VALUE-REQUEST FOR PA_FILE.

PERFORM GET_F4_FOR_FILE USING PA_FILE.

* START-OF-SELECTION. *

START-OF-SELECTION.

*WE NEED TO MOVE THE PA_FILE INTO ANOTHER VARIABLE OF TYPE STRING

*AS WE ARE GOING TO USE THE SAME IN THE FM : GUI_UPLOAD THERE THE ' '.

*FILE TYPE IS STRING.

V_FILE = PA_FILE.

*UPLOAD THE DATA FROM FLAT FILE TO <ITAB>

PERFORM UPLOAD_FILE_TO_ITAB USING V_FILE
CHANGING IT_DATA.

*FILL THE SCREEN AND FIELD DETAILS

LOOP AT IT_DATA INTO WA_DATA.

REFRESH IT_BDCDATA.

*FIST SCREEN DETAILS

PERFORM FILL_SCREEN_DETAILS USING 'SAPLKMA1' '0200' 'X'.

*CURSOR DETAILS

PERFORM FILL_FIELD_DETAILS USING 'BDC_CURSOR' 'CSKSZ-KOKRS'.

*OKCODE DETAILS

PERFORM FILL_FIELD_DETAILS USING 'BDC_OKCODE' '/00'.

**CONTROLLING AREA

PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-KOKRS' WA_DATA-KOKRS.

*COST CENTER DETAILS

PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-KOSTL' WA_DATA-KOSTL.

*START DATE

PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-DATAB_ANFO' WA_DATA-DATAB.

*END DATE

PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-DATBI_ANFO' WA_DATA-DATBI.

*NEXT SCREEN DETAILS

PERFORM FILL_SCREEN_DETAILS USING 'SAPLKMA1' '0299' 'X'.

*OKCODE DETAILS

PERFORM FILL_FIELD_DETAILS USING 'BDC_OKCODE' '=BU'.

*SUBSCR FIELD DETAILS

PERFORM FILL_FIELD_DETAILS USING 'BDC_SUBSCR' 'BDC_SUBSCR'.

*CURSOR DETAILS

PERFORM FILL_FIELD_DETAILS USING 'BDC_CURSOR' 'CSKSZ-WAERS'.

*NAME

PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-KTEXT' WA_DATA-KTEXT.

*DESCRIPTION

PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-LTEXT' WA_DATA-LTEXT.

*PERSON RESPONSIBLE

PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-VERAK' WA_DATA-VERAK.

*COST CENTER CATEGORY

PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-KOSAR' WA_DATA-KOSAR.

*HIERARCHY AREA

PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-KHINR' WA_DATA-KHINR.

*CURRENCY KEY

PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-WAERS' 'INR'.

*CALL THE TRANSACTION

CALL TRANSACTION C_KS01 USING IT_BDCDATA

 MODE C_A " A - ALL SCREENS

 " N - NO SCREENS

 " E - ERROR SCREENS ONLY

 UPDATE 'A' "Aynchronous

 " Synchronous

 MESSAGES INTO IT_BDCMSGCOLL.

ENDLOOP.

&-----

*& Form FILL_SCREEN_DETAILS

&-----

FORM FILL_SCREEN_DETAILS USING PROGRAM LIKE BDCDATA-
PROGRAM

 : DYNPRO LIKE BDCDATA-DYNPRO

 DYNBEGIN LIKE BDCDATA-DYNBEGIN.

CLEAR WA_BDCDATA.

WA_BDCDATA-PROGRAM = PROGRAM.

WA_BDCDATA-DYNPRO = DYNPRO.

WA_BDCDATA-DYNBEGIN = DYNBEGIN.

APPEND WA_BDCDATA TO IT_BDCDATA.

ENDFORM. " FILL_SCREEN_DETAILS

&-----

*& Form FILL_FIELD_DETAILS

&-----

* Fill Field Details

FORM FILL_FIELD_DETAILS USING FNAM
 FVAL.

CLEAR WA_BDCDATA.

WA_BDCDATA-FNAM = FNAM.

WA_BDCDATA-FVAL = FVAL.

APPEND WA_BDCDATA TO IT_BDCDATA.

ENDFORM.

```
*&-----*
*&   Form GET_F4_FOR_FILE
*&-----*
*   DISPLAY ALL FILES IN THE SYSTEM FOR SELECTION
*-----*
*   -->P_PA_FILE NAME OF THE FILE
*-----*
FORM GET_F4_FOR_FILE USING  P_PA_FILE.

CALL FUNCTION 'KD_GET_FILENAME_ON_F4'
  EXPORTING
    FIELD_NAME = 'PA_FILE'
  CHANGING
    FILE_NAME = PA_FILE.
IF SY-SUBRC <> 0.
  MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
    WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
ENDIF.

ENDFORM.          " GET_F4_FOR_FILE

*&-----*
*&   Form UPLOAD_FILE_TO_ITAB
*&-----*
*   SUBROUTINE TO UPLOAD THE DATA FROM PRE.SERVER FILE TO
ITAB
*-----*
*   -->P_V_FILE - FILE NAME
*   <-P_IT_DATA - INTERNAL TABLE TO STORE THE DATA
*-----*
FORM UPLOAD_FILE_TO_ITAB USING  FP_V_FILE
  CHANGING FP_IT_DATA LIKE IT_DATA.

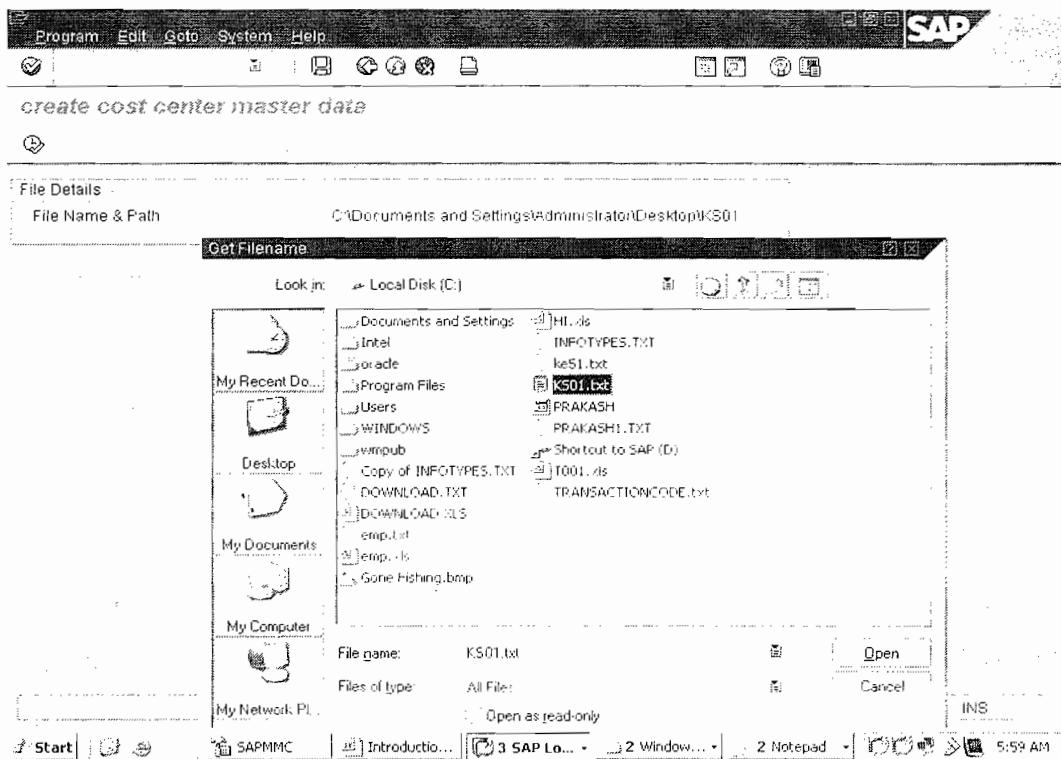
CALL FUNCTION 'GUI_UPLOAD'
  EXPORTING
    FILENAME      = FP_V_FILE
    HAS_FIELD_SEPARATOR = 'X'
  TABLES
    DATA_TAB      = FP_IT_DATA.

ENDFORM.          " UPLOAD_FILE_TO_ITAB
```

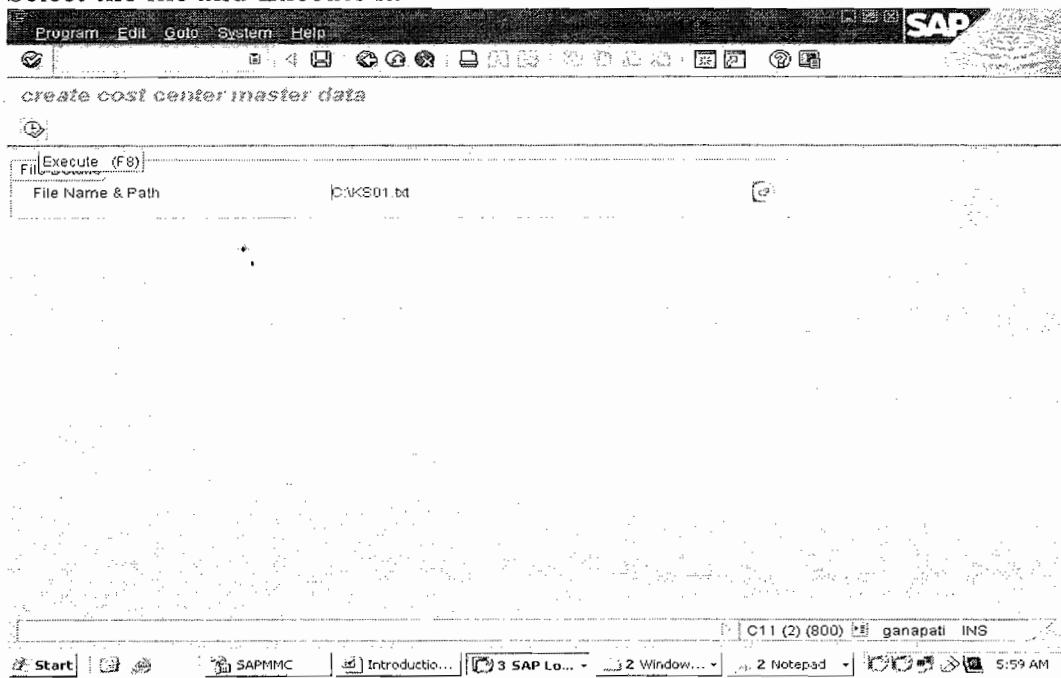
Execute the Program and Observe the Input and Output of the Program :
Step 1 : Execute the Program .

BDC (Batch Data Communication)

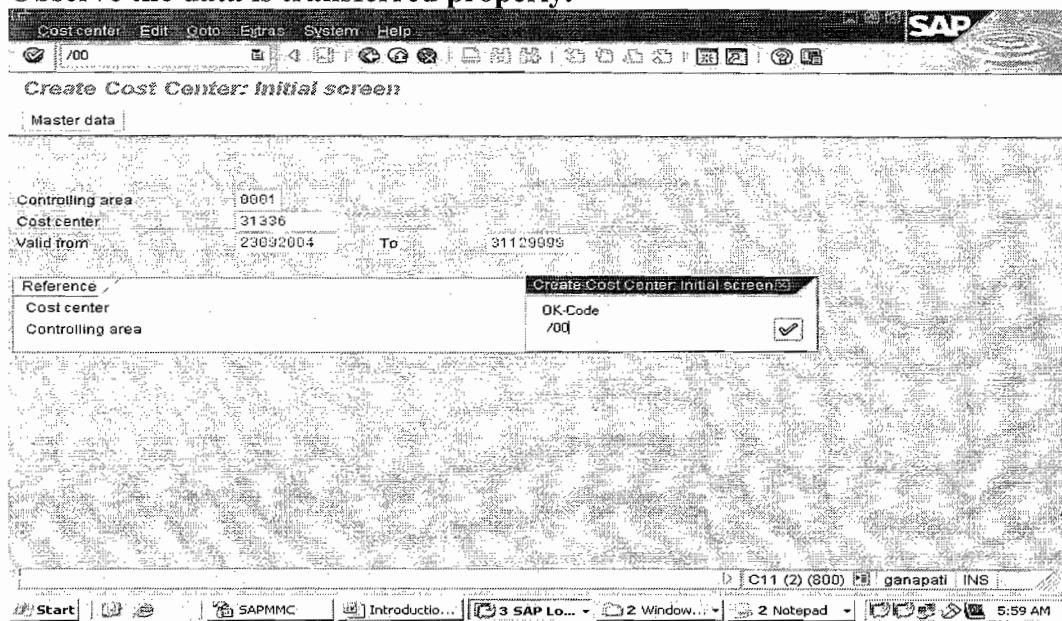
We Never Compromise in Quality, Would You?



Select the file and Execute it.

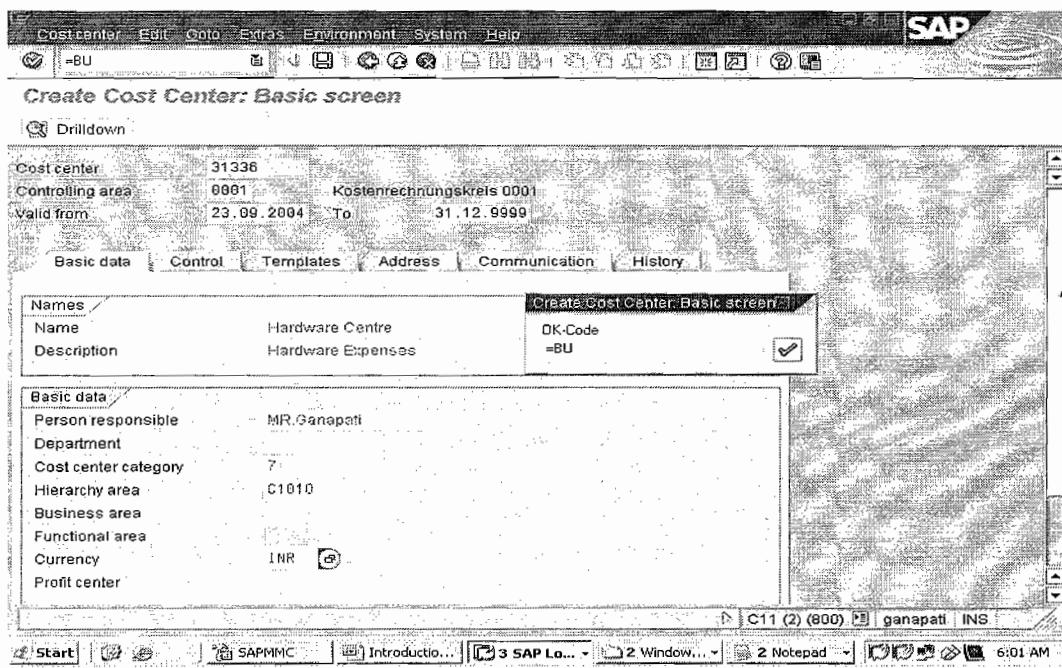


1st Screen :
Observe the data is transferred properly.



ENTER

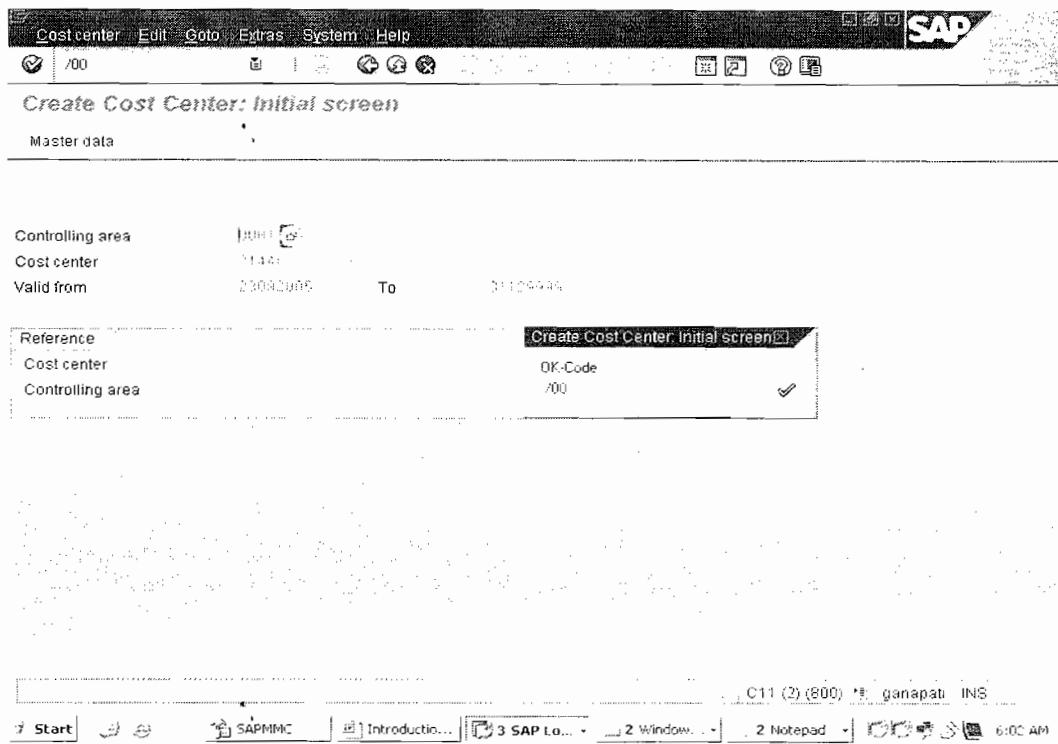
2nd Screen Details :



BDC (Batch Data Communication)

We Never Compromise in Quality, Would You?

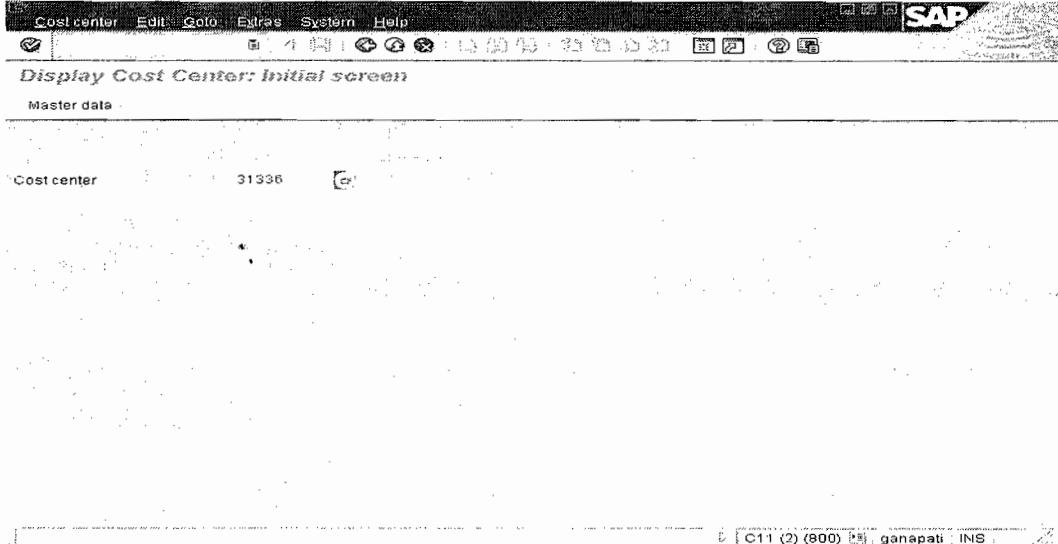
ENTER



Observe it , AGAIN it called the same screen/s for the next Cost center and it repeats the Same procedure for the rest of the records.

Check for the Successful Creation Of Records :

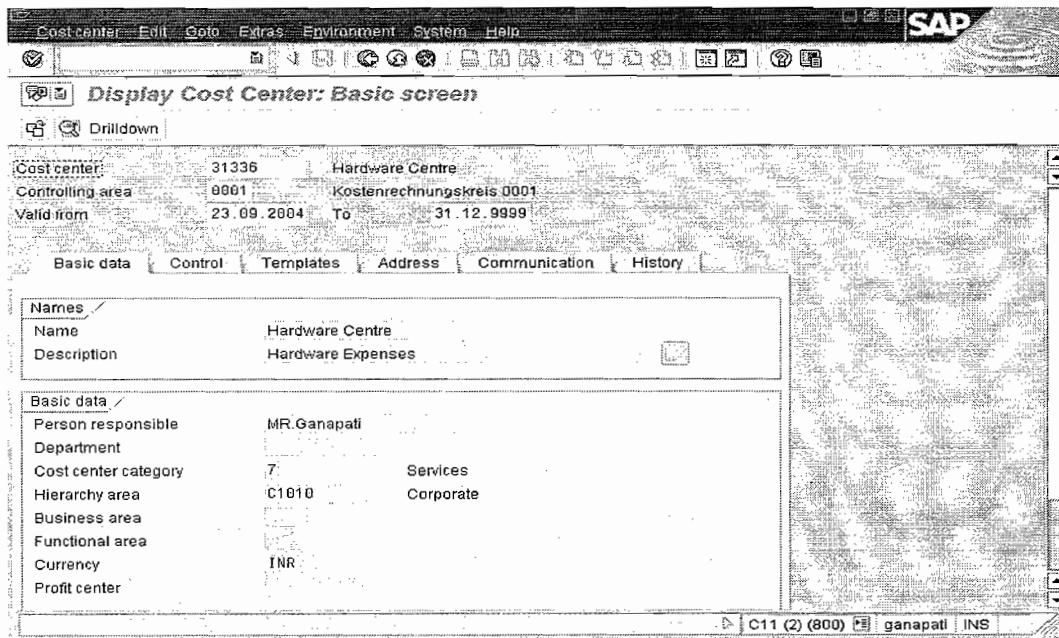
Execute KS03 (Display the Cost Center)



BDC (Batch Data Communication)

We Never Compromise in Quality, Would You?

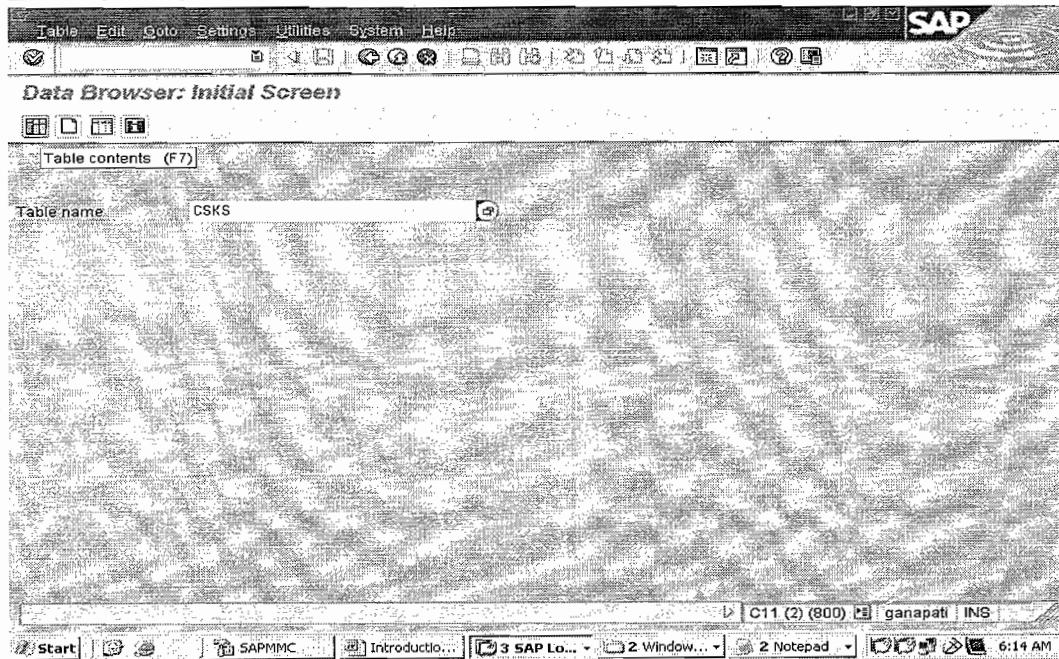
ENTER



Repeat the same Procedure for all records and the Other way is check the Corresponding Database table for all the records.

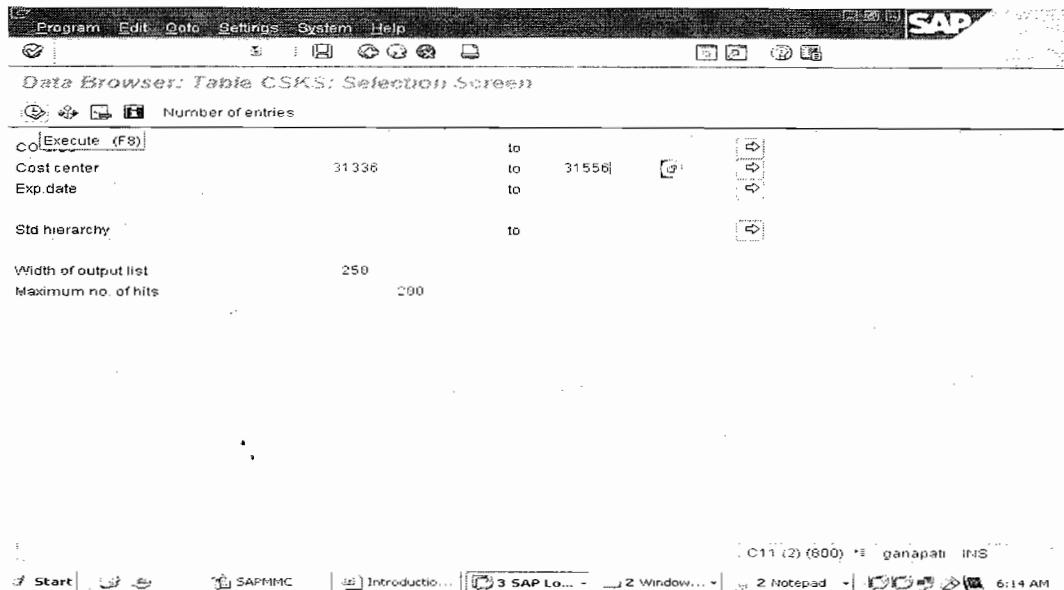
i.e Open the Database table **CSKS** Cost center master data.

Execute SE16 and Provide the table Name as CSKS and click On Table Contents.

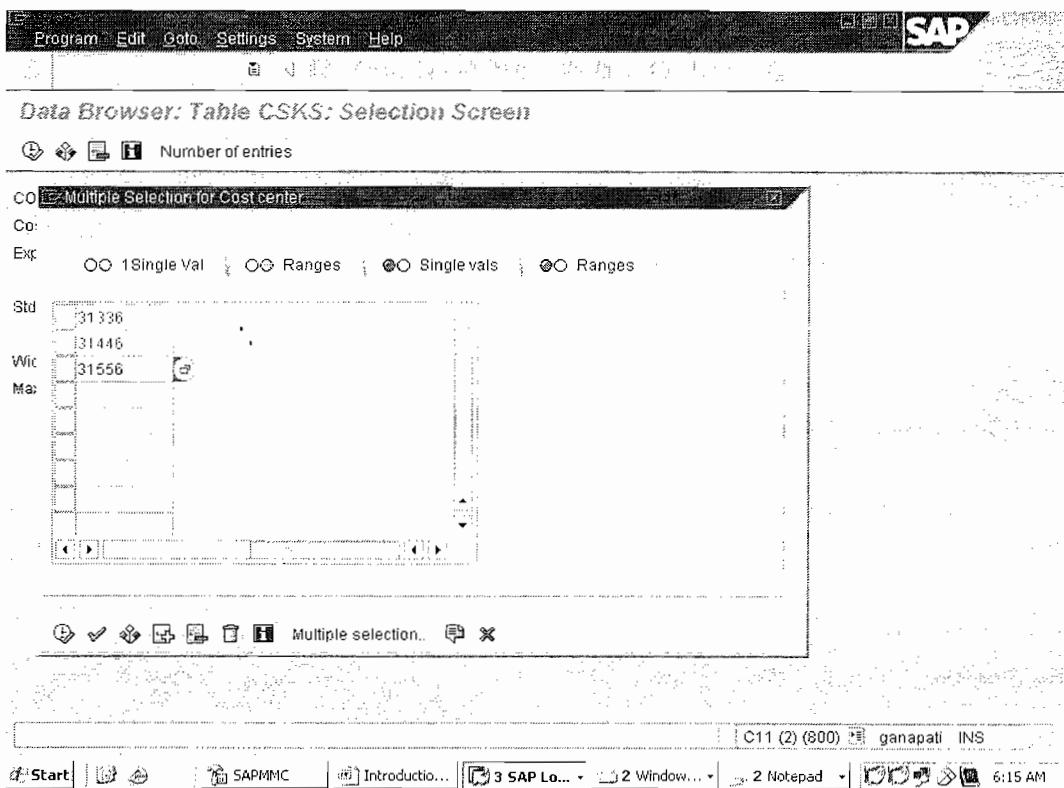


BDC (Batch Data Communication)

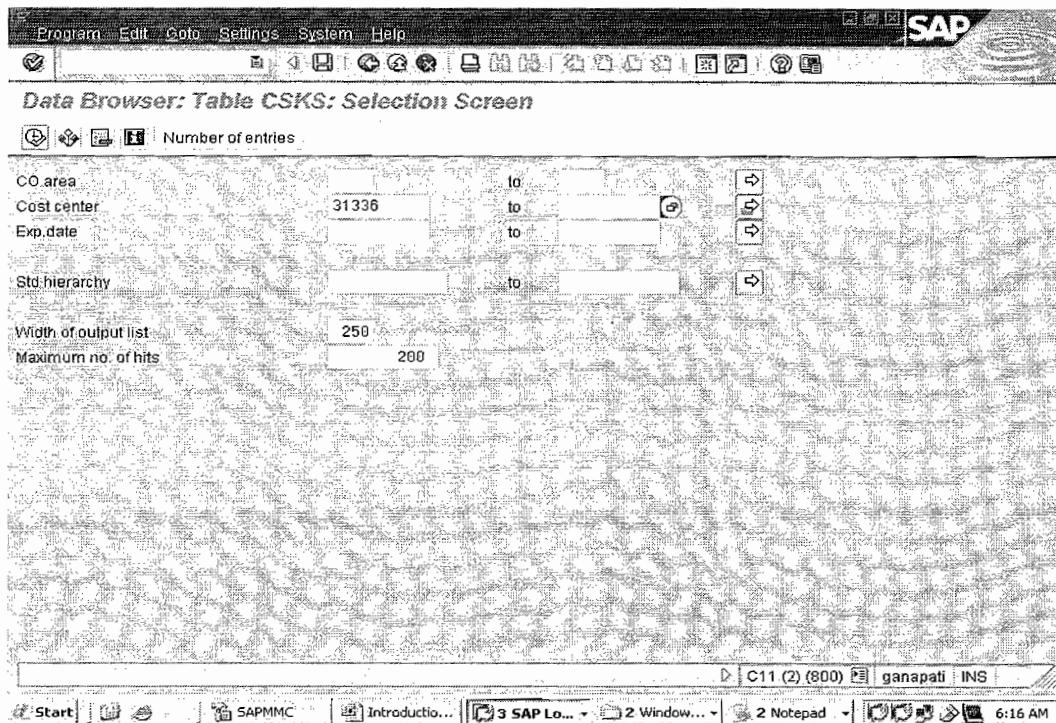
We Never Compromise in Quality, Would You?



Enter the Cost Centers from the File.



BDC (Batch Data Communication)
We Never Compromise in Quality, Would You?



This screenshot shows the SAP Data Browser interface for the CSKS table, titled "Table Browser: Table CSKS Select Entries". The table displays the following data:

| Client | CO-area | Cost_center | Exp.date | Valid_from | Lock_act | PCosts | Plan_pri | costs | Company_Code | Business_area |
|--------|---------|-------------|------------|------------|----------|--------|----------|-------|--------------|---------------|
| 800 | 0001 | 0000031336 | 31.12.9999 | 23.09.2004 | | | | | | |
| 800 | 0001 | 0000031446 | 31.12.9999 | 23.09.2005 | | | | | | |
| 800 | 0001 | 0000031556 | 31.12.9999 | 23.09.2006 | | | | | | |

A callout box points from the third row of the table to a message box containing the text: "Yes, All the Cost Centers are Uploaded Successfully".

The status bar at the bottom shows "C11 (2) (800) ganpati INS" and the system time "6:16 AM".

Session Method

Summary: Offers management of sessions, support for playing back and correcting sessions that contain errors, and detailed logging.

Data Transfer program Create a session on the batch input queue.

It prepares the data and stores it in a batch input session. A session is a collection of transaction data for one or more transactions. Batch input sessions are maintained by the system in the batch input queue. We can process batch input sessions in the background processing system.

The program must open a session in the queue before transferring data to it, and must close it again afterwards. All of these operations are performed by making function module calls from the ABAP program.

The most important aspects of the session interface are:

- Asynchronous processing
- Transfers data for multiple transactions
- Synchronous database update

During processing, no transaction is started until the previous transaction has been written to the database.

- A batch input processing log is generated for each session
- Sessions cannot be generated in parallel

The batch input program must not open a session until it has closed the preceding session.

One of the two recommended ways to process batch input data is to store the data in a batch input session. You can then run this session in the R/3 System to enter the batch input data into the system.

STEPS TO WORK WITH SESSION METHOD :

1. Generate the batch input session using function module **BDC_OPEN GROUP**.
2. Then proceed as follows for each transaction that the session contains:
 - a. In the BDCDATA structure, enter the value for all screens and fields that must be processed in the transaction. (I,e Fill the ITAB IT_BDCDATA).

Session Method

We Never Compromise in Quality, Would You?

- b. Use **BDC_INSERT** to transfer the transaction and the **IT_BDCDATA** to the session.
3. Close the batch input session with **BDC_CLOSE_GROUP**
4. Process the Session Online through **SM35/ In Background through program RSBDCSUB.**

Note : We need to repeat the Step 2, for each transaction , when we want to process multiple transactions through the same Session.

The above Steps in Detail : 1. Use the **BDC_OPEN_GROUP** function module to create a new session. Once you have created a session, then you can insert batch input data into it with **BDC_INSERT**.

| CALL FUNCTION 'BDC_OPEN_GROUP' | |
|--------------------------------|------------------------|
| EXPORTING | |
| CLIENT | = <client> |
| GROUP | = <session name> |
| HOLDDATE | = <lock date> |
| KEEP | = <deletion indicator> |
| USER | = <batch user name> |
| EXCEPTIONS RUNNING | |
| QUEUE_ERROR | = 1 |
| CLIENT_INVALID | = 2 |
| GROUP_INVALID | = 3 |
| . | |

You cannot re-open a session that already exists and has been closed. **If you call BDC_OPEN_GROUP with the name of an existing session, then an additional session with the same name is created.**

BDC_OPEN_GROUP takes the following EXPORTING parameters:

- **CLIENT**

Client in which the session is to be processed.

Default: If you don't provide a value for this parameter, the default is the client under which the batch input program runs when the session is created.

- **GROUP**

Name of the session that is to be created. **Default:** None. **You must specify a session name.**

- **HOLDDATE**

Lock date. The session is locked and may not be processed until the date that you specify. Only a system administrator with the **LOCK** authorization for the authorization object *Batch Input Authorizations* can unlock and run a session before this date.

Format: YYYYMMDD (8 digits).

Default: No lock date, session can be processed immediately. A lock date is optional.

- **KEEP**

Retain session after successful processing. Set this option to the value **X** to have a session kept after it has been successfully processed. A session that is kept remains in the input/output queue until an administrator deletes it.

Note: Sessions that contain **errors** in transactions are kept **even if KEEP is not set.**

Default: If not set, then sessions that are successfully processed are deleted. Only the batch input log is kept.

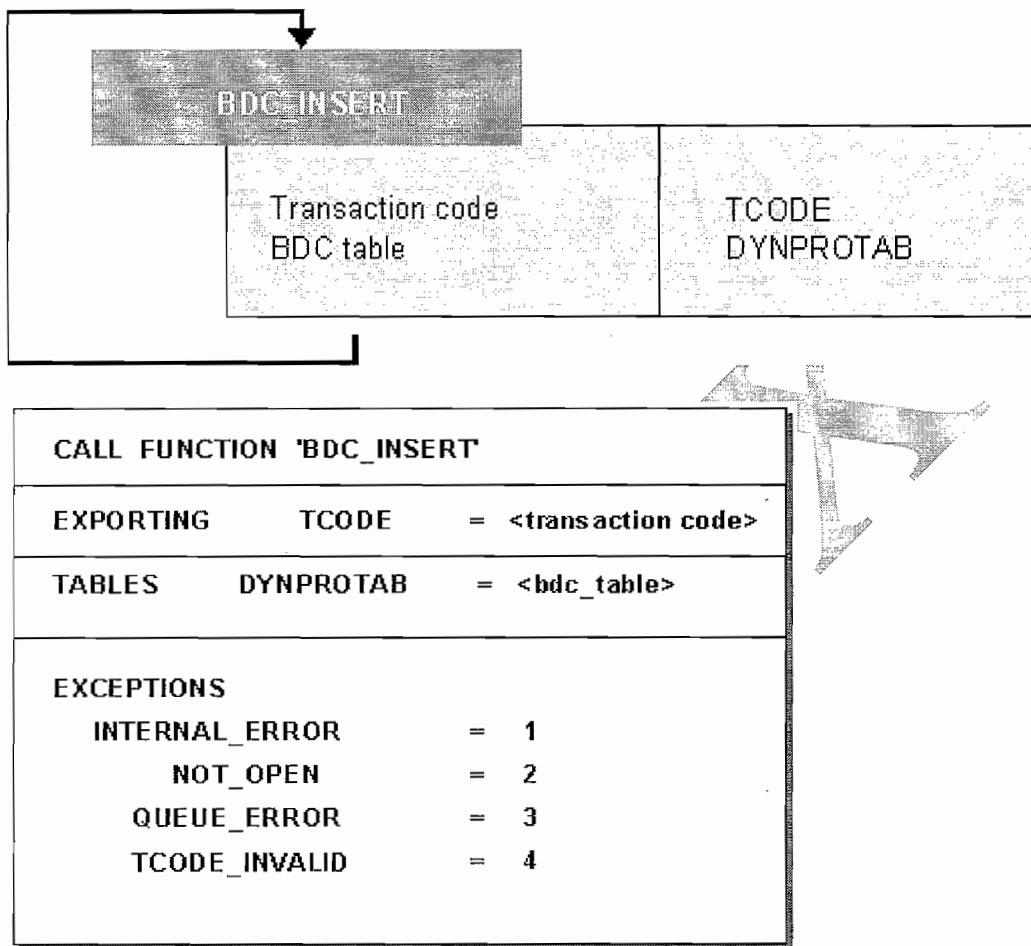
USER

Authorizations user for background processing. This is the user name that is used for checking authorizations if a session is started in background processing. The user must be authorized for all of the transactions and functions that are to be executed in a session. Otherwise, transactions will be terminated with "no authorization" errors.

5. The procedure as follows for each transaction that the session contains:
6.

- a. In the BDCDATA structure, enter the value for all screens and fields that must be processed in the transaction.
- b. Use BDC INSERT to transfer the transaction and the BDCDATA structure to the session.

Adding Data to a Session: BDC_INSERT



BDC_INSERT takes the following parameters:

- **TCODE**

The code of the transaction that is to be run.

- **DYNPROTAB**

The BDCDATA structure that contains the data that is to be processed by the transaction.

DYNPROTAB is a table parameter in the function module

3. Close the batch input session with BDC CLOSE GROUP

Use the BDC_CLOSE_GROUP function module to close a session after you have inserted all of your batch input data into it. Once a session is closed, it can be processed.

BDC CLOSE GROUP

BDC_CLOSE_GROUP needs no parameters. It automatically closes the session that is currently open in your program.

You must close a session before you can open another session from the same program.

You cannot re-open a session once it has been closed. **A new call to BDC_OPEN_GROUP with the same session name creates a new session with the same name.**

4. Start to process the generated session.

Session Can be Processed either Manually or in Background.
Execute the transaction SM35 if Manually Else Schedule the PR
RSBDCSUB.

Managing Batch Input Sessions

Includes : Processing the Sessions
Analysis of Session.

Processing Sessions

A batch input session is a set of one or more calls to transactions along with the data to be processed by the transactions. The system normally executes the transactions in a session non-interactively, allowing rapid entry of bulk data into an R/3 System.

A session records transactions and data in a special format that can be interpreted by the R/3 System. When the System reads a session, it uses the data in the session to simulate on-line entry of transactions and data.

For example, **the data that a session enters into transaction screens is subject to the same consistency checking as in normal interactive operation.** Further, batch input sessions are subject to the user-based authorization checking that is performed by the system.

A Session Can be Processed

Automatically Explicitly

Processing Sessions Automatically**Use**

In most cases, batch input sessions can be processed automatically. It is not necessary for a session to wait until a system administrator explicitly starts the processing of the session.

Prerequisites

The ABAP program **RSBDCSUB** must be scheduled as a periodic job in the R/3 background processing system. RSBDCSUB checks for and starts any batch input sessions that have not yet been run. It schedules such sessions for immediate execution in the background processing system.

Procedure

Schedule RSBDCSUB to run periodically in one or more background jobs.

If you have regularly scheduled batch input runs, you can schedule separate jobs for each of the scheduled data transfers. The start time for the RSBDCSUB job can be set according to the batch input schedule. And you can use a variant to restrict RSBDCSUB only to the batch input sessions that you expect.

Input For the Program RSBDCSUB:

- session name
- date and time of generation
- status: ready to run or held in the queue because of errors

Result

Batch input sessions are started automatically rather than by hand. The **RSBDCSUB** program can be set up to start all sessions that arrive in an R/3 System, or it can be fine-tuned to start only batch input sessions that you expect.

Starting Sessions Explicitly: Run Modes**Use**

Running a batch input session executes the transactions in the session and enters data into an R/3 System.

Usually, the system will run batch input sessions automatically. However, you can also start batch input sessions by hand. You may wish to do this for these and other reasons:

- To correct transactions that had errors
- To check that the transactions in a session have been generated correctly by running the first several transactions
- To start a session on special request (the session would not be started automatically or must be started right away).

Prerequisites

Start the batch input management tool: Select *System □ Service □ Batch input □ Sessions*. Alternate: Enter transaction **SM35**.

Procedure

To start a session, mark the session and choose *Process* from the tool bar. You can then choose how to run a session and with what logging and display options.

You can start any session in *new* status that is not locked.

Run Modes

There are three ways to run a session:



- Process/foreground
- Display errors only
- Background

Process/foreground: You can interactively correct transactions that contained errors and step through transactions that have not yet been executed.

- **Display errors only:** This mode is like *Process/foreground* except that transactions that have not yet been run and which do not contain errors are run non-interactively.

If an error occurs, processing stops and the screen upon which the error occurred is displayed.

Background: Use this mode to schedule a session for immediate processing in the background processing facility.

Note that your session is automatically released for processing in the background processing system. You need not explicitly release the session for processing.

Note : With *Process foreground* or *Display errors only* mode, you can also restart transactions that have the status *Incorrect*. Sessions with the status *Processed* cannot be run again.

- If a transaction in the session contained an error, the incorrect transaction is aborted. All other transactions are completed. The session remains in the queue and is held in the *Errors* section of the list. You can correct the session in one of the interactive modes and run it to completion.

A transaction contains an error if it issues a message of type E (error) or type A (abnormal termination). Other messages are ignored and do not affect the execution of a session.

Analyzing Sessions through SM35.

Use

Session Method

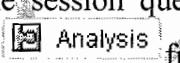
We Never Compromise in Quality, Would You?

Use the analysis functions to check on the actions performed by a batch input session. The analysis functions allow you to display the transactions, screens, and data in a session.

Procedure

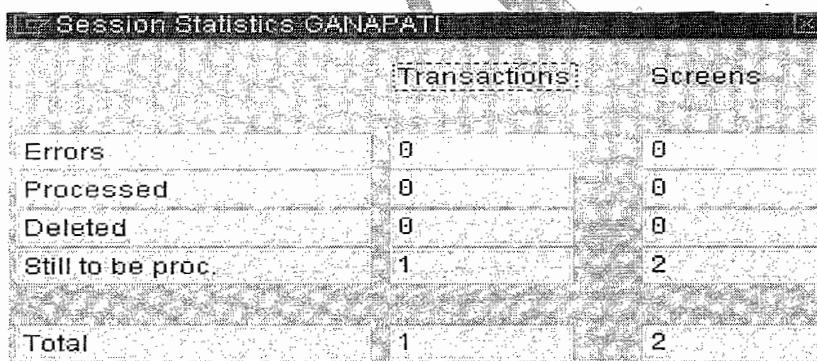
You can use the analysis functions before or after a session has been run.

Note : Analysis of sessions that have been run is available only for sessions that contained an error or which were generated with the KEEP option. The KEEP option prevents the system from deleting a session that has been successfully run.

You can analyze a session by marking the session in the session queue and selecting *Goto □ Analyze session* or by selecting *Analysis*  from the session log screen.

Statistics

For a quick summary of the size and contents of a session, you can use the statistics function: *Goto □ Session statistics* . The advantage over the analysis function is that the summary is displayed more quickly.



| | Transactions | Screens |
|-------------------|--------------|---------|
| Errors | 0 | 0 |
| Processed | 0 | 0 |
| Deleted | 0 | 0 |
| Still to be proc. | 1 | 2 |
| Total | 1 | 2 |

Session Data

You can display the data entered on each screen of a session by switching to the *Screens* tab .

Here, you can have the data presented to you in two formats:

- The first data display presents data in list format by field name. Choose this display by choosing *Options* from the tool bar and marking the *Field list* check box.

- data shown on it. Double-click on the screen in the *Dynpro* list. The system simulates the screen. No data is processed.

You can switch back and forth from one form of display to the other.

Correcting a Session

Use

This section explains how to correct errors in faulty transactions in a session.

Procedure

When a session is run in background-processing mode, the system marks transactions that contain errors as incorrect. All other transactions in the session are completed. The session itself is kept in the session queue in the *Errors* section of the list.

A transaction contains an error if it generates an error message of type E (error) or type A (abnormal termination). Messages of other types are ignored and do not affect the execution of a session.

You can correct and re-execute the incorrect transactions in an **"Errors" session**. There are two ways to do so:

1. Re-run the session in **display-all or error-display mode**. These modes offer the following advantages:

- The system skips transactions that were successfully completed. Only incorrect transactions or transactions that have not been executed are run.
- You can change inputs and add missing screens interactively as incorrect transactions run. The system logs all such changes.

The following topic provides more information on using display-all mode or error-display mode to correct a transaction.

2. As an alternative, you can analyze the session to determine what the error was. With the analysis displays, you can check the screen that contained the error and the values that were entered in it. You can then correct the program that generated the session and regenerate the session.

Note: You must ensure that the new session does not include transactions (Records) that were successfully completed. Otherwise, the updates made by the transactions will be performed again.

Interrupting a Session

You can interrupt the interactive execution of a session by entering the **/bend** OK code on any screen. **/bend** terminates the transaction currently being executed in the session and marks the transaction with the status **Incorrect**. The session is kept in the queue and is displayed in the *Errors* section of the list. Changes made by the interrupted transaction are rolled back as long as the transaction uses only the R/3 update facility.

You can use **/bend** when testing sessions. For example, you may wish to run the first transactions of a large session in display-all mode to make sure that the session has been generated correctly.

If the transactions are correct, you can then terminate the run with **/bend** and then submit the session for background execution. The transactions that have already been run will not be run again.

Restarting a Transaction

You can restart processing of a transaction by entering the **/bbeg** OK code on any screen. **/bbeg** terminates the transaction that is currently being processed and then restarts the transaction fresh, as it is recorded in the batch input session. Any changes made by the transaction are rolled back, as long as they were made only by way of the R/3 update facility. Changes made directly to the database are not rolled back.

Deleting a Transaction

You can delete a transaction from a session during interactive execution by entering the **/bdel** OK code. The transaction is removed from the session. The deleted transaction cannot be run even if you restart the session, and you cannot take back the deletion.

The transaction is, however, available for analysis if the session was generated with the KEEP option set. The transaction is marked with the status **Deleted** in the analysis display. The deletion also is recorded in the log generated by the session.

Similarly we have some other OK Codes and their Functionalities :.

| Function | OK Code |
|---|---------|
| Terminate current batch input and mark as incorrect | /n |
| Delete current batch input from session | /bdel |
| Restart a transaction | /bbeg |
| Terminate batch input processing and mark session as incorrect | /bend |
| Change display mode to process the on screen instead of displaying only | /bda |
| Change display mode to display only instead of processing the sessions on | /bde |

Locking and Unlocking Sessions

Use

You can lock a session to prevent the system from running it before the date that you specify in the lock.

For example, a session locked until the eleventh of November can be started only after that date.

You may wish to lock a session for such reasons as

- holding a session until a specified date
- preventing a session that contains errors from being re-started
- preventing a session that requires a special resource, such as printer forms or a specific tape, from being started unintentionally.

You can unlock a session by changing the lock date to the present date or by entering a space as the lock date.

Procedure

You can lock sessions in the session queue by marking the sessions and choosing **Lock** [] from the tool bar and similarly **Unlock** [].

Displaying Session Logs

Use

The batch input system keeps a detailed log of each session that is processed. The log contains not only progress messages from the batch input system itself, but also error messages from the transactions that are processed.

To analyze an error in a session, you should start by checking the session log for relevant messages.

A session log is kept only if the session was generated with the **KEEP** option or if the session is aborted or contains an error.

Prerequisites

Start the batch input management tool: Select **System □ Service □ Batch input □ Sessions or Logs**. Alternate: **Enter transaction SM35P**. You can display logs from the standard session overview, or from a special separate transaction for logs. The log functionality is in each case the same.

Procedure

To display a log, mark a session and choose **Log** [] from the tool bar.

Reorganizing the Batch Input Session Log File

You should periodically use the ABAP/4 program RSBDCREO to reorganize the batch input log file. You can run RSBDCREO in the background or interactively.

Running the program reduces the size of the log file .

Session Status(Before and After Processing the Session either of the ways).

Sessions in the session queue are sorted by date and time of generation and are grouped in different lists according to their status.



Possible statuses are as follows:

- **New**

Session was Created and recorded but not yet Processed.

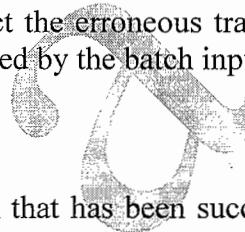
- **Incorrect**

Held in the session queue because of errors in transactions (Errors in sessions)

Transactions that contained errors are aborted; all correct transactions are processed. The *Tran.* and *Screen* fields in the session display show how many incorrect transactions were found in the session.

You can restart a session and correct the erroneous transactions with one of the interactive execution modes offered by the batch input system.

- **Processed**



For further information on a session that has been successfully run, you can display the log generated by the session. All completed sessions generate a log. You cannot run a completed session a second time.

Note: Only sessions that were generated with the KEEP option are held in the queue after processing. Other sessions are deleted after they are successfully completed.

- **in processing**

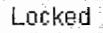
You will usually see this status only if you happen to display the queue while a session is being run.

- **In Background**

You will usually see this status only if you happen to Process the Session in the Mode Background Mode (**Display No Screens**).

- **Being Created**

You will usually see this status only if you happen to display the queue while a session is being generated (entered into the session queue).

Locked 

Status when the session is Locked i.e when the HOLDDATE in BDC_OPEN_GROUP is set.

Programming in Session Method :

Requirement: Upload all the List Of Profit Center Details into SAP using Session Method.

About Profit Center Master Data:**Profit Center****Definition**

A profit center is an organizational unit in accounting that reflects a management-oriented structure of the organization for the purpose of internal control.

You can analyze operating results for profit centers using either the cost-of-sales or the period accounting approach.

By calculating the fixed capital as well, you can use your profit centers as investment centers.

Use

Profit center Accounting at the profit center level is based on costs and revenues.

Structure**The master data of a profit center includes**

The name of the profit center,

the controlling area it is assigned to,

and the profit center's period of validity,

as well as information about the person responsible for the profit center,

the profit center's assignment to a node of the standard hierarchy, and data required for communication (address, telephone number and so on).

Every profit center is assigned to the organizational unit **Controlling area**. This assignment is necessary because Profit Center Accounting displays values in G/L accounts.

Session Method

We Never Compromise in Quality, Would You?

The system transfers all the data to Profit Center Accounting together with the G/L account to which the data was originally posted.

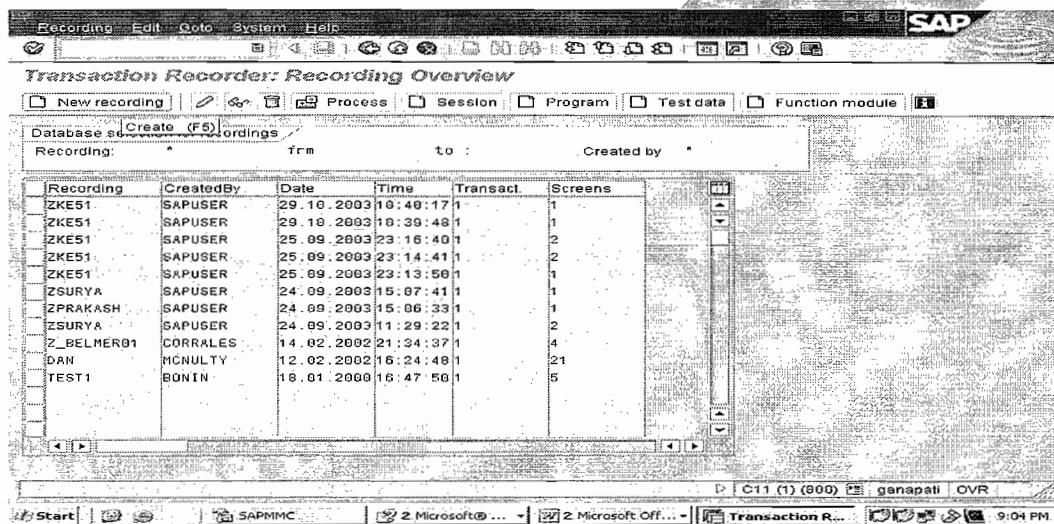
Transaction Code : KE51.

Note : SHDB is transaction to record all the steps while executing the Session Method.

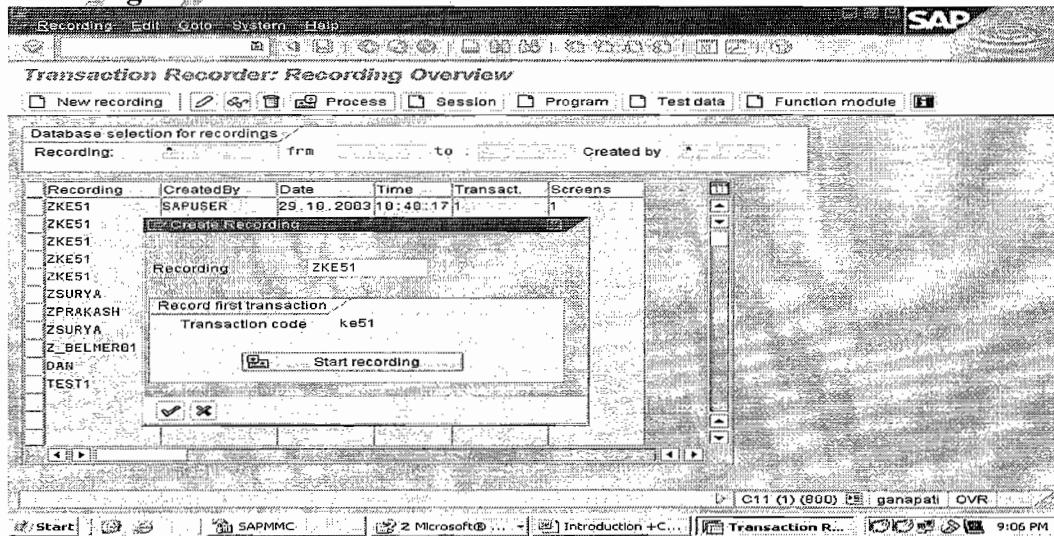
Recording the KE51:

Steps to Work with Recording(SHDB) :

1. Execute SHDB and Click On New Recording.



2. Provide the Recording name and Transaction Code for which the recording has to be initiated.

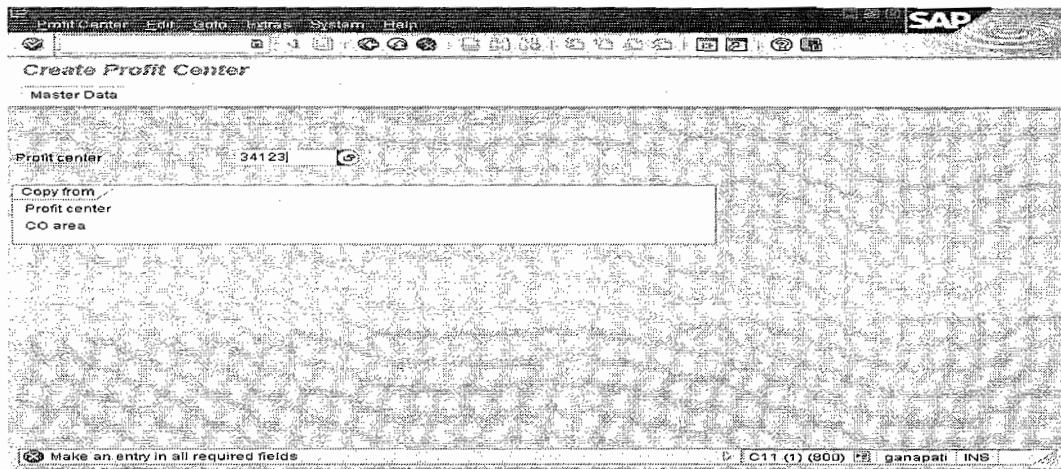


Session Method

We Never Compromise in Quality, Would You?

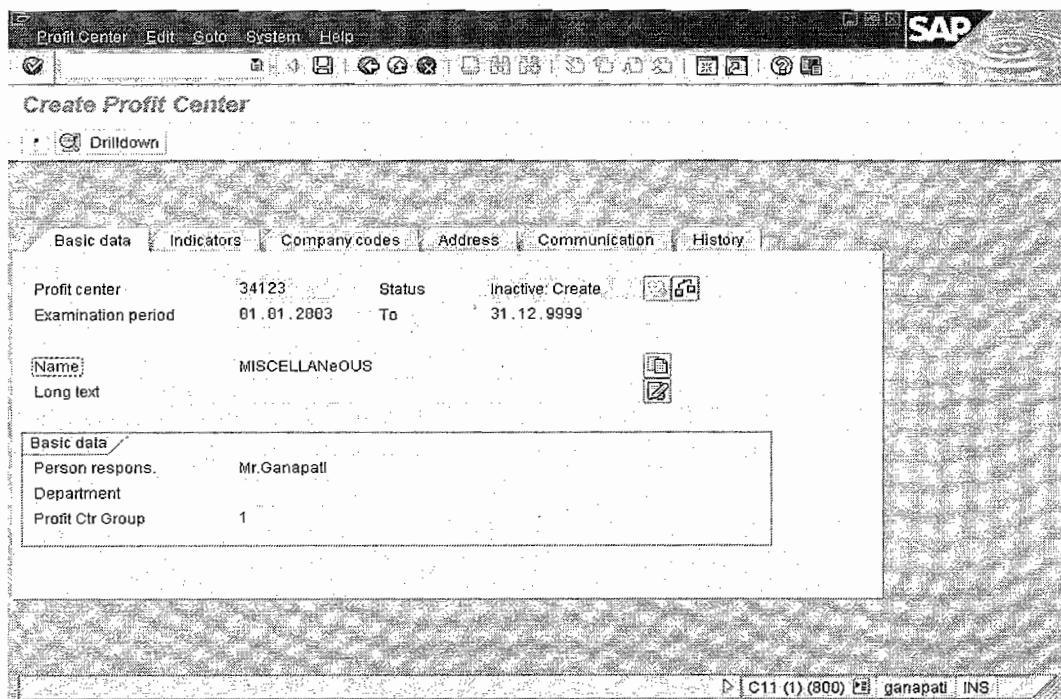
Press Enter and Start Execute the transaction so that the steps will be recorded.

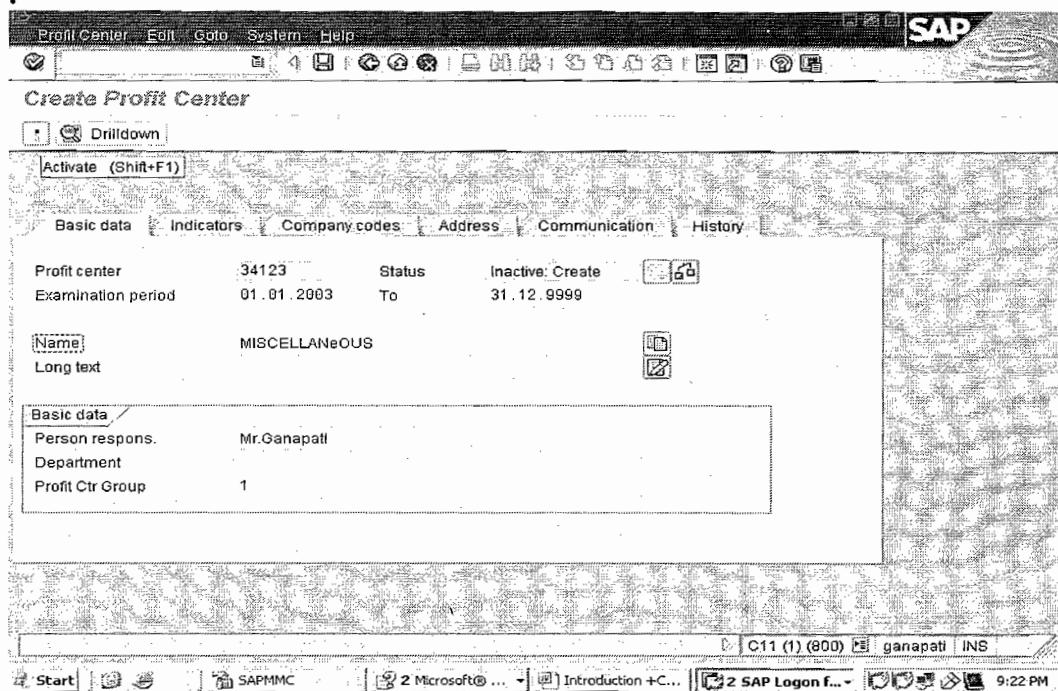
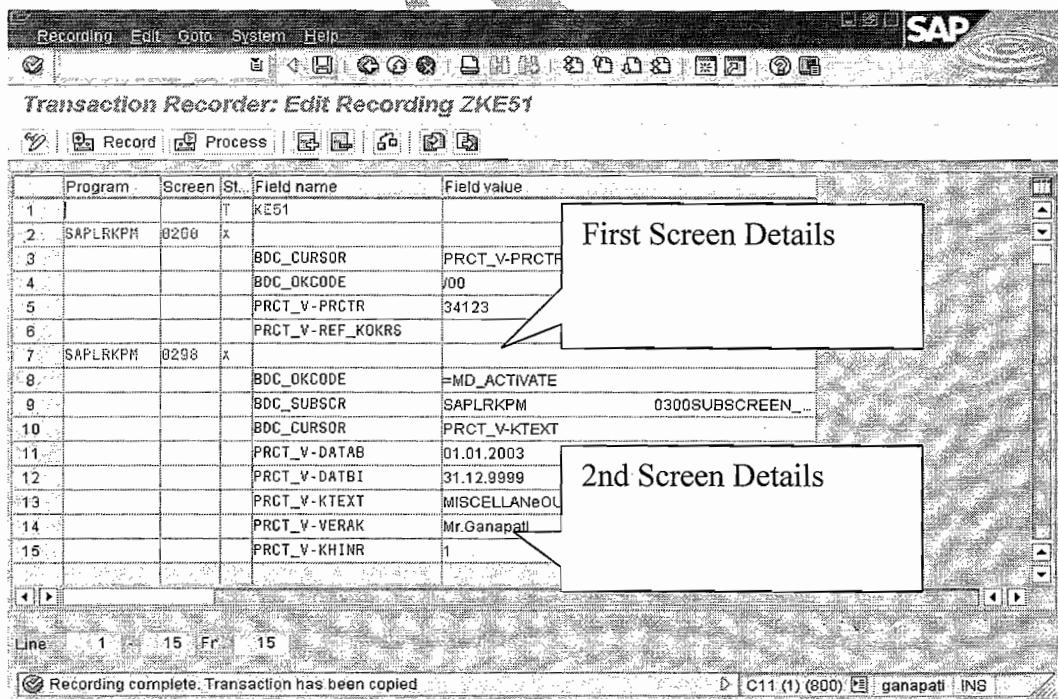
Enter the Profit Center number and press ENTER.



ENTER

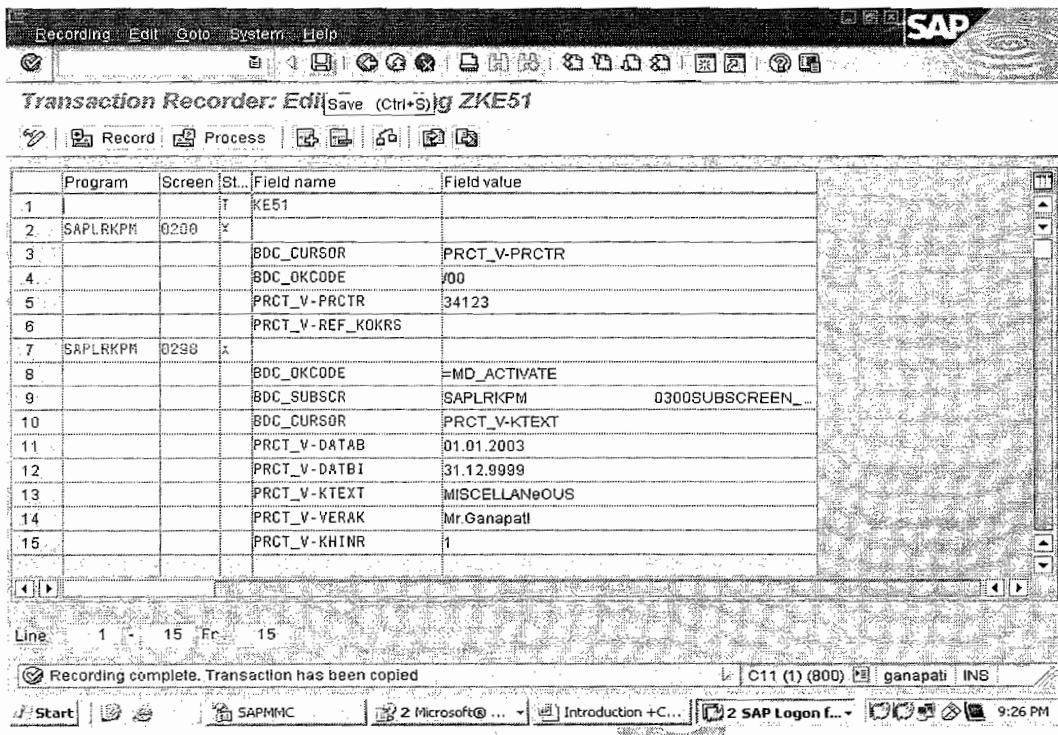
Enter all the mandatory fields i.e examination Periods(Valid From and Valid to),Name,Person responsible and Profit Center Group Details.



Click On Activate**Observe the Recording Steps.****SAVE the recording.**

Session Method

We Never Compromise in Quality, Would You?



Note: We Use the above steps while writing the BDC program.

```
*****
* PROGRAM : ZDEMO_UPLOAD_PROFIT_CENTER *
* AUTHOR : GANAPATI . ADIMULAM *
* PURPOSE : BDC PROGRAM TO UPLOAD ALL THE PROFIT *
*           CENTER MASTER DATA INTO SAP *
* REFERENCE : NA *
* COPIED FROM : NA *
* TRAPORT REQUEST NO : C11D2K9001 *
*****
```

REPORT ZDEMO_UPLOAD_PROFIT_CENTER .

```
*INCLUDE FOR ALL THE GLOBAL DECLARATIONS
INCLUDE ZDEMO_PROFIT_CENTER_TOP.
```

```
*****
* AT SELECTION-SCREEN ON VALUE-REQUEST      *
*****
```

*EVENT TO BE TRIGGERED WHEN WE PRESS F4.

AT SELECTION-SCREEN ON VALUE-REQUEST FOR PA_FILE.

PERFORM GET_F4_FOR_FILE USING PA_FILE.

* START-OF-SELECTION. *

START-OF-SELECTION.

*WE NEED TO MOVE THE PA_FILE INTO ANOTHER VARIABLE OF TYPE STRING

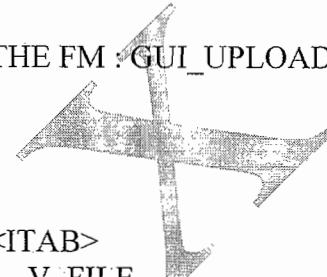
*AS WE ARE GOING TO USE THE SAME IN THE FM : GUI_UPLOAD THERE THE

*FILE TYPE IS STRING.

V_FILE = PA_FILE.

*UPLOAD THE DATA FROM FLAT FILE TO <ITAB>

PERFORM UPLOAD_FILE TO_ITAB USING V_FILE
CHANGING IT_DATA.



*OPENS A NEW SESSION

PERFORM OPEN_SESSION USING P_GROUP.

*FOR EACH PROFIT CENTER.

LOOP AT IT_DATA INTO WA_DATA,
REFRESH IT_BDCDATA.

*FIRST SCREEN DETAILS

PERFORM FILL_SCREEN_DETAILS USING 'SAPLRKPM' '0200' 'X'.

*PROFIT CENTER

PERFORM FILL_FIELD_DETAILS USING 'PRCT_V-PRCTR' WA_DATA-
PRCTR.

*ENTER

PERFORM FILL_FIELD_DETAILS USING 'BDC_OKCODE' '/00'.

*NEXT SCREEN DETAILS

PERFORM FILL_SCREEN_DETAILS USING 'SAPLRKPM' '0298' 'X'.

*CURSOR DETAILS

PERFORM FILL_FIELD_DETAILS USING 'BDC_CURSOR' 'PRCT_V-
KHNR'.

*OKCODE DETAILS

PERFORM FILL_FIELD_DETAILS USING 'BDC_OKCODE'
'=MD_ACTIVATE'.

Session Method

We Never Compromise in Quality, Would You?

```
*SUBSCR DETAILS
  PERFORM FILL_FIELD_DETAILS USING 'BDC_SUBSCR'
    'SAPLRKPM'          0300SUBSCREEN_EO'.

*START DATE
  PERFORM FILL_FIELD_DETAILS USING 'PRCT_V-DATAB' WA_DATA-
DATAB.

*END DATE
  PERFORM FILL_FIELD_DETAILS USING 'PRCT_V-DATBI' WA_DATA-
DATBI.

*NAME
  PERFORM FILL_FIELD_DETAILS USING 'PRCT_V-KTEXT' WA_DATA-
KTEXT.

*PERSON RESPONSIBLE
  PERFORM FILL_FIELD_DETAILS USING 'PRCT_V-VERAK' WA_DATA-
VERAK.

*HIERARCHIAL AREA
  PERFORM FILL_FIELD_DETAILS USING 'PRCT_V-KHINR' WA_DATA-
KHINR.

  PERFORM BDC_INSERT USING 'KE51'
    IT_BDCDATA.
```

ENDLOOP.

PERFORM CLOSE_SESSION.

* DEFINITIONS OF ALL THE ABOVE SUBROUTINES *

```
*&-----*
*&  Form FILL_SCREEN_DETAILS
*&-----*
*   text
*-----*
```

```
FORM FILL_SCREEN_DETAILS USING PROGRAM DYNPRO DYNBEGIN.
  WA_BDCDATA-PROGRAM = PROGRAM.
  WA_BDCDATA-DYNPRO = DYNPRO.
  WA_BDCDATA-DYNBEGIN = DYNBEGIN.
  APPEND WA_BDCDATA TO IT_BDCDATA.
  CLEAR WA_BDCDATA.
ENDFORM.           " FILL_SCREEN_DETAILS
*-----*
*&  Form FILL_FIELD_DETAILS
```

```
*&-----*
FORM FILL_FIELD_DETAILS USING FNAM FVAL.
WA_BDCDATA-FNAM = FNAM.
WA_BDCDATA-FVAL = FVAL.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
ENDFORM.           " FILL_FIELD_DETAILS
```

```
*&-----*
*&   Form UPLOAD_FILE_TO_ITAB
*&-----*
*   SUBROUTINE TO UPLOAD THE DATA FROM PRE SERVER FILE TO
ITAB
*-----*
*   -->P_V_FILE - FILE NAME
*   <-P_IT_DATA - INTERNAL TABLE TO STORE THE DATA
*-----*
FORM UPLOAD_FILE_TO_ITAB USING FP_V_FILE
CHANGING FP_IT_DATA LIKE IT_DATA.
```

```
CALL FUNCTION 'GUI_UPLOAD'
EXPORTING
  FILENAME      = FP_V_FILE
  HAS_FIELD_SEPARATOR = 'X'
TABLES
  DATA_TAB      = FP_IT_DATA.
ENDFORM.           " UPLOAD_FILE_TO_ITAB
```

```
*&-----*
*&   Form OPEN_SESSION
*&-----*
*   CREATES THE SESSION
*-----*
*   -->P_P_GROUP NAME OF THE SESSION
*-----*
FORM OPEN_SESSION USING FP_P_GROUP.
```

```
*CALL THE FUCTION MODEU BDC_OPEN_GROUP
CALL FUNCTION 'BDC_OPEN_GROUP'
EXPORTING
  CLIENT = SY-MANDT
  GROUP = FP_P_GROUP
  KEEP = 'X'
```

```
USER = SY-UNAME.  
IF SY-SUBRC = 0.  
    WRITE :'/PROCESS THE SESSION', FP_P_GROUP, 'USING SM35'.  
ENDIF.  
  
ENDFORM.          " OPEN_SESSION  
*&-----*  
*&  Form GET_F4_FOR_FILE  
*&-----*  
*  DISPLAY ALL FILES IN THE SYSTEM FOR SELECTION  
*-----*  
*  -->P_PA_FILE NAME OF THE FILE  
*-----*  
FORM GET_F4_FOR_FILE USING P_PA_FILE.  
  
CALL FUNCTION 'KD_GET_FILENAME_ON_F4'  
    EXPORTING  
        FIELD_NAME = 'PA_FILE'  
    CHANGING  
        FILE_NAME = PA_FILE.  
IF SY-SUBRC <> 0.  
    MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO  
        WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.  
ENDIF.  
  
ENDFORM.          " GET_F4_FOR_FILE  
*&-----*  
*&  Form CLOSE_SESSION  
*&-----*  
*  TO CLOSE THE ACTIVE SESSION  
*-----*  
FORM CLOSE_SESSION.  
  
*CALL FUCNTION MODULE 'BDC_COLSE_GROUP'  
    CALL FUNCTION 'BDC_CLOSE_GROUP'  
* EXCEPTIONS  
*  NOT_OPEN      = 1  
*  QUEUE_ERROR   = 2  
*  OTHERS        = 3  
  
IF SY-SUBRC <> 0.  
    MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO  
        WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
```

```
ENDIF.  
  
ENDFORM.           " CLOSE_SESSION  
  
*&-----*  
*&  Form BDC_INSERT  
*&-----*  
*   text  
*-----*  
*   -->P_TCODE text  
*   -->P_IT_BDCDATA text  
*-----*  
FORM BDC_INSERT USING  FP_TCODE TYPE SYTCODE  
                      FP_IT_BDCDATA LIKE IT_BDCDATA.  
  
*CALL FUCTION MODULE 'BDC_INSERT'  
CALL FUNCTION 'BDC_INSERT'  
  EXPORTING  
    TCODE    = FP_TCODE  
  TABLES  
    DYNPROTAB = FP_IT_BDCDATA.  
IF SY-SUBRC <> 0.  
  MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO  
    WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.  
ENDIF.  
  
ENDFORM.           " BDC_INSERT
```

The Declarations in the TOP INCLUDE:

```
*-----*  
*  INCLUDE ZDEMO_PROFIT_CENTER_TOP  
*-----*
```

```
DATA : BEGIN OF WA_DATA,  
      PRCTR TYPE PRCTR, "Profit Center  
      DATAB TYPE DATAB, "START DATE  
      DATBI TYPE DATBI, "END DATE  
      KTEXT TYPE KTEXT, "NAME  
      VERAK TYPE VERAK, "PERSON RESPONSIBLE  
      KHINR TYPE KHINR, "HIERARCHY AREA  
      END OF WA_DATA.  
DATA : IT_DATA LIKE TABLE OF WA_DATA.
```

DATA : IT_BDCDATA LIKE TABLE OF BDCDATA,
WA_BDCDATA LIKE LINE OF IT_BDCDATA.

DATA V_FILE TYPE STRING.

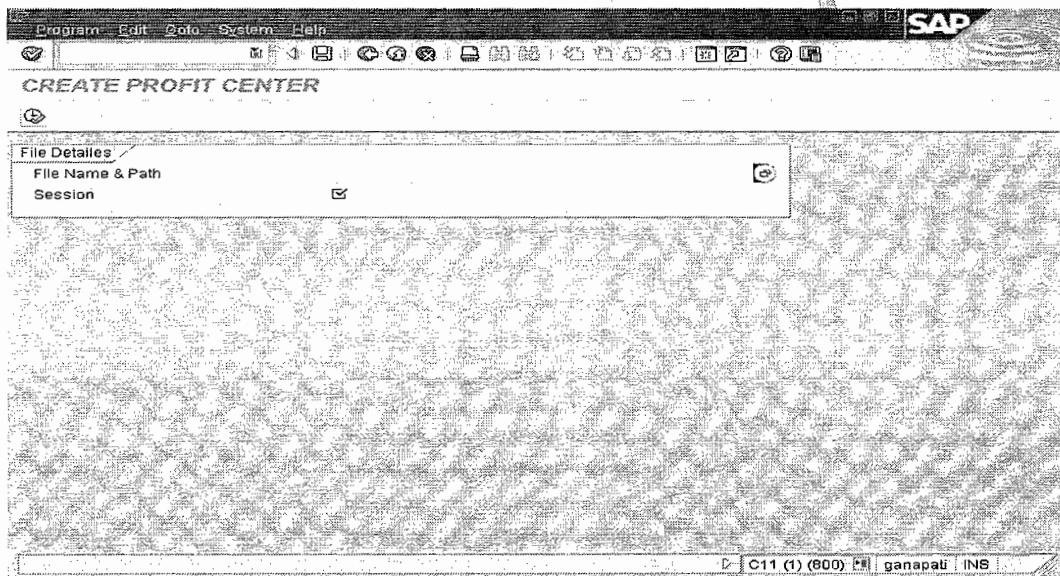
SELECTION-SCREEN BEGIN OF BLOCK B1 WITH FRAME TITLE TEXT-000.

PARAMETER : PA_FILE LIKE FC03TAB-PL00_FILE OBLIGATORY ,
P_GROUP LIKE APQI-GROUPID OBLIGATORY.

SELECTION-SCREEN END OF BLOCK B1.

Execution the Program , Observe the Input and Output of the Program :

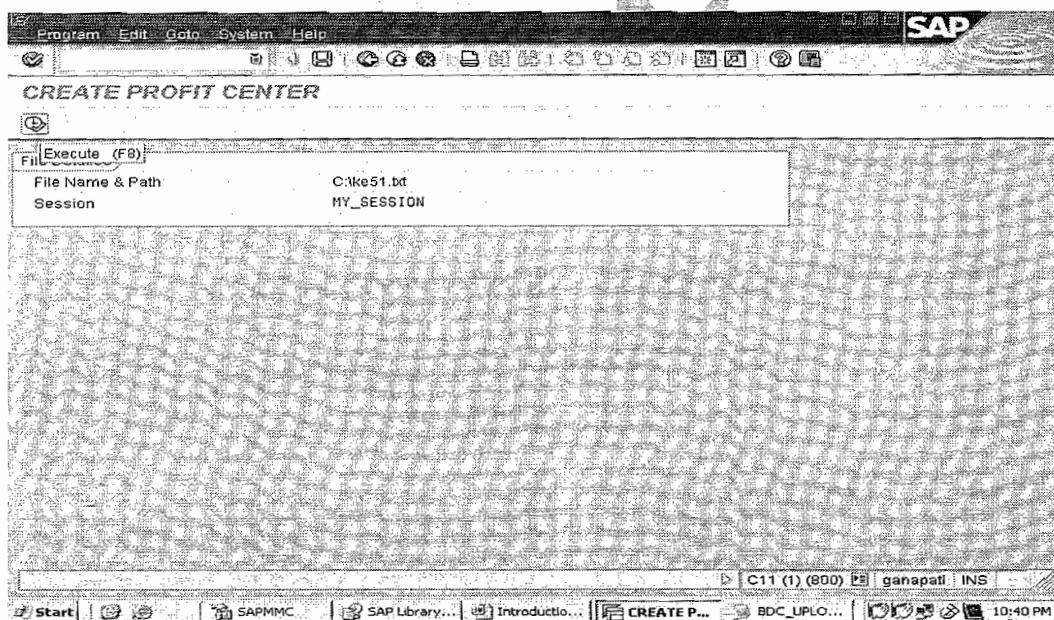
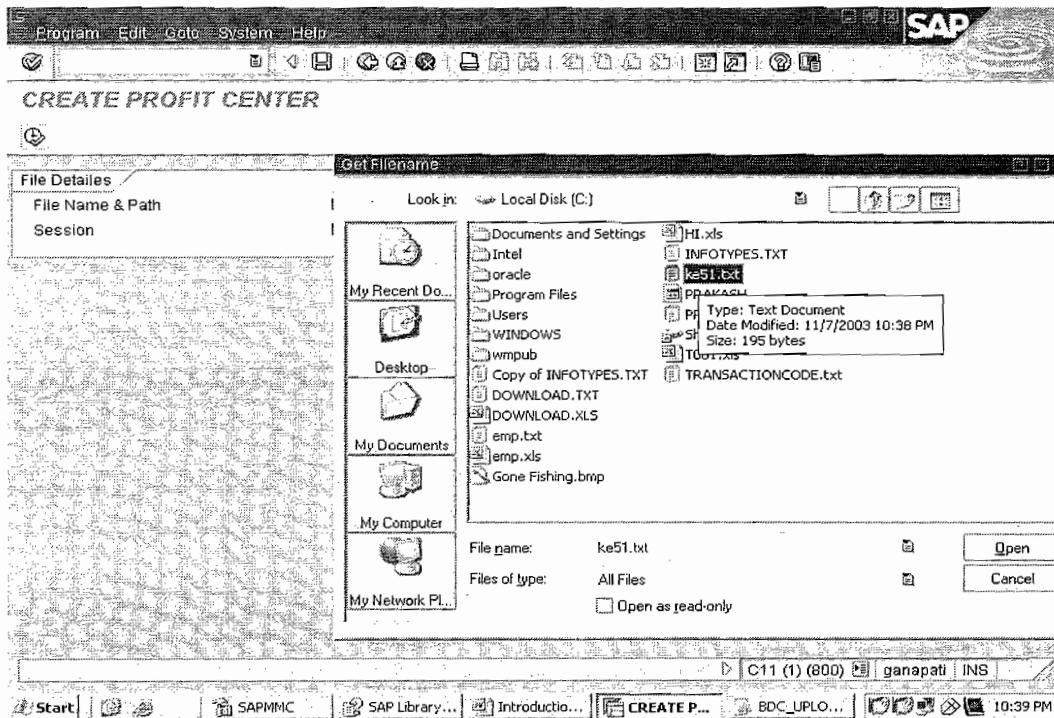
Execute ZDEMO_UPLOAD_PROFIT_CENTER in SE38.



Select the File and Provide the Session Name and Execute it.

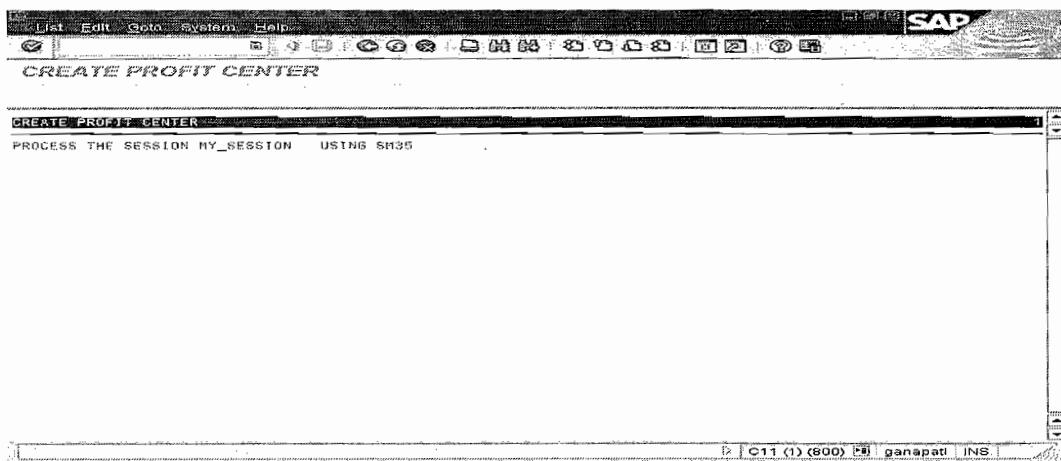
Session Method

We Never Compromise in Quality, Would You?



Session Method

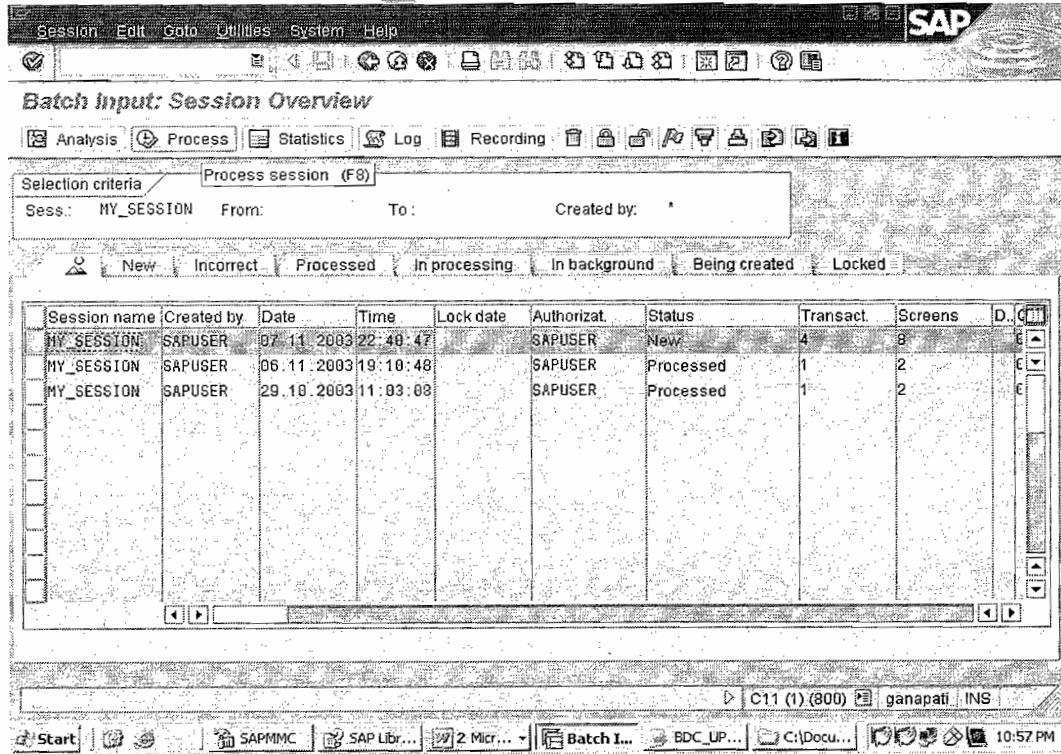
We Never Compromise in Quality, Would You?



Session Processing through SM35 (Process the Session Explicitly) :

1) Execute the Transaction SM35.

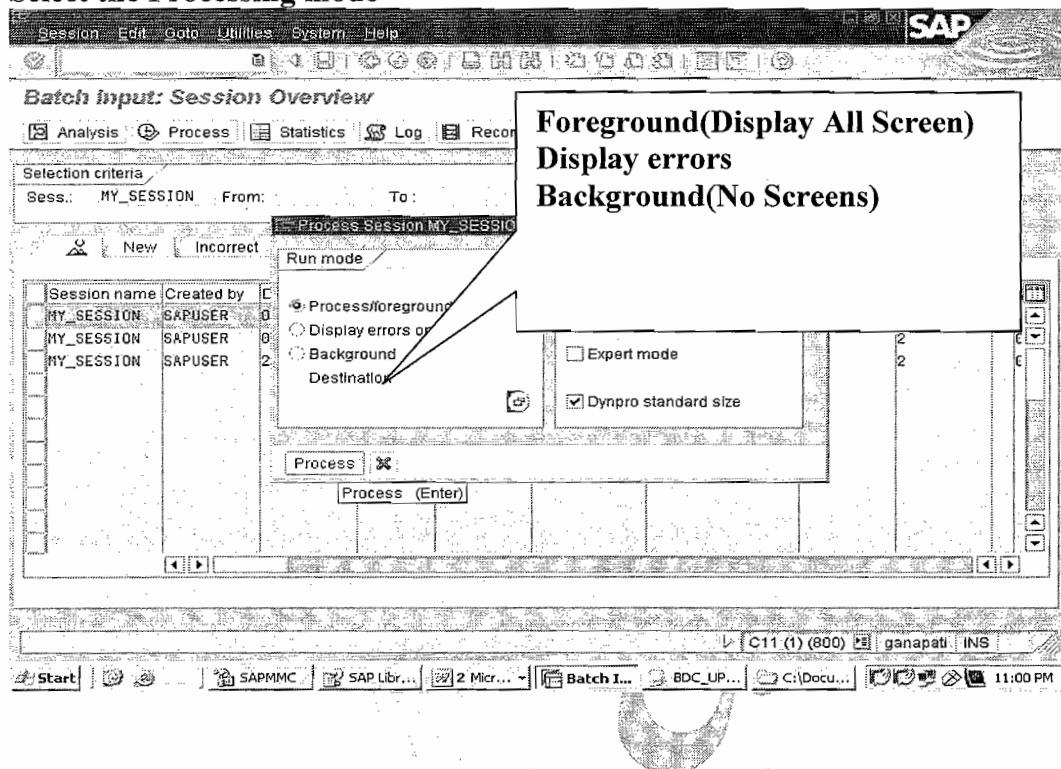
Provide the Session name and Execute .



Session Method

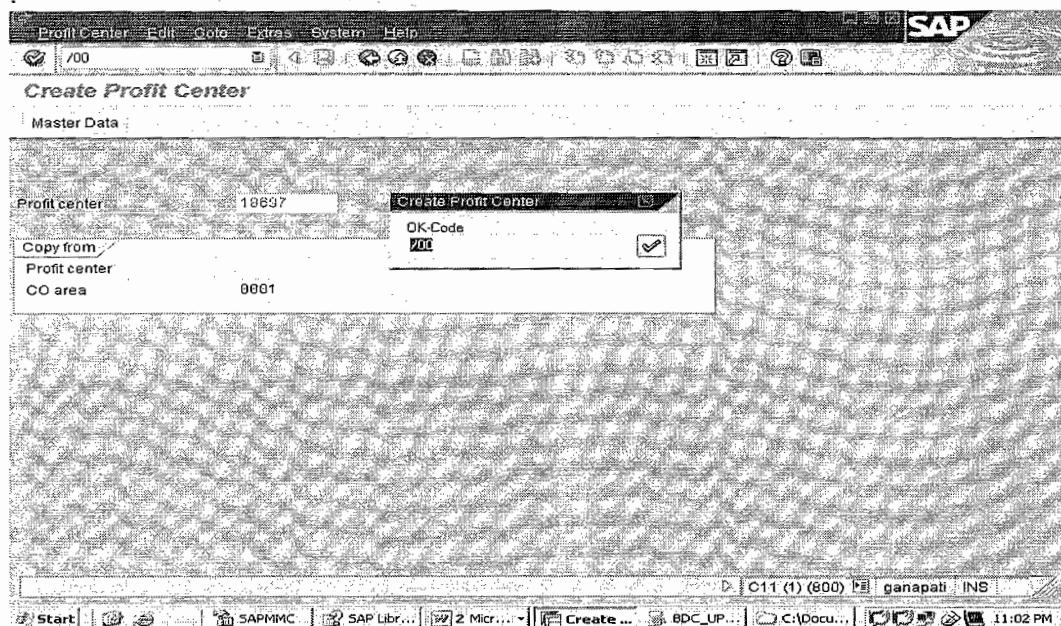
We Never Compromise in Quality, Would You?

Select the Processing mode



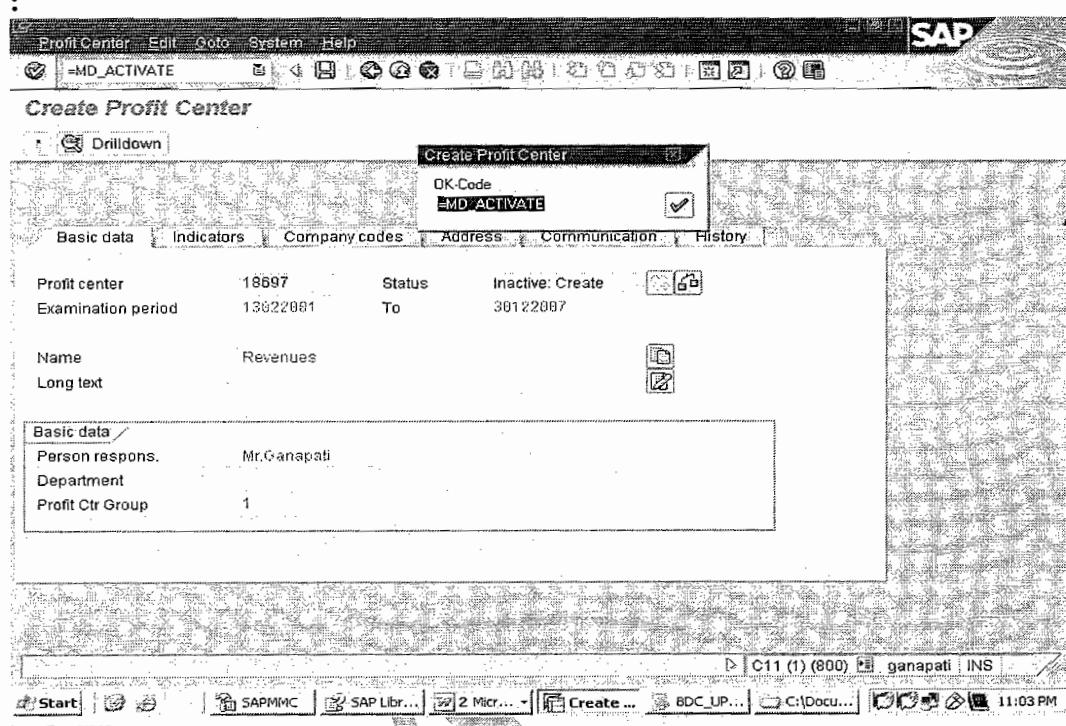
Click On Process.

First Screen Details



ENTER

2nd Screen Details



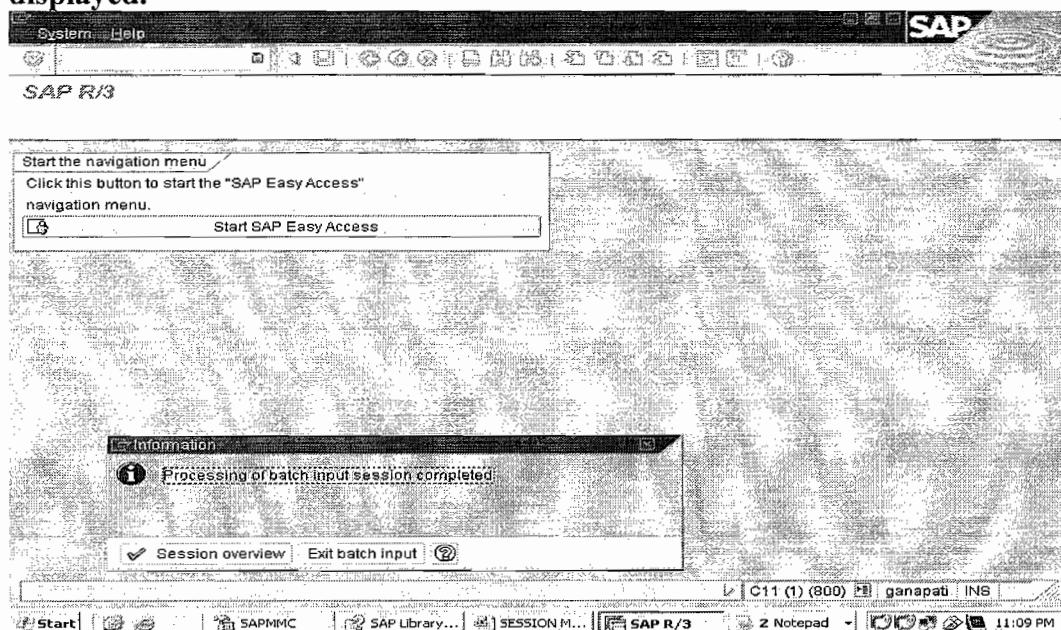
ENTER

Note: Observe the Profit Center 18697 is created and calls the same sequence of screens For the rest of the Profit centers too.

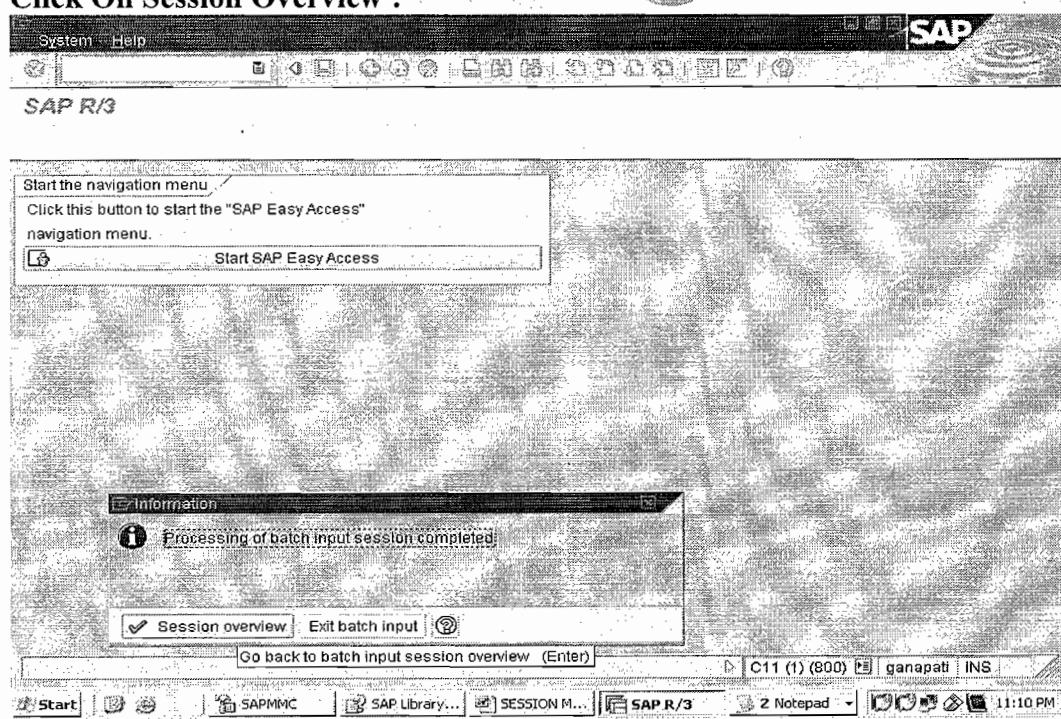
Input File :

| File Edit Format View Help | | | |
|----------------------------|----------|----------|-------------|
| 18697 | 13022001 | 30122007 | Revenues |
| 18567 | 13022005 | 30122008 | Furniture |
| 16697 | 13022001 | 30122007 | Hardware |
| 14697 | 13022001 | 30122007 | S/W Product |

After Processing all the records from the above screens , a screen will be displayed.



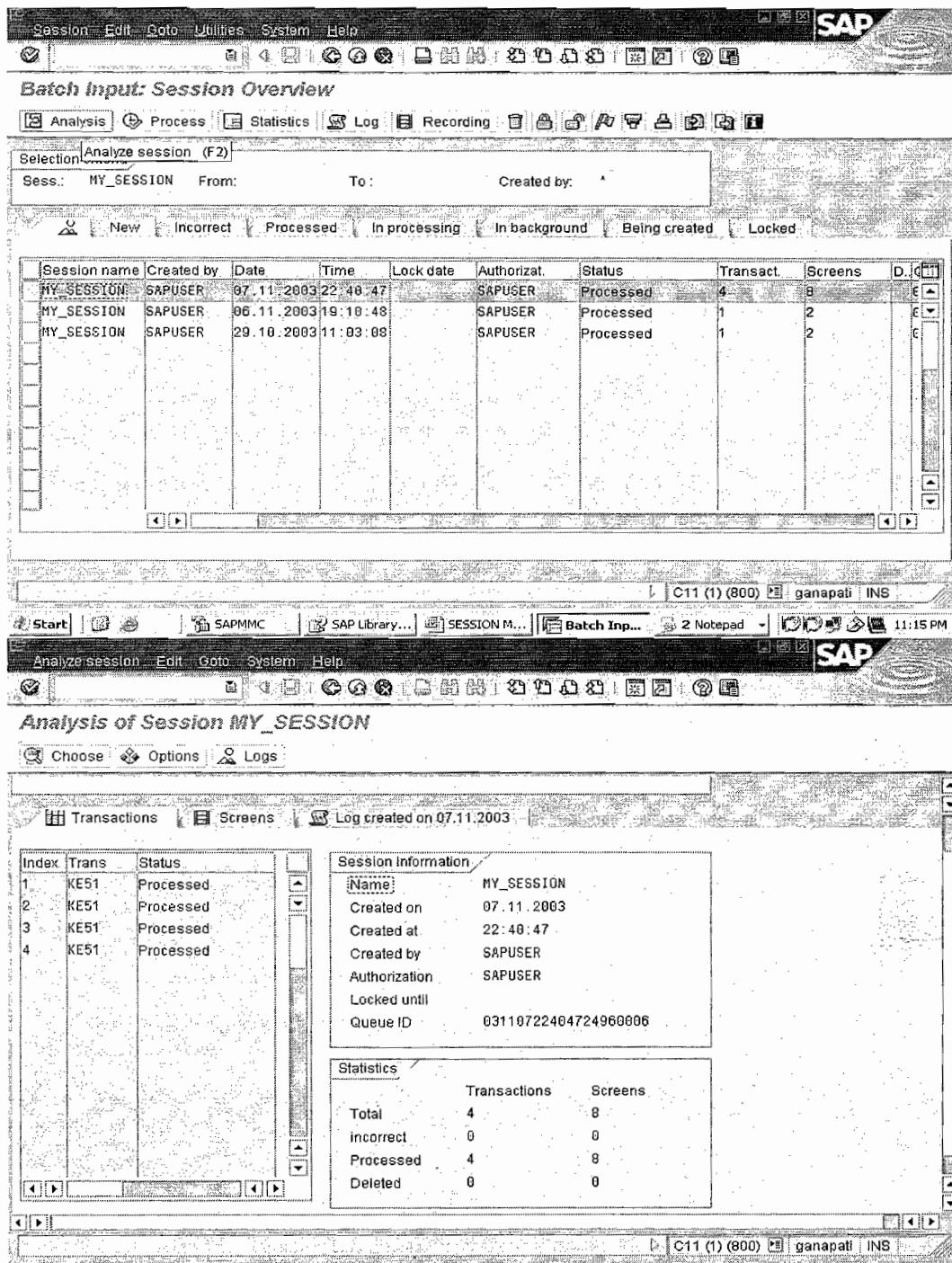
Click On Session Overview :



Select the processed session and click on Analysis.

Session Method

We Never Compromise in Quality, Would You?



Note : For More details about Batch Input Session, go through the **Managing Batch Input Sessions** in the same chapter.

Requirement :

Write a program to process more than one transaction through the same session(Upload Profit Center + Cost Center Master Data).

```
*****  
* PROGRAM : ZDEMO_UPLOAD_PROFIT_CENTER *  
* AUTHOR : GANPATI . ADIMULAM  
* PURPOSE : BDC PROGRAM TO UPLOAD ALL THE PROFIT *  
* CENTER AND COST CENTER MASTER DATA INTO *  
* SAP THROUGH THE SAME SESSION  
*
```

```
* REFERENCE : NA
```

```
* COPIED FROM : NA
```

```
* TRAPORT REQUEST NO : C11D2K9001
```

```
*****  
REPORT ZDEMO_UPLOAD_PROFIT_CENTER
```

```
*INCLUDE FOR ALL THE GLOBAL DECLARATIONS  
INCLUDE ZDEMO_PROFIT_CENTER_TOP.
```

```
*****  
* AT SELECTION-SCREEN ON VALUE-REQUEST *  
*****  
*EVENT TO BE TRIGGERED WHEN WE PRESS F4.
```

```
AT SELECTION-SCREEN ON VALUE-REQUEST FOR PA_FILE.
```

```
PERFORM GET_F4_FOR_FILE USING PA_FILE.
```

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

```
*WE NEED TO MOVE THE PA_FILE INTO ANOTHER VARIABLE OF TYPE STRING  
*AS WE ARE GOING TO USE THE SAME IN THE FM : GUI_UPLOAD THERE THE  
*FILE TYPE IS STRING.
```

```
V_FILE = PA_FILE.
```

```
*UPLOAD THE DATA FROM FLAT FILE TO <ITAB>  
PERFORM UPLOAD_FILE_TO_ITAB USING V_FILE  
CHANGING IT_DATA.
```

```
*OPENS A NEW SESSION , SAME SESSION FOR BOTH  
*COST CENTER AND PROFIT CENTER.
```

Session Method

We Never Compromise in Quality, Would You?

```
PERFORM OPEN_SESSION USING P_GROUP.

*FOR EACH PROFIT CENTER.
LOOP AT IT_DATA INTO WA_DATA.
REFRESH IT_BDCDATA.

*FIRST SCREEN DETAILS
PERFORM FILL_SCREEN_DETAILS USING 'SAPLRKPM' '0200' 'X'.

*PROFIT CENTER
PERFORM FILL_FIELD_DETAILS USING 'PRCT_V-PRCTR' WA_DATA-
PRCTR.
*ENTER
PERFORM FILL_FIELD_DETAILS USING 'BDC_OKCODE' '/00'.

*NEXT SCREEN DETAILS
PERFORM FILL_SCREEN_DETAILS USING 'SAPLRKPM' '0298' 'X'.
*CURSOR DETAILS
PERFORM FILL_FIELD_DETAILS USING 'BDC_CURSOR' 'PRCT_V-KHINR'.
*OKCODE DETAILS
PERFORM FILL_FIELD_DETAILS USING 'BDC_OKCODE' '=MD_ACTIVATE'.
*SUBSCR DETAILS
PERFORM FILL_FIELD_DETAILS USING 'BDC_SUBSCR'
'SAPLRKPM
0300SUBSCREEN_EO'.
*START DATE
PERFORM FILL_FIELD_DETAILS USING 'PRCT_V-DATAB' WA_DATA-
DATAB.
*END DATE
PERFORM FILL_FIELD_DETAILS USING 'PRCT_V-DATBI' WA_DATA-
DATBI.
*NAME
PERFORM FILL_FIELD_DETAILS USING 'PRCT_V-KTEXT' WA_DATA-
KTEXT.
*PERSON RESPONSIBLE
PERFORM FILL_FIELD_DETAILS USING 'PRCT_V-VERAK' WA_DATA-
VERAK.
*HIERARCHIAL AREA
PERFORM FILL_FIELD_DETAILS USING 'PRCT_V-KHINR' WA_DATA-
KHINR.

*INSERT THE SCREEN AND FIELD DETAILS INTO THE SESSION FOR KE51
PERFORM BDC_INSERT USING 'KE51'
IT_BDCDATA.

ENDLOOP.

*BEFORE CLOSING THE SESSION , WE NEED TO INSERT
*NEXT TRANSACTION DETAILS INTO THE SAME SESSION
*****
* TO MAKE IT CLEAR WE DELCARE ALL THE RALATED THINGS *
```

Session Method

We Never Compromise in Quality, Would You?

```
* TO UPLOAD THE COST CENTER MASTER DATA, BUT IN          *
* REAL TIME WE DON'T DELARE THE VARIABLES IN BETWEEN   *
* THE PROGRAM AND IT SHOULD BE ALWAYS AT THE BEGINNING*
* OF THE PROGRAM.                                     *
*****
```

```
DATA : BEGIN OF WA_KS01,
      KOKRS TYPE KOKRS, "CONTROLLING AREA
      KOSTL TYPE KOSTL, "COST CENTER
      DATAB TYPE DATAB, "START DATE
      DATBI TYPE DATBI, "END DATE
      KTEXT TYPE KTEXT, "NAME
      LTEXT TYPE LTEXT, "DESCRIPTION
      VERAK TYPE VERAK, "PERSON RESPONSIBLE
      KOSAR TYPE KOSAR, "COST CENTER CATEGORY
      KHINR TYPE KHINR, "HIERARCHY AREA
      END OF WA_KS01.

DATA : IT_KS01 LIKE TABLE OF WA_KS01.
```

```
*CONSTANTS
CONSTANTS : C_KS01(4) TYPE C VALUE 'KS01',
             C_X(1)    TYPE C VALUE 'X',
             C_A(1)    TYPE C VALUE 'A'.
```

```
SELECTION-SCREEN BEGIN OF BLOCK B2 WITH FRAME TITLE TEXT-001.
PARAMETER : PA_KS01  LIKE FC03TAB-PL00_FILE OBLIGATORY .
SELECTION-SCREEN END OF BLOCK B2.
```

```
*WE NEED TO MOVE THE PA_FILE INTO ANOTHER VARIABLE OF TYPE STRING
*AS WE ARE GOING TO USE THE SAME IN THE FM : GUI_UPLOAD THERE THE
*FILE TYPE IS STRING.
V_FILE = PA_KS01.
```

```
*UPLOAD THE FILE DATA INTO ITAB
CALL FUNCTION 'GUI_UPLOAD'
  EXPORTING
    FILENAME           = V_FILE
    HAS_FIELD_SEPARATOR = 'X'
  TABLES
    DATA_TAB           = IT_KS01.
```

```
*FILL THE SCREEN AND FIELD DETAILS
LOOP AT IT_KS01 INTO WA_KS01.
```

```
REFRESH IT_BDCDATA.
```

```
*FIRST SCREEN DETAILS
PERFORM FILL_SCREEN_DETAILS USING 'SAPLKMA1' '0200' 'X'.
```

```
*CURSOR DETAILS
PERFORM FILL_FIELD_DETAILS USING 'BDC_CURSOR' 'CSKSZ-KOKRS'.
```

```
*OKCODE DETAILS
PERFORM FILL_FIELD_DETAILS USING 'BDC_OKCODE' '/00'.
```

Session Method

We Never Compromise in Quality, Would You?

```
**CONTROLLING AREA
    PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-KOKRS' WA_KS01-
KOKRS.

*COST CENTER DETAILS
    PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-KOSTL' WA_KS01-KOSTL.

*START DATE
    PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-DATAB_ANFO' WA_KS01-
DATAB.

*END DATE
    PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-DATBI_ANFO' WA_KS01-
DATBI.

*NEXT SCREEN DETAILS
    PERFORM FILL_SCREEN_DETAILS USING 'SAPLKMA1' '0299' 'X'.

*OKCODE DETAILS
    PERFORM FILL_FIELD_DETAILS USING 'BDC_OKCODE' '=BU'.

*SUBSCR FIELD DETAILS
    PERFORM FILL_FIELD_DETAILS USING 'BDC_SUBSCR' 'BDC_SUBSCR'.

*CURSOR DETAILS
    PERFORM FILL_FIELD_DETAILS USING 'BDC_CURSOR' 'CSKSZ-WAERS'.

*NAME
    PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-KTEXT' WA_KS01-KTEXT.

*DESCRIPTION
    PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-LTEXT' WA_KS01-LTEXT.

*PERSON RESPONSIBLE
    PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-VERAK' WA_KS01-VERAK.

*COST CENTER CATEGORY
    PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-KOSAR' WA_KS01-KOSAR.

*HIERARCHY AREA
    PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-KHINR' WA_KS01-KHINR.

*CURRENCY KEY
    PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-WAERS' 'INR'.

*INSERT THE SCREEN AND FIELD DETAILS INTO THE SESSION FOR KS01
    PERFORM BDC_INSERT USING 'KS01'
        IT_BDCDATA.
    REFRESH IT_BDCDATA.
ENDLOOP. "END OF IT_KS01

*CLOSE THE SESSION
```

Session Method

We Never Compromise in Quality, Would You?

PERFORM CLOSE_SESSION.

```
*****  
*      DEFINITIONS OF ALL THE ABOVE SUBROUTINES      *  
*****
```

```
*&-----  
*&      Form  FILL_SCREEN_DETAILS  
*&-----  
*      FILL SCREEN AND FIELD DETAILS  
*-----  
FORM FILL_SCREEN_DETAILS USING PROGRAM DYNPRO DYNBEGIN.  
    WA_BDCDATA-PROGRAM = PROGRAM.  
    WA_BDCDATA-DYNPRO = DYNPRO.  
    WA_BDCDATA-DYNBEGIN = DYNBEGIN.  
    APPEND WA_BDCDATA TO IT_BDCDATA.  
    CLEAR WA_BDCDATA.  
ENDFORM.          " FILL_SCREEN_DETAILS  
*&-----  
-----*  
*&      Form  FILL_FIELD_DETAILS  
*&-----  
-----*  
*      FILL FIELD DETAILS  
*-----  
-----*  
FORM FILL_FIELD_DETAILS USING FNAM_FVAL.  
    WA_BDCDATA-FNAM = FNAM.  
    WA_BDCDATA-FVAL = FVAL.  
    APPEND WA_BDCDATA TO IT_BDCDATA.  
    CLEAR WA_BDCDATA.  
ENDFORM.          " FILL_FIELD_DETAILS  
*&-----  
-----*  
*&      Form  UPLOAD_FILE_TO_ITAB  
*&-----  
-----*  
*      SUBROUTINE TO UPLOAD THE DATA FROM PRE.SERVER FILE TO  
ITAB  
*-----  
-----*  
*      -->P_V_FILE - FILE NAME  
*      <--P_IT_DATA - INTERNAL TABLE TO STORE THE DATA  
*-----  
-----*  
FORM UPLOAD_FILE_TO_ITAB USING     FP_V_FILE  
                           CHANGING FP_IT_DATA LIKE IT_DATA.  
  
CALL FUNCTION 'GUI_UPLOAD'  
  EXPORTING  
    FILENAME           = FP_V_FILE  
    HAS_FIELD_SEPARATOR = 'X'
```

Session Method

We Never Compromise in Quality, Would You?

```
TABLES
      DATA_TAB          = FP_IT_DATA.

ENDFORM.                      " UPLOAD_FILE_TO_ITAB

*&-----*
*&     Form  OPEN_SESSION
*-----*
*           Creates the session
*-----*
*     -->P_P_GROUP  Name of the session
*-----*
FORM OPEN_SESSION USING      FP_P_GROUP.

*CALL THE FUNCTION MODEU BDC_OPEN_GROUP
  CALL FUNCTION 'BDC_OPEN_GROUP'
    EXPORTING
      CLIENT = SY-MANDT
      GROUP  = FP_P_GROUP
      KEEP    = 'X'
      USER   = SY-UNAME
IF SY-SUBRC = 0.
  WRITE :'/PROCESS THE SESSION', FP_P_GROUP, 'USING SM35'.
ENDIF.

ENDFORM.                      " OPEN_SESSION
*-----*
*&     Form  GET_F4_FOR_FILE
*-----*
*           DISPLAY ALL FILES IN THE SYSTEM FOR SELECTION
*-----*
*     -->P_PA_FILE  NAME OF THE FILE
*-----*
FORM GET_F4_FOR_FILE USING    P_PA_FILE.

  CALL FUNCTION 'KD_GET_FILENAME_ON_F4'
    EXPORTING
      FIELD_NAME = 'PA_FILE'
    CHANGING
      FILE_NAME  = PA_FILE.
IF SY-SUBRC <> 0.
  MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
    WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
ENDIF.

ENDFORM.                      " GET_F4_FOR_FILE
*-----*
*&     Form  CLOSE_SESSION
```

Session Method

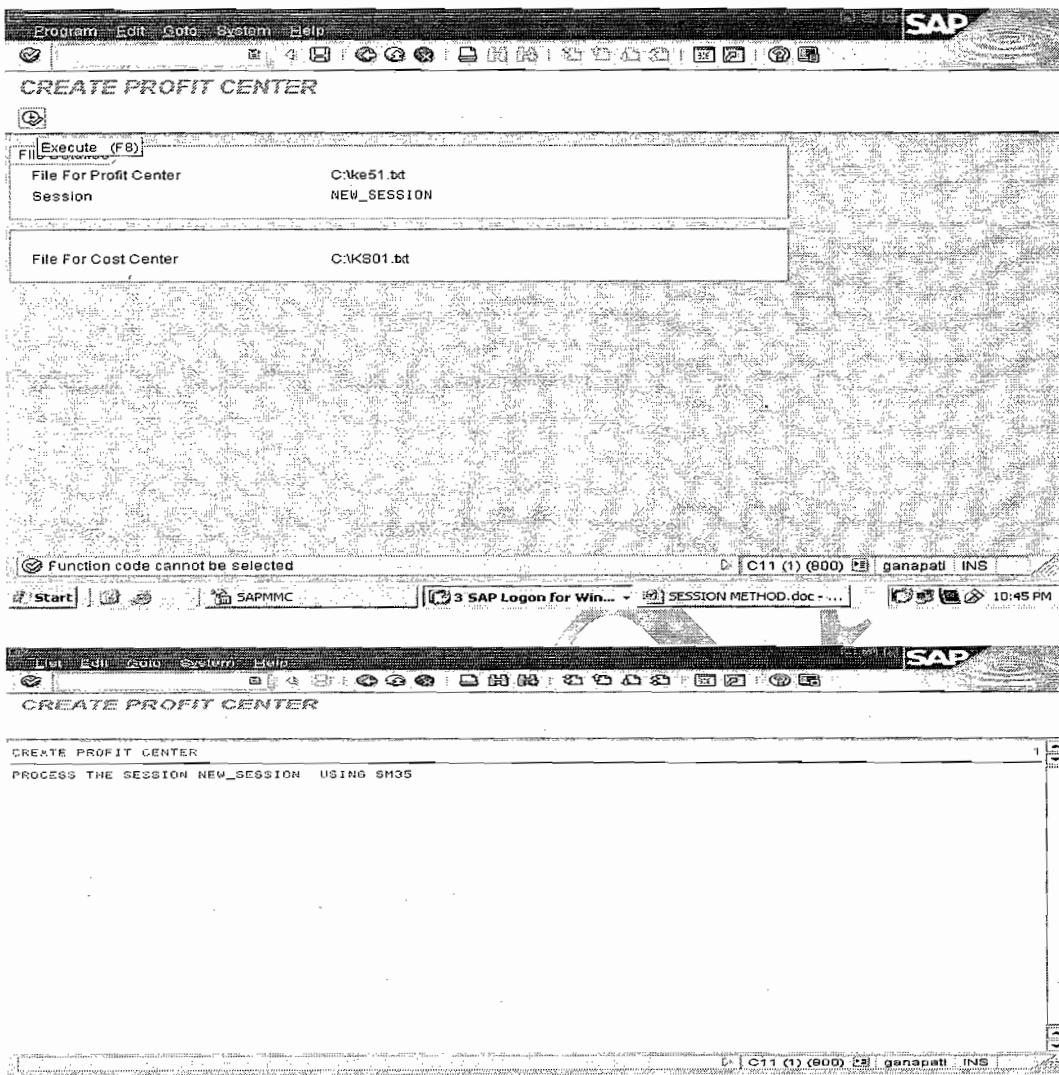
We Never Compromise in Quality, Would You?

```
*&-----  
-----*  
*      TO CLOSE THE ACTIVE SESSION  
*-----  
-----*  
FORM CLOSE_SESSION.  
  
*CALL FUCTION MODULE 'BDC_COLSE_GROUP'  
  CALL FUNCTION 'BDC_CLOSE_GROUP'  
* EXCEPTIONS  
*   NOT_OPEN          = 1  
*   QUEUE_ERROR        = 2  
*   OTHERS             = 3  
  
  IF SY-SUBRC <> 0.  
    MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO  
    WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.  
  ENDIF.  
ENDFORM.                                     " CLOSE_SESSION  
  
*&-----  
-----*  
* &     Form  BDC_INSERT  
* &-----  
-----*  
*     text  
*-----  
-----*  
*     -->P_TCODE  text  
*     -->P_IT_BDCDATA text  
*-----  
-----*  
FORM BDC_INSERT USING      FP_TCODE TYPE SYTCODE  
                      FP_IT_BDCDATA LIKE IT_BDCDATA.  
*CALL FUCTION MODULE 'BDC_INSERT'  
  CALL FUNCTION 'BDC_INSERT'  
    EXPORTING  
      TCODE      = FP_TCODE  
    TABLES  
      DYNPROTAB = FP_IT_BDCDATA.  
  IF SY-SUBRC <> 0.  
    MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO  
    WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.  
  ENDIF.  
ENDFORM.                                     " BDC_INSERT
```

Execute the Program

Session Method

We Never Compromise in Quality, Would You?



The Profit Center Input File is :

A screenshot of a Microsoft Notepad window titled 'IceSick - Notepad'. The window contains the following text:

| | | | |
|-------|----------|----------|-----------|
| 16397 | 13022001 | 30122007 | Revenues |
| 14567 | 13022005 | 30122008 | Furniture |

Session Method

We Never Compromise in Quality, Would You?

Cost Center File:

| ECSOLTHE NOTEPAD | | | | |
|------------------|-------|----------|----------|----------|
| File | Edit | Format | View | Help |
| 0001 | 12336 | 23092004 | 31129999 | Hardware |
| 0001 | 23446 | 23092005 | 31129999 | Software |

Process the Session through SM35 :

Execute SM35.

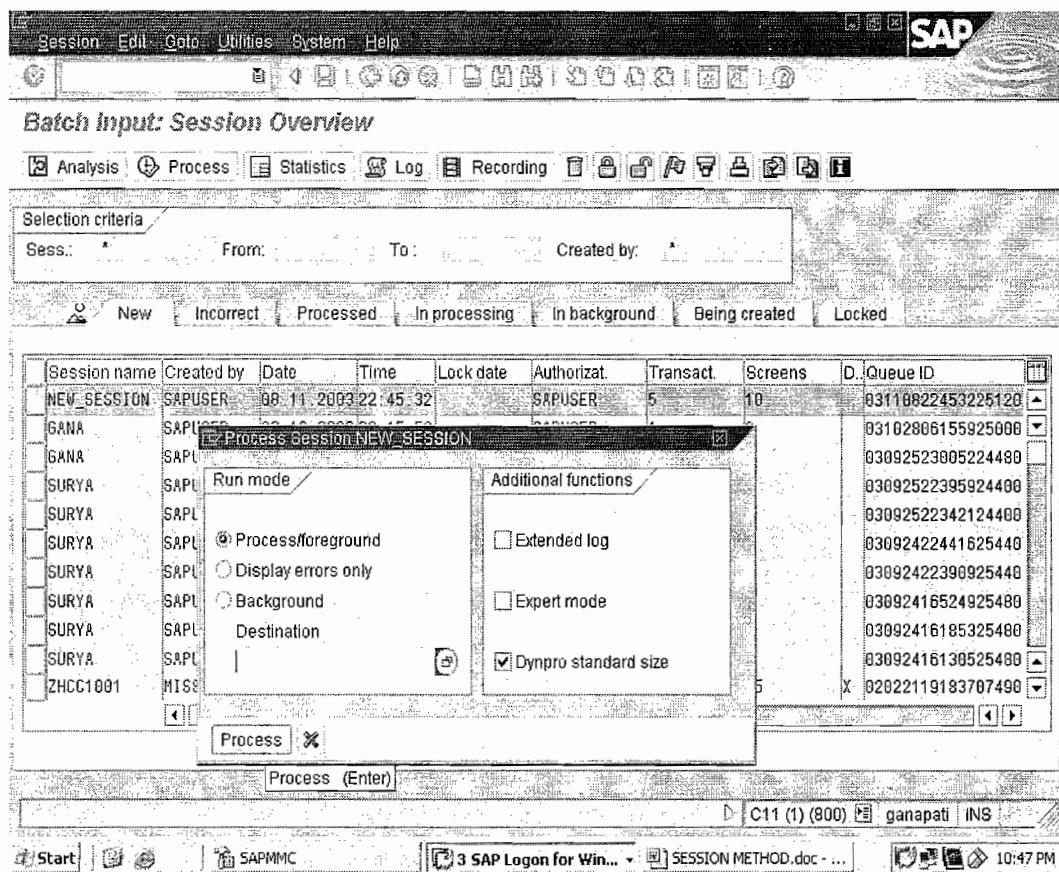
Select the already created session, and click on Process.

The screenshot shows the SAP Session Overview screen. At the top, there's a toolbar with various icons. Below it, a menu bar includes Session, Edit, Goto, Utilities, System, and Help. The main area is titled "Batch Input: Session Overview". A sub-menu bar below the title includes Analysis, Process, Statistics, Log, Recording, and several other icons. A "Selection criteria" section has fields for "Sess.", "From:", "To:", and "Created by:". Below this is a filter bar with buttons for New, Incorrect, Processed, In processing, In background, Being created, and Locked. The main part of the screen is a grid table with columns: Session name, Created by, Date, Time, Lock date, Authorizat., Transact., Screens, D, Queue ID. There are 13 rows of data, each representing a session. The last row is highlighted with a yellow background. At the bottom of the grid, there are navigation buttons (first, previous, next, last) and a scroll bar. The status bar at the bottom right shows "C11 (1) (800) ganpati INS" and the system time "10:47 PM".

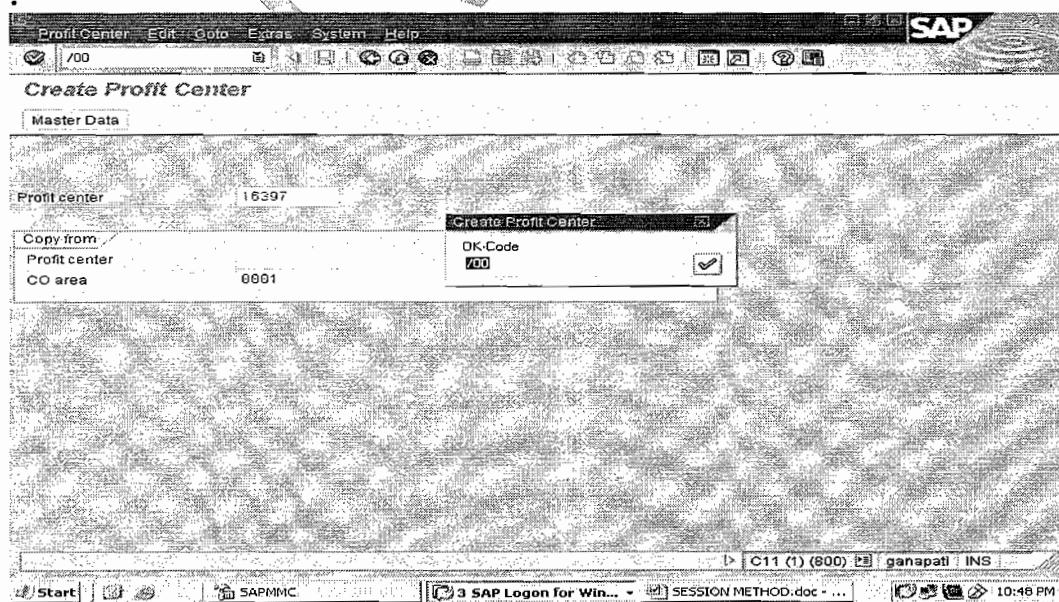
| Session name | Created by | Date | Time | Lock date | Authorizat. | Transact. | Screens | D | Queue ID |
|--------------|------------|------------|----------|-----------|-------------|-----------|---------|---|-------------------|
| NEW SESSION | SAPUSER | 08.11.2003 | 22:45:32 | | SAPUSER | 5 | 10 | | 03110822453225120 |
| GANA | SAPUSER | 28.10.2003 | 06:15:59 | | SAPUSER | 1 | 2 | | 03102806155925000 |
| GANA | SAPUSER | 25.09.2003 | 23:00:52 | | SAPUSER | 1 | 1 | | 03092523005224400 |
| SURYA | SAPUSER | 25.09.2003 | 22:39:59 | | SAPUSER | 0 | 0 | | 03092522395924400 |
| SURYA | SAPUSER | 25.09.2003 | 22:34:21 | | SAPUSER | 1 | 2 | | 03092522342124400 |
| SURYA | SAPUSER | 24.09.2003 | 22:44:16 | | SAPUSER | 1 | 2 | | 03092422441625440 |
| SURYA | SAPUSER | 24.09.2003 | 22:39:09 | | SAPUSER | 0 | 0 | | 03092422390925440 |
| SURYA | SAPUSER | 24.09.2003 | 16:52:49 | | SAPUSER | 2 | 3 | | 03092416524925480 |
| SURYA | SAPUSER | 24.09.2003 | 16:18:53 | | SAPUSER | 1 | 2 | | 03092416185325480 |
| SURYA | SAPUSER | 24.09.2003 | 16:13:05 | | SAPUSER | 1 | 2 | | 03092416130525480 |
| ZHCC1001 | MISSIG | 21.02.2002 | 19:18:37 | | MISSIG | 5 | 15 | X | 02022119183707490 |

Session Method

We Never Compromise in Quality, Would You?



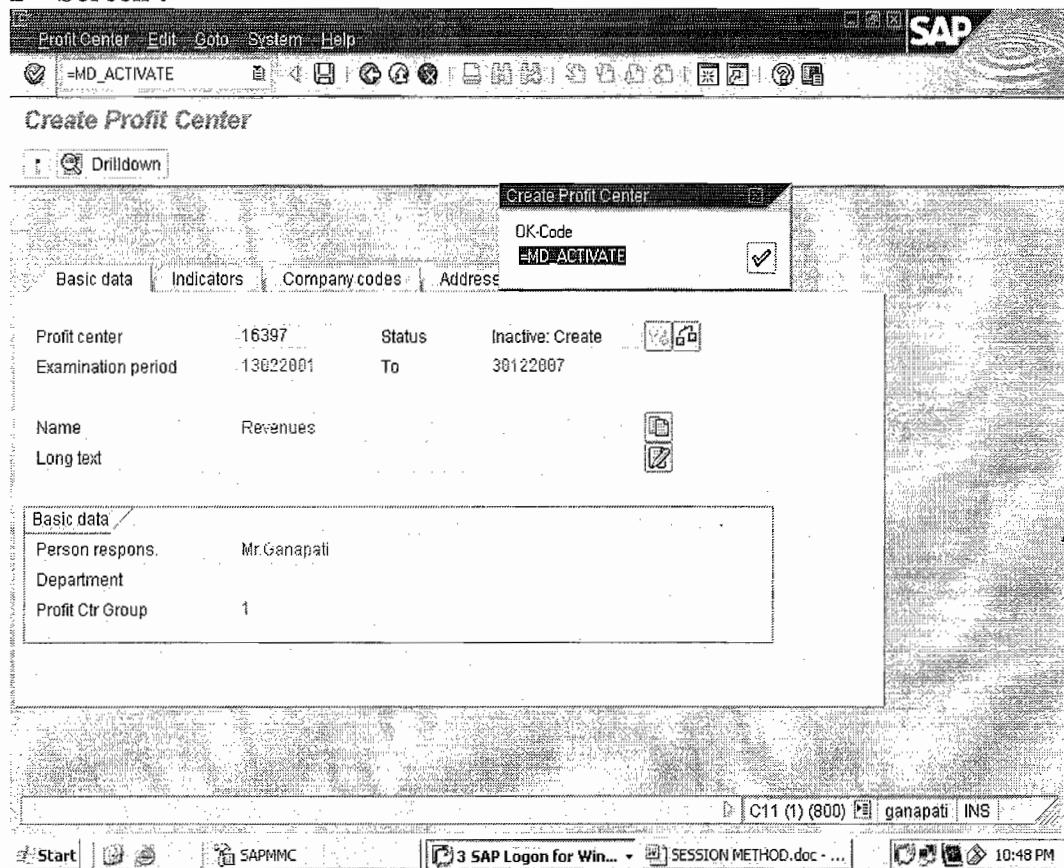
1st Screen:



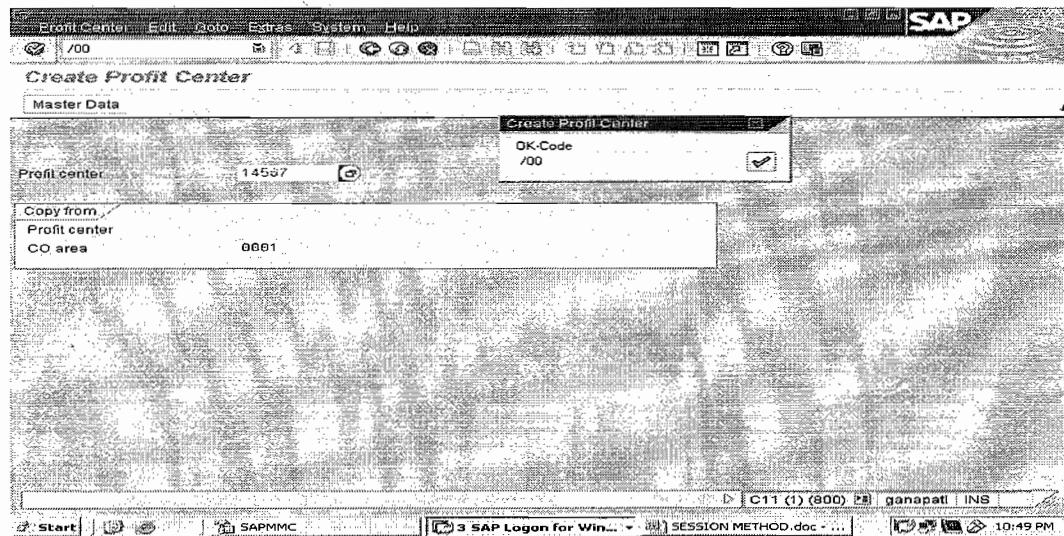
Session Method

We Never Compromise in Quality, Would You?

2nd Screen :

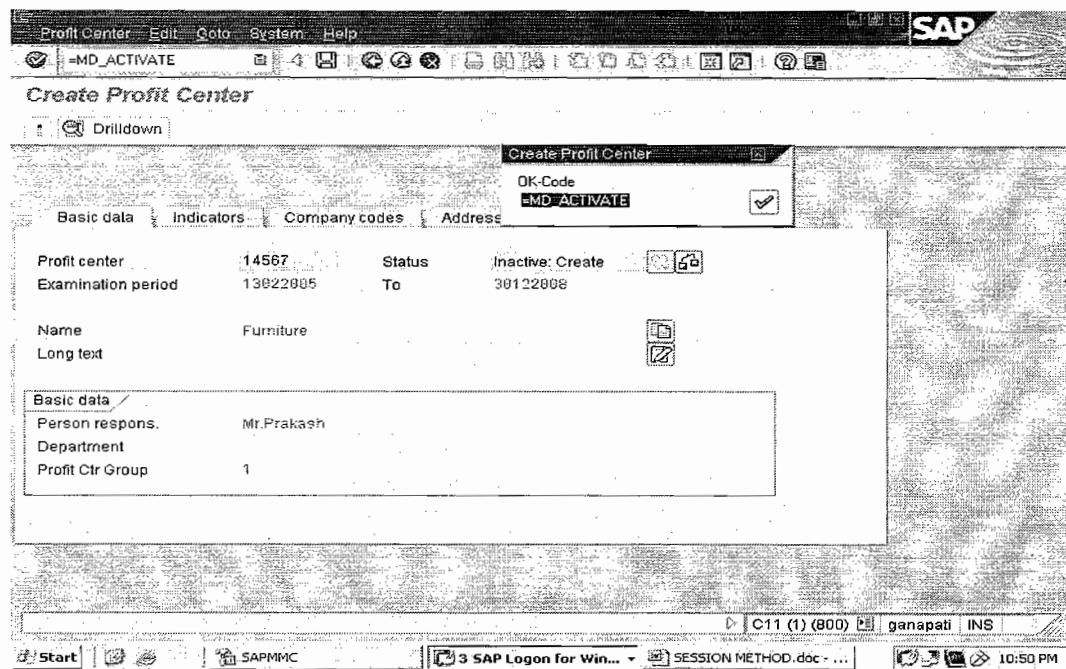


2nd Profit Center Master :

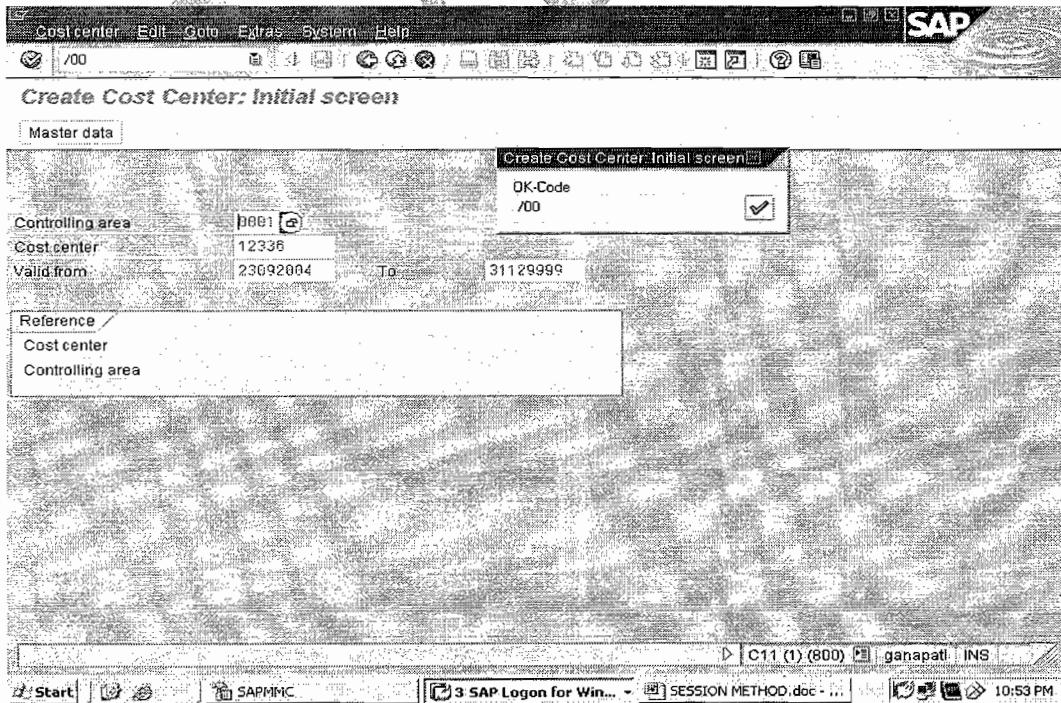


Session Method

We Never Compromise in Quality, Would You?

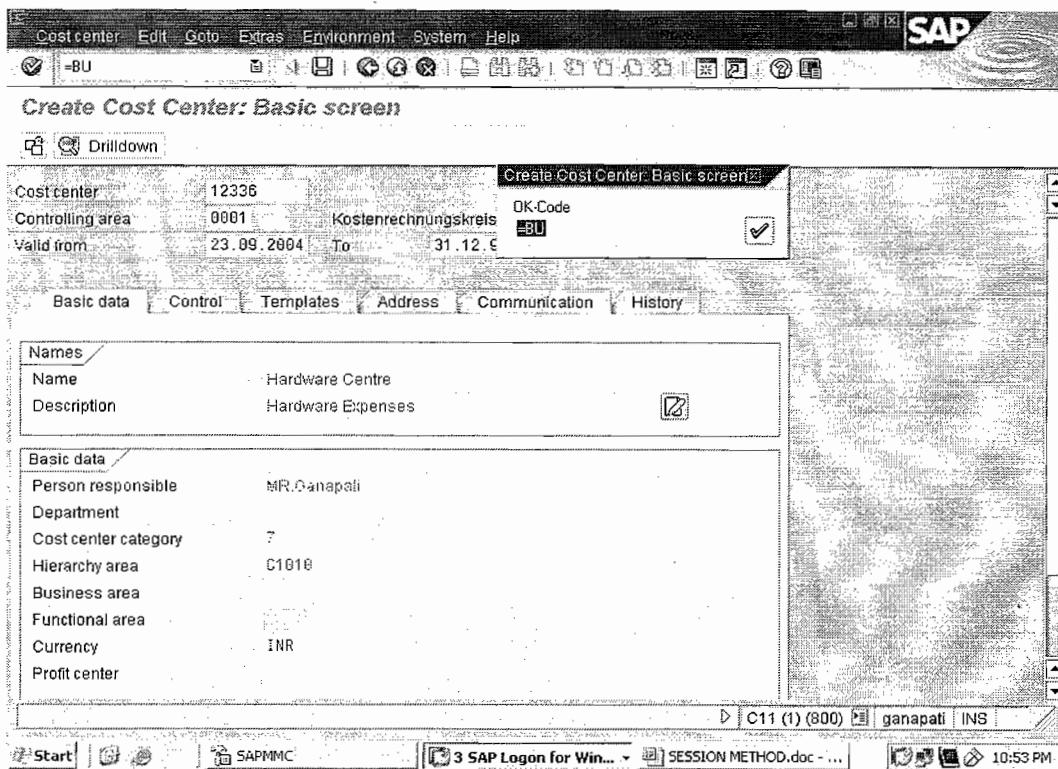


After it Process the 2 records for Profit Center and it continues with the Cost Center Details.

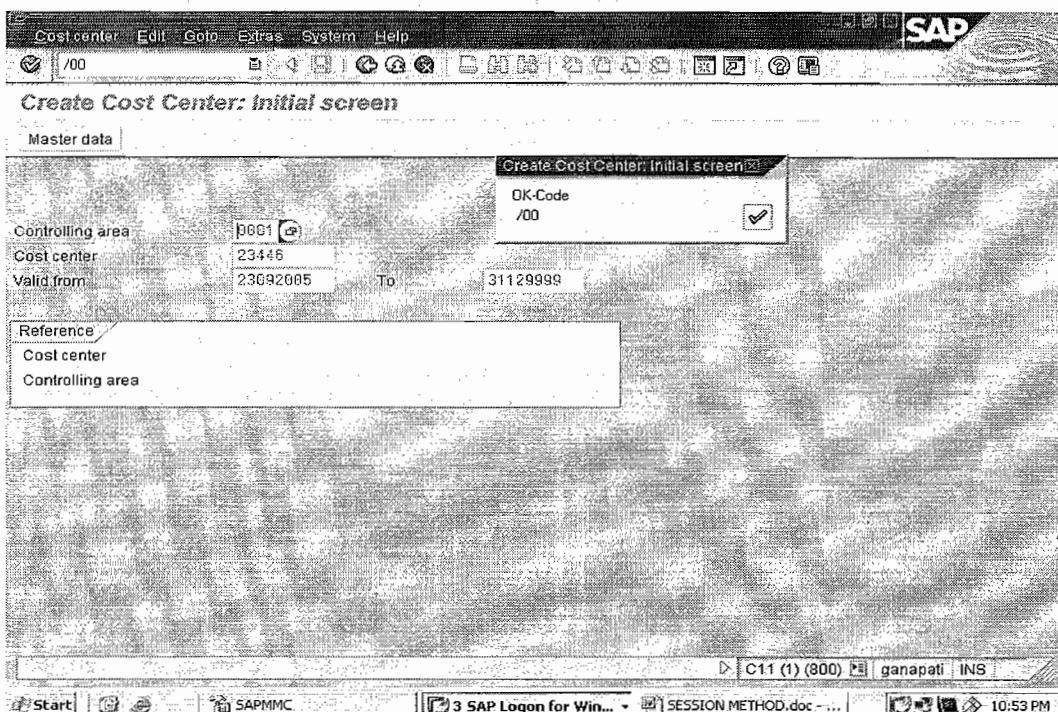


Session Method

We Never Compromise in Quality, Would You?

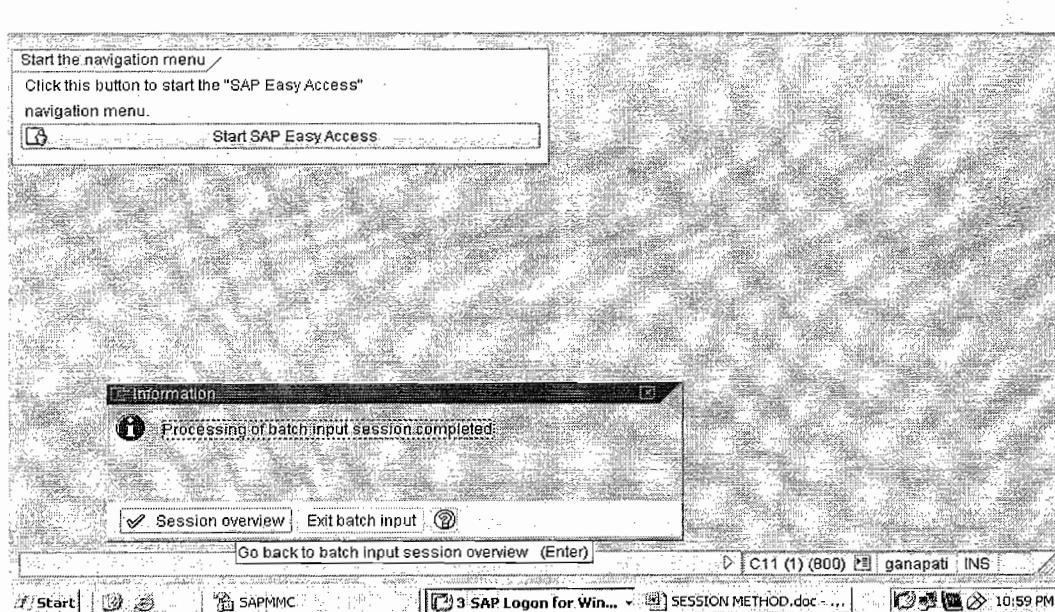
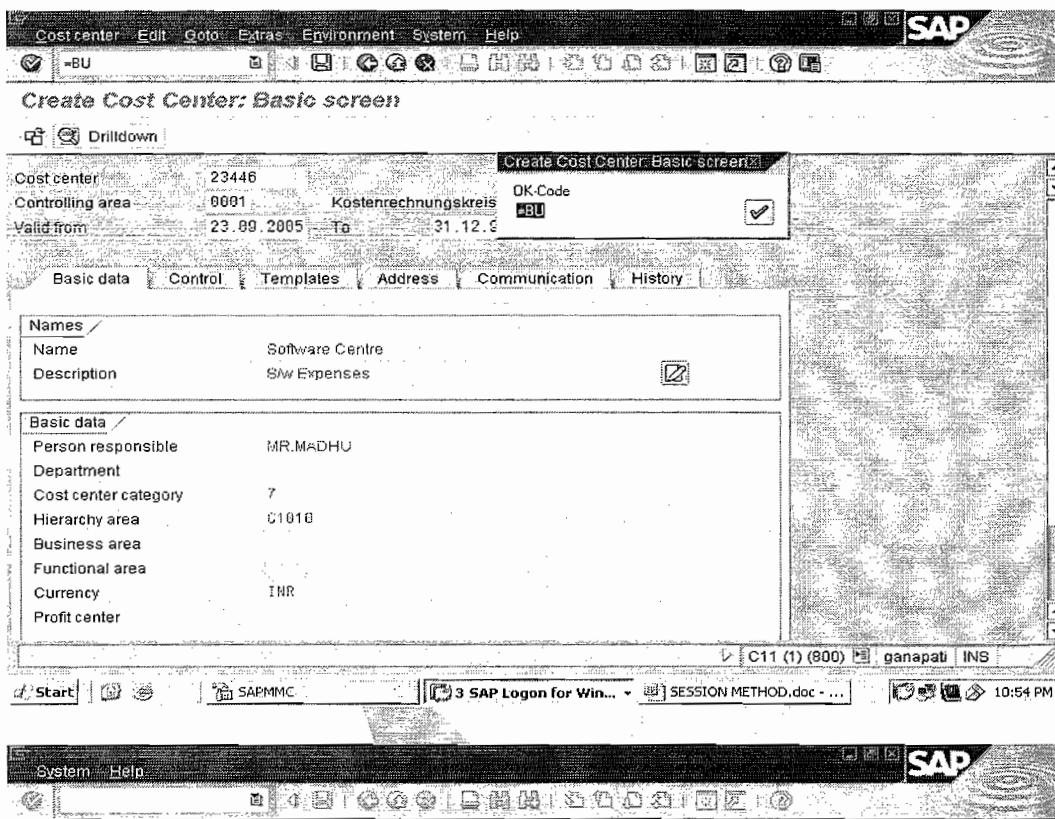


Next Cost Center Details :



Session Method

We Never Compromise in Quality, Would You?



Session Method

We Never Compromise in Quality, Would You?

Batch Input: Session Overview

Selection Analyze session (F2)

| Sess.. | From: | To: | Created by: | New | Incorrect | Processed | In processing | In background | Being created | Locked |
|-------------|---------|---------------------|-------------|-----------|-----------|-----------|---------------|---------------|---------------|--------|
| NEW_SESSION | SAPUSER | 08.11.2003 22:59:03 | SAPUSER | Processed | | 4 | 8 | | | |
| NEW_SESSION | SAPUSER | 08.11.2003 22:56:42 | SAPUSER | Errors | | 5 | 10 | | | |
| NEW_SESSION | SAPUSER | 08.11.2003 22:45:32 | SAPUSER | Errors | | 5 | 10 | | | |
| MY_SESSION | SAPUSER | 07.11.2003 22:40:47 | SAPUSER | Processed | | 4 | 8 | | | |
| MY_SESSION | SAPUSER | 06.11.2003 19:10:48 | SAPUSER | Processed | | 1 | 2 | | | |
| MY_SESSION | SAPUSER | 29.10.2003 11:03:08 | SAPUSER | Processed | | 1 | 2 | | | |
| NEW_SESSION | SAPUSER | 29.10.2003 11:00:06 | SAPUSER | Processed | | 1 | 2 | | | |
| SESSION | SAPUSER | 29.10.2003 10:53:56 | SAPUSER | Processed | | 1 | 2 | | | |
| GANA | SAPUSER | 28.10.2003 06:15:59 | SAPUSER | New | | 1 | 2 | | | |
| GANA | SAPUSER | 25.09.2003 23:17:15 | SAPUSER | Errors | | 1 | 1 | | | |
| GANA | SAPUSER | 25.09.2003 23:00:52 | SAPUSER | New | | 1 | 1 | | | |

C11 (1) (800) ganapati INS
Start SAPMMC 3 SAP Logon for Win... SESSION METHOD.doc - ... 10:59 PM

Analyze session NEW_SESSION

Choose Options Logs

| Index | Trans. | Status |
|-------|--------|-----------|
| 1 | KE51 | Processed |
| 2 | KE51 | Processed |
| 3 | KS01 | Processed |
| 4 | KS01 | Processed |

Session Information

- Name: NEW_SESSION
- Created on: 08.11.2003
- Created at: 22:59:03
- Created by: SAPUSER
- Authorization: SAPUSER
- Locked until: 03110822590325120003

Statistics

| | Transactions | Screens |
|-----------|--------------|---------|
| Total | 4 | 8 |
| incorrect | 0 | 0 |
| Processed | 4 | 8 |
| Deleted | 0 | 0 |

Observe, All the 4 records(2 – Profit center, 2-Cost Center s) are successfully Processed.

C11 (1) (800) ganapati INS

Requirement :

Upload the Cost Center Master Data through Call Transaction , if there are any errors in processing the Call transaction ,

Based On the Radio button Selected to Handle the errors

IF Call Transaction Selected.

 Collect the error Messages and display them.

ELSEIF Session Method Selected.

 INSERT that Error record into the Session .

ENDIF.

Program : ZDEMO_BDC_XD01_HNDALE_ERRORS

```
*****  
* PROGRAM   : ZDEMO_BDC_XD01_HNDALE_ERRORS          *  
* AUTHOR    : GANAPATI . ADIMULAM  
* PURPOSE   : BDC PROGRAM TO UPLOAD ALL THE COST      *  
*              CENTER MASTER DATA INTO SAP USING  
*              CALL TRANSACTION METHOD  
*              PROCESS THE ERRORS EITHER THROUGH SESSION *  
*              OR CALL TRANSACTION FOR SELECTION  
* REFERENCE : NA  
* COPIED FROM : NA  
* TRAPORT REQUEST NO : C11D2K9001  
*****
```

REPORT ZDEMO_BDC_XD01_HNDALE_ERRORS .

```
DATA : BEGIN OF WA_DATA,  
        KOKRS TYPE KOKRS, "CONTROLLING AREA  
        KOSTL TYPE KOSTL, "COST CENTER  
        DATAB TYPE DATAB, "START DATE  
        DATBI TYPE DATBI, "END DATE  
        KTEXT TYPE KTEXT, "NAME  
        LTEXT TYPE LTEXT, "DESCRIPTION  
        VERAK TYPE VERAK, "PERSON RESPONSIBLE  
        KOSAR TYPE KOSAR, "COST CENTER CATEGORY  
        KHINR TYPE KHINR, "HIERARCHY AREA  
    END OF WA_DATA.
```

```
DATA : BEGIN OF WA_ERROR,  
        INDEX      TYPE C,  
        KOKRS      TYPE KOKRS,  
        TEXT(100)   TYPE C,  
    END OF WA_ERROR.
```

DATA : IT_ERROR LIKE TABLE OF WA_ERROR.

```
DATA : IT_DATA LIKE TABLE OF WA_DATA,  
      IT_BDCDATA  LIKE TABLE OF BDCDATA,
```

Session Method

We Never Compromise in Quality, Would You?

```
WA_BDCDATA      LIKE LINE OF IT_BDCDATA,
IT_BDCMSGCOLL  LIKE TABLE OF BDCMSGCOLL,
WA_BDCMSGCOLL  LIKE LINE OF IT_BDCMSGCOLL.
```

```
DATA : V_FILE TYPE STRING,
V_INDEX TYPE I,
V_TEXT(100) TYPE C,
V_SESS(1)  TYPE C.
```

*CONSTANTS

```
CONSTANTS : C_KS01(4) TYPE C VALUE 'KS01',
C_X(1)      TYPE C VALUE 'X',
C_A(1)      TYPE C VALUE 'A'.
```

SELECTION-SCREEN BEGIN OF BLOCK B1 WITH FRAME TITLE TEXT-000.

PARAMETER : PA_FILE LIKE FC03TAB-PL00_FILE OBLIGATORY,
P_GROUP LIKE APQI-GROUPID MODIF ID M1.

SELECTION-SCREEN END OF BLOCK B1.

SELECTION-SCREEN BEGIN OF BLOCK B2 WITH FRAME TITLE TEXT-001.

PARAMETER : R_CALL RADIobutton GROUP G1 USER-COMMAND C1 DEFAULT
'X',

R_SESS RADIobutton GROUP G1.

SELECTION-SCREEN END OF BLOCK B2.

```
*****
```

```
*          AT SELECTION-SCREEN ON VALUE-REQUEST           *
```

```
*****
```

*EVENT TO BE TRIGGERED WHEN WE PRESS F4.

AT SELECTION-SCREEN ON VALUE-REQUEST FOR PA_FILE.

PERFORM GET_F4_FOR_FILE USING PA_FILE.

*MODIFY THE SESSION FIELD FOR THE SELECTED RADIO BUTTON

AT SELECTION-SCREEN OUTPUT.

*IF CALL TRANSACTION SELECTED, TURN THE SESSION NAME IN DISABLE MODE

*ELSEIF SESSION METHOD SELECTED, ENABLE THE SESSION + TURN INTO
*MANDATORY FIELD.

PERFORM MODIFY_SCREEN.

```
*****
```

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

```
*
```

```
*****
```

START-OF-SELECTION.

*WE NEED TO MOVE THE PA_FILE INTO ANOTHER VARIABLE OF TYPE STRING
*AS WE ARE GOING TO USE THE SAME IN THE FM : GUI_UPLOAD THERE THE
*FILE TYPE IS STRING.

Session Method

We Never Compromise in Quality, Would You?

```
V_FILE = PA_FILE.

*UPLOAD THE DATA FROM FLAT FILE TO <ITAB>
  PERFORM UPLOAD_FILE_TO_ITAB USING V_FILE
    CHANGING IT_DATA.

*FILL THE SCREEN AND FIELD DETAILS
  LOOP AT IT_DATA INTO WA_DATA.
    V_INDEX = SY-TABIX.
    REFRESH IT_BDCDATA.

*FIST SCREEN DETAILS
  PERFORM FILL_SCREEN_DETAILS USING 'SAPLKMA1' '0200' 'X'.

*CURSOR DETAILS
  PERFORM FILL_FIELD_DETAILS USING 'BDC_CURSOR' 'CSKSZ-KOKRS'.

*OKCODE DETAILS
  PERFORM FILL_FIELD_DETAILS USING 'BDC_OKCODE' '/00'.

**CONTROLLING AREA
  PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-KOKRS' WA_DATA-KOKRS.

*COST CENTER DETAILS
  PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-KOSTL' WA_DATA-KOSTL.

*START DATE
  PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-DATAB_ANFO' WA_DATA-DATAB.

*END DATE
  PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-DATBI_ANFO' WA_DATA-DATBI.

*NEXT SCREEN DETAILS
  PERFORM FILL_SCREEN_DETAILS USING 'SAPLKMA1' '0299' 'X'.

*OKCODE DETAILS
  PERFORM FILL_FIELD_DETAILS USING 'BDC_OKCODE' '=BU'.

*SUBSCR FIELD DETAILS
  PERFORM FILL_FIELD_DETAILS USING 'BDC_SUBSCR' 'BDC_SUBSCR'.

*CURSOR DETAILS
  PERFORM FILL_FIELD_DETAILS USING 'BDC_CURSOR' 'CSKSZ-WAERS'.

*NAME
  PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-KTEXT' WA_DATA-KTEXT.

*DESCRIPTION
  PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-LTEXT' WA_DATA-LTEXT.

*PERSON RESPONSIBLE
```

Session Method

We Never Compromise in Quality, Would You?

```
PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-VERAK' WA_DATA-VERAK.  
  
*COST CENTER CATEGORY  
    PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-KOSAR' WA_DATA-KOSAR.  
  
*HIERARCHY AREA  
    PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-KHINR' WA_DATA-KHINR.  
  
*CURRENCY KEY  
    PERFORM FILL_FIELD_DETAILS USING 'CSKSZ-WAERS' 'INR'.  
  
*CALL THE TRANSACTION  
  
* MODE : A - ALL SCREENS N - NO SCREENS E - ERROR SCREENS  
ONLY  
* UPDATE : S - SYNCHRONOUS A - ASYNCHRONOUS  
    CALL TRANSACTION C_KS01 USING IT_BDCDATA  
        MODE 'N'  
        UPDATE C_A  
        MESSAGES INTO IT_BDCMSGCOLL.  
  
*IF THE CALL TRANSACTION FAILS  
    IF SY-SUBRC <> 0 .  
  
        IF R_CALL = C_X. "ERRORS THROUGH CALL TRANSACTION  
        LOOP AT IT_BDCMSGCOLL INTO WA_BDCMSGCOLL.  
            CALL FUNCTION 'FORMAT_MESSAGE'  
                EXPORTING  
                    ID      = WA_BDCMSGCOLL-MSGID  
                    LANG   = SY-LANGU  
                    NO     = WA_BDCMSGCOLL-MSGNR  
                    V1     = WA_BDCMSGCOLL-MSGV1  
                    V2     = WA_BDCMSGCOLL-MSGV2  
                    V3     = WA_BDCMSGCOLL-MSGV3  
                    V4     = WA_BDCMSGCOLL-MSGV4  
                IMPORTING  
                    MSG    = V_TEXT  
                EXCEPTIONS  
                    NOT_FOUND = 1  
                    OTHERS    = 2.  
  
        IF SY-SUBRC = 0.  
            WA_ERROR-INDEX  = V_INDEX. "RECORD NO  
            WA_ERROR-TEXT   = V_TEXT. "MESSAGE TEXT  
            WA_ERROR-KOKRS = WA_DATA-KOKRS. "COST CENTER  
            APPEND WA_ERROR TO IT_ERROR.  
        ENDIF.  
    ENDDO.  
    REFRESH IT_BDCMSGCOLL.  
ENDIF.  
IF R_SESS = C_X. "ERRORS THROUGH SESSION  
*INSERT THE DATA INTO THE SESSION  
    CALL FUNCTION 'BDC_INSERT'  
        EXPORTING
```

Session Method

We Never Compromise in Quality, Would You?

```
TCODE      = 'KS01'
TABLES
        DYNPROTAB = IT_BDCDATA.
V_SESS = 'X'.
ENDIF.

ENDIF.
CLEAR WA_BDCDATA.
ENDLOOP.

*CLOSE THE SESSION
CALL FUNCTION 'BDC_CLOSE_GROUP'
* EXCEPTIONS
* NOT_OPEN          = 1
* QUEUE_ERROR       = 2
* OTHERS            = 3

*DISPLAY ERROR MESSAGES
IF NOT IT_ERROR IS INITIAL.
    WRITE : /30 'ERROR RECORDS DETAILS'.
    SKIP 1.
    WRITE : /5 'RECORD NO', 15 SY-VLINE,
            16 'COST CENTER', 25 SY-VLINE, 26 'ERROR MESSAGE'.
    ULINE.
    LOOP AT IT_ERROR INTO WA_ERROR.
        WRITE : /5 WA_ERROR-INDEX LEFT-JUSTIFIED,
                15 SY-VLINE,
                16 WA_ERROR-KOKRS,
                25 SY-VLINE,
                26 WA_ERROR-TEXT.
        CLEAR WA_ERROR.
    ENDLOOP.
ENDIF.

*IF ERRORS THROUGH SESSION
IF V_SESS = 'X'.
    WRITE : / 'ERROR RECORDS ARE INSERTED INTO SESSION'.
ENDIF.

*&-----*
*&     Form FILL_SCREEN_DETAILS
*&-----*
-----*

FORM FILL_SCREEN_DETAILS USING      PROGRAM LIKE      BDCDATA-PROGRAM
                                DYNPRO  LIKE      BDCDATA-DYNPRO
                                DYNBEGIN LIKE      BDCDATA-DYNBEGIN.

CLEAR WA_BDCDATA.
WA_BDCDATA-PROGRAM = PROGRAM.
WA_BDCDATA-DYNPRO = DYNPRO.
WA_BDCDATA-DYNBEGIN = DYNBEGIN.
APPEND WA_BDCDATA TO IT_BDCDATA.
```

Session Method

We Never Compromise in Quality, Would You?

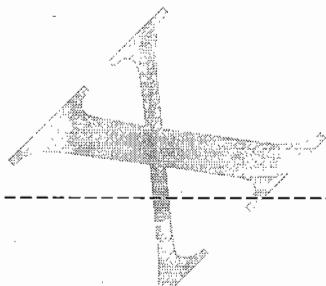
```
ENDFORM.          " FILL_SCREEN_DETAILS
```

```
*&-----  
-----*  
*&      Form  FILL_FIELD_DETAILS  
*&  
-----*
```

```
*      Fill Field Details  
*-----
```

```
-----*  
FORM FILL_FIELD_DETAILS USING      FNAM  
                                FVAL .
```

```
CLEAR WA_BDCDATA.  
WA_BDCDATA-FNAM = FNAM.  
WA_BDCDATA-FVAL = FVAL.  
APPEND WA_BDCDATA TO IT_BDCDATA.  
ENDFORM.
```



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

```
*&      Form  GET_F4_FOR_FILE  
*&  
-----*
```

```
*      DISPLAY ALL FILES IN THE SYSTEM FOR SELECTION  
*-----
```

```
-----*  
*      -->P_PA_FILE  NAME OF THE FILE  
*-----
```

```
-----*  
FORM GET_F4_FOR_FILE USING      P_PAFILE.  
CALL FUNCTION 'KD_GET_FILENAME_ON_F4'  
    EXPORTING  
        FIELD_NAME = 'PAFILE'  
    CHANGING  
        FILE_NAME = PAFILE.  
IF SY-SUBRC <> 0.  
    MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO  
        WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.  
ENDIF.  
ENDFORM.          " GET_F4_FOR_FILE
```

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

```
*&      Form  UPLOAD_FILE_TO_ITAB  
*&  
-----*
```

```
*      SUBROUTINE TO UPLOAD THE DATA FROM PRE.SERVER FILE TO  
ITAB  
*-----
```

```
-----*  
*      -->P_VFILE - FILE NAME  
*      -->P_ITDATA - INTERNAL TABLE TO STORE THE DATA  
*-----
```

Session Method

We Never Compromise in Quality, Would You?

```
FORM UPLOAD_FILE_TO_ITAB USING      FP_V_FILE
                           CHANGING FP_IT_DATA LIKE IT_DATA.

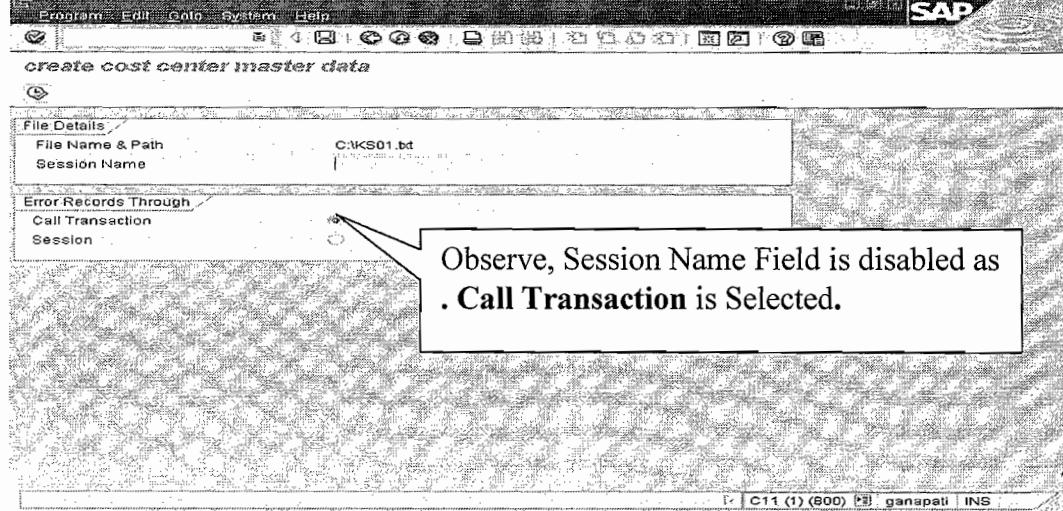
   CALL FUNCTION 'GUI_UPLOAD'
      EXPORTING
         FILENAME           = FP_V_FILE
         HAS_FIELD_SEPARATOR = 'X'
      TABLES
         DATA_TAB          = FP_IT_DATA.

ENDFORM.                      " UPLOAD_FILE_TO_ITAB
*&-----
-----*
*&     Form MODIFY_SCREEN
*&-----
-----*
*     MODIFY THE SCREEN FOR SESSION NAME
*-----
-----*
```

FORM MODIFY_SCREEN.
LOOP AT SCREEN.
 IF SCREEN-NAME = 'P_GROUP'.
 IF R_CALL = 'X'.
 SCREEN-INPUT = 0.
 ELSEIF R_SESS = 'X'.
 SCREEN-INPUT = 1.
 ENDIF.
 ENDIF.
 MODIFY SCREEN.
ENDLOOP.
ENDFORM. " MODIFY_SCREEN

Execute the Program : ZDEMO_BDC_XD01_HNDALE_ERRORS.

Selection Screen :



Session Method

We Never Compromise in Quality, Would You?

Data in Input File :

| DATA | | | | |
|------|-------|----------|----------|----------|
| File | Edit | Format | View | Help |
| 0001 | 15736 | 23092004 | 31129999 | Hardware |
| 0001 | 23446 | 23092005 | 31129999 | Software |

Cost Center 23446 is Already Created, So that
It should be displayed as Error Record:

Output: (Handle Errors through Call transaction)

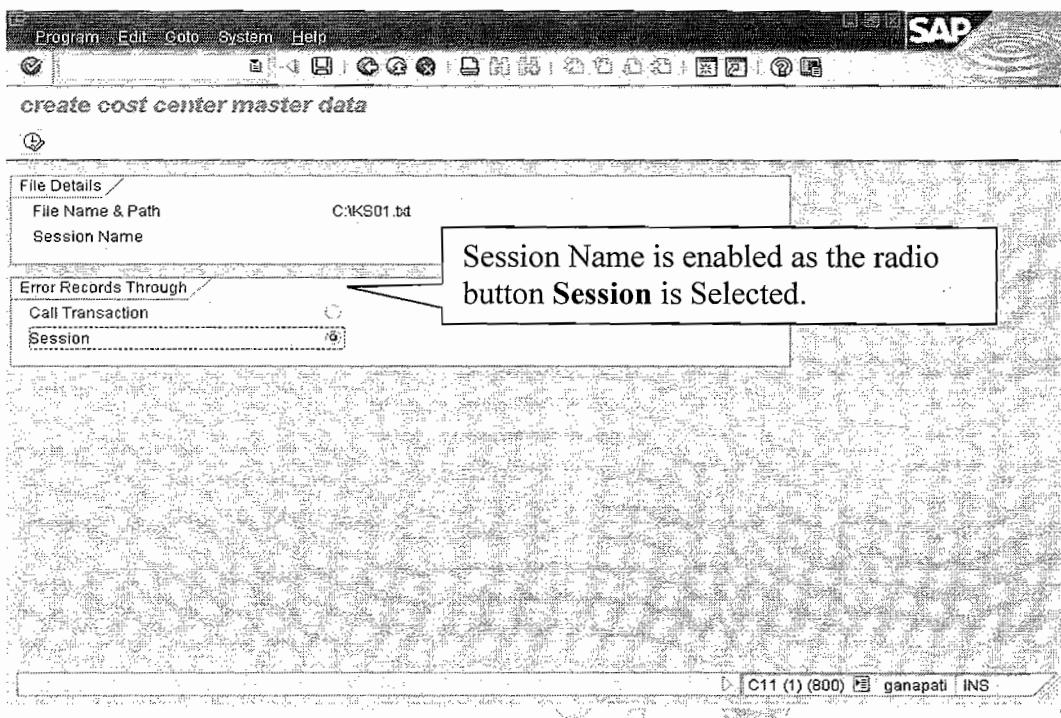
| create cost center master data | | |
|--------------------------------|-----------|--|
| ERROR RECORDS DETAILS | | |
| RECORD NO | COST CENT | ERROR MESSAGE |
| 2 | 23446 | Cost center already exists from 23.09.2005 to 31.12.2005 |

Note :

Execute the same Program and this time INSERT the error records into Session.

Session Method

We Never Compromise in Quality, Would You?



Input File :

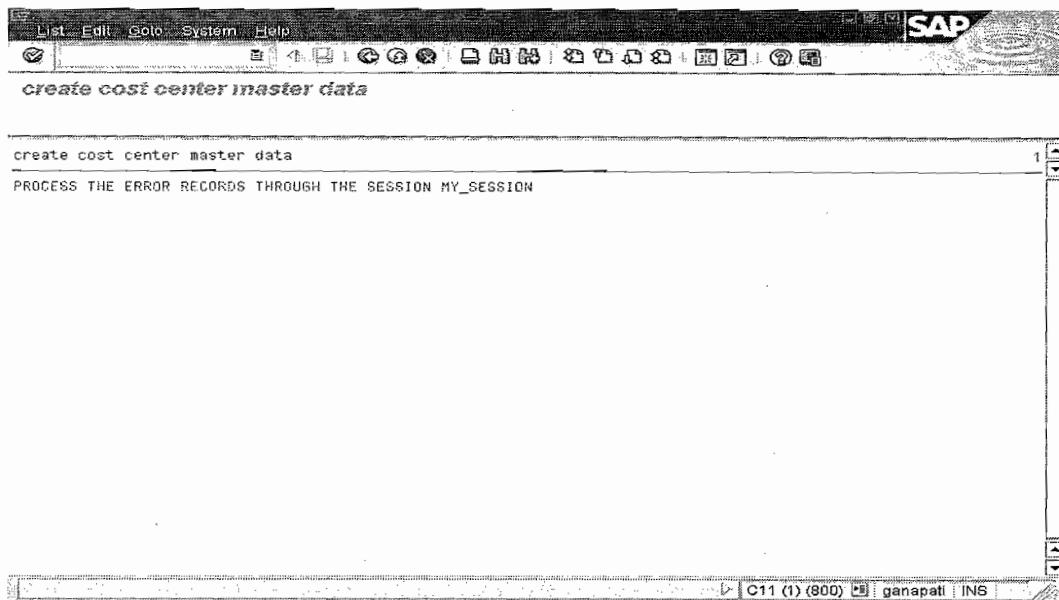
| KS01.txt - Notepad | | | | |
|--------------------|-------|----------|----------|----------|
| File | Edit | Format | View | Help |
| 0001 | 15736 | 23092004 | 31129999 | Hardware |
| 0001 | 23446 | 23092005 | 31129999 | Software |
| 0001 | 17796 | 23092004 | 31129999 | Hardware |
| 0001 | 98672 | 23092005 | 31129999 | Software |

Execute the Program :

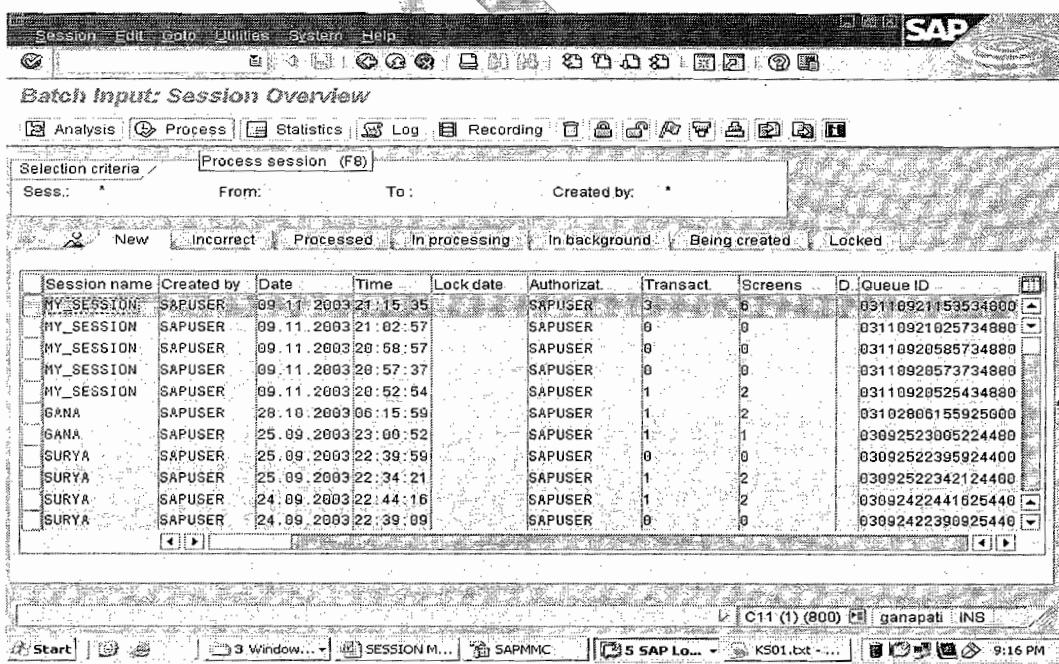
Output :

Session Method

We Never Compromise in Quality, Would You?

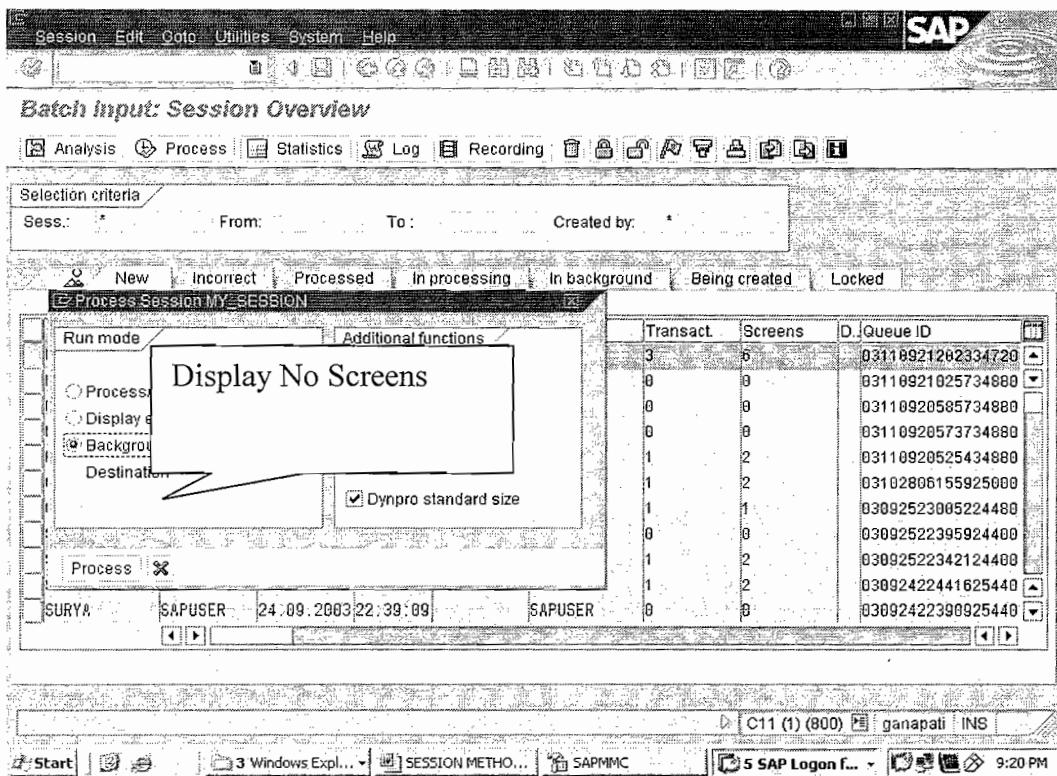


Process the Session :
Execute SM35
Select the Session Name MY_SESSION and Execute.



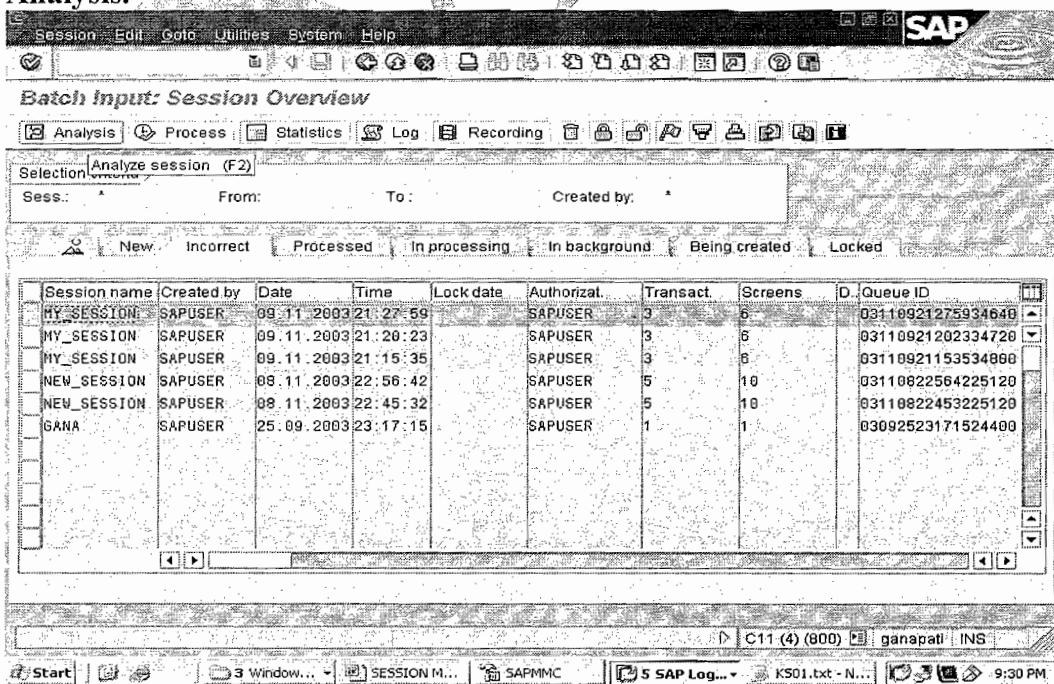
Session Method

We Never Compromise in Quality, Would You?



Click On Process .

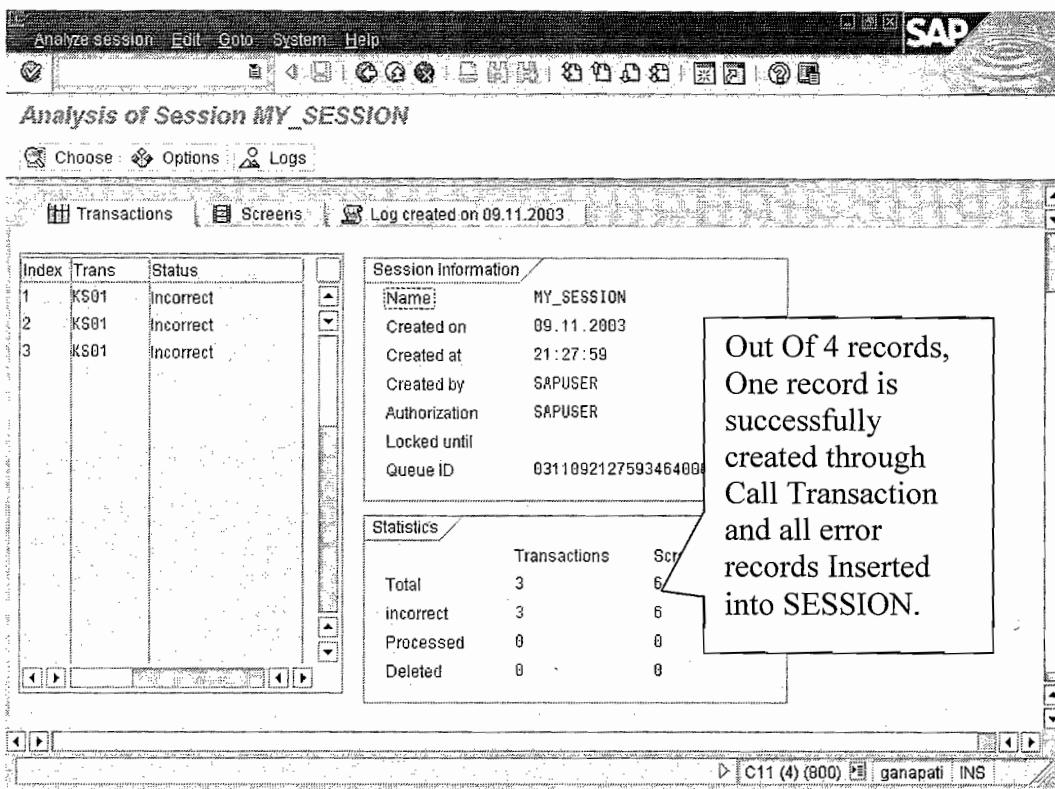
After Processing Select the Session Name(MY_SESSION) and Click On Analysis.



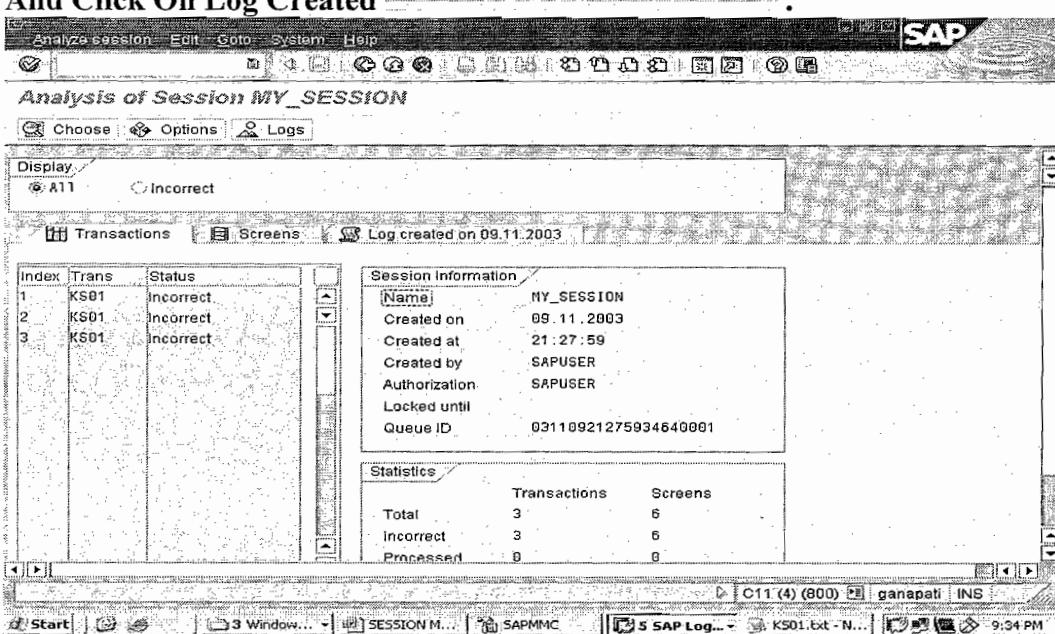
Session Method

We Never Compromise in Quality, Would You?

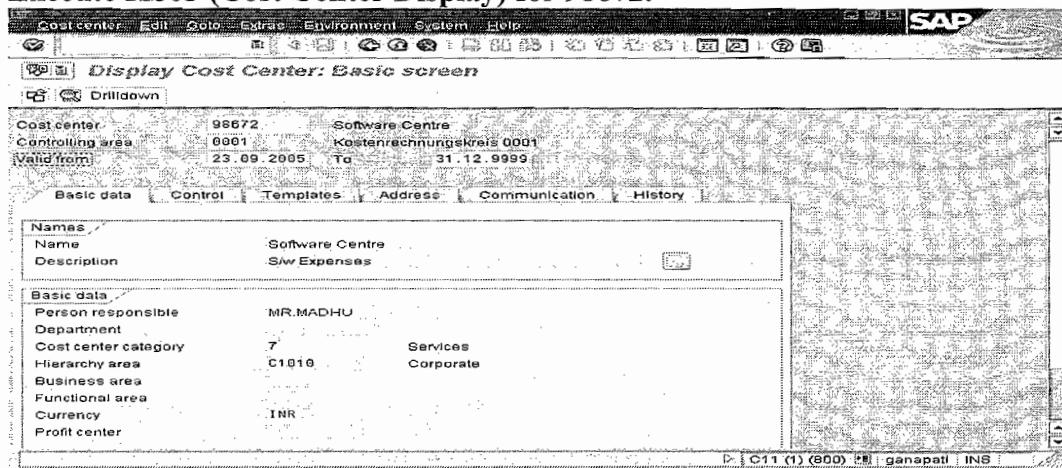
Observe the Error LOG



And Click On Log Created



Check whether the Cost Center 98672 is really created or Not ?
 Execute KS03 (Cost Center Display) for 98672.

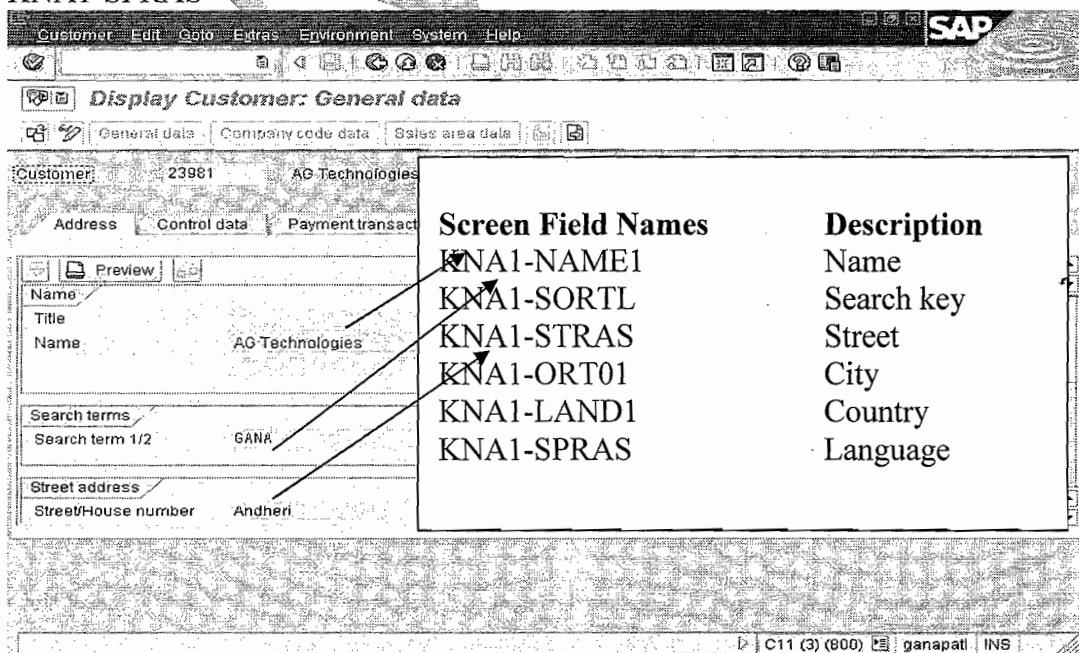


Working with Table Control in BDC :

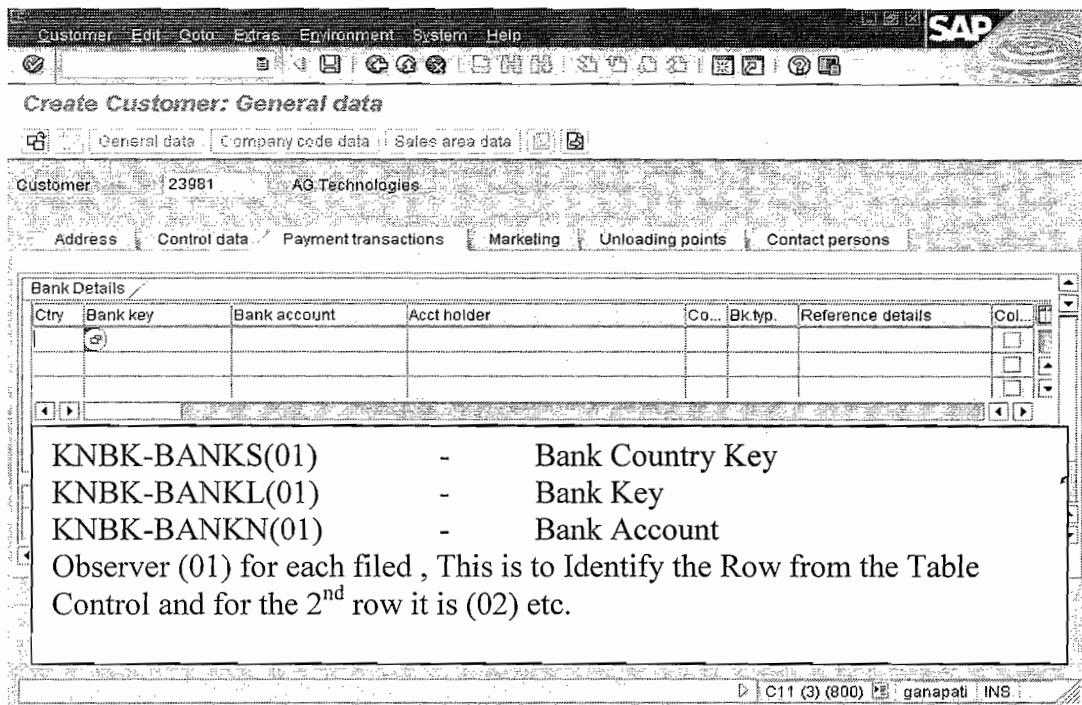
Table Control is used to Input/Out multiple records.

Observe the Screen from Customer Master Creation without Table Control.

KNA1-NAME1
 KNA1-SORTL
 KNA1-STRAS
 KNA1-ORT01
 KNA1-LAND1
 KNA1-SPRAS



Screen with Table Ctrl For Customer Bank Details:



Note : Here in the table control, field names depends on the no of Bank details for each customer , i.e the row no in each field name is Dynamic.

Processing Logic to build the field names in the table Control.

Ex : IT_KNB1 is the table for list of Customer bank details.

```

DATA : V_INDEX(2) TYPE N.
DATA V_FNAM LIKE BDCDATA-FNAM.
LOOP AT IT_BANKS INTO WA_BANKS.

*IS INCREMENTED BY 1 EACH TIME FOR ALL RECORDS
V_INDEX = SY-TABIX. "LOOP COUNTER

*FOR BANKS
CONCATENATE      'KNBK-BANKS(' V_INDEX ')' INTO
V_FNAM.
WA_BDCDATA-FNAM = V_FNAM.
WA_BDCDATA-FVAL = WA_BANKS-BANKS.
APPEND WA_BDCDATA TO IT_BDCDATA.

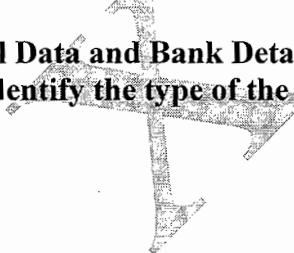
*NOTE : REPEAT THE SAME PROCEDURE FOR ALL FIELDS
ENDLOOP.
  
```

Result of CONCATENATE
 Is KNNK-BANKS(01)
 for 1st record
 KNNK-BANKS(02) for
 2nd record
 Etc..

Session Method

We Never Compromise in Quality, Would You?

Input File :



```
File Edit Format View Help
10,21879,rel,fresh,re1,Ameerpet,HYD,IN,EN
20,21879,DE,20050000,01122334
20,21879,DE,23022200,1122334455
20,21879,DE,23984899,2233445566
```

The file contains both Customer Master General Data and Bank Details.
In each record , the first two characters are to identify the type of the Data.
i.e 10 for Customer Master General and
20 for Customer Bank Details.

Customer General Data File Structure

| Field Position | SAP Field Name | Length | Description | Example |
|----------------|----------------|--------|----------------|------------|
| 1 | ID | 2 | Identification | 10(Header) |
| 2 | KUNNR | 10 | Customer No | 21169 |
| 3 | NAME1 | 35 | Name | Rel.Fresh |
| 4 | SORTL | 10 | Search Key | Rel |
| 5 | STRAS | 35 | Street | Ameerpet |
| 6 | ORT01 | 35 | City | HYD |
| 7 | LAND1 | 4 | Country Key | IN |
| 8 | SPRAS | 2 | Language Key | EN |

Customer Master Bank Details :

| Field Position | SAP Field Name | Length | Description | Example |
|----------------|----------------|--------|------------------|------------------|
| 1 | ID | 2 | Identification | 20(Bank Details) |
| 2 | KUNNR | 10 | Customer No | 21169 |
| 3 | BANKS | 3 | Bank Country Key | DE |
| 4 | BANKL | 15 | Bank Key | 50013050 |
| 5 | BANKN | 18 | Bank Account No | 123456789 |

Session Method

We Never Compromise in Quality, Would You?

**Note : This File structure is helpful to declare the corresponding Internal table and also to SPLIT the file according to the file structure.
Execute the Program : ZDEMO_BDC_USING_TABC_IN_XD01.**

```
REPORT ZDEMO_BDC_USING_TABC_IN_XD01 .  
  
DATA : IT_BDCDATA LIKE TABLE OF BDCDATA,  
      WA_BDCDATA LIKE BDCDATA.  
DATA : IT_BDCMSGCOLL LIKE TABLE OF BDCMSGCOLL,  
      WA_BDCMSGCOLL LIKE BDCMSGCOLL.  
  
DATA : BEGIN OF WA_FILE,  
      TEXT(150) TYPE C,  
      END OF WA_FILE.  
DATA IT_FILE LIKE TABLE OF WA_FILE.  
  
DATA : BEGIN OF WA_KNA1,  
      KUNNR TYPE KUNNR, "CUSTOMER  
      NAME1 TYPE NAME1, "NAME  
      SORTL TYPE SORTL, "SEARCH TERM  
      STRAS TYPE STRAS, "STREET  
      ORT01 TYPE ORT01, "CITY  
      LAND1 TYPE LAND1, "COUNTRY  
      SPRAS TYPE SPRAS, "LANGU  
      END OF WA_KNA1.  
  
DATA IT_KNA1 LIKE TABLE OF WA_KNA1.  
  
DATA : BEGIN OF WA_KNBK,  
      KUNNR TYPE KUNNR,  
      BANKS LIKE KNBK-BANKS, "BANK COUNTRY  
      BANKL LIKE KNBK-BANKL, "BANK KEY  
      BANKN LIKE KNBK-BANKN, "ACCNO  
      END OF WA_KNBK.  
  
DATA IT_KNBK LIKE TABLE OF WA_KNBK.  
DATA V_COUNTER(2) TYPE N.  
DATA V_FNAM LIKE BDCDATA-FNAM.  
DATA V_ID(2).  
  
*****  
*          START-OF-SELECTION.          *  
*****  
START-OF-SELECTION.  
*UPLOAD  
  CALL FUNCTION 'GUI_UPLOAD'  
    EXPORTING  
      FILENAME = 'C:\XD01_KNBK.TXT'  
    TABLES  
      DATA_TAB = IT_FILE.  
*SPLIT THE FILE ITAB AND MOVE TO CORRESPONDING ITABS.  
  LOOP AT IT_FILE INTO WA_FILE.  
    IF WA_FILE+0(2) = '10'.
```

Session Method

We Never Compromise in Quality, Would You?

```
SPLIT WA_FILE AT ','  
    INTO V_ID  
        WA_KNA1-KUNNR  
        WA_KNA1-NAME1  
        WA_KNA1-SORTL  
        WA_KNA1-STRAS  
        WA_KNA1-ORT01  
        WA_KNA1-LAND1  
        WA_KNA1-SPRAS .  
APPEND WA_KNA1 TO IT_KNA1.  
ELSEIF WA_FILE+0(2) = '20'.  
    SPLIT WA_FILE AT ','  
        INTO V_ID  
            WA_KNBK-KUNNR  
            WA_KNBK-BANKS  
            WA_KNBK-BANKL  
            WA_KNBK-BANKN.  
APPEND WA_KNBK TO IT_KNBK.  
ENDIF.  
ENDLOOP.  
  
PERFORM FILL_FIELD_DETAILS USING 'BDC_OKCODE' '/00'  
    CHANGING IT_BDCDATA.  
*KUNNR RF02D-KUNNR  
    PERFORM FILL_FIELD_DETAILS USING 'RF02D-KUNNR' WA_KNA1-KUNNR  
    CHANGING IT_BDCDATA.  
*KTOKD  
    PERFORM FILL_FIELD_DETAILS USING 'RF02D-KTOKD' '0004'  
    CHANGING IT_BDCDATA.  
*2ND SCREEN DETAILS  
    PERFORM FILL_SCREEN_DETAILS USING 'SAPMF02D' '0110' 'X'  
    CHANGING IT_BDCDATA.  
*CURSOR  
    PERFORM FILL_FIELD_DETAILS USING 'BDC_CURSOR' 'KNA1-SPRAS'  
    CHANGING IT_BDCDATA.  
*OK CODE FOR NEXT  
    PERFORM FILL_FIELD_DETAILS USING 'BDC_OKCODE' '=VW'  
    CHANGING IT_BDCDATA.  
    PERFORM FILL_FIELD_DETAILS USING 'KNA1-NAME1' WA_KNA1-NAME1  
    CHANGING IT_BDCDATA.  
    PERFORM FILL_FIELD_DETAILS USING 'KNA1-SORTL' WA_KNA1-SORTL  
    CHANGING IT_BDCDATA.  
    PERFORM FILL_FIELD_DETAILS USING 'KNA1-ORT01' WA_KNA1-ORT01  
    CHANGING IT_BDCDATA.  
    PERFORM FILL_FIELD_DETAILS USING 'KNA1-STRAS' WA_KNA1-STRAS  
    CHANGING IT_BDCDATA.  
    PERFORM FILL_FIELD_DETAILS USING 'KNA1-LAND1' WA_KNA1-LAND1  
    CHANGING IT_BDCDATA.  
    PERFORM FILL_FIELD_DETAILS USING 'KNA1-SPRAS' WA_KNA1-SPRAS  
    CHANGING IT_BDCDATA.  
  
*2ND SCREEN DETAILS
```

Session Method

We Never Compromise in Quality, Would You?

```
PERFORM FILL_SCREEN_DETAILS USING 'SAPMF02D' '0120' 'X'
CHANGING IT_BDCDATA.

*CURSOR
  PERFORM FILL_FIELD_DETAILS USING 'BDC_CURSOR' 'KNA1-LIFNR'
CHANGING IT_BDCDATA.

*OK CODE FOR NEXT
  PERFORM FILL_FIELD_DETAILS USING 'BDC_OKCODE' '=VW'
CHANGING IT_BDCDATA.

*NEXT SCREEN
  PERFORM FILL_SCREEN_DETAILS USING 'SAPMF02D' '0125' 'X'
CHANGING IT_BDCDATA.

*CURSOR
  PERFORM FILL_FIELD_DETAILS USING 'BDC_CURSOR' 'KNA1-NIELS'
CHANGING IT_BDCDATA.

*OK CODE FOR NEXT
  PERFORM FILL_FIELD_DETAILS USING 'BDC_OKCODE' '=VW'
CHANGING IT_BDCDATA.

*TABLE CTRL SCREEN
  PERFORM FILL_SCREEN_DETAILS USING 'SAPMF02D' '0130' 'X'
CHANGING IT_BDCDATA.

*OK CODE FOR SAVE
  PERFORM FILL_FIELD_DETAILS USING 'BDC_OKCODE' '=UPDA'
CHANGING IT_BDCDATA.

  V_COUNTER = 0.

*Handling Table Control
  LOOP AT IT_KNBK INTO WA_KNBK WHERE KUNNR = WA_KNA1-KUNNR.
  V_COUNTER = V_COUNTER + 1.

*BANKS : COUNTRY KEY
  CONCATENATE 'KNBK-BANKS(' V_COUNTER ')' INTO V_FNAM.
  PERFORM FILL_FIELD_DETAILS USING V_FNAM WA_KNBK-BANKS
CHANGING IT_BDCDATA.

  CLEAR : V_FNAM.

* KNBK-BANKL
  CONCATENATE 'KNBK-BANKL(' V_COUNTER ')' INTO V_FNAM.
  PERFORM FILL_FIELD_DETAILS USING V_FNAM WA_KNBK-BANKL
CHANGING IT_BDCDATA.

  CLEAR V_FNAM.

* KNBK-BANKN
  CONCATENATE 'KNBK-BANKN(' V_COUNTER ')' INTO V_FNAM.
  PERFORM FILL_FIELD_DETAILS USING V_FNAM WA_KNBK-BANKN
CHANGING IT_BDCDATA.

ENDLOOP.
CALL TRANSACTION 'XD01' USING IT_BDCDATA MODE 'A'
UPDATE 'S'
MESSAGES INTO IT_BDCMSGCOLL.

REFRESH IT_BDCDATA.
ENDLOOP.
```

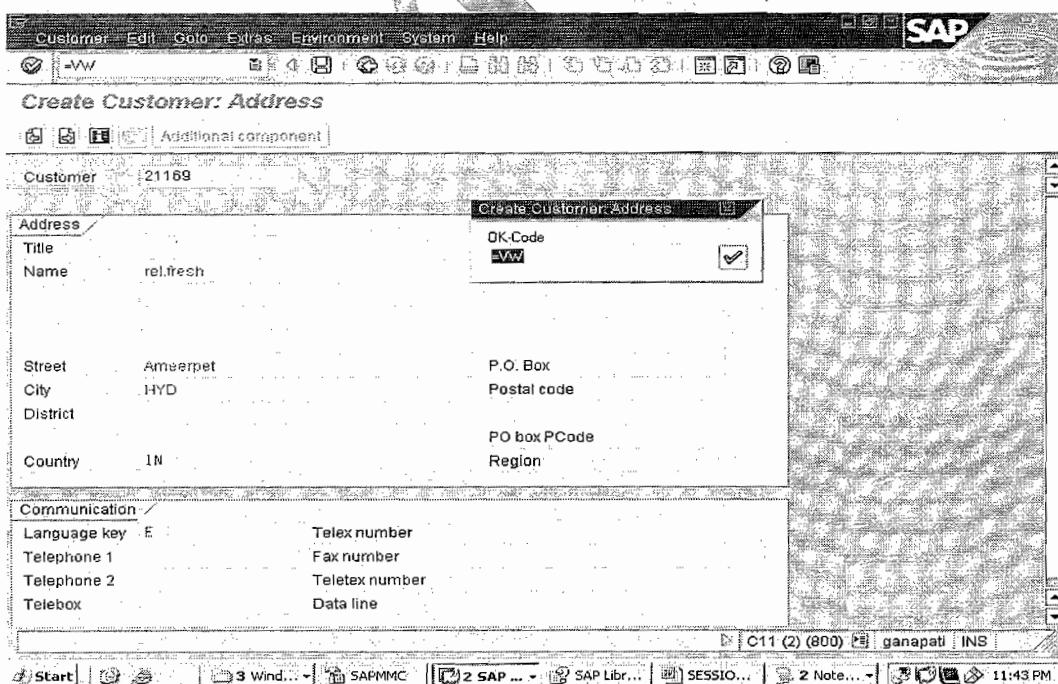
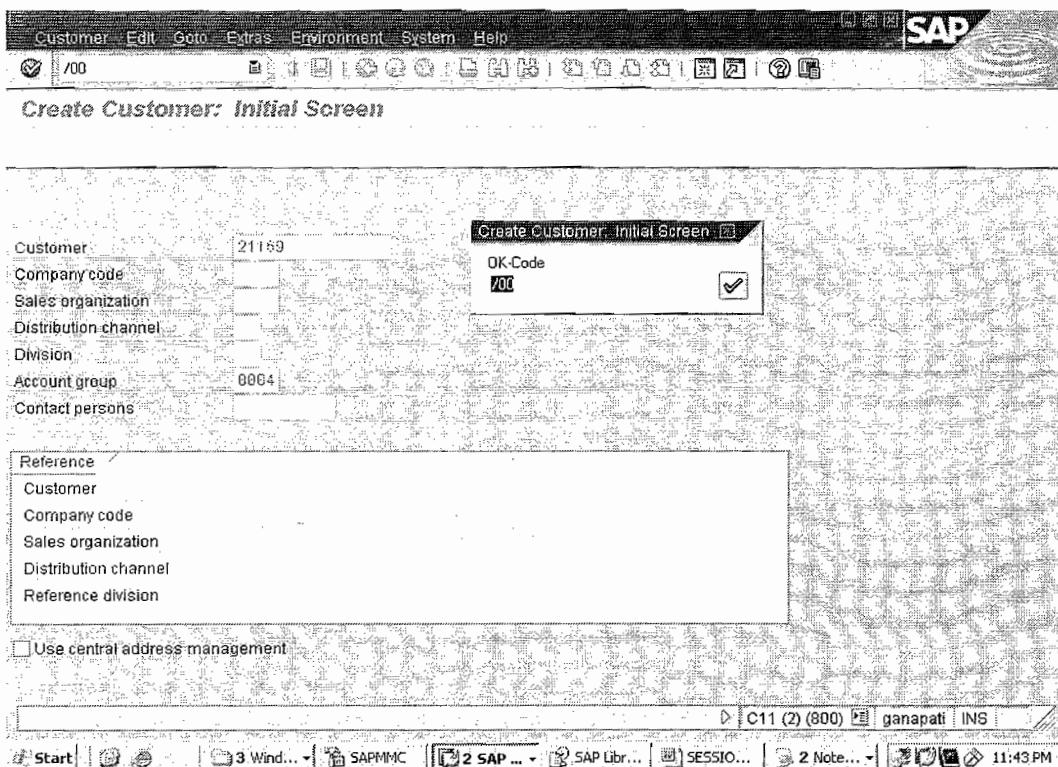
Session Method

We Never Compromise in Quality, Would You?

```
*&-----  
*-----*  
*&      Form  FILL_SCREEN_DETAILS  
*&-----  
-----*  
*      text  
*-----  
-----*  
*      -->P_PROGRAM  text  
*      -->P_DYNPRO   text  
*      -->P_DYNBEGIN  text  
*      <--P_IT_BDCDATA text  
*-----  
-----*  
FORM FILL_SCREEN_DETAILS USING      FP_PROGRAM LIKE BDCDATA-PROGRAM  
                                    FP_DYNPRO  LIKE BDCDATA-DYNPRO  
                                    FP_DYNBEGIN LIKE BDCDATA-  
DYNBEGIN  
                                         CHANGING FP_IT_BDCDATA LIKE IT_BDCDATA.  
  
CLEAR WA_BDCDATA.  
WA_BDCDATA-PROGRAM = FP_PROGRAM.  
WA_BDCDATA-DYNPRO = FP_DYNPRO.  
WA_BDCDATA-DYNBEGIN = FP_DYNBEGIN.  
APPEND WA_BDCDATA TO FP_IT_BDCDATA.  
  
ENDFORM.          " FILL_SCREEN_DETAILS  
*&-----  
-----*  
*&      Form  FILL_FIELD_DETAILS  
*&-----  
-----*  
*      text  
*-----  
-----*  
*      -->P_FNAM  text  
*      -->P_FVAL   text  
*      <--P_IT_BDCDATA text  
*-----  
-----*  
FORM FILL_FIELD_DETAILS USING      FP_FNAM  
                                    FP_FVAL  
                                         CHANGING FP_IT_BDCDATA LIKE IT_BDCDATA.  
  
CLEAR WA_BDCDATA.  
WA_BDCDATA-FNAM = FP_FNAM.  
WA_BDCDATA-FVAL = FP_FVAL.  
APPEND WA_BDCDATA TO FP_IT_BDCDATA.ENDFORM.  
" FILL_FIELD_DETAILS
```

Session Method

We Never Compromise in Quality, Would You?



Next Screen :

Session Method

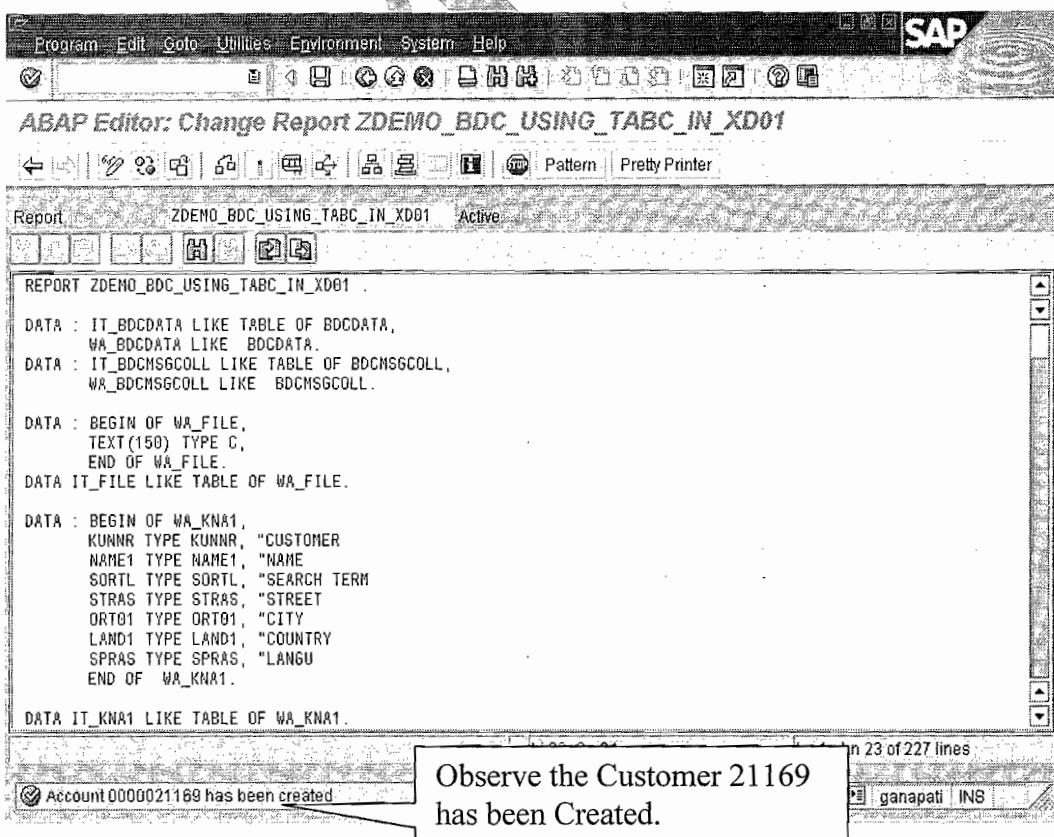
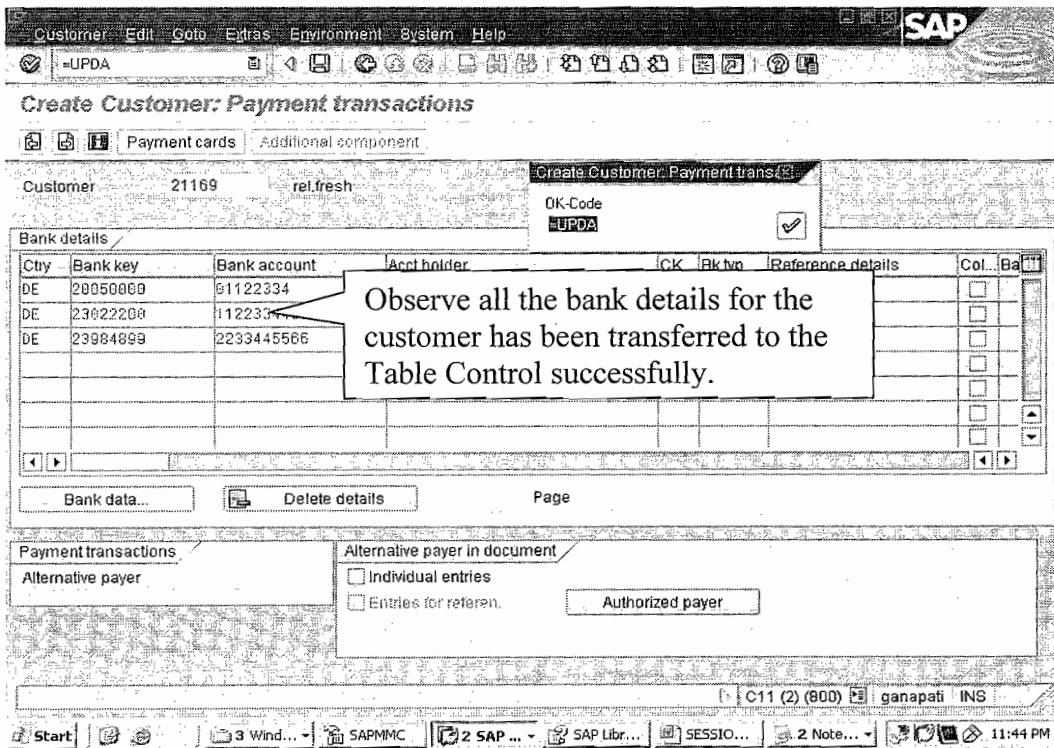
We Never Compromise in Quality, Would You?

The screenshot shows the SAP Customer Control interface. The title bar reads "Customer Control". The main area is titled "Create Customer: Control". The "Customer" field contains "21169" and the "HYD" field is selected. The "Tax categories" and "Additional component" buttons are visible. The "Account control" section includes fields for "Vet" (set to "Create Customer: Control"), "Tra" (set to "OK-Code"), and "WW". The "Tax administration" section contains "Tax code 1" and "Tax code 2" fields, each with checkboxes for "Equalizatn tax", "Natural person", and "Tax on sis/pur.". The "Fiscal address" section includes "County code" and "City code" fields, and "VAT reg.no." and "Jurisdic. code" fields. The "Reference data/area" section includes "Location no. 1" and "Location no. 2" fields, and a "Check digit" button. The bottom status bar shows "C11 (2) (800) ganapati INS" and the system time "11:43 PM".

The screenshot shows the SAP Customer Marketing interface. The title bar reads "Customer Marketing". The main area is titled "Create Customer: Marketing". The "Customer" field contains "21169" and the "HYD" field is selected. The "Other industries" button is visible. The "Classification" section includes fields for "Nle" (set to "Create Customer: Marketing"), "Cu" (set to "OK-Code"), and "Ind" (set to "WW"). The "Industry codes" button is present. The "Key figures" section includes "Annual sales" and "Employees" fields, and a "Fi.year variant" button. The "Sales prospecting" section includes a "Legal status" button. The bottom status bar shows "C11 (2) (800) ganapati INS" and the system time "11:43 PM".

Session Method

We Never Compromise in Quality, Would You?

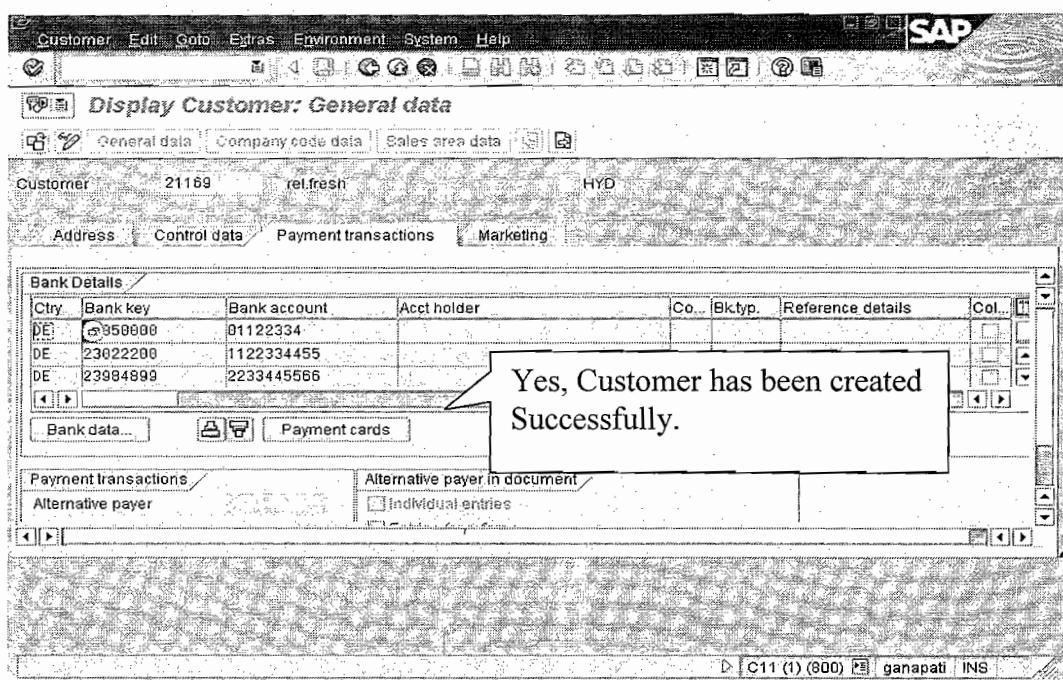
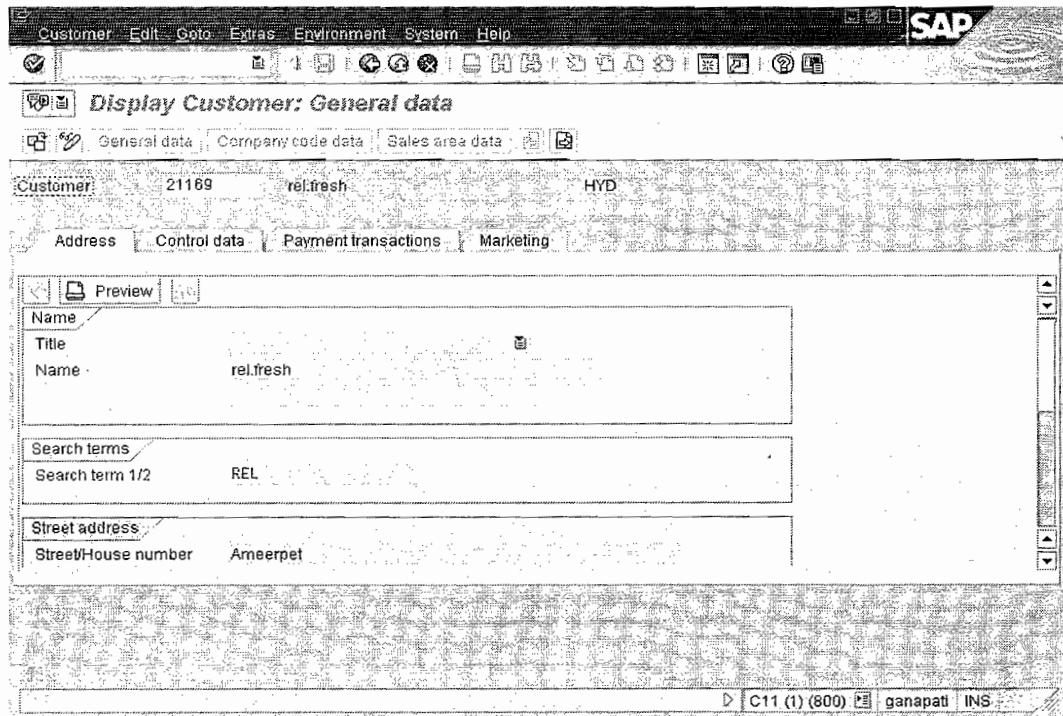


Session Method

We Never Compromise in Quality, Would You?

Testing :

Execute XD03 to display the above Customer.



Session Method

We Never Compromise in Quality, Would You?

Recording Details of the above transaction:

The screenshot shows the SAP Transaction Recorder interface with the title "Transaction Recorder: Edit Recording ZXD01". The menu bar includes "Recording", "Edit", "Goto", "System", and "Help". The toolbar contains various icons for file operations like Open, Save, Print, and Copy. The main area is a grid table with columns: Program, Screen, St., Field name, and Field value. The table contains 16 rows of data. Row 17 is highlighted.

| Program | Screen | St. | Field name | Field value |
|---------|----------|------|-------------|---------------------------|
| 1 | | T | XSD01 | BS |
| 2 | SAPMF02D | 0100 | X | |
| 3 | | | BDC_CURSOR | RF02D-KTOKD |
| 4 | | | BDC_OKCODE | /00 |
| 5 | | | RF02D-KUNNR | 38911 |
| 6 | | | RF02D-KTOKD | 0004 |
| 7 | SAPMF02D | 0110 | X | |
| 8 | | | BDC_CURSOR | KNA1-SPRAS |
| 9 | | | BDC_OKCODE | =VV |
| 10 | | | KNA1-NAME1 | CONVERGENT COMMUNICATIONS |
| 11 | | | KNA1-SORTL | CC |
| 12 | | | KNA1-STRAS | TOPAZ BLD |
| 13 | | | KNA1-ORT01 | HYDERABAD |
| 14 | | | KNA1-LAND1 | IN |
| 15 | | | KNA1-SPRAS | EN |
| 16 | SAPMF02D | 0120 | X | |

Line 1 - 16 Fr 26

C11 (3) (800) ganapati INS

The screenshot shows the SAP Transaction Recorder interface with the title "Transaction Recorder: Edit Recording ZXD01". The menu bar includes "Recording", "Edit", "Goto", "System", and "Help". The toolbar contains various icons for file operations like Open, Save, Print, and Copy. The main area is a grid table with columns: Program, Screen, St., Field name, and Field value. The table contains 26 rows of data. Rows 17 through 26 are highlighted.

| Program | Screen | St. | Field name | Field value |
|---------|----------|------|-----------------|-------------|
| 17 | | | BDC_CURSOR | KNA1-LIFNR |
| 18 | | | BDC_OKCODE | =VV |
| 19 | SAPMF02D | 0135 | X | |
| 20 | | | BDC_CURSOR | KNA1-NIELS |
| 21 | | | BDC_OKCODE | =VV |
| 22 | SAPMF02D | 0130 | X | |
| 23 | | | BDC_OKCODE | =UPDA |
| 24 | | | KNBK-BANKS (01) | DE |
| 25 | | | KNBK-BANKL (01) | 50013050 |
| 26 | | | KNBK-BANKN (01) | 123456789 |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Line 17 - 26 Fr 26

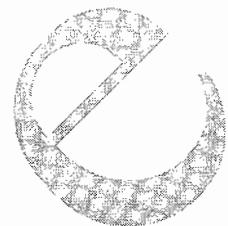
C11 (3) (800) ganapati INS

Direct Input

To enhance the batch input procedure, the system offers the direct input technique, especially for transferring large amounts of data. In contrast to batch input, this technique does not create sessions, but stores the data directly. It does not process screens. To enter the data into the corresponding database tables directly, the system calls a number of function modules that execute any necessary checks. In case of errors, the direct input technique provides a restart mechanism. However, to be able to activate the restart mechanism, direct input programs must be executed in the background only. To maintain and start these programs, use program RBMVSHOW or Transaction *BMV0*.

Examples for direct input programs are:

| Program | Application |
|----------|-------------|
| RFBIBL00 | FI |
| RMDATIND | MM |
| RVAFSS00 | SD |
| RAALTD11 | AM |
| RKEVEXT0 | CO-PA |



BDC Programming:

Theory:

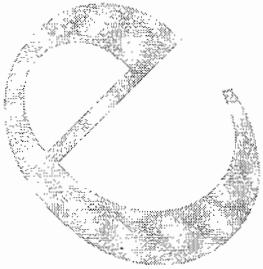
- 1) Explain the types of BDC Programming Techniques?
- 2) Differences between Call Transaction and Session Method?
- 3) Different ways of Handling Errors In Call Transaction?
- 4) Explain the steps to write a BDC Program in Detail?
- 5) Explain the steps to work with Session method for One Transaction and also for more than one transaction through the same Session?
- 6) Explain the Different Ways to Process/Run the Session?
- 7) What is Direct Input method and List out the Standard Programs available?
- 8) Explain the Role of CTUPARAMS Structure in Call Transaction?
- 9) Explain the Importance of the structures **BDCDATA** and **BDCMSGCOLL** in BDC?
- 10) Explain how to Handle the Table Control in BDC?
- 11) While uploading a flat file through BDC Call Transaction, the system suddenly get CRASHED. How do I know how many records have been updated?

Practical:

- 1) Write a BDC Program to upload Cost Centers through Call Transaction?
- 2) Write a BDC Program to Upload Purchasing Info Records through Session Method?
- 3) Write a BDC Program to upload General Ledger Accounts through Call Transaction and Push the UnSuccessful records into Session Method?
- 4) Write a BDC Program to Upload the Employee Previous Experience Details?
- 5) Write a BDC Program to Upload the Bill of materials through Both Call Transaction and Session Method?
- 6) Upload Credit Master Records?

2. SAP Scripts

1. Introduction SAP Script
2. SAP Script Components
3. SAP Script Symbols
4. SAP Script Control Commands
5. Custom Scripts
6. Standard Scripts



SAP Script :

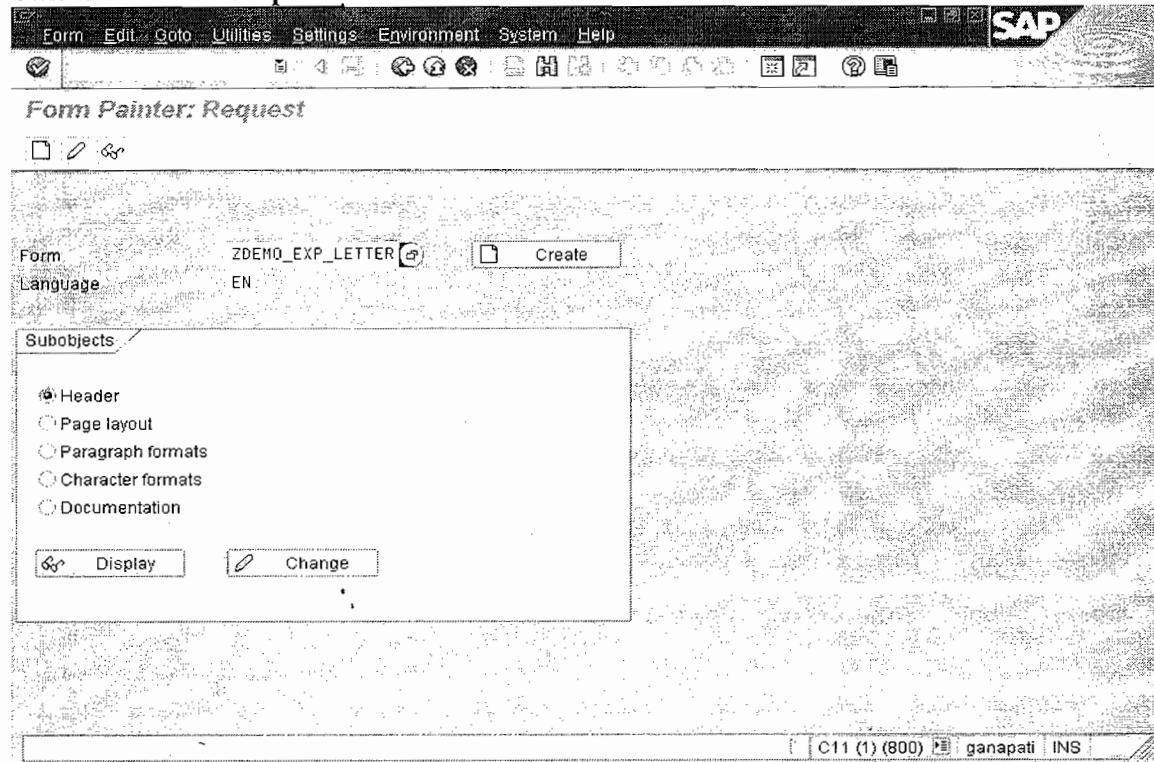
Definition :

SAP Script is SAP's Word Processing tool for Generating the Business documents that are frequently used in the Manufacturing Sectors, Service Sectors and Trading Sectors. Some of the Business documents are Sales Order, Invoice, Purchase Order, Purchase Requisitions etc,

Often there are instances where an output from a SAP program is required on a physical paper in a pre-designed format. Using normal ABAP code this is not possible. Instead SAP provides an object called SAPSCRIPT to generate such kind of documents which can contain logos, tables and other objects and which can look like pre-printed documents.

A layout set is a template designed in SAP to place the stream of data coming from a SAP program on different parts of a physical page. The designer needs to *lay out* the various elements that need to be printed on the page and store it as an object in the SAP system. An ABAP program will subsequently call this object to generate an instance of the template – thus generating an output document from the program.

SAP SCRIPT Components:



Language

Language in which the data coming on to the layout set will be printed. Generally, this will be the language that has been set up as default in the SAP system

Header

Section to define the various attributes of the layout set on a global level. Changing these attributes will affect all the components of the layout set.

The various components of the header are explained below

- **Administration Information** Administration data

This shows the information about the layout set – details of the designer, details of changes occurring to the design, development class of the layout set and the language details for the layout set

- **Standard Attributes** Basic settings

1. Description - Brief description or title of the layout set
2. Default paragraph - The base paragraph that is globally applicable to the document. This can be overridden at lower level of the layout set by using other paragraphs
3. Tab Stop - The base tab-stop that is globally applicable to the document. These can be overridden at lower level of the layout set by using other tab stops
4. First Page - The start page of the layout set
5. Page Format
6. Orientation - The direction of printing the data on a page – P for portrait (vertical) and L for landscape (horizontal)
7. Lines per inch
8. Characters/inch

- **Font Attributes**

Note Here the various attributes and the base font applicable to the document can be defined. This font setting can be overridden at a lower level using the character strings

Paragraphs Paragraph formats

Used to define the start and end positions for the different texts and objects that need to be printed on the output document.

Character Strings Character formats

Used to define the fonts and the formatting and printing styles for each and every character that needs to be printed on the output document. The start of the character string is indicated by <string name>, while the end of the character string is indicated by </>

Pages Pages

The designer needs to organize the template as a series of pages. When an actual output document is printed, it will refer to each page for putting the data coming from the ABAP program. The order of pages is also taken from the template i.e the layout set defined.

Windows Windows

Various parts of the output document can be conveniently organized on the pages using windows. Thus the data stream coming from the ABAP program can be logically grouped into various parts and can be placed on different locations on a page

There are 2 main types of windows that can be used in a layout set:

MAIN - A layout set can have only **one** MAIN window which is created by default. This window can flow over multiple pages.

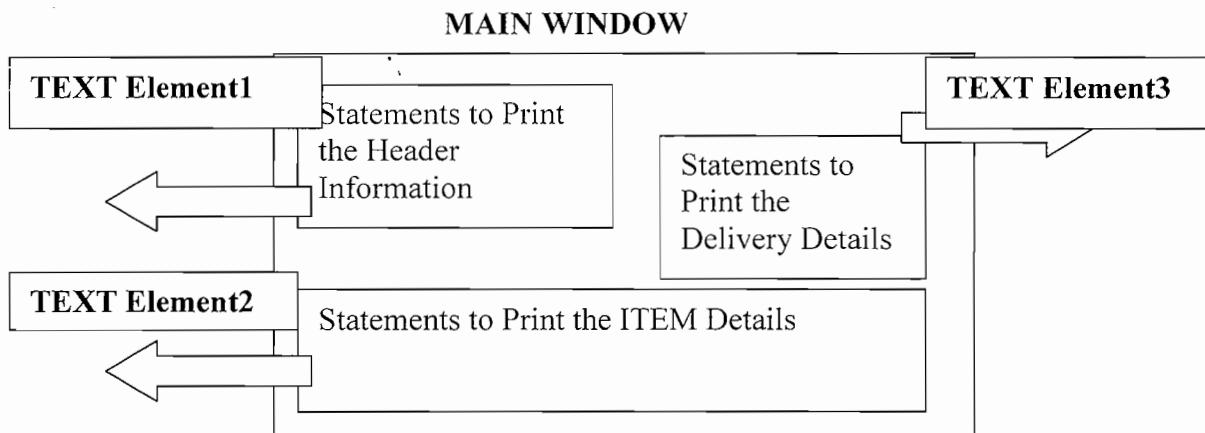
VARIABLE - A layout set can have any number of VARIABLE windows. A VARIABLE window can be used once per page.

Text Elements

Any text that needs to be written on the output document should be placed within a **text element**. This includes **constant** text as well as **variable data** like internal table data coming from the ABAP program.

Note: It is advisable to group logically related data within one text element.

Example:



The fields of various tables defined in the ABAP program will be included under these text elements. These fields are carriers of data. Every field should be included in a pair of & characters. (e.g. &aufk-aufnr&)

Page Windows

All the windows that form a page of the layout set.

Choose the window and click the **Text Elements button** to go to the Layout Set Editor. This consists of 2 parts

The small space on the left is for specifying the type of command, while the window adjacent to it is for writing the command or the text that needs to go under a text element.

The various types of commands that can be used within a layout set are tabulated below

| Command | Purpose |
|---------|-----------------------------|
| * | Default paragraph |
| Blank | Continuous text |
| = | Extended Line |
| (| Raw Line |
| / | Line Feed |
| /= | Line feed and extended line |

| | |
|------|---|
| /C | Line Feed and Raw Line |
| /: | Command Line |
| /* | Comment Line |
| /E | Text Element |
| <PN> | This is either the name of the paragraph that should be applicable from that line of the layout set |

steps for including graphical elements in the layout set are as follows

We Can INCLUDE the Graphic Elements in the layout in 2 ways.

We can always INCLUDE either .TIFF files and .BMP Files.

Working with INCLUDING the .TIFF Files :

- The graphical element (like company logo) should be in valid graphic file format like .bmp or .jpg
- Use appropriate software to convert the above file into a .TIFF file
- Use report RSTXLDMC to upload this file as a text module in SAP
- Execute the above program from the ABAP /4 editor

Initial Screen of RSTXLDMC:

Upload TIFF files to SAPscript texts

| | |
|--|---------------|
| File name and parameters for TIFF conversion | |
| File name | /tmp/file.tif |
| Type (BMON=b/w, BCOL=color) | BMON |
| Resolution for graphic (dpi) | |
| <input type="checkbox"/> Graphic resident on printer | |
| Param. for standard text | |
| Text name | ZHEX-MACRO-* |
| Text ID | ST |
| Text language | EN |
| Text title | |
| Line width for text | 132 |

- Enter the location of the .TIFF file on the PC

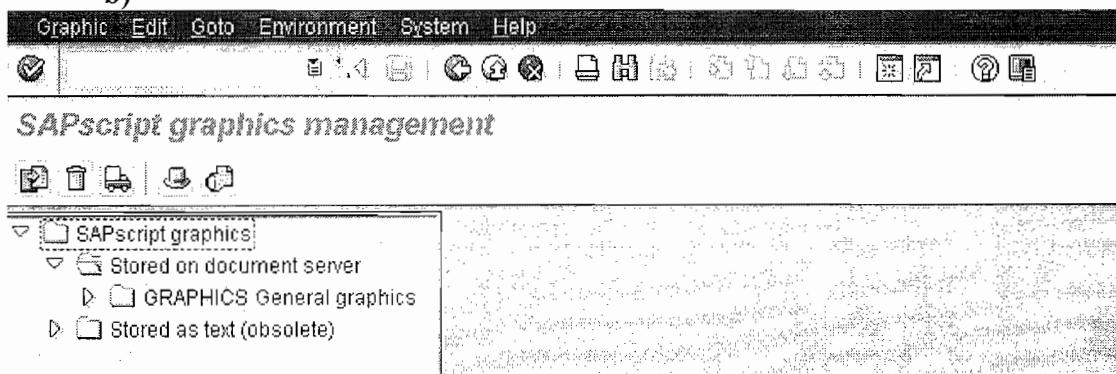
- Specify BMON or BCOL as the raster image type
- The SAP system suggests a name for the file (like ZHEX-MARCO-). The * indicates the type of file. For e.g. if the file contains a logo then the name can be ZHEX-MACRO-LOGO
- The ID should be 'ST' and give the logon language
- Running the program will convert this .TIFF file into a text element
- Incorporate this converted logo in the appropriate window under the appropriate text element by giving

INCLUDE ZHEX-MACRO-LOGO OBJECT TEXT ID ST in the first line

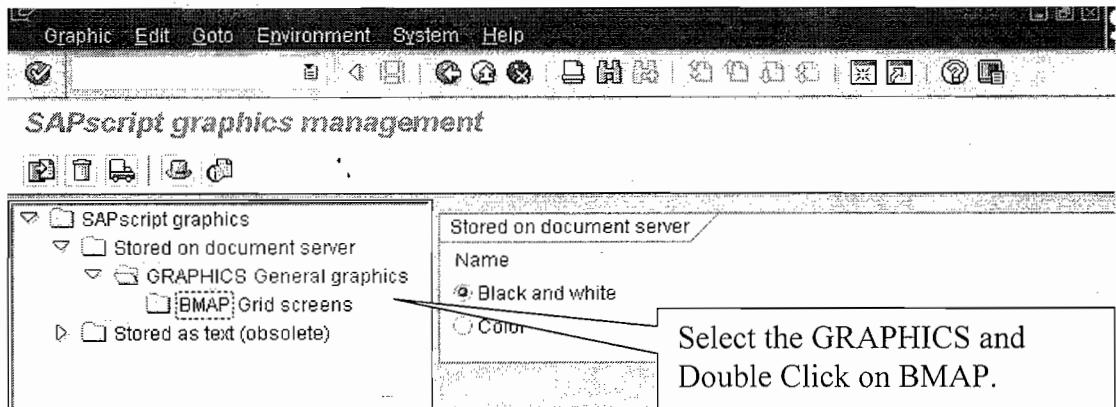
2) Another way to INSERT the Graphic element into Layout :

a) Execute the transaction SE78

b)

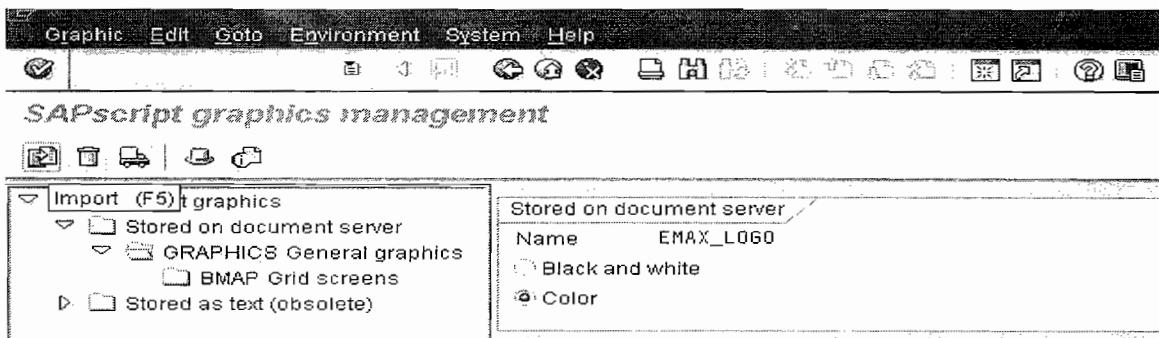


c)

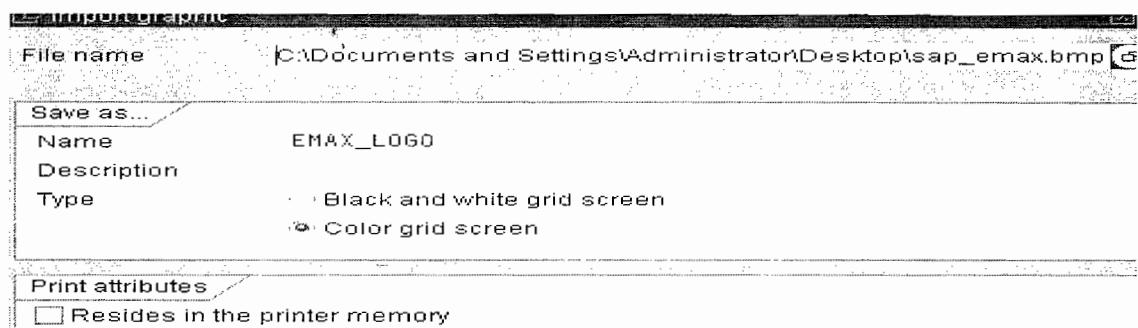


d) Provide the Name of the Graphic Element and Click on IMPORT.

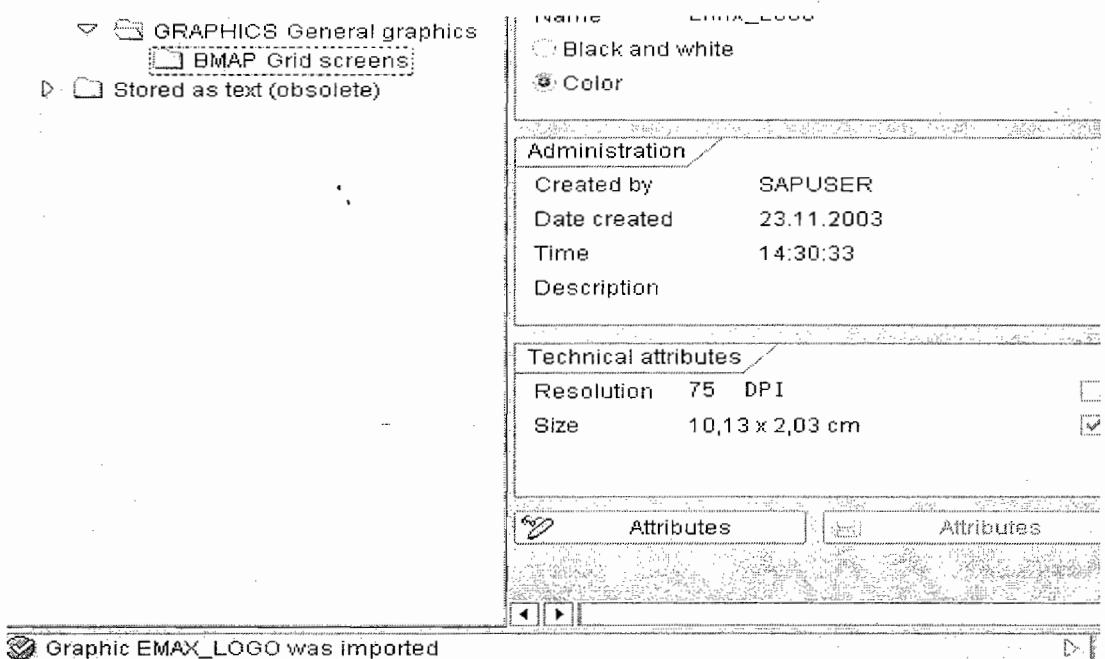
SAP Scripts
We Never Compromise in Quality, Would You?



e) Provide the File path and ENTER.



f) Make sure that the Graphic is imported Correctly.



SAP Script Symbols

1 SAP script Symbols: Text in the SAP system does not usually exist independently of other objects, but often contains a reference to some other stored object. For example, for a letter this could be the address data in the vendor master record or information in the material master record that is to be included in a purchase order text. You solve this problem by using placeholders for the data rather than entering the actual values into the text. Thus, you can create flexible text modules by using these placeholders at all points where the text needs to be variable. Since much of the data to be inserted in the text reflects the contents of fields in SAP tables, this technique ensures that the text modules always contain the current values of these fields when printed.

In SAPscript, these placeholders are known as symbols. They represent data that will not be added to the text until a later point. This is normally the point at which the output is formatted. All symbols occurring in the text are then replaced with their current values. This replacement is performed only in the output text. The original version of the text module is unaffected.

SAPscript recognizes four different kinds of symbols

- System symbols
- Standard symbols
- Program symbols
- Text symbols.

The main difference between these is the source of their values. SAPscript provides values for the system symbols. Standard symbols and their values are defined in the TTDTG table. Program symbols represent data supplied by the program that is currently executing. The values for text symbols come either from control commands in the text itself or are supplied dynamically by the Include function in the text editor.

SAPscript automatically recognizes the type of a symbol. First of all, it checks whether the symbol is a system symbol. If not, then it checks whether the symbol name is defined in the data area of the calling program. In this case, it is a program symbol. Otherwise, SAPscript reads table TTDTG. If the symbol is found in this table, then it is a standard symbol. If a symbol is neither a system symbol nor a program symbol nor a standard symbol, then it is a text symbol.

1.1 The Syntax of Symbols

Each symbol has a name that is used when the symbol is called. A call to a symbol is normally made in a line of text that also includes other kinds of text. Therefore it is necessary that symbols can be distinguished from normal text, and that the call is structured in such a way that it is possible to identify it as a call to a symbol.

- Use the delimiter & both immediately before and after the symbol.
- Do not use blank characters in the name of a symbol. Moreover, since the characters '+()' are used for defining formatting options, you must not use any of these a symbol name either
- Make sure that no SAPscript editor line break occurs between the symbol delimiters. If necessary, use a long line to avoid this (paragraph format = or /=).
- Enclose additional formatting options in round brackets and insert them immediately after the symbol name. The code letters identifying these options must be given in capitals.

A string that does not satisfy all the above conditions is not interpreted as a symbol but is copied directly into the output text.

Examples of valid symbols:

&symbol&

&MY_symbol&

&KNA1-NAME1&

&DATE&

&KNA1-UMSAT(I)&

Examples of invalid symbols:

&mysymbol closing delimiter missing

&my symbol& name contains blanks

&mysymbol)& name contains an invalid character

&symbol(Z& closing bracket of formatting option missing

&KNA1-UMSAT(i)& formatting option not in capitals

Note : The symbol names themselves are not case-sensitive, that is, SAP script does not distinguish between capital and lower case letters in symbol names.

Page 9 of 81 By Ganapati Adimulam

These symbol names are identical:

&mysymbol& &Mysymbol& &MYSYMBOL&

A symbol name can contain a maximum of 130 characters. However, only the first 32 characters are used for unique identification.

1.2 System Symbols

You can use system symbols in all kinds of text. **SAPscript** supplies the values for system symbols. **The names of the system symbols are fixed.**

1.2.1 Current Date &DATE& The current date is displayed.

It is formatted according to the specifications found in the user master data. You can adapt this format to your own requirements by specifying a date mask (**SET DATE MASK**) or by using a country-specific formatting option (**SET COUNTRY**). The current value for this symbol is taken from the **SY-DATUM** field. This value is not copied every time that the date is called, but only at the following times:

- When printing starts (**OPEN_FORM, PRINT_TEXT**)
- When symbols are replaced in the text editor
- When a text is exported in the ASCII or RTF format
- When the **TEXT_SYMBOL_REPLACE** function module is called (optional)

1.2.2 Current Day Number &DAY&

The current day number is printed. The display includes leading zeros.

1.2.3 Current Month Number &MONTH&

The current month number is printed. The display includes leading zeros.

1.2.4 Current Year Number &YEAR&.

This symbol is used to print the current year as a four digit number.

1.2.5 Current Day Name (Long Form) &NAME_OF_DAY&.

The name of the current day is written out in full. The language used for the output is determined by the appropriate text language or form language. The names of the days are stored in the TTDTG table under the key **%%SAPSCRIPT_DDD_dd**, where dd is the day number (01 = Monday, ..., 07 = Sunday).

1.2.6 Current Month Name (Long Form) &NAME_OF_MONTH& .

The name of the current month is written out in full. The language used for the output is determined by the appropriate text language or form language. The names of the months

are stored in the TTDTG table under the key %%SAPSCRIPT_MMMM_mm, where mm is the month number (01,.., 12).

1.2.7 Current Time &TIME& The current time is printed in the form hours: minutes: seconds. Each of the components for hours, minutes, and seconds consists of two digits, using a leading zero if necessary. You can adapt this format to your own requirements by specifying a time mask (SET TIME MASK). The value for the time field is taken from the SY-UZEIT field. This value can be copied over only at particular times (c.f. DATE).

1.2.8 Hours Component of Current Time &HOURS& .

The component of the current time referring to hours is printed. The display includes leading zeros.

1.2.9 Minutes Component of Current Time &MINUTES&.

The component of the current time referring to minutes is printed. The display includes leading zeros.

1.2.10 Seconds Component of Current Time &SECONDS&.

The component of the current time referring to seconds is printed. The display includes leading zeros.

1.2.11 Current Page Number &PAGE& .

You can use this symbol to insert into the text the page number that the current page will have when printed. You can specify the formatting option for the page number in the form for each page type.

1.2.12 Page Number of the Next Page &NEXTPAGE&

This symbol is used to print the number of the following page. The output format is the same as with &PAGE& . Note that on the last page of the output, in each window that is not of type MAIN, &NEXTPAGE& has the value 0.

1.2.13 Selected Device Type &DEVICE&

The &DEVICE& symbol is used to print the type of the output device. This type is passed in the DEVICE parameter when the SAPscript output (OPEN_FORM, PRINT_TEXT) is started, and it specifies the device for which the output should be formatted.

Possible values are:

PRINTER

SCREEN

TELEX

TELEFAX

ABAP (ABAP list display)

1.2.14 Spaces &SPACE&

You can use this symbol to generate a string of space characters. You must pass the number of space characters required with the symbol. If you leave out the number, then no spaces are printed.

1.2.15 Underline &ULINE&

You can use this symbol to insert a string of underline characters into the output text. You must pass the number of underline characters required with the symbol. If you leave out the number, then just one underline character is printed.

1.2.16 Vertical Line &VLINE&

You can use this symbol to insert a string of vertical line characters into the output text. You must pass the number of vertical line characters required with the symbol. If you leave out the number, then just one vertical line character is printed.

1.3 Program Symbols

The integration of SAPscript allows to link data that is stored in various applications of the SAP system into text modules; for example a form letter to be sent to several customers. The address information of these customers is in the SAP database and must be incorporated into the letter. **SAPscript cannot read this data out of the SAP database itself, but has to call another program to do this. The data is then copied into work areas declared with TABLES.**

Starting with Release 3.1G, you are no longer restricted to the TABLES statement. You can address any global variable using a program symbol. The system can evaluate the ABAP Dictionary information (output length, number of decimal places, and so on) not only for TABLES fields, but also for INFOTYPES fields and variables with a LIKE reference. Example: DATA: MYCOUNTRY LIKE USR03-LAND1. The system considers all output characteristics that can be retrieved using the ABAP statement DESCRIBE. If SAPscript is now called from this program to format a text, it can copy the data out of these work areas. Symbols that obtain their values from this kind of data area are called program symbols. The value of a program symbol is limited up to a maximum of 255 characters. The name of a program symbol, when using TABLES statements, consists of the table name and the field name, separated by a hyphen. Examples of customer address fields are: &KNA1-NAME1&, &KNA1-ORT01&, &KNA1-PFACH&. Otherwise, the symbol is used in the way it is defined in the print program (for example, &MYCOUNTRY&). When SAPscript encounters a symbol, it first checks whether the symbol is a system symbol. If it finds no corresponding entry, it tries to find a table or structure in the calling ABAP program (declared with TABLES). If there is, the symbol is a program symbol and SAPscript next checks in the Dictionary to see whether this table contains the specified field. If there is no table or structure in the calling ABAP program, SAPscript checks whether the symbol is a standard symbol. If no entry is found in table TTDTG, the system checks the calling program for global definitions (DATA, CONSTANTS, INFOTYPE, PARAMETER). If it finds any definitions, SAPscript processes the symbol as program symbol. Only if no global definitions are found either, does SAPscript process the symbol as text symbol. Basically, a defined text symbol remains a text symbol even if in the print program, for example, a DATA statement with the same name is used.

For replacing the variables, the sequence of the variables in the corresponding text is decisive.

Form/text: /: DEFINE &mysymbol& = 'abc'

- &mysymbol&

.... Print program: Data: mysymbol(5) value 'xyz'. In this example, in the form/text instead of &mysymbol& the value of the text symbol defined with DEFINE is printed: abc

Form/text:

- &mysymbol&

/: DEFINE &mysymbol& = 'abc' Print program: Data: mysymbol(5) value 'xyz'. In this example, in the form/text instead of &mysymbol& the value of the program symbol defined in the print program is printed: xyz Usually, SAPscript looks for the table work areas and the global variables in the main part of the ABAP program that was started. For example, if you call function modules that you want to use the table work

areas, enter the name of the corresponding program in the field TDPROGRAM of the structure you can enter in the OPTIONS parameter at OPEN_FORM. This definition is then valid for all program symbols up to the next CLOSE_FORM. However, you can also specify the name of the program in the PROGRAM parameter of the START_FORM function module. If you do this, it will be valid up to the next END_FORM. If this parameter is not set in START_FORM, then the setting in OPEN_FORM applies. For formatting a program symbol, SAPscript uses the specifications found in the Dictionary (output length, formatting routine, and so on). Certain data types require additional information out of other fields to format a value. For example, the contents of a currency field can be formatted correctly only when the information from the currency field key is also available. SAPscript itself is responsible for obtaining the necessary information from the Dictionary.

NOTE : For printing the program symbols, SAPscript uses the WRITE statement of the ABAP programming language. The effect of this statement is controlled externally via parameters (such as specifications in the user master record, table T005X), depending on the data type. If a program symbol is not printed in the way you expected it, first check whether the control parameters stated above are set correctly.

To fields of the table work areas shown below, you can refer in all SAPscript text modules:

1.3.1 SYST: System Fields in the ABAP Programming Environment

You can refer to all the fields in this table. You should, however, note that some of the fields are specific to a certain environment. They contain values that do not come from your application but have been set by the SAPscript programming environment (for example, SYST-REPID).

1.3.2 USR03: User Address Data

This structure contains information from the user master record for a given user:

- Business address
- Telecommunication (telephone, telefax)
- Other data such as user language, department, cost center. You can maintain the contents of these fields in the ‘Address’ section of the user maintenance. If this table exists in the calling program, then SAPscript copies the data from the work area of this program. Otherwise, SAPscript uses the values for the currently active user.

1.3.3 SAPSCRIPT: General SAPscript Fields

You can print the following fields of structure SAPSCRIPT as program symbols in SAPscript forms:

- **&SAPSCRIPT-SUBRC&**: Receives a value after executing an INCLUDE statement. The value shows whether the INCLUDE was found (that is, the INCLUDE text exists) or not. You can query this value in an IF statement. INCLUDE was found: &SAPSCRIPT-SUBRC& = 0 INCLUDE was not found: &SAPSCRIPT-SUBRC& = 4

- **&SAPSCRIPT-DRIVER&**: SAPscript formats a text for a specific output device. The initial formatting is independent of the specific language of this device. SAPscript then calls a driver to convert the device-independent format to device-specific control commands. This field contains the name of the driver.

POST Postscript driver HPL2 HP Laserjet driver for the PCL4/PCL5 languages PRES Driver for output devices using the PRESCRIBE language The available drivers are stored in table TSP09.

- **&SAPSCRIPT-FORMPAGES&**: This field contains a number representing the total number of pages of the currently formatted form (any output between START_FORM and END_FORM). The page counter mode (START, INC, HOLD) of the individual pages is ignored. You can use this symbol to formulate information like 'Page x of y' for your output.

- **&SAPSCRIPT-JOBPAGES&**: This field contains a number representing the total number of pages of all forms contained in the currently formatted print request, in other words, of all forms created using the OPEN_FORM, START_FORM.. ENDFORM, START_FORM.. END_FORM,..., CLOSE_FORM function modules.

When using the SAPSCRIPT-FORMPAGES or SAPSCRIPT-JOBPAGES symbols, SAPscript leads all output pages of the current form or current print request into main memory to replace the symbol by the appropriate value. For large output jobs, this can mean a very large amount of memory.

- **&SAPSCRIPT-COUNTER_x& (x = 0.. 9)**: These fields represent ten counter variables that you can use in your text and forms for any counting purposes. You can use the '+' and '-' formatting options to increment or decrement a counter before its value is printed. You can use the DEFINE control command to assign any specific value to a counter.
- **&SAPSCRIPT-TELELAND&**: This field contains the country key of the fax target address when using fax output via SAPscript (field ITCPO-TDTELELAND of parameter OPTIONS of function module OPEN_FORM).
- **&SAPSCRIPT-TELENUM&**: This field contains the local fax number of the fax target address when using fax output via SAPscript (field ITCPO-TDTELENUM of parameter OPTIONS of function module OPEN_FORM).
- **&SAPSCRIPT-TELENUME&**: This field contains the complete fax number of the fax target address when using fax output via SAPscript

(field ITCPO-TDTELENUME of parameter OPTIONS of the function module OPEN_FORM).

1.4 Standard Symbols

Standard symbols are defined in table TTDTG. This table contains both the name of each symbol and its value. The value, which is language-dependent, can contain up to 60 characters. SAP supplies this table filled with standard entries. You can extend it with customer-specific symbols. You can use standard symbols in all kinds of text.

1.5 Text Symbols All symbols that do not correspond to one of the three types of symbol described above are text symbols. You define the value of a text symbol yourself in the text module.

There are two ways of doing this:

- In the Text. All the text symbols contained□ Symbols □text editor, choose Include either in the current text or in a form assigned to the text are displayed. You can assign a value of up to 60 characters to a text symbol. Enter this value in the same form as it is to appear in the output. The effect of assigning a value is temporary, since the values assigned are not stored with the text but are lost as soon as you leave the editor. You use this kind of value assignment if you had a ready-made text containing symbols that you want to print with specific values in place of the symbols, and you want to do this only once without storing the 'changed' text.
- In the text, use the control command DEFINE. Since control commands are stored with the text module, any value you assign in this way is preserved when you save the text. You can change the value assigned to a symbol in the text at any time simply by issuing another DEFINE command. Remember always to use the ' (inverted comma) character to delimit a value. The maximal length for these values is also 60 characters.

A text in the editor contains the following DEFINE commands: /: DEFINE &mysymbol& = 'xxx xxx xxxx xxxx' &mysymbol& /: DEFINE &mysymbol& = 'yyyy yyyy yyyy' / &mysymbol& The printed text appears like this: xxx xxx xxxx xxxx yyyy yyyy yyyy

1.6 Formatting Options

The value of a symbol is normally printed using its full length, although trailing spaces are removed. An exception are program symbols of these data types: CURR, DEC, QUAN, INT1 INT2, INT4, PREC, and FLTP. These are formatted right-justified with an output length as specified in the Dictionary. **You can adapt the standard formatting to your own requirements by using one of the additional formatting options available.** You supply the parameters for these options together with the symbol itself. **Many of these options are abbreviated to a single letter, which has to be given as a capital letter.** You can combine two or more options on a single symbol, as long as the result still makes sense.

List Of Possible Formatting Options :

Offset

Output Length

Omitting the Leading Sign

Leading Sign to the Left

Leading Sign to the Right

Omitting Leading Zeros

Space Compression

Number of Decimal Places

Omitting the Separator for 'Thousands'

Specifying an Exponent for Floating Point Numbers

Right-Justified Output

Fill Characters

Suppressing Output of Initial Values

Ignoring Conversion Routines

Preceding and Subsequent Texts (Pre-Text / Post-Text)

Country-Dependent Formatting

Date Mask

Time

1.6.1 Offset:

Specifying an offset has the effect that a certain number of bytes of the symbol value, starting with the first byte on the left, will not be displayed. If the offset specified is greater than the length of the value, nothing is printed.

Syntax &symbol+offset&

If <symbol> has the value 123456789,

the following will be displayed:

&symbol& -> 123456789

&symbol+3& -> 456789

&symbol+7& -> 89

&symbol+12& -> &symbol+0& -> 123456789

(As 12 in &symbol+12& is greater than the total length).

1.6.2 Output Length:

If you need only a part of the symbol value, or the output has to fit into a box or a field on the screen without overlapping the edges of this area, then you can use an output length specification to define how many bytes of the value should be copied. **Syntax &symbol(length)&**

If <symbol> has the value 123456789.

&symbol(3)& -> 123

&symbol(7)& -> 1234567

You can combine an output length specification with an offset specification. The specified length is then counted from the specified offset position. &symbol+4(3)& -> 567 If a length specified is greater than the current length of the value, then spaces are appended to the symbol value.

You can use the character * to specify the length of a program symbol. The value of the symbol is then printed using the output length defined in the ABAP Dictionary.

Syntax &symbol(*)&

The SYST-UNAME field contains the logon name of a user called Einstein . The Dictionary entry for this field contains an output length of 12.

```
&SYST-UNAME&...      -> Einstein...
&SYST-UNAME(9)&...    -> Einstein...
&SYST-UNAME(*)&...    -> Einstein ...
```

1.6.3 Omitting the Leading Sign :

Program symbols with numeric values can have a leading sign. This sign usually appears to the right of the numeric value, either as a space for positive numbers, or as a minus sign for negative numbers . **You can use the S option to ensure that the value is formatted without the sign .**

Syntax &symbol(S)&

The ITCDP-TDULPOS field contains the value -100.00.

The ABAP Dictionary definition for this field includes a leading sign.

```
&ITCDP-TDULPOS&        -> 100.00-
&ITCDP-TDULPOS(S)&     -> 100.00
```

1.6.4 Leading Sign to the Left :

The leading sign is normally displayed to the right of a numeric value, except in the case of a floating point number. This option enables you to specify that the leading sign should be placed to the left of the number.

Syntax &symbol(<)&

```
&ITCDP-TDULPOS&        -> 100.00-
&ITCDP-TDULPOS(<)&     -> -100.00
```

The **SET SIGN LEFT** control command specifies that all subsequent symbols with a numeric value should have a left-justified leading sign. **If you use this control command, you must no longer repeat the < option for each individual symbol.**

1.6.5 Leading Sign to the Right

The default setting is to print the leading sign to the right of a numeric value. If you used the SET SIGN LEFT control command to specify that the leading sign should be printed in front of the value, you can override this specification for individual symbols. The symbols specified with the > option are then printed with the leading sign to the right.

Syntax: **&symbol(>)&**

You can use the SET SIGN RIGHT control command to switch back to the default setting for the output of the leading sign.

1.6.6 Omitting Leading Zeros:

Certain symbol values are printed with leading zeros. If you want to suppress these, use the Z option.

Syntax **&symbol(Z)&**

Assuming the current date is 1.1.1994,

&DAY& -> 01

&DAY(Z)& -> 1

1.6.7 Space Compression:

The symbol value is viewed as a sequence of ‘words’, each separated from the next by either one or a string of space characters.

The C option has the effect of **replacing each string of space characters with a single space and shifting the ‘words’ to the left as necessary to close up the gaps.** Leading spaces are completely removed. The results are the same as those of the ABAP command CONDENSE.

Syntax: **&symbol(C)&**

Assuming ' Albert Einstein ' is the symbol value,

&symbol& -> Albert Einstein

&symbol(C)& -> Albert Einstein

1.6.8 Number of Decimal Places:

Page 20 of 81

By Ganapati Adimulam

A program symbol of one of the data types DEC, QUAN, and FLTP can contain decimal place data. Use the option below to override the Dictionary definition for the number of decimal places for the formatting of this symbol value.

Syntax &symbol(N)&

The EKPO-MENGE field contains the value 1234.56.

The Dictionary definition specifies 3 decimal places and an output length of 17.

| | |
|---------------------|------------|
| &EKPO-MENGE& -> | 1,234.560 |
| &EKPO-MENGE(.1) -> | 1,234.6 |
| &EKPO-MENGE&(.4) -> | 1,234.5600 |
| &EKPO-MENGE&(.0) -> | 1,235 |

1.6.9 Omitting the Separator for ‘Thousands’:

Symbols of the DEC, CURR, INT, and QUAN data types are normally formatted with the a ‘thousands’ separator character.

The T option allows you to omit this separator character.

Syntax: &symbol(T)&

The EKPO-MENGE field contains the value 1234.56.

The Dictionary definition specifies 3 decimal places and an output length of 17.

| | |
|--------------------|---|
| &EKPO-MENGE& -> | 1,234.560 |
| &EKPO-MENGE(T)& -> | 1234.560 (No Comma Separator for thousands) |

1.6.10 Specifying an Exponent for Floating Point Numbers:

The way a floating point number is formatted depends on whether an exponent is specified. The mantissa is adjusted by shifting the decimal point and, if necessary, introducing leading zeros, according to the exponent chosen.

Using an exponent value of 0 means that the exponent representation will not be used for displaying the symbol.

Syntax **&symbol(EN)&**

If you specify an exponent of 0, then the number is displayed without using the exponent representation.

This has the same effect as completely omitting the specification of an exponent:

&symbol(E0)& has the same effect as **&symbol(E)&**

In this example,

the PLMK-SOLLWERT field is assumed to have the value 123456.78 and to be of data type FLTP.

| | | |
|------------------------------------|----|-----------------------|
| &PLMK-SOLLWERT& | -> | +1.23456780000000E+05 |
| &PLMK-SOLLWERT(E3)& | -> | +123.456780000000E+03 |
| &PLMK-SOLLWERT(E6)& | -> | +0.12345678000000E+06 |
| &PLMK-SOLLWERT(E0)& | -> | +123456.780000000 |
| &PLMK-SOLLWERT(E)& | -> | +123456.780000000 |

1.6.11 Right-Justified Output :

Symbol values other than numeric values are normally formatted left-justified. To specify right-justified formatting, use the R option. **You must use this option in conjunction with an output length specification.**

Syntax **&symbol(R)&**

If symbol has the value 1234.

| | | |
|-------------------------|----|------|
| &symbol& | -> | 1234 |
| &symbol(8R) | -> | 1234 |

For program symbols, the length specification contained in the Dictionary definition may be used instead of an explicit length.

1.6.12 Fill Characters :

You can replace leading spaces in a value with a fill character.

Use the **F** option with the character immediately following the F in the specification as the fill character.

Syntax &symbol(F f)& f = fill character

The figure for customer sales in the KNA1-UMSAT field is \$700.

The Dictionary description of the field specifies an output length 8.

&KNA1-UMSAT& -> 700.00

&KNA1-UMSAT(F*)& -> **700.00

(Since it is right justified, it is added the *'s left to it for the remaining spaces.)

&KNA1-UMSAT(F0)& -> 00700.00 (Zeros are added , left to it)

1.6.13 Suppressing Output of Initial Values:

Use the I option to suppress the output of symbols that still contain their initial values.

Syntax &symbol(I)&

Assuming KNA1-UMSAT contains the value 0 and the currency is DEM.

&KNA1-UMSAT& -> 0,00

&KNA1-UMSAT(I)& -> If the field contains an amount other than 0, this value is printed in the normal way.

&KNA1-UMSAT& -> 700,00

&KNA1-UMSAT(I)& -> 700,00

1.6.14 Ignoring Conversion Routines :

SAPscript conversion routines specified in the Dictionary are automatically recognized and used when program symbols are formatted. To suppress conversion, use the **K** option.

Syntax &symbol(K)&

1.6.15 Changing the Value of a Counter

You can increase or decrease the value of a SAPSCRIPT-COUNTER_x (x=0.. 9) counter variable by 1, before the current counter value is printed.

Syntax: **&SAPSCRIPT-COUNTER_x(+)&**

Increases by 1 the contents of the counter variable x (x=0.. 9)

&SAPSCRIPT-COUNTER_x(-)& Decreases by 1 the contents of the counter variable x (x=0.. 9).

If you want to change the value of a counter variable without actually printing the new value, use this formatting option together with an additional option to set the output length to 0 (see above). If you want to set a counter variable to some specific value, use the DEFINE control command.

Assume that &SAPSCRIPT-COUNTER_1& initially has the value 2.

&SAPSCRIPT-COUNTER_1& -> 2

&SAPSCRIPT-COUNTER_1(+)& -> 3

&SAPSCRIPT-COUNTER_1(-)& -> 2

&SAPSCRIPT-COUNTER_1(-)& -> 1

&SAPSCRIPT-COUNTER_1(+0)& -> &SAPSCRIPT-COUNTER_1(+)&

-> 3 (as the Initial Value is 2).

1.6.16 Preceding and Subsequent Texts (Pre-Text / Post-Text) :

In addition to using initial values for symbols, **you can specify additional texts that are printed only when the value of the symbol is no longer the initial value.**

You can specify a text to be printed immediately before the symbol value (**the pre-text**), and a text to be printed immediately after the symbol value (**the post-text**).

If the symbol contains its initial value, these texts are suppressed.

Syntax: **&'pre-text'symbol'post-text'&**

Make sure that the symbol, the pre-text, and the post-text all appear on a single line of the editor. You may have to use a long line (paragraph attribute = or /=) in the editor. The inverted comma ' is used as a delimiter for these texts. If this character is also part of one

of these texts, enter it twice at this point, so that it is not interpreted as a delimiter character.

A pre-text or post-text may itself contain symbols in addition to normal text. However, these symbols may not have a pre-text or a post-text.

The KNA1-PFACH field contains a customer P.O. Box number.

Since the text "P.O. Box" is not stored in the field along with the value, you would normally write the following for

the P.O. Box line of an address: **P.O. Box &KNA1-PFACH&**

However, if no P.O. Box has been specified,

the text "P.O. Box" would still appear on its own in the address.

P.O. Box &KNA1-PFACH& -> P.O. Box <No Value for PO Box>.

To prevent this, use pre-text or post-text (in this case, pre-text).

&'P.O. Box 'KNA1-PFACH& -> **If a P.O. Box has been specified, then this will be displayed together with the appropriate text in the normal way.**

&'P.O. Box 'KNA1-PFACH& -> P.O. Box 123456

1.7 Country-Dependent Formatting :

Certain fields are formatted specific to a particular country.

These include fields for displaying a date and numeric fields containing either a decimal point or a 'thousands' separator character. The formatting applied is usually determined by the definitions contained in the user master record.

You can use the SET COUNTRY control command to choose a different formatting operation. The various country-dependent formatting options are stored in table **T005X**.

Syntax /: SET COUNTRY country_key

You can specify this country key either by quoting it directly enclosed in inverted commas or by using a symbol.

/: SET COUNTRY 'CAN' OR /: SET COUNTRY &KNA1-LAND1&

You can revert to the settings of the user master record by using the SET COUNTRY control command again with an empty country name. /: SET COUNTRY '' When

SAPscript encounters this command it calls the corresponding ABAP command internally. The effect of the SAPscript command is thus identical with that of the ABAP command.

If the formatting turns out other than expected, check the settings in table T005X.

1.7.1 Date Mask :

To format date fields, use the SAPscript SET DATE MASK command.

Executing this command causes all subsequent date fields to be printed with the specified formatting.

Syntax /: SET DATE MASK = 'date mask'

The following templates may be used in the date mask:

DD day (two digits)

DDD name of day (abbreviated)

DDDD name of day (written out in full)

MM month (two digits)

MMM name of month (abbreviated)

MMMM name of month (written out in full)

YY year (two digits)

YYYY year (four digits)

LD day (formatted as for the L option)

LM month (formatted as for the L option)

LY year (formatted as for the L option)

Any other characters occurring in the mask are interpreted as simple text and are copied directly to the output.

Assuming a current system date of March 1st, 1997.

/: SET DATE MASK = 'Foster City, MM.DD.YY'

&DATE& -> Foster City, 03.01.97

&DATE(Z)& -> Foster City, 3.1.97(No Leading Zeros)

/: SET DATE MASK = 'MMMM DD, YYYY'

&DATE& -> March 01, 1997

You can revert to the standard setting by using the SET DATE MASK command again with an empty string in place of the date mask:

/: SET DATE MASK = ''

1.7.2 Time Mask

You can use the SAPscript SET TIME MASK command to format time fields in a way that differs from the standard setting.

Executing this command causes all subsequent time fields to be printed with the specified formatting.

Syntax: /: SET TIME MASK = 'time_mask'

The following templates may be used in the time mask:

HH hours (two digits)

MM minutes (two digits)

SS seconds (two digits)

Any other characters occurring in the mask are interpreted as simple text and are copied directly to the output.

Assuming the current time is 10:08:12.

&TIME& -> 10:08:12

/: SET TIME MASK = 'HH:MM'

&TIME& -> 10:08

/: SET TIME MASK = 'HH hours MM minutes'

&TIME& -> 10 hours 08 minutes

&TIME(Z)& -> 10 hours 8 minutes (No Leading Zeros).

You can revert to the standard setting by using the SET TIME MASK command again with an empty string in place of the time mask:

/: SET TIME MASK = ''

SAPscript Control Commands:

The functionality of the SAPscript editor is made available through a set of commands. These commands give you full editing control over your text. They are executed immediately when called.

There is, however, another kind of SAPscript command, namely the control commands. The purpose of these is to allow control of the output formatting. These commands are not interpreted by the SAPscript editor, but are passed through to the SAPscript Composer for processing. The Composer is the program that converts text from the form displayed in the editor to the form used for printing. This includes, for example, line and page formatting, the replacement of symbols with their current values and the formatting of text according to the paragraph and character formats specified.

Explicit Page Break: NEW-PAGE

Preventing Page Breaks: PROTECT

Next Main Window: NEW-WINDOW

Assigning a Value to a Text Symbol: DEFINE

Including Other Texts: INCLUDE

Changing the Style: STYLE

Formatting Addresses: ADDRESS

Setting a Header Text in the Main Window: TOP

Setting a Footer Text in the Main Window: BOTTOM

Conditional Text: IF

Finding a Match: CASE

Calling ABAP Subroutines: PERFORM

Inserting Print Controls: PRINT-CONTROL

Syntax of Control Commands

SAPscript control commands are entered and edited in the text editor in the same way as a normal line of text. They do, however, differ from normal lines of text:

- Enter the paragraph format **/:** in the format column to identify a control command.
- Make sure that a control command, together with any parameters it requires, does not occupy more than a single line.
- Enter only one control command in each line.
- Note that the editor formatting has no effect on lines containing control commands.

Note : If a control command is unknown or it contains syntax errors, the line containing it is treated as a comment line. It is neither interpreted nor printed.

Explicit Page Break: NEW-PAGE

SAPscript automatically inserts a page break when the main window of a page (MAIN) is full. You can use the NEW-PAGE command to force a page break in the text at any point you want one. The text following this command then appears on a new page. The page break is always performed (it is an unconditional page break).

The NEW-PAGE command completes the current page. This means that all the windows that are still on the page are printed immediately. If you use the NEW-PAGE command without parameters, the page defined in the current form as the next page will be taken next. If, however, your form contains a number of different pages, then you can specify any one of these as the next page to be used.

Syntax:

/: NEW-PAGE [page_name]

/: NEW-PAGE

The current page will be completed and the text in the following lines will be written to the page specified in the form.

/: NEW-PAGE S1

As above, except that the page S1 will be taken as the next page.

- If, in a NEW-PAGE command, you specify a page not contained in the form, the specification is ignored.
- Take care that there are no blank lines immediately before a NEW-PAGE command. If an implicit page break occurs within the blank lines, an unexpected blank page may be printed.

Preventing Page Breaks: PROTECT

You can specify, either in the style or in the form, that a particular paragraph should not be split in two by a page break. If this page protect attribute is set, then the complete paragraph is always printed on one page. This property applies only to that particular paragraph.

To allow you to define the areas to be protected against a page break on an individual basis, SAPscript provides the PROTECT.. ENDPROTECT command pair. If you enclose the text to be protected in these commands, then SAPscript will ensure that each line of this text is printed together on the same page.

Note : If the complete text fits in the space remaining on the current page, then it is printed on this page just as it would be if no PROTECT command had been used. If, however, the remaining space is not sufficient for the text, then the PROTECT command has the same effect as a NEW-PAGE command and text is printed on a new page.

Note : Thus the PROTECT/ENDPROTECT commands may be regarded as a kind of conditional NEW-PAGE command, the condition being whether or not the lines enclosed between the two commands fit in the space remaining in the current main window.

Syntax:

```
/:PROTECT  
:  
:  
/: ENDPROTECT
```

The text lines to be protected are enclosed between the two commands.

- An ENDPROTECT command without a preceding PROTECT command has no effect.

- If the terminating ENDPROTECT is missing, SAPscript assumes it at the end of the text.
- PROTECT.. ENDPROTECT command pairs cannot be nested. If a second PROTECT command occurs before the first one has been terminated by an ENDPROTECT, it is ignored.

Note : If the text enclosed by a PROTECT.. ENDPROTECT pair is itself too long for a page break is generated immediately before the text and the text is printed in the normal way. It is then unavoidable that a page break will occur at some point within the text.

Next Main Window: NEW-WINDOW

Each page can consist of up to 99 main windows. Each main window is assigned a consecutive identifying number (0..98), and the windows are filled in this order. This feature enables SAPscript to print labels and to output multi-column text. When one main window fills up, the next main window on that page is taken, if there is a next one. A page break is inserted after the last main window.

You can use the NEW-WINDOW command to call the next main window explicitly, even if the current main window is not yet full. If you are in the last main window of the page, the command has the same effect as the NEW-PAGE command.

Syntax:

/: NEW-WINDOW

Including Other Texts: INCLUDE

To include the contents of another text into the current text, use the INCLUDE control command. SAPscript still treats the text to be included as a separate text. The text is copied over only at the point at which the output is formatted.

Thus the use of the INCLUDE command always ensures that the most current version of a text is included into the output, since the text is not read and inserted until the output is formatted.

Syntax:

**/: INCLUDE name [OBJECT o] [ID i] [LANGUAGE l] [PARAGRAPH p]
[NEW-PARAGRAPH np]**

/: INCLUDE MYTEXT LANGUAGE 'E' PARAGRAPH 'A1'

The text with the name MYTEXT and the language E is included, regardless of the language of the calling text. The paragraph format A1 will be used as the standard paragraph type for this call.

The INCLUDE command returns a status code in the SAPSCRIPT-SUBRC symbol:

- 0: the text include was successful.
- 1: the command could not be executed because it contained syntax errors.
- 2: the rules governing the text to be included were not followed (see above).

This value cannot occur if the command is used in a SAPscript form.

- 4: the specified text could not be found.

Setting a Header Text in the Main Window: TOP - ENDTOP

You can use the TOP.. ENDTOP control command to specify lines of text that you want to print always at the top of the main window. These text lines are also known as header texts. For example, you would use header texts in the case of a very long table covering several pages of output to ensure that the table heading information were repeated at the start of each new page of output.

Syntax:

```
/:TOP  
:  
:  
/: ENDTOP
```

The lines of text enclosed between the two control commands will be output from now on at the start of the main window.

An existing header text can be disabled by using the TOP.. ENDTOP command pair without enclosing any text lines between the two command lines:

```
/:TOP  
/: ENDTOP
```

Subsequent pages will contain no header text.

- If the main window already contains some output then a newly specified header text takes effect on the next page only.

- The same applies to the deletion of a header text. If a header text has already been output on the current page then it cannot be retracted.
- Header texts should not be employed in texts that are printed from applications programs, such as reminder texts, order texts. These applications programs can also work with header texts via the form interface, which may lead to unexpected results.

Setting a Footer Text in the Main Window: BOTTOM - ENDBOTTOM

You can specify footer texts for the main window in a similar way to header texts. Footer texts are always printed at the bottom of the window.

Syntax:

/:BOTTOM

:

:

/:ENDBOTTOM

The lines of text enclosed between the two control commands will be output from now on at the bottom of the main window.

An existing footer text can be disabled by using the BOTTOM.. ENDBOTTOM command pair without enclosing any text lines between the two command lines:

/:BOTTOM

/:ENDBOTTOM

This and subsequent pages will contain no footer text.

- Assuming there is still sufficient space in the main window, a newly specified footer text will also be printed on the current page.
- Footer texts should not be employed in texts that are printed from applications programs, such as reminder texts, order texts. These applications programs can also work with footer texts via the form interface, which may lead to unexpected results.

Conditional Text: IF

You can use the IF control command to specify that text lines should be printed only when certain conditions are met. If the logical expression contained within the IF

command is true, then the text lines enclosed by the IF... ENDIF command pair are printed. Otherwise they are ignored.

Syntax:

/:IF condition

:

:

/: ENDIF

,

The logical expression can use the following comparison operators:

= EQ equal to

< LT less than

> GT greater than

<= LE less than or equal to

>= GE greater than or equal to

\diamond NE not equal to

The following logical operators can be used to combine conditions:

- NOT
- AND
- OR

Evaluation of both the individual logical expressions and of the combinations of expressions is performed strictly from left to right. There are no precedence rules. Bracketed expressions are not supported.

The comparison is always performed on literal values, that is, the symbols are formatted as character strings before they are compared. This is particularly significant in the case of program symbols, because the formatting of these may depend on various parameters. For example, the formatted form of a currency field employs a variable number of decimal places and a variable ‘decimal point’ symbol (a period or a comma) depending on the applicable currency key.

You can extend the IF command with the ELSE command to allow text lines to be specified that you want to print in case the condition is false. If the condition is true, the

text lines enclosed by the IF and ELSE commands are formatted; otherwise the text lines enclosed by the ELSE and ENDIF commands are formatted.

Syntax:

```
/: IF condition  
:  
/: ELSE  
:  
/: ENDIF
```

The ELSEIF command allows you to specify multiple cases.

Syntax:

```
/: IF condition  
:  
/: ELSEIF condition  
:  
/: ELSE  
:  
/: ENDIF
```

You can use any number of ELSEIF commands within one compound IF..ENDIF control command. The use of an ELSE command is then optional.

- You must not extend a condition over more than one line. Both the IF or ELSEIF command and the attached condition must be completely contained within a single line.
- You can nest IF commands.
- You must terminate an IF command with an ENDIF command. If you forget this, there will be no more output following the IF command if the condition is false.
- If a syntax error occurs in the interpretation of this command, then the command is not executed. This may have an unexpected effect on the subsequent text output. For example, if the IF statement is incorrect, then all following ELSEIF and ELSE commands will be ignored, since the opening IF command is ‘missing’. This will cause all the text lines attached to the ELSEIF and ELSE commands to be printed.

Finding a Match: CASE

The CASE command covers a special case of the multiple case IF command. Rather than allowing an arbitrary condition to be tested in each of the individual cases (as in the general multiple case IF command), the CASE command allows a given symbol to be tested against specific values until a matching value is found.

Syntax:

```
/: CASE symbol  
/: WHEN value1  
:  
/: WHEN value2  
:  
/: WHEN valuen  
:  
/: WHEN OTHERS.  
:  
/: ENDCASE
```

The symbol in the CASE line is formatted. If its value is found in one of the WHEN lines, then the text lines following this WHEN line are printed. If no matching value is found then the text lines enclosed by the WHEN OTHERS line and the ENDCASE command are printed. The WHEN OTHERS section is optional.

As with the IF command, the comparison is always performed on literal values.

- A CASE command must be terminated by an ENDCASE command.
- The WHEN OTHERS section is optional.

Inserting Print Controls: PRINT-CONTROL

You can use this command to call certain printer functions from a SAPscript text. Although you cannot enter control characters for the printer directly in your text, you can define a print control via the spool maintenance transaction SPAD that contains the printer commands you want. You can then call a print control using the PRINT-CONTROL SAPscript command.

Syntax:

```
/: PRINT-CONTROL name
```

Specify the name of the print control either with or without inverted commas.

The print control printed via PRINT-CONTROL should always be a print control created in the customer name area (Zxxx) for the device type used. Use no Sxxx print controls, since their contents may be changed by SAP at any time with a release or driver modification. To modify SAP device types, refer to Note 3166.

The printer commands in the print control must never effect the current printer settings, since the SAPscript printer driver has no information on any changes triggered by a print control, which would have to be undone by commands of the driver itself. The printer driver assumes that the print control has no effect on any texts or graphics printed afterwards.

Never use PRINT-CONTROL to control print attributes covered by the SAPscript driver. To change such print attributes, use the usual method of defining form and style and the device type accordingly.

- The contents of the print control called are transparent to SAPscript. SAPscript cannot check whether the printer commands contained in the control are correct. Therefore, if you experience problems when printing a text containing these commands, you should first try to print the text without the print controls. Then, reintroduce the PRINT-CONTROL commands one by one until the command causing the problem is identified.
- You should ensure that the printer control sequences you define leave the printer afterwards in a well-defined state. SAPscript assumes that after completing each text output certain settings (for example, font, current page) are still valid for subsequent printing. If your printer commands change these settings without resetting them again afterwards, the results may be unpredictable.

After executing a PRINT-CONTROL command, SAPscript inserts a space character at the start of the next text line. If you do not want this, you should give this text line the '=' paragraph format.

Boxes, Lines, Shading: BOX, POSITION, SIZE

Use the BOX, POSITION, and SIZE commands for drawing boxes, lines, and shading to print particular windows within a form or passages of text within a window in a frame or with shading.

The SAP printer drivers that are based on page-oriented printers (the HP LaserJet driver HPL2, the Postscript driver POST, the Kyocera Prescribe driver PRES) employ these

commands when printing. Line printers and page-oriented printers not supported in the standard ignore these commands. You can view the resulting printer output in the SAPscript print preview.

Syntax:

1. **/: BOX [XPOS] [YPOS] [WIDTH] [HEIGHT] [FRAME] [INTENSITY]**
2. **/: POSITION [XORIGIN] [YORIGIN] [WINDOW] [PAGE]**
3. **/: SIZE [WIDTH] [HEIGHT] [WINDOW] [PAGE]**

BOX Command

Syntax

/: BOX [XPOS] [YPOS] [WIDTH] [HEIGHT] [FRAME] [INTENSITY]

Effect: draws a box of the specified size at the specified position.

Parameters: For each of XPOS, YPOS, WIDTH, HEIGHT, and FRAME, you must specify both a measurement and a unit of measurement. Specify the INTENSITY parameter as a percentage between 0 and 100.

XPOS, YPOS

Upper left corner of the box, relative to the values of the POSITION command.

Default: Values specified in the POSITION command.

The following calculation is performed internally to determine the absolute output position of a box on the page:

$$X(\text{abs}) = \text{XORIGIN} + \text{XPOS}$$

$$Y(\text{abs}) = \text{YORIGIN} + \text{YPOS}$$

WIDTH

Width of the box. Default: WIDTH value of the SIZE command.

HEIGHT

Height of the box. Default: HEIGHT value of the SIZE command.

FRAME

Thickness of frame.

Default: 0 (no frame).

INTENSITY

Grayscale of box contents as %.

Default: 100 (full black)

Measurements: You must specify decimal numbers as literal values (like ABAP numeric constants) by enclosing them in inverted commas. Use the period as the decimal point character. See also the examples listed below.

Units of measurement: The following units of measurement may be used:

- TW (twip)
- PT (point)
- IN (inch)
- MM (millimeter)
- CM (centimeter)
- LN (line)
- CH (character).

The following conversion factors apply:

- 1 TW = 1/20 PT
- 1 PT = 1/72 IN
- 1 IN = 2.54 CM
- 1 CM = 10 MM
- 1 CH = height of a character relative to the CPI specification in the form header
- 1 LN = height of a line relative to the LPI specification in the form header

/: BOX FRAME 10 TW

Draws a frame around the current window with a frame thickness of 10 TW (= 0.5 PT).

/: BOX INTENSITY 10

Fills the window background with shading having a gray scale of 10 %.

/: BOX HEIGHT 0 TW FRAME 10 TW

Draws a horizontal line across the complete top edge of the window.

/: BOX WIDTH 0 TW FRAME 10 TW

Draws a vertical line along the complete height of the left hand edge of the window.

/: BOX WIDTH '17.5' CM HEIGHT 1 CM FRAME 10 TW INTENSITY 15

/: BOX WIDTH '17.5' CM HEIGHT '13.5' CM FRAME 10 TW

/: BOX XPOS '10.0' CM WIDTH 0 TW HEIGHT '13.5' CM FRAME 10 TW

/: BOX XPOS '13.5' CM WIDTH 0 TW HEIGHT '13.5' CM FRAME 10 TW

Draws two rectangles and two lines to construct a table of three columns with a highlighted heading section.

POSITION Command

Syntax

/: POSITION [XORIGIN] [YORIGIN] [WINDOW] [PAGE]

Effect: Sets the origin for the coordinate system used by the XPOS and YPOS parameters of the BOX command. When a window is first started, the POSITION value is set to refer to the upper left corner of the window (default setting).

Parameters: If a parameter value does not have a leading sign, then its value is interpreted as an absolute value, in other words, as a value that specifies an offset from the upper left corner of the output page. If a parameter value is specified with a leading sign, then the new value of the parameter is calculated relative to the old value. If one of the parameter specifications is missing, then no change is made to this parameter.

XORIGIN, YORIGIN

Origin of the coordinate system.

Page 40 of 81

By Ganapati Adimulam

eMax Technologies, Ameerpet, Hyderabad

Ph No :+91 40 -65976727, Cell No : 99484 44808, 98490 34399, www.emaxtech.com

WINDOW

Sets the values for the left and upper edges to match those of the current window (default setting).

PAGE

Sets the values for the left and upper edges to match those of the current output page (XORIGIN = 0 cm, YORIGIN = 0 cm).

/: POSITION WINDOW

Sets the origin for the coordinate system to the upper left corner of the window.

/: POSITION XORIGIN 2 CM YORIGIN '2.5 CM'

Sets the origin for the coordinate system to a point 2 cm from the left edge and 2.5 cm from the upper edge of the output page.

/: POSITION XORIGIN '-1.5' CM YORIGIN -1 CM

Shifts the origin for the coordinates 1.5 cm to the left and 1 cm up.

SIZE Command

Syntax

/: SIZE [WIDTH] [HEIGHT] [WINDOW] [PAGE]

Effect: Sets the values of the WIDTH and HEIGHT parameters used in the BOX command. When a window is first started, the SIZE value is set to the same values as the window itself (default setting).

Parameters: If one of the parameter specifications is missing, then no change is made to the current value of this parameter. If a parameter value does not have a leading sign, then its value is interpreted as an absolute value. If a parameter value is specified with a leading sign, then the new value of the parameter is calculated relative to the old value.

WIDTH, HEIGHT

Dimensions of the rectangle or line.

WINDOW

Sets the values for the width and height to the values of the current window (default setting).

PAGE

Sets the values for the width and height to the values of the current output page.

/: SIZE WINDOW

Sets WIDTH and HEIGHT to the current window dimensions.

/: SIZE WIDTH '3.5' CM HEIGHT '7.6' CM

Sets WIDTH to 3.5 cm and HEIGHT to 7.6 cm.

/: POSITION WINDOW

/: POSITION XORIGIN -20 TW YORIGIN -20 TW

/: SIZE WIDTH +40 TW HEIGHT +40 TW

/: BOX FRAME 10 TW

A frame is added to the current window. The edges of the frame extend beyond the edges of the window itself, so as to avoid obscuring the leading and trailing text characters.

Steps to work with Custom Scripts:

Let us assume that we need to design the document to print the Employee Experience Letter for **eMAX Technologies**.

1. Understand the structure of the document that needs to be generated page by page.
2. Find out the different pages that form the document.
3. Decide the FIRST page of the document and the pages that are going to follow.
4. Find out the various fonts and styles (bold, italics, etc) that are used in the document.
5. Also try to group the data printed on the document into logical parts.
6. Create all the character strings that have been used in the document
7. Create all the paragraphs that have been used in the document

8. Create all the VARIABLE windows that have been uniquely identified in the document
9. Identify the MAIN window of the every page of the document
10. Define the pages that form the parts of the document
11. Assign the windows to each page
12. Define the text elements within each window
13. Use function module **OPEN_FORM** to open the layout set.
14. Use function module **WRITE_FORM** to write the text elements in various windows
15. Use function module **CLOSE_FORM** to close the layout set .

Note: Always remember to check and activate the layout set when any change is done to it, otherwise the change will not appear on the output document that is printed!!!

Details about Function Modules Used in SAP Scripts :

OPEN_FORM

The function module OPEN_FORM opens form printing. You must call this function module before you can use any other form function (WRITE_FORM, START_FORM, CONTROL_FORM...).

You need not specify a form name. If you omit the name, you must use the function module START_FORM to open a form before starting the output.

You must end form printing by using the function module CLOSE_FORM. Otherwise, the system does not print or display anything.

Function call:

CALL FUNCTION 'OPEN_FORM'

EXPORTING FORM = SPACE

LANGUAGE = SY-LANGU

DEVICE = 'PRINTER'

OPTIONS = SPACE

Export parameters:

| | |
|----------|---|
| FORM | Name of the Layout Name(FORM). If you leave the parameter blank, you must call START_FORM with a valid form name before starting any output functions. |
| LANGUAGE | Default value: SPACE Forms are language-dependent. Enter the desired language. If a form does not exist in this language, the system tries to call the form in its original language. |
| DIALOG | Default value: SY-LANGU Use parameter DIALOG to determine whether to display a dialog box before printing, in which the user can set several spool parameters for print formatting. |
| OPTIONS | Possible values: ' ' display no print parameter screen 'X' display print parameter screen Default value: 'X' Use parameter OPTIONS to set several options for print formatting. The parameter has the structure ITCPO. The user can change some of the defined settings on the print control screen. Structure: ITCPO |
| OPTIONS | The parameter OPTIONS contains invalid values for the formatting options. Possible errors: <ul style="list-style-type: none">• The output device specified in field TDDEST does not exist.• The field TDPAGESLCT for selecting the pages |

to be printed contains invalid characters.

UNCLOSED

The system was told to open a new form even though an old form is still active. The old form must be closed first (CLOSE_FORM or END_FORM).

WRITE FORM

The system outputs the form element specified in parameter ELEMENT into the currently opened form.

In the parameter WINDOW you can specify the name of a window for the output. Remember that the form element must be defined in this window. The parameter FUNCTION specifies how to merge the text lines to be output with any existing contents in the window. In this case, there are differences between the different window types or areas.

Function call:

CALL FUNCTION 'WRITE_FORM'

EXPORTING ELEMENT = SPACE

WINDOW = 'MAIN'

FUNCTION = 'SET'

TYPE = 'BODY'

IMPORTING PENDING_LINES =

EXCEPTIONS ELEMENT =

FUNCTION =

TYPE =

UNOPENED =

UNSTARTED =

WINDOW =

Export parameters:

| | |
|----------|---|
| ELEMENT | Specify the name of the text element you want to output into the form window specified in the parameter WINDOW. The element must be defined in that form window. If you specify no element, the system uses the default element, if one is defined in the form. Default value: SPACE |
| WINDOW | Specify the name of the window into which you want to output the form element specified in the parameter ELEMENT. Default value: 'MAIN' |
| FUNCTION | The parameter determines how to output the text element into the respective window. The output type depends on the window type and area: Window type MAIN, area BODY: 'SET' append to previous output 'APPEND' same as SET 'DELETE' no effect Window type MAIN, areas TOP and BOTTOM; all other windows: 'SET' delete old window or area contents and output the element 'APPEND' append the element to the existing elements 'DELETE' delete the specified element from the window or area DELETE in the TOP area (headings) takes effect only on the next page. You can no longer delete any heading from the TOP area after outputting text to the BODY |

area.

Default value: 'SET'

TYPE

The system interprets this parameter only for output to the main window.

The parameter determines the area of the main window into which you want to output the element.

Possible values:

'TOP' header area

'BODY' main area

'BOTTOM' footer area

Default value: 'BODY'

START_FORM

In-between the function modules OPEN_FORM and CLOSE_FORM, you can use different forms. This allows you to combine several different forms into one print output. **However, you can combine only those forms that have the same page format.**

To switch forms, use the function module **START_FORM**. If another form is still open, you must close it first using END_FORM.

If you specify no form name when calling **START_FORM**, the system restarts the last open form. If after OPEN_FORM no form was activated yet, the system leaves the function module with the exception UNUSED.

Function call:

CALL FUNCTION 'START_FORM'

EXPORTING FORM = SPACE

LANGUAGE = SPACE

STARTPAGE = SPACE

PROGRAM = SPACE

ARCHIVE_INDEX = SPACE

IMPORTING LANGUAGE =

EXCEPTIONS FORM =

FORMAT =

UNENDED =

UNOPENED =

UNUSED =

Export parameters:

FORM

The parameter contains the name of the form you want to use for printing. If you specify no form here, the system restarts the last active form.

Default value: SPACE

LANGUAGE

Forms are language-dependent. Enter the desired language here. If the form does not exist in this language, the system tries to call the form in its original language. If the parameter LANGUAGE is empty, the system uses the language of the last active form.

Reference field: THEAD-TDSPRAS

Default value: SY-LANGU

STARTPAGE

Usually, SAPscript starts with the page specified as start page in the form definition. If you want to start output with another form page, enter the name of the desired form page here. If the desired page is not defined, the system uses the start page defined in the form.

Default value: SPACE

UNENDED

The last form opened is still open. It must first be closed using END_FORM or the form output must be closed using CLOSE_FORM.

UNOPENED

The current form function could not be executed, since the form output was not yet initialized using OPEN_FORM.

UNUSED

One of the parameters FORM or LANGUAGE contains only blanks and no form has been opened yet whose name or language could be used as defaults.

Starting a Form Again

Usually a print program does not print only one urging letter or one account statement, but several forms for different customers. To have the output for each customer begin with the start page of the form, you must start the current form again and again.

To start a form again, you must first end the current form and then open the form again. Within one print request, first call the function module **END_FORM**. It executes the final processing for the current form. Then start the form again using **START_FORM**. Output then begins again on the start page of the desired form.

CALL FUNCTION 'OPEN_FORM'

:

CALL FUNCTION 'START_FORM'

:

CALL FUNCTION 'END_FORM'

:

CALL FUNCTION 'START_FORM'

:

CALL FUNCTION 'END_FORM'

:

CALL FUNCTION CLOSE_FORM

If you use **START_FORM** and **END_FORM**, you must not specify a form for **OPEN_FORM**. However, in this case you can use the SAPscript output functions only after opening a form with **START_FORM**.

END_FORM

END_FORM ends the currently open form and executes the required termination processing. After calling this function module, no more form is active. For further output, you must start a new form using **START_FORM**.

END_FORM does not replace CLOSE_FORM, that is, you must always close any SAPscript output using CLOSE_FORM.

Function call:

CALL FUNCTION 'END_FORM'

IMPORTING RESULT =

EXCEPTIONS UNOPENED =

Import parameters:

RESULT

The parameter contains results of the print formatting process. By comparing the corresponding fields of parameter OPTIONS with those of parameter RESULT, you can determine whether the user made changes to any settings on the print control screen.

Structure: ITCPP

CLOSE_FORM

The function module closes the form opened using OPEN_FORM. The system executes any terminating processing steps for the last opened form.

You must use this function module to close form printing. Otherwise, no output appears on printer or screen.

Function call:

CALL FUNCTION 'CLOSE_FORM'

IMPORTING RESULT =

TABLES OTFDATA = ?...

EXCEPTIONS UNOPENED =

Import parameters:

RESULT

The parameter contains results of the print formatting process. By comparing the corresponding fields of parameter OPTIONS with those of parameter RESULT, you can determine whether the user made changes to any settings on the print control screen.

Structure: ITCPP

Among others, the structure ITCPP contains a field with the name of USEREXIT. This field tells you how the user left the print view:

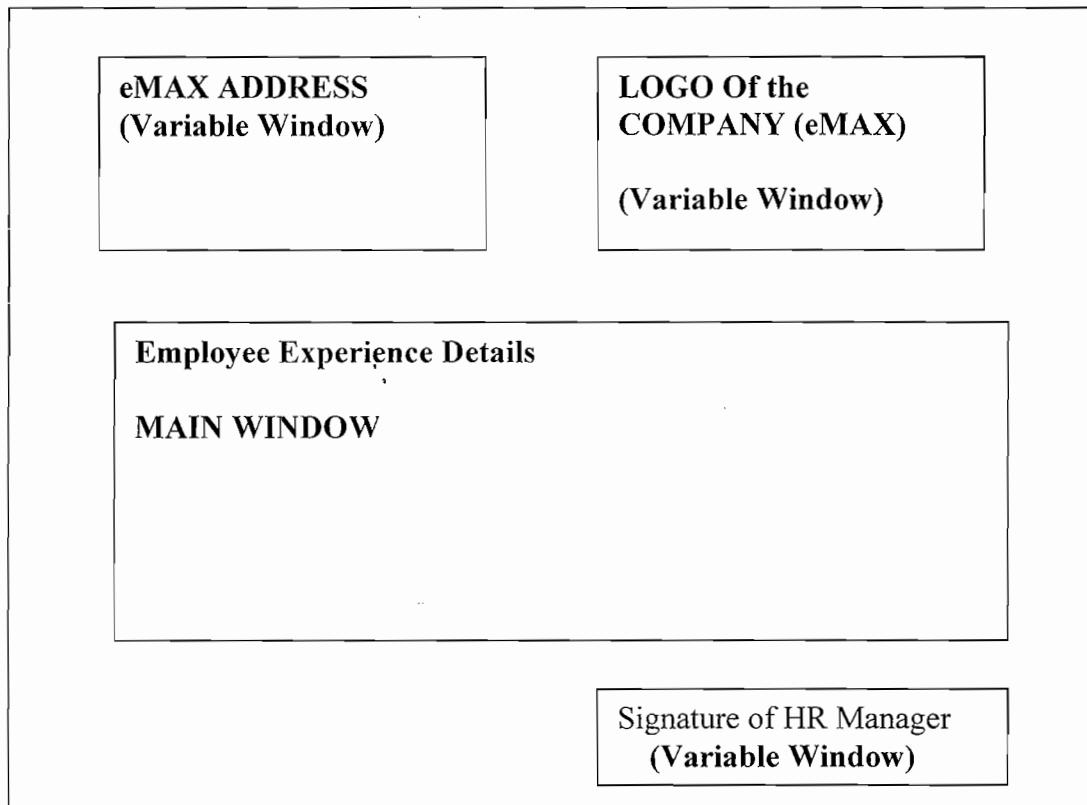
Characters E, B, or C:

EXIT <-> E

BACK <-> B

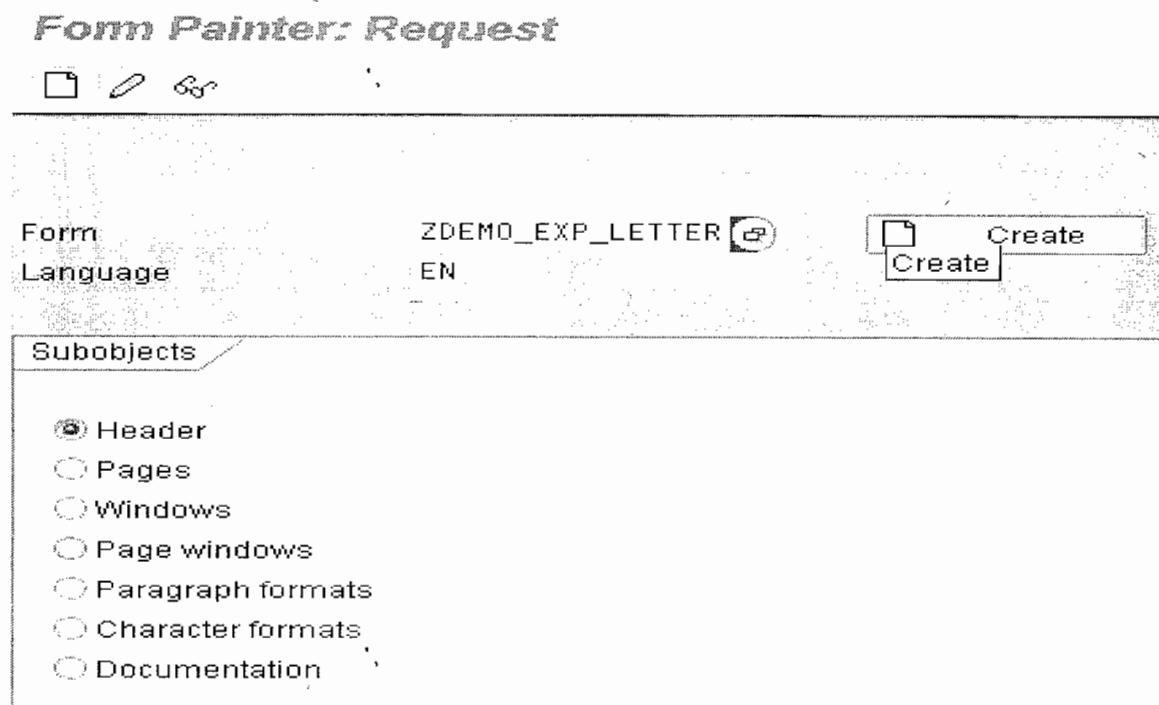
CANCEL <-> C

Step 1 : Design the Desired Layout i.e. PAGE1



Steps to Create the Layout:

Step 1 : Execute the Transaction SE71 to Design the Layout.



Enter the Form Name and Language and Click on Create [Create].

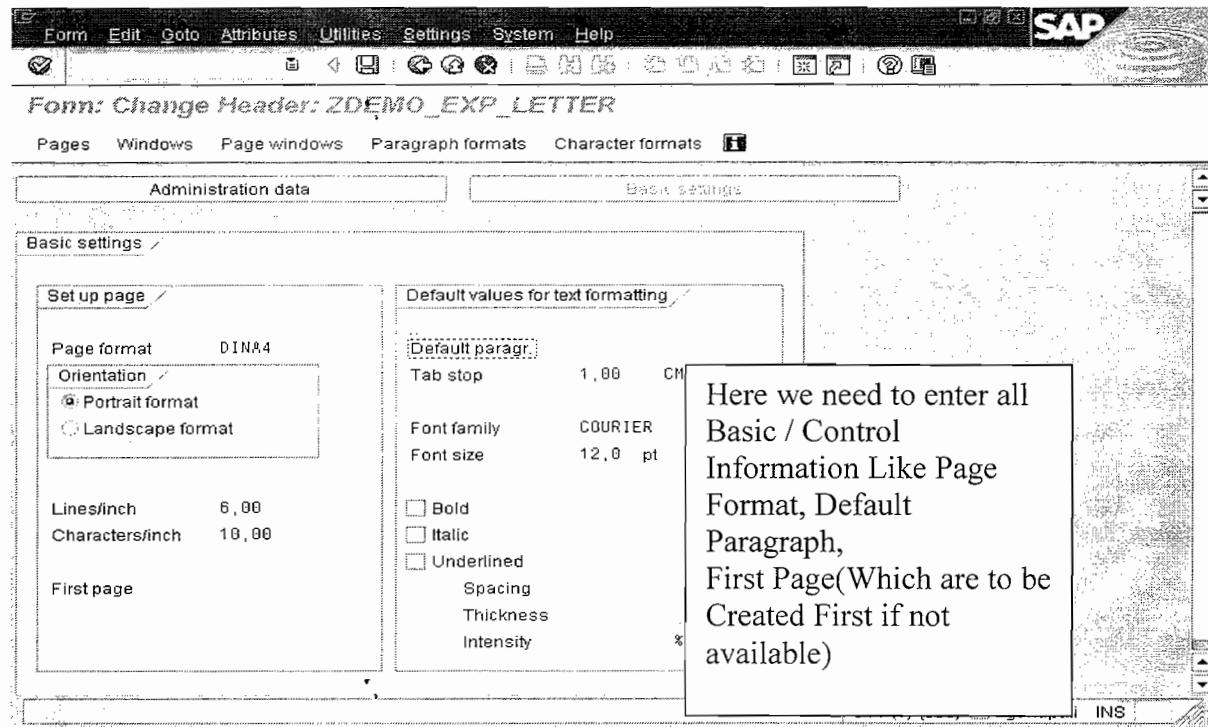
Form: Change Header: ZDEMO_EXP LETTER

This screenshot shows the 'Change Header' screen for the form ZDEMO_EXP LETTER. The top navigation bar includes tabs for Pages, Windows, Page windows, Paragraph formats, Character formats, and a search field. The main area is divided into two sections: 'Administrative data' and 'Basic settings'. Under 'Administrative data', the form details are listed: Form ZDEMO_EXP LETTER, Description Employee Experience Letter, and Status New - Not saved. Under 'Classification', the development class is listed as 800. The 'Basic settings' section contains information about the creation and last change of the form.

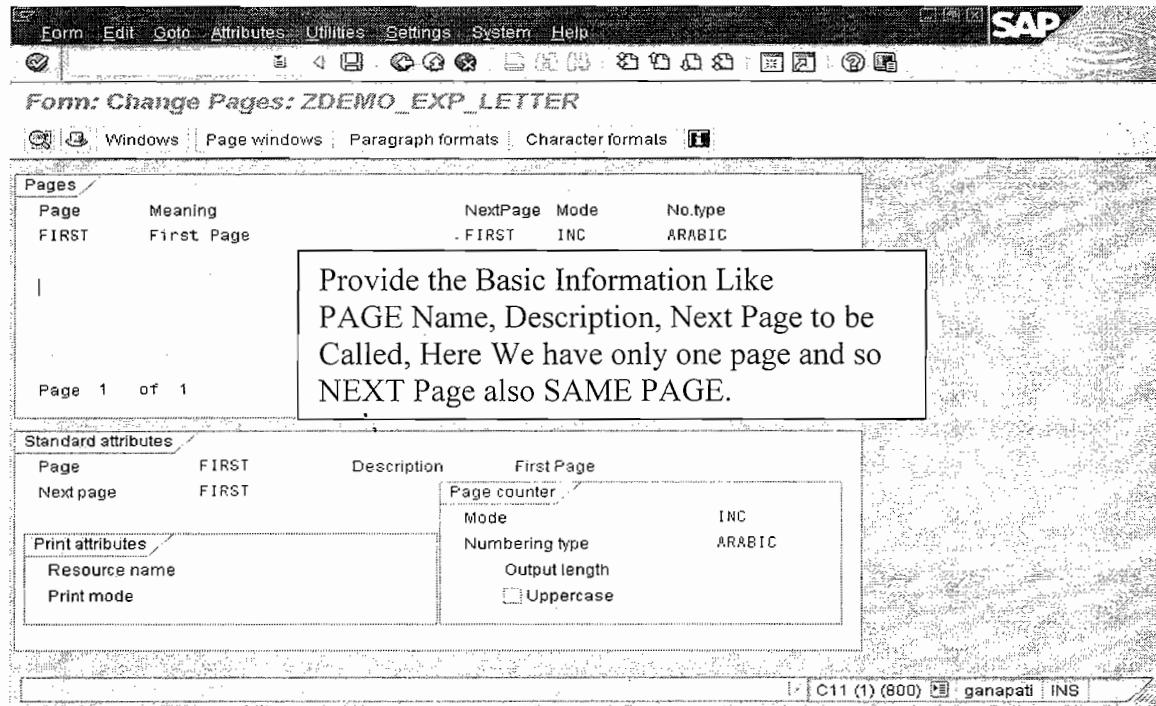
| Administrative data | | Basic settings | |
|----------------------------|----------------------------|----------------|----------|
| Administration information | | | |
| Form | ZDEMO_EXP LETTER | | |
| Description | Employee Experience Letter | | |
| Status | New - Not saved | | |
| Classification | | | |
| Development class | 800 | | |
| Client number | 800 | | |
| Created on | 00:00:00 | by | Release: |
| Changed on | 00:00:00 | by | Release |

SAP Scripts
We Never Compromise in Quality, Would You?

Enter the Short Text and Click on Basic Settings Basic settings .

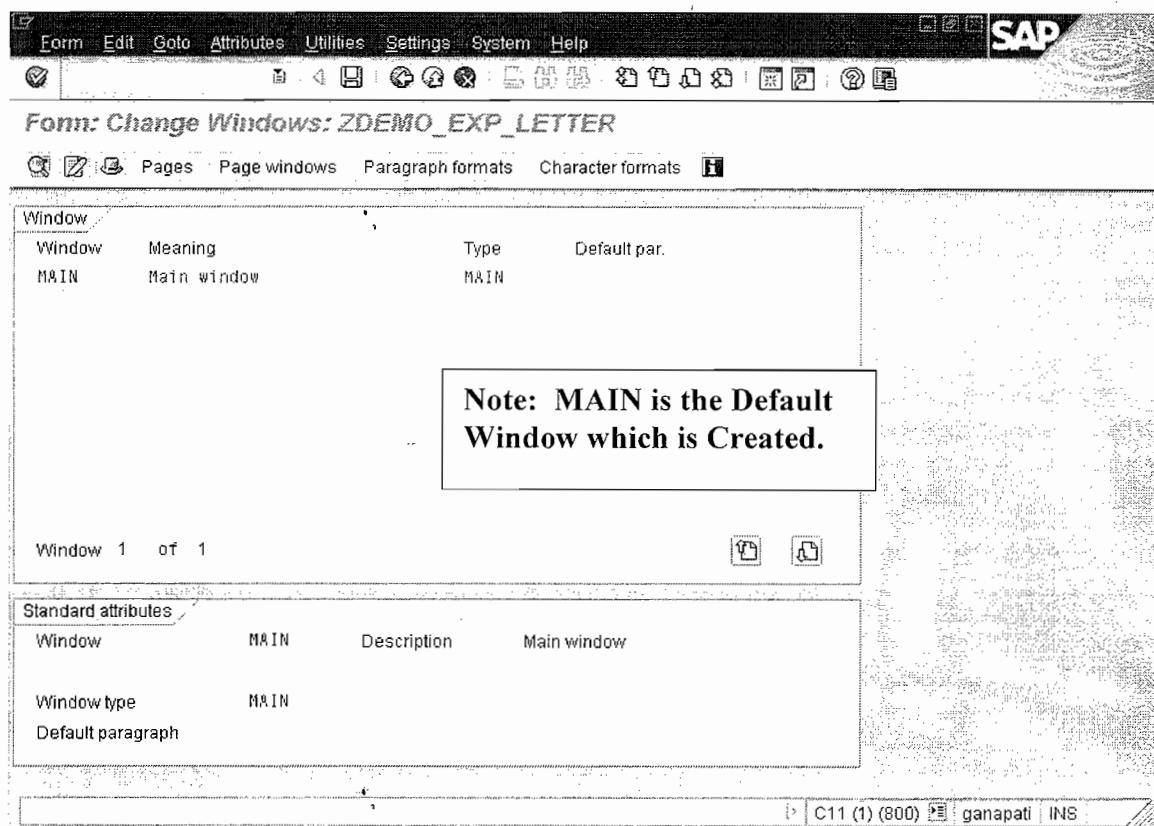


Click On Pages TAB to Create the Pages .



SAP Scripts
We Never Compromise in Quality, Would You?

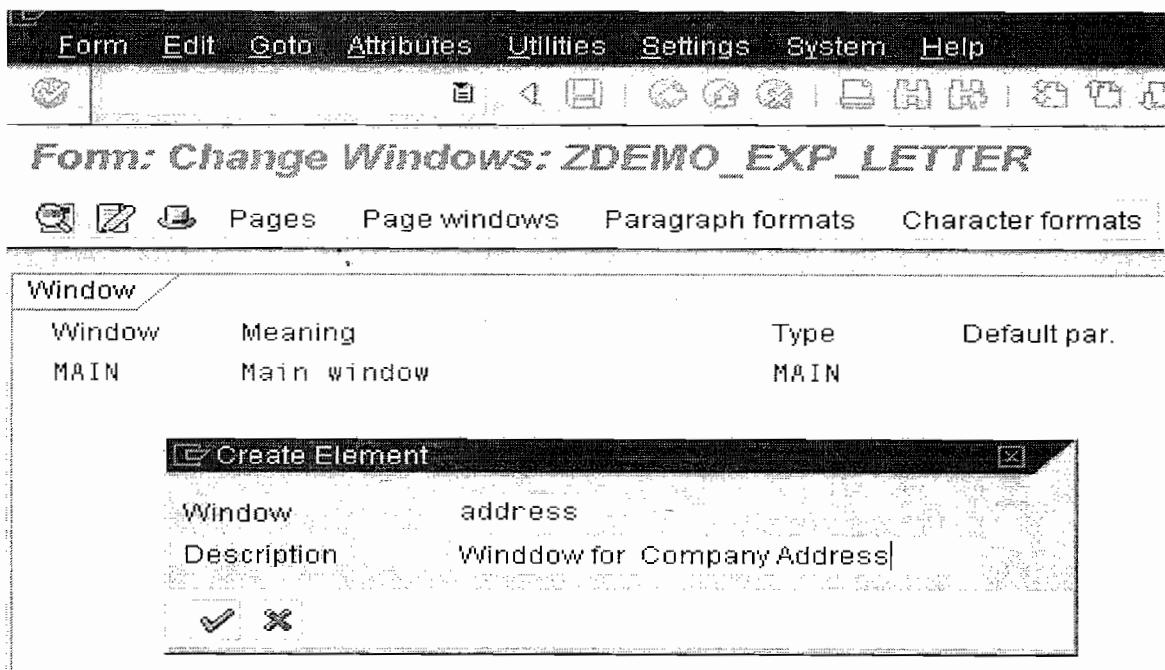
Click On Windows TAB  to Create Windows.



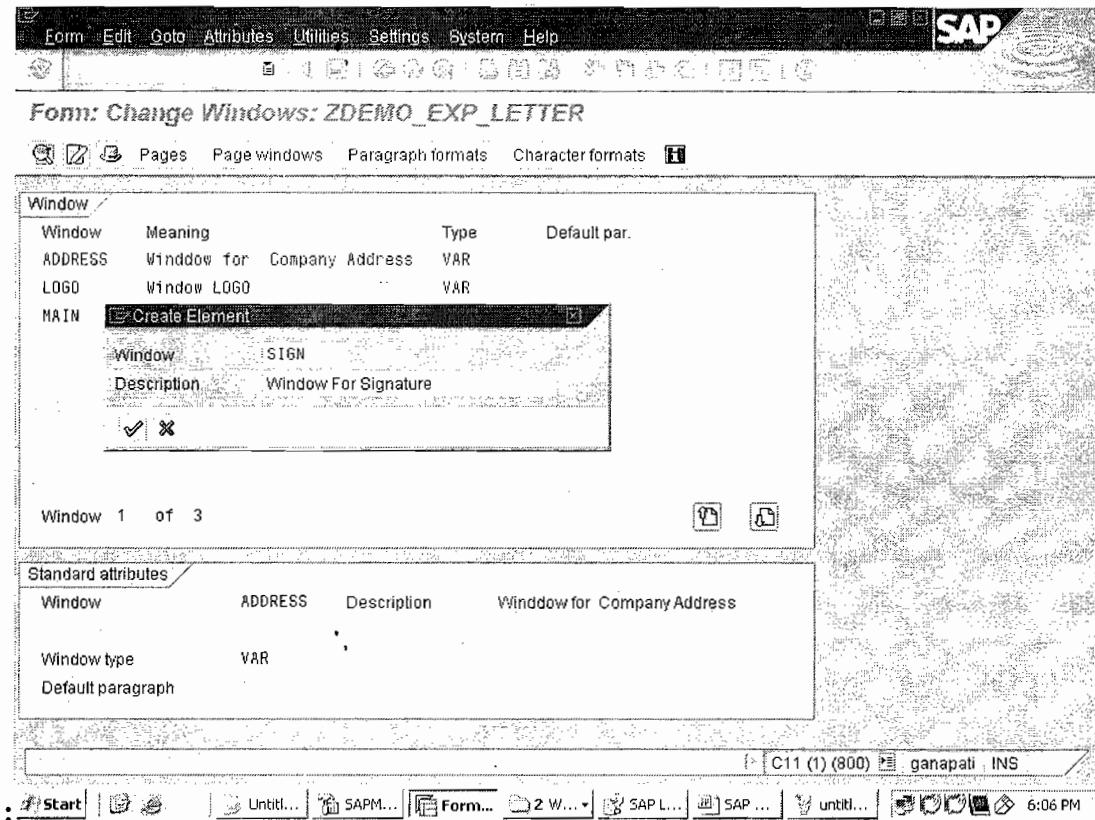
From the Same Screen Edit → Create element Create another Window.

NOTE: Edit → Create element Creates the Respective SAP Script Component. I.e If PAGES TAB is Active, it Creates PAGES and If PAGEWINDOWS TAB is Active it Creates PAGEWINDOWS.

SAP Scripts
We Never Compromise in Quality, Would You?

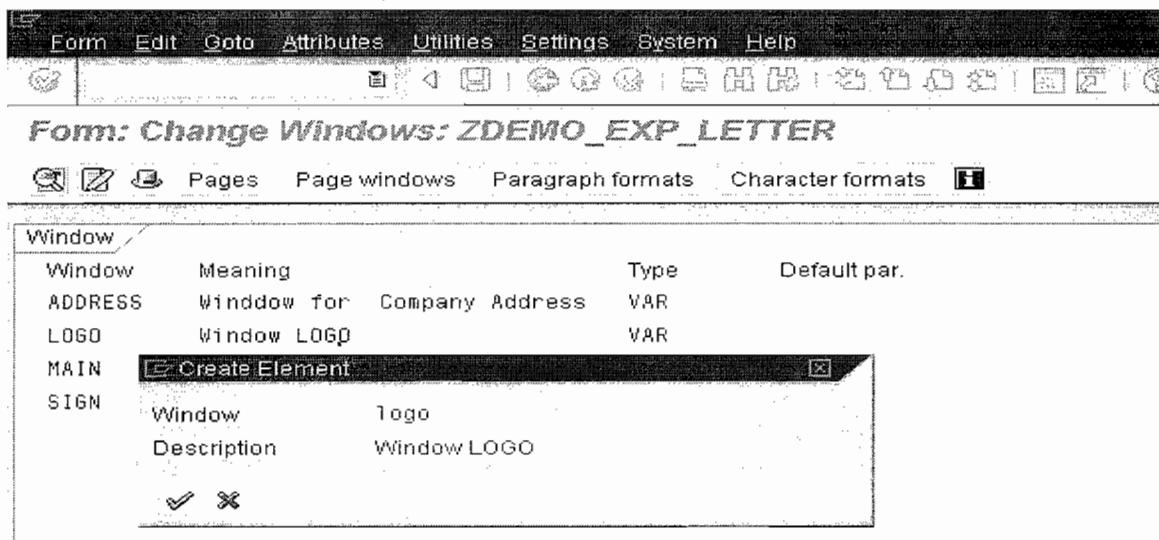


And repeat the same for LOGO and Signature Windows. **For Signature**



For LOGO:

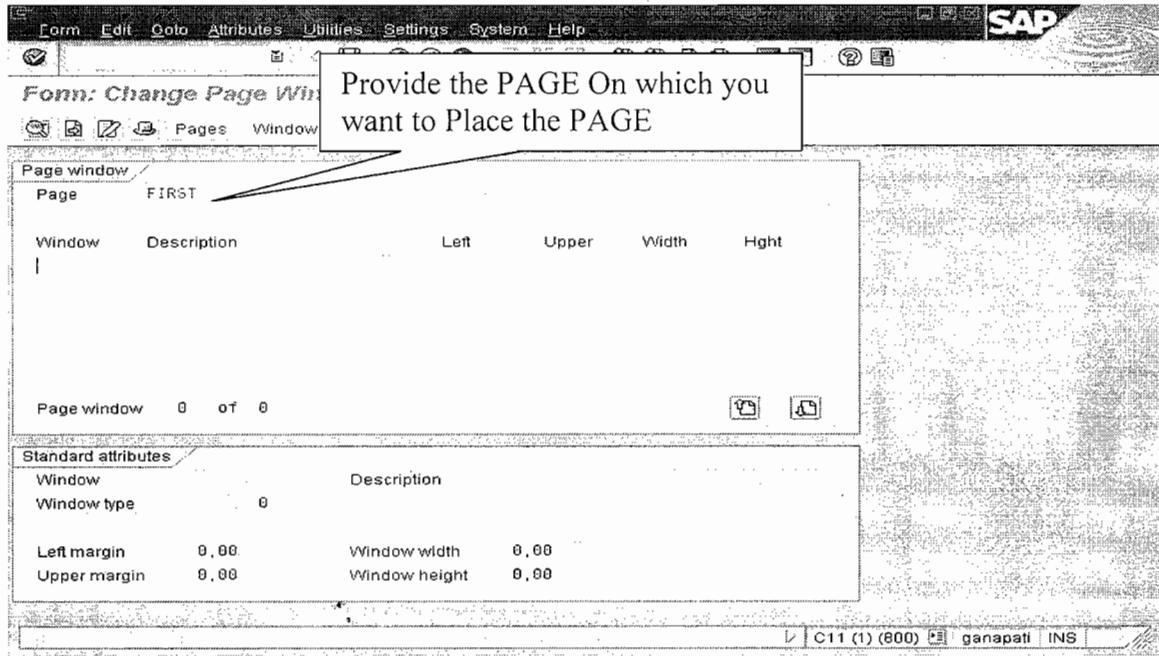
SAP Scripts
We Never Compromise in Quality, Would You?



SAVE it.

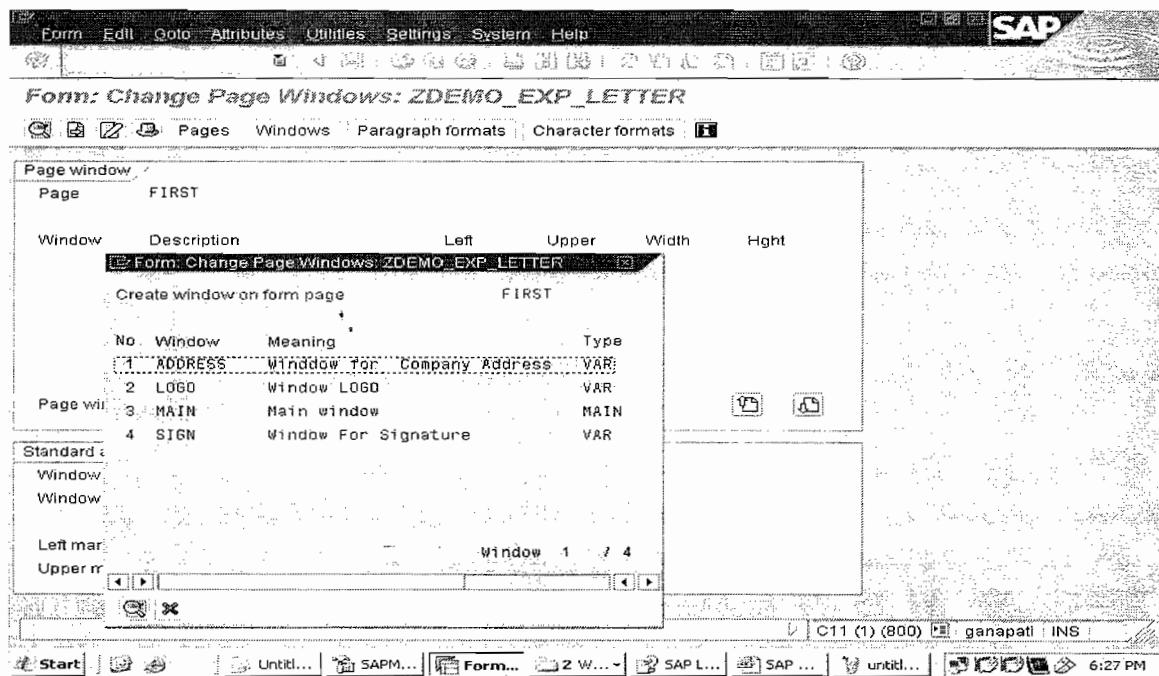
Create PAGE WINDOWS. Click on the TAB Page windows.

Note : Creating PAGEWINDOWS is Nothing but Providing the Wndow Co-ordinates on the PAGE.



Since we are in PAGE WINDOWS TAB, Edit -> Create Element Creates PAGE WINDOWS.

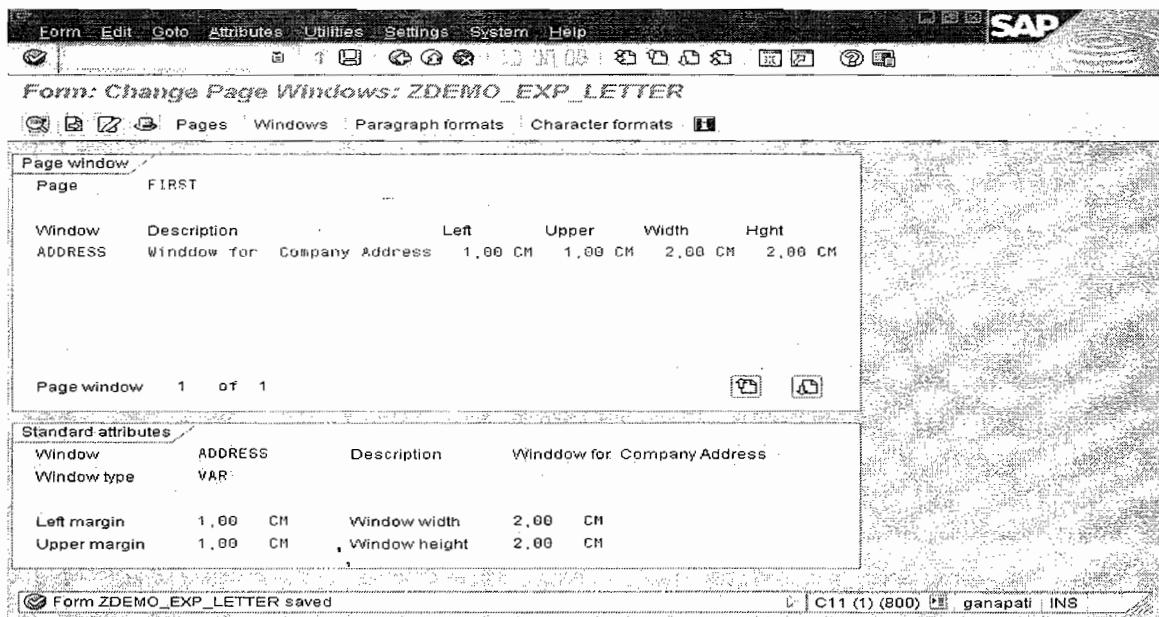
SAP Scripts
We Never Compromise in Quality, Would You?



It displays all the windows that can be placed on the PAGE, FIRST.

Double Click On the Window and Provide the Co-ordinates.

Address WINDOW :



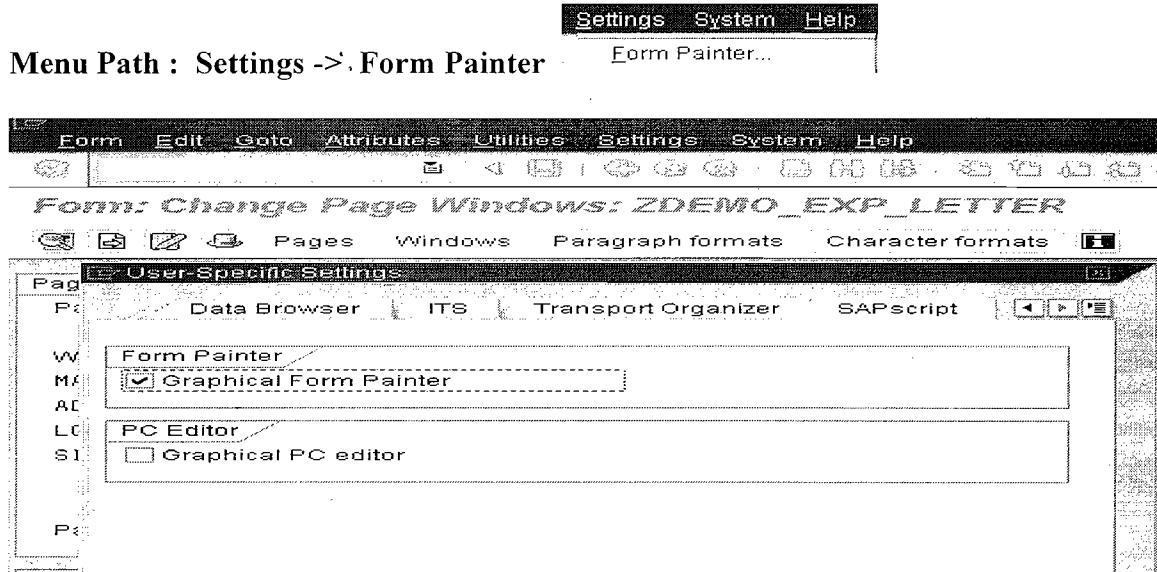
Repeat the Same for all the Windows.

After Providing all the Co-ordinates , We Can Change the Layout according to the requirement by simply Drag and Drop.

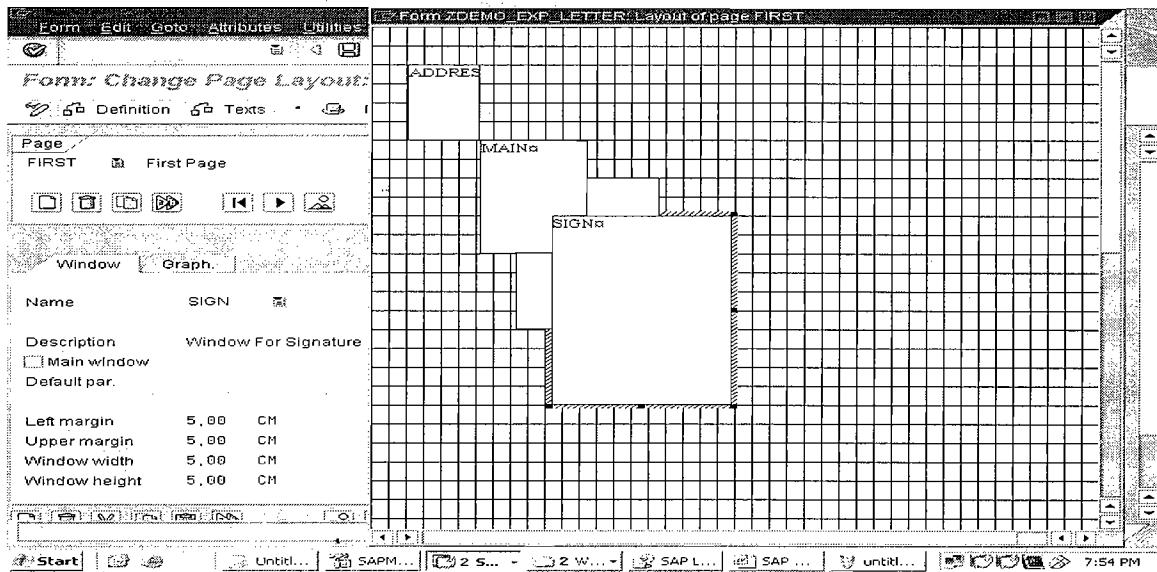
Note : In Real time we get the Exact Co-ordinates so that we can directly provide .

Other wise we can align it using the Graphical Painter until you feel GOOD.

To get the Graphical Painter Activated,

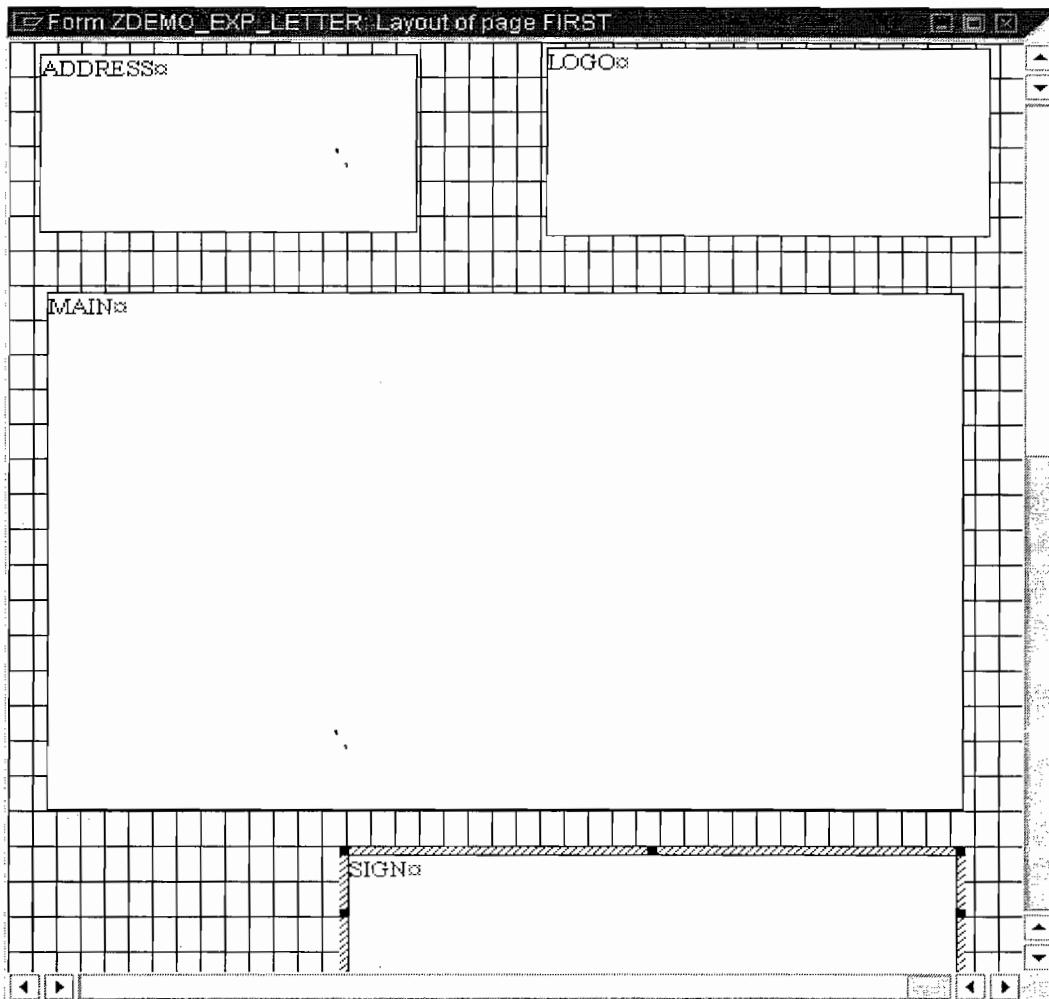


Observe the layout before we align it.



Now you Select the Corresponding Windows and Drag and Place according to the requirement and the co-ordinates will be automatically changed when we drag and Place.

Layout After the alignment .



Once you finish the Layout Design, it is better to un check the Graphical Form Painter

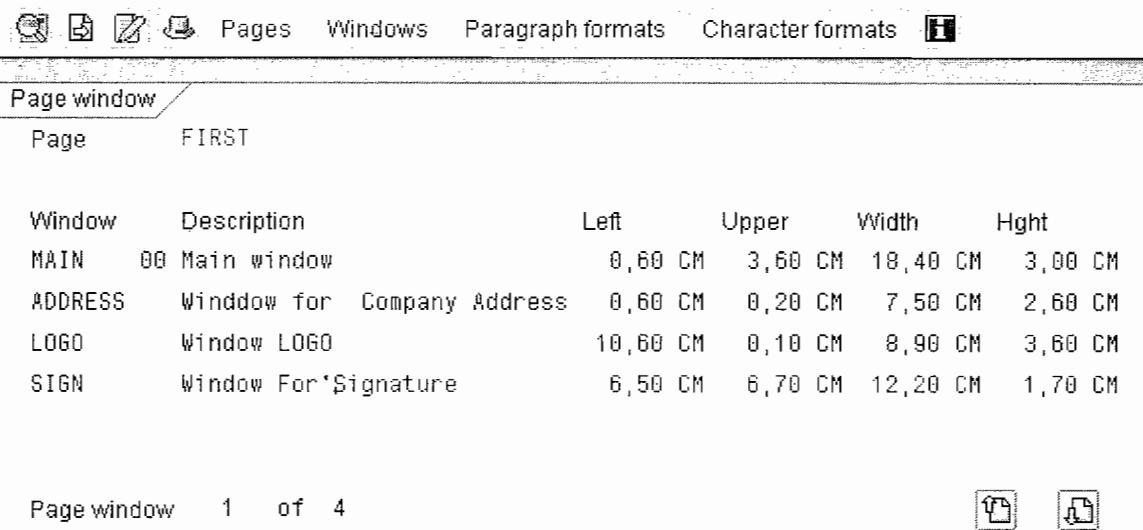
TEXT Elements Logic for Each PAGE Window:

General Procedure to check the TEXT Elements :

Open the Form , and Select the TAB Page windows

SAP Scripts
We Never Compromise in Quality, Would You?

Form: Change Page Windows: ZDEMO_EXP LETTER

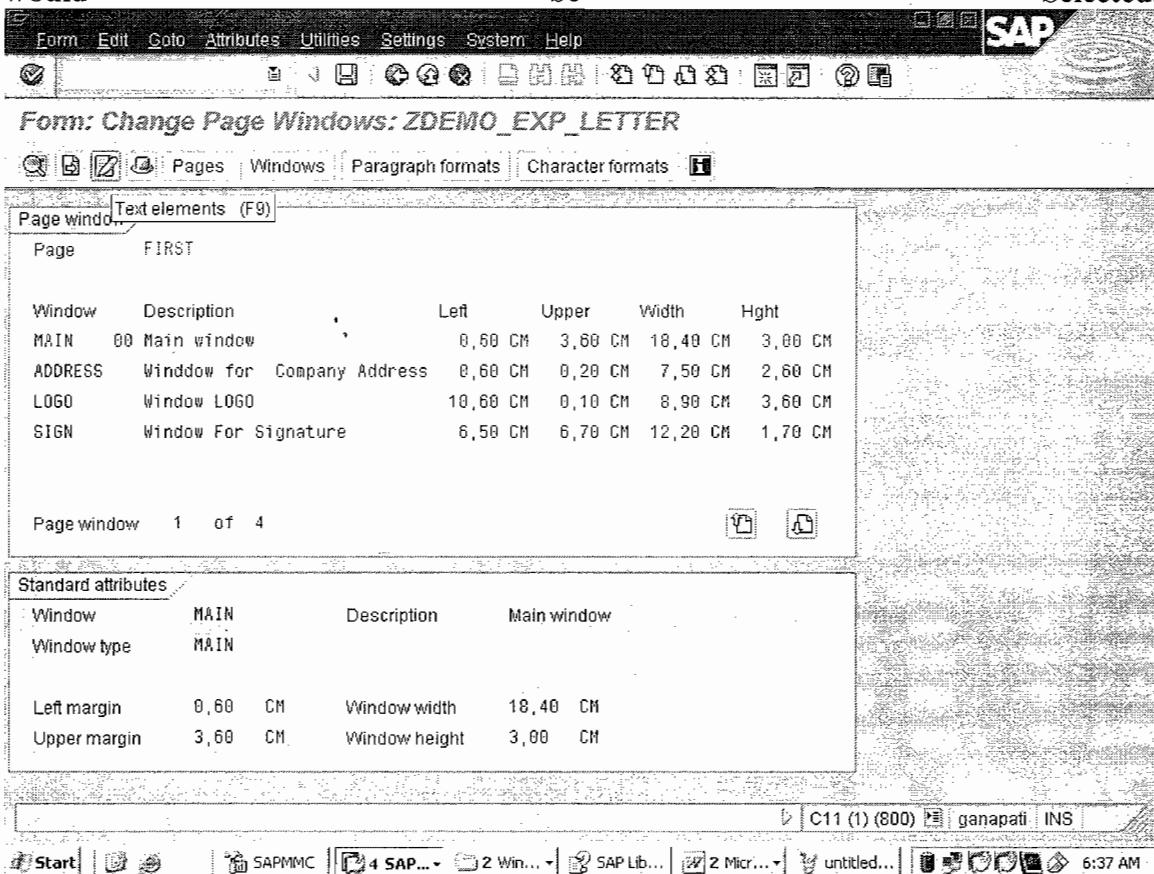


Page window

| Page | FIRST | | | | |
|---------|-----------------------------|----------|---------|----------|---------|
| Window | Description | Left | Upper | Width | Hght |
| MAIN | 00 Main window | 0,60 CM | 3,60 CM | 18,40 CM | 3,00 CM |
| ADDRESS | Winddow for Company Address | 0,60 CM | 0,20 CM | 7,50 CM | 2,60 CM |
| LOGO | Window LOGO | 10,60 CM | 0,10 CM | 8,90 CM | 3,60 CM |
| SIGN | Window For Signature | 6,50 CM | 6,70 CM | 12,20 CM | 1,70 CM |

Page window 1 of 4

Provide the Page and Double Click on the required Window so that the Window would be Selected.



Form: Change Page Windows: ZDEMO_EXP LETTER

Page window

| Page | FIRST | | | | |
|---------|-----------------------------|----------|---------|----------|---------|
| Window | Description | Left | Upper | Width | Hght |
| MAIN | 00 Main window | 0,60 CM | 3,60 CM | 18,40 CM | 3,00 CM |
| ADDRESS | Winddow for Company Address | 0,60 CM | 0,20 CM | 7,50 CM | 2,60 CM |
| LOGO | Window LOGO | 10,60 CM | 0,10 CM | 8,90 CM | 3,60 CM |
| SIGN | Window For Signature | 6,50 CM | 6,70 CM | 12,20 CM | 1,70 CM |

Page window 1 of 4

Standard attributes

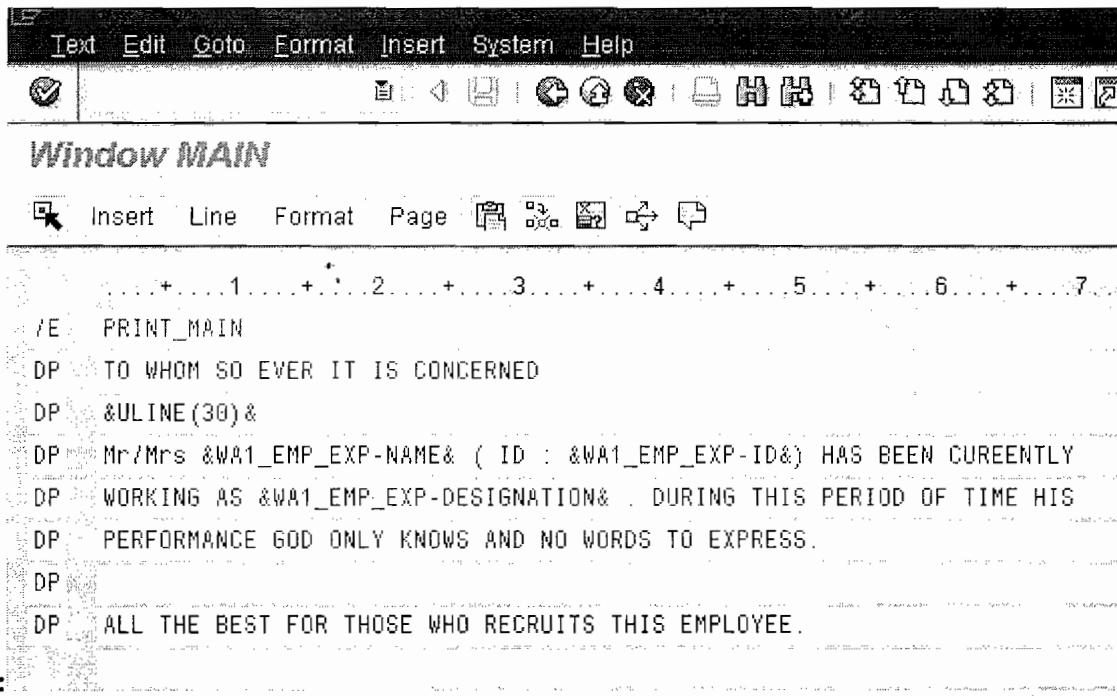
| | | | |
|--------------|---------|---------------|-------------|
| Window | MAIN | Description | Main window |
| Window type | MAIN | | |
| Left margin | 0,60 CM | Window width | 18,40 CM |
| Upper margin | 3,60 CM | Window height | 3,00 CM |

C11 (1) (800) ganapati INS
Start SAPMMC 4 SAP... 2 Win... SAP Lib... 2 Micr... untitled... 6:37 AM

SAP Scripts
We Never Compromise in Quality, Would You?

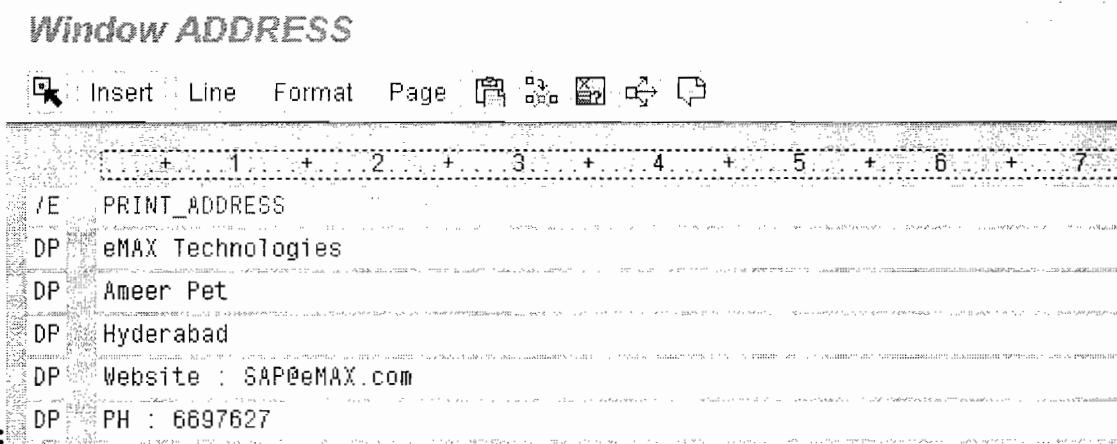
Once the Window is Selected, Select the Text Elements .

For MAIN Window



Repeat the Same Procedure for all the Windows.

For ADDRESS Window



For LOGO Window:

Window LOGO

```
+...1....+...2....+...3....+...4....+...5....+...6....+...7.  
/E PRINT_LOGO  
/: BITMAP EMAX_LOGO OBJECT GRAPHICS ID BMAP TYPE BCOL  
P1 WE NEVER COMPROMISE IN QUALITY. WOULD YOU ?
```

For SIGN (Signature) Window:

Window SIGN

```
+...1....+...2....+...3....+...4....+...5....+...6....+...7.  
/E PRINT_SIGN  
DP HR Manager  
/: BITMAP RAM_SIGN OBJECT GRAPHICS ID BMAP TYPE BCOL
```

Note :

2. Develop the Print Program .

EMAX is maintaining all the EMP Details in one Custom Table instead of Investing for HR Module.

Custom Table Details:

Dictionary: Maintain Table

ZEMP_EXP_DETAILS Active

Employee Experience Letter

Attributes Fields Currency/quant. fields

| Fields | Key | Init | Field type | Data... | Lgth. | Dec.p... | Check table | Short text |
|-------------|-------------------------------------|-------------------------------------|--------------|----------|-------|----------|-------------|-----------------|
| ID | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | ZEMPID | CHAR | 10 | 0 | | Employee ID |
| NAME | <input type="checkbox"/> | <input type="checkbox"/> | ZGNAME | CHAR | 40 | 0 | | Name |
| DOB | <input type="checkbox"/> | <input type="checkbox"/> | ZDOB | DATETIME | 8 | 0 | | Date Of Birth |
| DOJ | <input type="checkbox"/> | <input type="checkbox"/> | ZDOJ | DATETIME | 8 | 0 | | Date Of Joining |
| DESIGNATION | <input type="checkbox"/> | <input type="checkbox"/> | ZDESIGNATION | CHAR | 40 | 0 | | Designation |

Step for writing the Print Program :

- a) Design the Selection Screen to accept the range of Emp.IDs
- b) Read the emp Designation details for the given Input
- c) OPEN the FORM.
 - For each emp.
 - START the FORM.
 - Print the Data on all the WINDOWS
 - END the FORM

CLOSE the FORM.

```
*****  
* PROGRAM :ZDEMO_EXP LETTER *  
* AUTHOR :GANPATI . ADIMULAM *  
* PURPOSE :PRINT PROGRAM TO PRINT THE EMP.EXP. LETTER*  
* *  
* REFERENCE : NA *  
* COPIED FROM : NA *  
* TRAPORT REQUEST NO : C11D2K9001 *  
*****
```

REPORT ZDEMO_EXP LETTER .

```
DATA : BEGIN OF WA_EMP_EXP,  
      ID TYPE ZEID,  
      NAME TYPE ZGNAME,  
      DOJ TYPE ZDOJ,  
      DESIGNATION TYPE ZDESIGNATION,  
      END OF WA_EMP_EXP.
```

```
DATA IT_EMP_EXP LIKE TABLE OF WA_EMP_EXP.  
DATA WA1_EMP_EXP LIKE WA_EMP_EXP.
```

```
SELECTION-SCREEN BEGIN OF BLOCK B1 WITH FRAME TITLE TEXT-000.  
SELECT-OPTIONS : S_ID FOR WA_EMP_EXP-ID.  
SELECTION-SCREEN END OF BLOCK B1 .
```

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

```
START-OF-SELECTION.  
PERFORM READ_EMP_EXP_DATA.
```

PERFORM OPEN_FORM.

SORT IT_EMP_EXP BY ID.

*NOTE : FOR EACH NEW EMPLOYEE, CALL THE SAME LAYOUT.
LOOP AT IT_EMP_EXP INTO WA_EMP_EXP.

WA1_EMP_EXP = WA_EMP_EXP.

*NOTE : PRINT THE DATA FROM WA1_EMP_EXP AS

*ALL THE FIELDS RIGHT TO THAT BECOMES *S(CHAR.FIELDS)

AT NEW ID.

PERFORM START_FORM.

PERFORM WRITE_FORM_ADDRESS.

PERFORM WRITE_FORM_LOGO.

PERFORM WRITE_FORM_PRINT_MAIN.

PERFORM WRITE_PRINT_SIGN.

PERFORM END_FORM.

ENDAT.

ENDLOOP.

PERFORM CLOSE_FORM.

```
*&-----*  
*&  Form READ_EMP_EXP_DATA  
*&-----*  
*  text  
*-----*  
* --> p1  text  
* <-- p2  text  
*-----*
```

FORM READ_EMP_EXP_DATA.

SELECT ID

NAME

DOJ

DESIGNATION

INTO TABLE IT_EMP_EXP

FROM ZEMP_EXP_DETAILS WHERE ID IN S_ID.

ENDFORM. " READ_EMP_EXP_DATA

&-----

```
*&  Form OPEN_FORM
*&-----*
FORM OPEN_FORM.
CALL FUNCTION 'OPEN_FORM'.
ENDFORM.      " OPEN_FORM

*&-----*
*&  Form WRITE_FORM_ADDRESS
*&-----*
*   text
*-----*
FORM WRITE_FORM_ADDRESS.

CALL FUNCTION 'WRITE_FORM'
EXPORTING
ELEMENT          = 'PRINT_ADDRESS'
*  FUNCTION       = 'SET'
*  TYPE           = 'BODY'
WINDOW          = 'ADDRESS'

IF sy-subrc <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*      WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
ENDIF.

ENDFORM.      " WRITE_FORM_ADDRESS

*&-----*
*&  Form WRITE_FORM_PRINT_MAIN
*&-----*
*   text
*-----*
* --> p1    text
* <- p2    text
*-----*
FORM WRITE_FORM_PRINT_MAIN.

CALL FUNCTION 'WRITE_FORM'
EXPORTING
ELEMENT          = 'PRINT_MAIN'
*  FUNCTION       = 'SET'
*  TYPE           = 'BODY'
WINDOW          = 'MAIN'
```

```
IF sy-subrc <> 0.  
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO  
*      WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.  
ENDIF.  
  
ENDFORM.          " WRITE_FORM_PRINT_MAIN  
  
*&-----  
*&  Form START_FORM  
*&-----  
*  text  
*-----  
* --> p1    text  
* <- p2    text  
*-----  
FORM START_FORM.  
  
CALL FUNCTION 'START_FORM'  
EXPORTING  
  FORM           = 'ZDEMO_EXP_LETTER'  
  
IF SY-SUBRC <> 0.  
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO  
*      WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.  
ENDIF.  
  
ENDFORM.          " START_FORM  
  
*&-----  
*&  Form END_FORM  
*&-----  
*  text  
*-----  
* --> p1    text  
* <- p2    text  
*-----  
FORM END_FORM.  
  
CALL FUNCTION 'END_FORM'.  
  
IF SY-SUBRC <> 0.  
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO  
*      WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.  
ENDIF.
```

```
ENDFORM.          " END_FORM

*&-----*
*&  Form CLOSE_FORM
*&-----*
*  text
*-----*
* --> p1    text
* <-- p2    text
*-----*

FORM CLOSE_FORM.

CALL FUNCTION 'CLOSE_FORM'

IF SY-SUBRC <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*      WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
ENDIF.

ENDFORM.          " CLOSE_FORM

*&-----*
*&  Form WRITE_FORM_LOGO
*&-----*
*  text
*-----*
* --> p1    text
* <-- p2    text
*-----*

FORM WRITE_FORM_LOGO.

CALL FUNCTION 'WRITE_FORM'
EXPORTING
ELEMENT           = 'PRINT_LOGO'
*  FUNCTION        = 'SET'
*  TYPE            = 'BODY'
WINDOW           = 'LOGO'

IF sy-subrc <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*      WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
ENDIF.
```

ENDFORM. " WRITE_FORM_LOGO

```
*&-----*
*&  Form WRITE_PRINT_SIGN
*&-----*
*   text
*-----*
* --> p1    text
* <-- p2    text
*-----*
FORM WRITE_PRINT_SIGN.

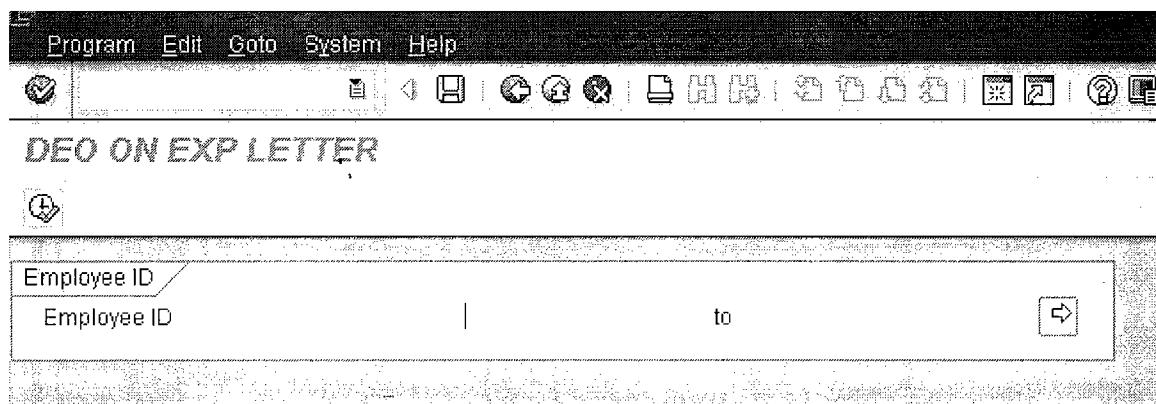
CALL FUNCTION 'WRITE_FORM'
EXPORTING
ELEMENT          = 'PRINT_SIGN'
*  FUNCTION        = 'SET'
*  TYPE            = 'BODY'
WINDOW           = 'SIGN'

IF SY-SUBRC <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*      WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
ENDIF.

ENDFORM.          " WRITE_PRINT_SIGN
```

Steps to Print the Layout :

Execute the Program:



Execute it so that it prints the Documents for all the Employee as nothing is entered in the SELECT-OPTIONS for Employee

SAP Scripts
We Never Compromise in Quality, Would You?

ID.

| | | |
|---|---------------|--|
| Output device | LP01 | <input checked="" type="checkbox"/> Beispieldrucker. Mit SPAD anpas... |
| Number of copies | 1 | |
| Page selection: | | |
| Spool request | | |
| Name | SCRIPT | SAPUSER |
| Title | | |
| Authorization | | |
| Output options | Cover page | |
| <input type="checkbox"/> Print immediately | No cover page | |
| <input type="checkbox"/> Delete after print | | |
| <input type="checkbox"/> New spool request | Recipient | |
| <input type="checkbox"/> Close spool request | Department | |
| Spool retention per. | 8 Day(s) | |
| Archiving mode | Print only | <input type="checkbox"/> |
| <input type="button" value="Print preview"/> <input type="button" value="Print"/> <input type="button" value="Cancel"/> | | |

Provide the Output Device and Click On Print Preview



Print Preview for LP01 Page 00001 of 00002

No of Pages Generate is 2 , as we have TWO Employees in the Database.

eMAX Technologies
Ameer Pet
Hyderabad
Website : SAP@eMAX.com
PH : 6697627

WE NEVER COMPROMISE IN QUALITY. WOULD YOU ?

TO WHOM SO EVER IT IS CONCERNED

Mr/Mrs GANPATI . ADIMULAM (ID : 1) HAS BEEN CUREENTLY WORKING AS SR. S/W ENGINEER . DURING THIS PERIOD OF TIME HIS PERFORMANCE GOD ONLY KNOWS AND NO WORDS TO EXPRESS.

ALL THE BEST FOR THOSE WHO RECRUTTS THIS EMPLOYEE.

HR Manager

Working with Standard Script:

Note : With Standard Scripts , We Can work in two ways.

- 1) Modifying the Layout Design and to add new windows to accommodate Logos.
- 2) Incorporating additional Functionalities through PERFROMS in Layout and FORM – ENDFORM in Subroutine Pool Programs.

1) Working With Modifying the Layout :

- a) Since SAP Script is Client dependent , We Need to Copy the standard Layout and Convert the Original Language from DE to EN as We Can always change the Layout in Original Language Only.
- B) Modify the Layout according to the Client's Requirement and Change the Script Configuration i.e Attach the Modified Layout with the Original(Standard) Layout.

Steps Find out the Standard Layout and Driver Program Details :

This Information is required to Copy the Standard Layout and also to Change the Configuration.

- a) Execute NACE Transaction. and Select the Corresponding Application

The screenshot shows a SAP interface titled "Conditions for Output Control". At the top, there are tabs: "Condition records", "Procedures", "Output types", and "Access sequences". Below the tabs is a table with two columns: "Application" and "Description". The table contains the following data:

| Application | Description |
|-------------|----------------------|
| CF | Error control |
| CV | Document administr. |
| E1 | Inbound delivery |
| EA | Purchasing RFQ |
| EF | Purchase Order |
| EL | Purch. SchAgrRelease |
| ES | Data entry sheet |
| EV | Purch. Outline Agr. |
| K1 | Sales activities |
| M1 | Rough Goods Receipt |
| MD | Direct buying |
| ME | Inventory management |
| MR | Invoice verification |
| NF | Nota fiscal |

Click On Output types.

Display View "Output Types": Overview



| Dialog Structure <ul style="list-style-type: none"> ▼ <input checked="" type="checkbox"/> Output Types <ul style="list-style-type: none"> <input type="checkbox"/> Mail title and texts <input type="checkbox"/> Processing routines <input type="checkbox"/> Partner functions | <p style="margin-bottom: 5px;">Application EF</p> <p style="margin-bottom: 5px;">Output Types</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 15%;">OutputType</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>AUFB</td> <td>Dunning Ord Confirm</td> </tr> <tr> <td>ERIN</td> <td>Reminder</td> </tr> <tr> <td>MAHN</td> <td>Dunning</td> </tr> <tr> <td>NEU</td> <td>New PO printout</td> </tr> </tbody> </table> <p style="text-align: right; margin-top: 5px;"><input type="button" value="Position..."/> <input type="button" value="New..."/> <input type="button" value="Edit..."/> <input type="button" value="Delete..."/></p> | OutputType | Name | AUFB | Dunning Ord Confirm | ERIN | Reminder | MAHN | Dunning | NEU | New PO printout |
|---|---|------------|------|------|---------------------|------|----------|------|---------|-----|-----------------|
| OutputType | Name | | | | | | | | | | |
| AUFB | Dunning Ord Confirm | | | | | | | | | | |
| ERIN | Reminder | | | | | | | | | | |
| MAHN | Dunning | | | | | | | | | | |
| NEU | New PO printout | | | | | | | | | | |

Select the New PO Printout and Double Click on Processing routines |

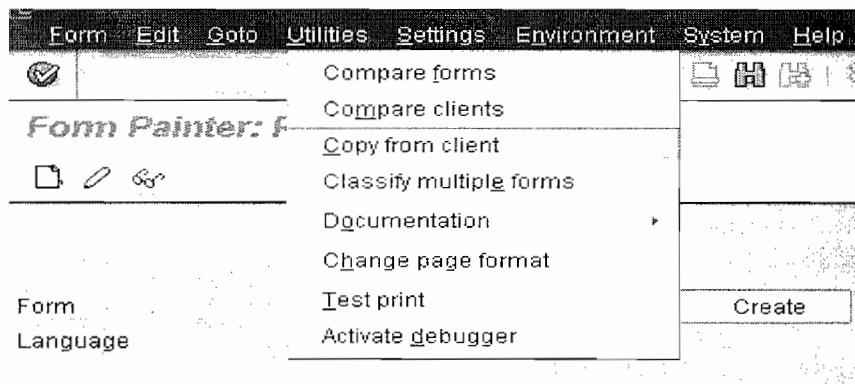
| Dialog Structure <ul style="list-style-type: none"> ▼ <input type="checkbox"/> Output Types <ul style="list-style-type: none"> <input type="checkbox"/> Mail title and texts <input checked="" type="checkbox"/> Processing routines <input type="checkbox"/> Partner functions | <p style="margin-bottom: 5px;">Outputtype NEU</p> <p style="margin-bottom: 5px;">Application EF</p> <p style="margin-bottom: 5px;">Driver Program : SAPFM06P Form Name : MEDRUCK</p> <p style="margin-bottom: 5px;">Processing routines</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 15%;">Medium</th> <th style="width: 20%;">Short text</th> <th style="width: 20%;">Program</th> <th style="width: 20%;">FORM routine</th> <th style="width: 15%;">Form</th> <th style="width: 15%;">Smart</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Print output</td> <td>SAPFM06P</td> <td>ENTRY_NEU</td> <td>MEDRUCK</td> <td></td> </tr> <tr> <td>2</td> <td>Fax</td> <td>SAPFM06P</td> <td>ENTRY_NEU</td> <td>MEDRUCK</td> <td></td> </tr> <tr> <td>3</td> <td>EDI</td> <td>RSNASTED</td> <td>EDI_PROCESSING</td> <td></td> <td></td> </tr> </tbody> </table> | Medium | Short text | Program | FORM routine | Form | Smart | 1 | Print output | SAPFM06P | ENTRY_NEU | MEDRUCK | | 2 | Fax | SAPFM06P | ENTRY_NEU | MEDRUCK | | 3 | EDI | RSNASTED | EDI_PROCESSING | | |
|---|--|----------|----------------|---------|--------------|------|-------|---|--------------|----------|-----------|---------|--|---|-----|----------|-----------|---------|--|---|-----|----------|----------------|--|--|
| Medium | Short text | Program | FORM routine | Form | Smart | | | | | | | | | | | | | | | | | | | | |
| 1 | Print output | SAPFM06P | ENTRY_NEU | MEDRUCK | | | | | | | | | | | | | | | | | | | | | |
| 2 | Fax | SAPFM06P | ENTRY_NEU | MEDRUCK | | | | | | | | | | | | | | | | | | | | | |
| 3 | EDI | RSNASTED | EDI_PROCESSING | | | | | | | | | | | | | | | | | | | | | | |

Note : This is same Place we Change Configuration and most of the times the Functional Consultants do this.

Requirement : Add the Window For Logo and print the Logo.

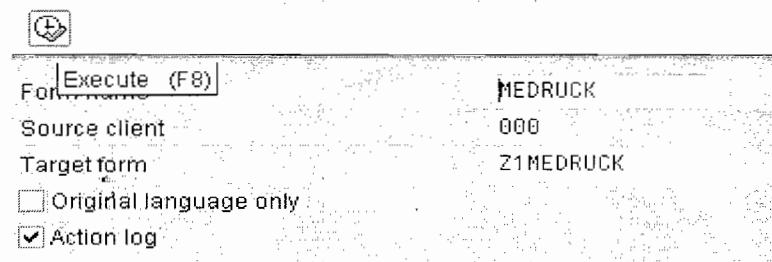
Steps to Copy the Layout for Modifications :

- a) Execute SE71 and Utilities -> Copy from Client



Provide the Source Form and Target Form and Execute it.

Copy Forms Between Clients



Copy Forms Between Clients

We need to Change the
Original Language to EN
for the Changes in English

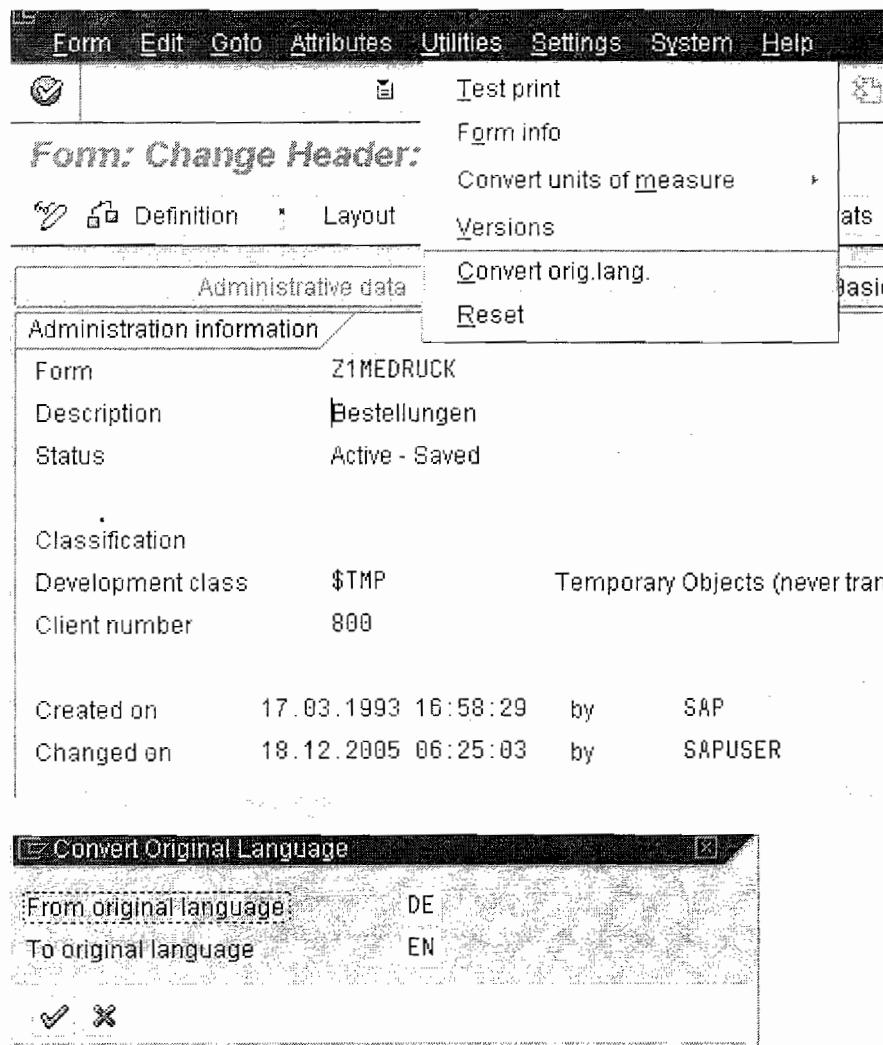
Copy Forms Between Clients

```
Z1MEDRUCK : Original language set to D
Z1MEDRUCK : Definition D copied
Z1MEDRUCK : Language P copied
Z1MEDRUCK : Language O copied
Z1MEDRUCK : Language N copied
Z1MEDRUCK : Language M copied
Z1MEDRUCK : Language K copied
Z1MEDRUCK : Language V copied
Z1MEDRUCK : Language U copied
Z1MEDRUCK : Language S copied
Z1MEDRUCK : Language R copied
Z1MEDRUCK : Language Q copied
Z1MEDRUCK : Language D copied
Z1MEDRUCK : Language C copied
Z1MEDRUCK : Language B copied
Z1MEDRUCK : Language 4 copied
Z1MEDRUCK : Language 1 copied
Z1MEDRUCK : Language J copied
Z1MEDRUCK : Language I copied
Z1MEDRUCK : Language H copied
```

Convert the Original Languages:

Open the Layout in DE (Change Mode) in SE71.

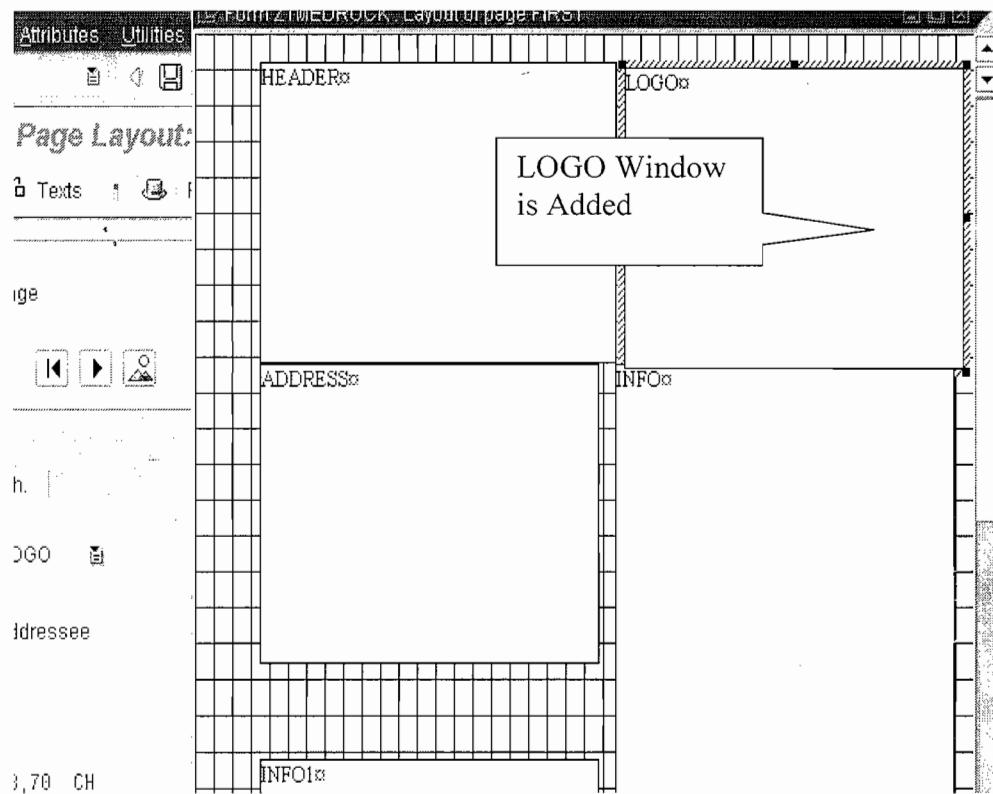
Utilities -> Convert Original Language.



ENTER.

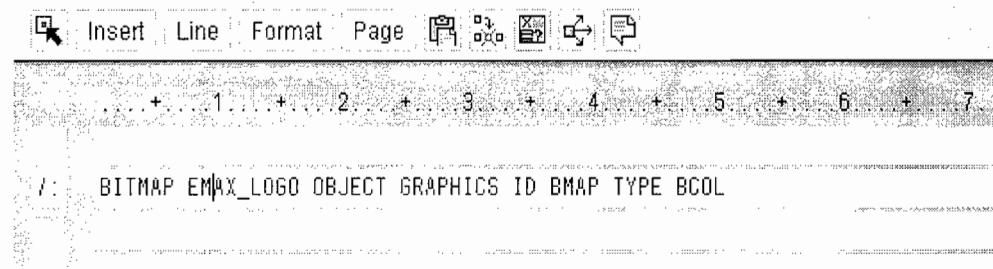
Original language of form Z1MEDRUCK converted from DE to EN

Open the English version in Change Mode and Add the page Window LOGO.



INSERT the GRAPHICS , Which is already Converted for Custom Scripts.

Window LOGO



SAVE, Check and Activate it.

Note: Once it is Activated, Link the New Layout with the Standard Driver Program.

SAP Scripts
We Never Compromise in Quality, Would You?

Execute NACE and repeat the steps to find out the standard Driver Programs and layouts.

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|--|----------------------------|----------------|---------------------|--|--------|------------|---------|--------------|---|---|--------------|----------|-----------|-----------|---|-----|----------|-----------|---------|---|-----|----------|----------------|--|
| <p>Dialog Structure</p> <ul style="list-style-type: none">▼ <input type="checkbox"/> Output Types<input type="checkbox"/> Mail title and texts<input type="checkbox"/> Processing routines<input type="checkbox"/> Partner functions | <table border="1"><tr><td style="padding: 5px;">Output type Application</td><td style="padding: 5px;">NEU EF</td></tr><tr><td colspan="2" style="padding: 5px;">Processing routines</td></tr><tr><td style="padding: 5px;">Medium</td><td style="padding: 5px;">Short text</td><td style="padding: 5px;">Program</td><td style="padding: 5px;">FORM routine</td><td style="padding: 5px;">P</td></tr><tr><td style="padding: 5px;">1</td><td style="padding: 5px;">Print output</td><td style="padding: 5px;">SAPFM06P</td><td style="padding: 5px;">ENTRY_NEU</td><td style="padding: 5px;">Z1MEDRUCK</td></tr><tr><td style="padding: 5px;">2</td><td style="padding: 5px;">Fax</td><td style="padding: 5px;">SAPFM06P</td><td style="padding: 5px;">ENTRY_NEU</td><td style="padding: 5px;">MEDRUCK</td></tr><tr><td style="padding: 5px;">6</td><td style="padding: 5px;">EDI</td><td style="padding: 5px;">RSNASTED</td><td style="padding: 5px;">EDI_PROCESSING</td><td style="padding: 5px;"></td></tr></table> | Output type Application | NEU EF | Processing routines | | Medium | Short text | Program | FORM routine | P | 1 | Print output | SAPFM06P | ENTRY_NEU | Z1MEDRUCK | 2 | Fax | SAPFM06P | ENTRY_NEU | MEDRUCK | 6 | EDI | RSNASTED | EDI_PROCESSING | |
| Output type Application | NEU EF | | | | | | | | | | | | | | | | | | | | | | | | |
| Processing routines | | | | | | | | | | | | | | | | | | | | | | | | | |
| Medium | Short text | Program | FORM routine | P | | | | | | | | | | | | | | | | | | | | | |
| 1 | Print output | SAPFM06P | ENTRY_NEU | Z1MEDRUCK | | | | | | | | | | | | | | | | | | | | | |
| 2 | Fax | SAPFM06P | ENTRY_NEU | MEDRUCK | | | | | | | | | | | | | | | | | | | | | |
| 6 | EDI | RSNASTED | EDI_PROCESSING | | | | | | | | | | | | | | | | | | | | | | |

Provide the new Layout name and SAVE it.

Entry 1 of 4

Test the Changes :

We Can test through New PO or we can change the existing PO and test it.

Execute ME22N to change the PO.

Change Some Data and Click on Print Preview.

Electronic commerce 4500012166 Created by Ulrich Grauenhorst

Document overview on | Print preview | Messages | Help | Personal setting

Electronic commerce 4500012166 Vendor | Print preview (Shift+F8) | Doc. date

Header

| S.. | Item | A | I | Material | Short text | PO quantity | O... | C | Deliv. date | Net price |
|-----|------|---|---|----------|--------------|-------------|------|----|-------------|-----------|
| 10 | K | D | | | Service item | | 1 | AU | 11.10.2002 | |
| | | | | | | | | | | |
| | | | | | | | | | | |

Item [10] Service item |

Company
Delta Steel Inc.
Baker Str.
HOUSTON AK 99694



4500012166 / 10/18/2001
Contact person/Telephone
Garner, J/551-0013

Yes. We Can Observe the LOGO on the Page.

Passing Additional Data to Standard Layout:

Note :

Here the Key thing is we need to pass the additional Data to the Layout without Changing the Driver Program.

Solution : Having the Data Retrieval Logic through FORM –ENDFORM in

Subroutine Pool Programs and Calling the FORMs , i.e Having the PERFORM in Text Elements of the Layout.

Calling ABAP Subroutines: PERFORM

You can use the PERFORM command to call an ABAP subroutine (form) from any program, subject to the normal ABAP runtime authorization checking. You can use such calls to subroutines for carrying out calculations, for obtaining data from the database that is needed at display or print time, for formatting data, and so on.

PERFORM commands, like all control commands, are executed when a document is formatted for display or printing. Communication between a subroutine that you call and the document is by way of symbols whose values are set in the subroutine.

Syntax in a FORM window:

```
/: PERFORM <form> IN PROGRAM <prog>
/: USING &INVAR1&
/: USING &INVAR2&
.....
/: CHANGING &OUTVAR1&
/: CHANGING &OUTVAR2&
.....
/: ENDPERFORM
```

INVAR1 and **INVAR2** are variable symbols and may be of any of the four SAPscript symbol types.

OUTVAR1 and **OUTVAR2** are local text symbols and must therefore be character strings.

The ABAP subroutine called via the command line stated above must be defined in the ABAP report **prog** (Subroutine Pool) as follows:

```
FORM <form> TABLES IN_TAB STRUCTURE ITCSY
  OUT_TAB STRUCTURE ITCSY.
...
ENDFORM.
```

The values of the SAPscript symbols passed with **/: USING...** are now stored in the internal table **IN_TAB**. Note that the system passes the values as character string to the subroutine, since the field **FIELD VALUE** in structure **ITCSY** has the domain **TDSYMVALUE(CHAR'80)**. See the example below on how to access the variables.

The internal table **OUT_TAB** contains names and values of the CHANGING parameters in the PERFORM statement. These parameters are local text symbols, that is, character fields. See the example below on how to return the variables within the subroutine.

From within a SAPscript form, a subroutine **GET_BARCODE** in the ABAP program **QCJPERFO** is called. Then the simple barcode contained there ('First page', 'Next page', 'Last page') is printed as local variable symbol.

Definition in the SAPscript form:

```
/: PERFORM GET_BARCODE IN PROGRAM QCJPERFO
/: USING &PAGE&
/: CHANGING &BARCODE&
/: ENDPERFORM

/ &BARCODE&
```

Coding of the calling ABAP program:

REPORT ZBARCODE .

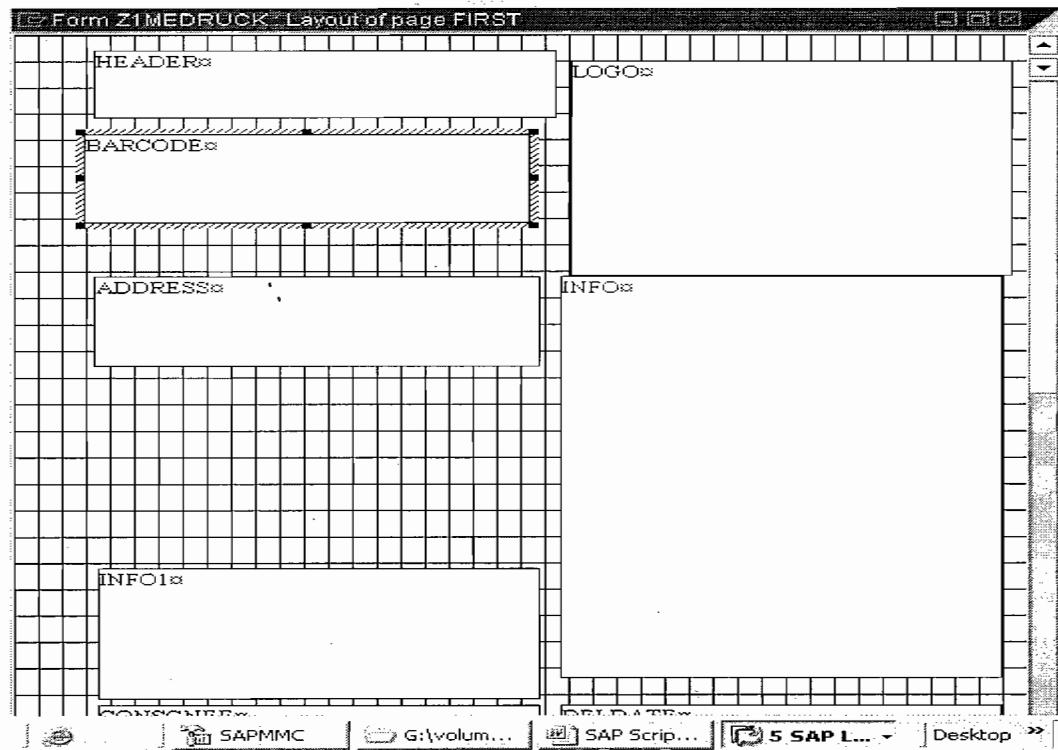
**FORM PRINT_BARCODE TABLES IN_TABLE STRUCTURE ITCSY
OUT_TABLE STRUCTURE ITCSY.**

***READ THE FIRST RECORD FROM INPUT TABLE
READ TABLE IN_TABLE INDEX 1.**

```
IF IN_TABLE-VALUE = '1'. "PAGE 1
    OUT_TABLE-VALUE = '|||||||'.
ELSEIF IN_TABLE-VALUE = '2'.
    OUT_TABLE-VALUE = '|| || || || |'.
ENDIF.
```

***MODIFY THE CHANGING VARIABLE
MODIFY OUT_TABLE INDEX 1 TRANSPORTING VALUE.
ENDFORM.**

Note : To Pass the additional Information , We have placed one more window called BARCODE to the Copied Layout.



Save, Check for Syntax and Activate it.

Window Barcode Logic in FORM :

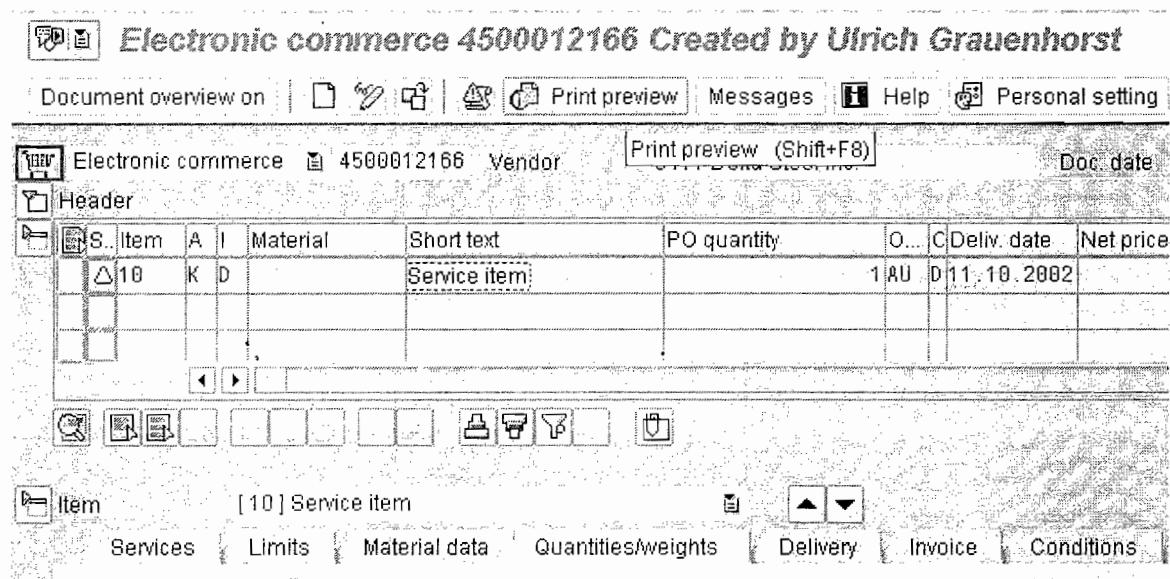
Window BARCODE

```
Insert Line Format Page  | . . . + . . . 1 . . . + . . . 2 . . . + . . . 3 . . . + . . . 4 . . . + . . . 5 . . . + . . . 6 . . . + . . . 7 . . .  
/| DEFINE &BARCODE& = ''  
/| PERFORM PRINT_BARCODE IN PROGRAM ZBARCODE  
/| USING &PAGE&  
/| CHANGING &BARCODE&  
/| ENDPERFORM  
/| &BARCODE&
```

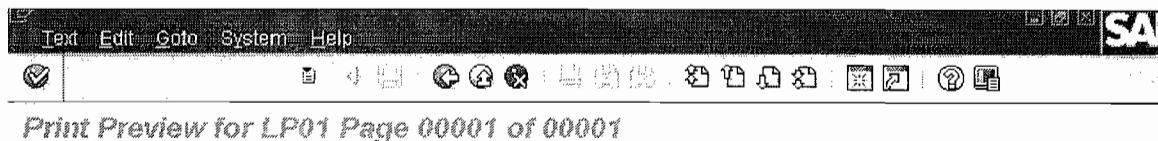
Testing :

Execute ME22N to change the PO.

Change Some Data and Click on Print Preview.



SAP Scripts
We Never Compromise in Quality, Would You?



Note : We can Test all the Purchasing Documents through Transaction ME9F.

EXERCISES

- 1) Differences between Reports and Scripts?
- 2) Can we create a Layout MAIN Window?
- 3) How Many times a MAIN Window Can be Placed on the Same Page in a Layout?
- 4) Explain the Different Ways Of Uploading Logos into Scripts?
- 5) SAP SCRIPTS are Client Independent. (TRUE/FALSE)
- 6) What is the table name that will contain all the script form names and print program names?
- 7) What is sub routine pool in sap script? When it is used?
- 8) What is Standard Script and the Transaction Code to Create?
- 9) Explain the Types Of Windows, More about MAIN Window?
- 10) Explain the Control Commands PROTECT-ENDPROTECT, SET DATE MASK, SET TIME MASK, TOP - ENDTOP, BOTTOM -ENDBOTTOM.
- 11) Explain All the Function Modules used in SAP Scripts and the Importance Of START_FORM and END_FORM?
- 12) Explain the Syntax of the Subroutines Calls in the Layout?
- 13) Explain the SAP Script Components?
- 14) Explain How to Print the Signature Only in the Last page and Address and Logo Only in First Page?
- 15) Explain What is Symbol and Types Of Symbols?
- 16) Explain the Procedure to Modify the Standard Script layout?
- 17) How to DEBUG the SAP Scripts?

EXERCISES:

- 1) Develop a Custom layout to Issues the Offer Letter for Any no Of Employees at a time?
- 2) Develop a Custom layout to Issues the Experience Letter for Any no Of Employees at a time?
- 3) Change the Standard Layout MEDRUCK (Purchase Order) to Include the Scanned Signature of the Purchasing Manager if the Total Purchaser orders Amount is more than 1 lakh Document Currency ?
- 4) Modify the Standard Layout RVINVOICE01 and Create the Standard Texts For All the Hard Coded Texts?
- 5) Modify the Standard Layout DELNOTE and Create the Standard Texts for all the Hard Coded text and maintain the Same in More than One Language?

3. MODULE POOL PROGRAMMING

- 1. Introduction to Module Pool Programming**
- 2. Data Transfer between Screen and program.**

3. Field Checks



4. Working with screen painter

5. Events

6. Sub Screens

7. Table Control

8. Tabstrip Controls

Transactions/Dialog Programming/Module Pool Programming**General Introduction to Transaction****About Transaction.**

TRANSACTION , In R/3 system is an operation that lets the user to make necessary

Changes to the database. The entire R/3 system is nothing but set of business Transaction. The data transfer from old system to SAP R/3 database or modifying data,

Or deleting data. Which is not required, is done through transaction.

For SAP system, TRANSACTION is nothing but sequence of steps called as dialog

Steps and for user it is sequence of screens which appears one after the other depending

Upon the option he is selecting. The special transaction monitor called the SAP Dispatcher handles the sequence of steps that takes place in any transaction. The main

Task of transaction is completed. All changes can be rolled back if the transaction has

Not finished.

The transaction contains two steps which are as following:

- **Interactive phase** in this step user enters the data, which needs to be inserted or deleted or modified on to the screen. These can be single screen or multiple screens depending upon the transaction. So this step can consist of single step or multiple steps. In this phase you prepare database record
- **Update phase.** This phase processes the database record and updates the database tables. Actual updation of database table takes place in this phase.

All the transactions are associated with transaction code. And all these codes are stored in table called TSTC.

Logical unit of work

The R/3 system is multi user system and many users access same information at the same time, which is mainly DATA. Consider the case where one user is modifying a record, and second user is trying to delete the same record. If the second user is successful in deleting the record then the first user will face problem for modifying the record that is already deleted. To avoid such situation, R/3 system has provided logical unit of work is defined as a locking mechanism to protect transaction integrity. Of course, these are other measures, which ensures

data integrity like check table I.e. foreign key relationship. Within SAP system there are three types of transaction and may be distinguished as:

Database transaction known as LUW. It can be defined as a period in which operation requested must be performed as a unit, i.e. all or nothing operation. At the end of LUW, either the database changes are committed or rolled back.

Update transaction or SAP LUW. One SAP LUW can have several databases LUW. So a set of a database is either committed or rolled back. The special ABAP/4 command COMMIT WORK, marks the end of a SAP LUW.

ABAP/4 transaction. is made up of a set of related task combined under one transaction code. ABAP/4 transaction functions like one complete object containing screen, menus, Transaction codes.

R/3 system has provided in built locking mechanism, which defines the logical unit of work. Also user can set his own locking mechanism. The LUW starts when a lock entry in the system table is created, and it ends when the lock is released.

To provide the user the facility to communicate with the table in order to modify or delete or insert data, R/3 has provided tool called SCREEN PAINTER. This tool allows you to design screen, process screen through program and update the database table. SAP has provided one and only one way to update the database table i.e. transaction. through you can update database table by using open SQL statement through program, SAP usually doesn't recommend this kind of updating. Many standard transactions are available to update standard table but if the need arises, the developer should be able to develop entirely new transaction, which allows the updating of database tables. This can be achieved by using various components of screen painter.

Following are the few concepts and steps for creating entire new transaction.

DYNPRO concept.

A dynpro refers to the screen + flow logic. With screen painter you can develop screen and flow logic. The relationship between screen, flow logic, and program can be shown as follows:

Dynpro, is figure indicates consist of screen and flow logic and places exactly one call to module pool program. A transaction consists of many screens and for each screen flow logic is attached. When the transaction is executed, the screen places a call to flow logic and flow logic in turn places a call to module pool program.

- A module program is nothing but usual ABAP/4 program which consist nothing but **set of modules and data declaration**.
- ABAP/4 is event driven language and for screen, also event gets triggered and these events are handled in flow logic . flow logic editor is subset of ABAP/4 editor. The system automatically displays the two important events for the flow logic.
- Screen is the important component of dynpro and can be created, designed by screen painter.

Screen painter

Starting screen painter

A screen painter can be started by

Development workbench -----→ screen painter

Or

SE51 transaction code .

Using screen painter

The process of creating a dynpro includes the creation and definition of all the needed screen components

The steps involved in creating the dynpro are as follows:

- Create screen and attributes by using screen attribute screen
- Select and place the needed field within the screen by using dict/program fields.
- Establish the field attributes to which the screen belongs by using field list.
- Define the flow logic with respect to the transaction to which it belongs by using flow logic.

About Screen Painter:

The Screen Painter is a ABAP Workbench tool that allows you to create screens for your transactions. You use it both to create the screen itself, with fields and other graphical elements, and to write the flow logic behind the screen.

Screen Painter Architecture

You use the Screen Painter to create and maintain all elements of a screen. These are:

| | |
|--------------------------|--|
| Screen attributes | Describe a screen object in the R/3 System. 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. Screen elements include checkboxes and radio buttons. |
| Elements | Correspond to screen elements. Fields are defined in the ABAP Dictionary or in your program. |
| Flow logic | Controls the flow of your program. |

Two Screen Painter Modes

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

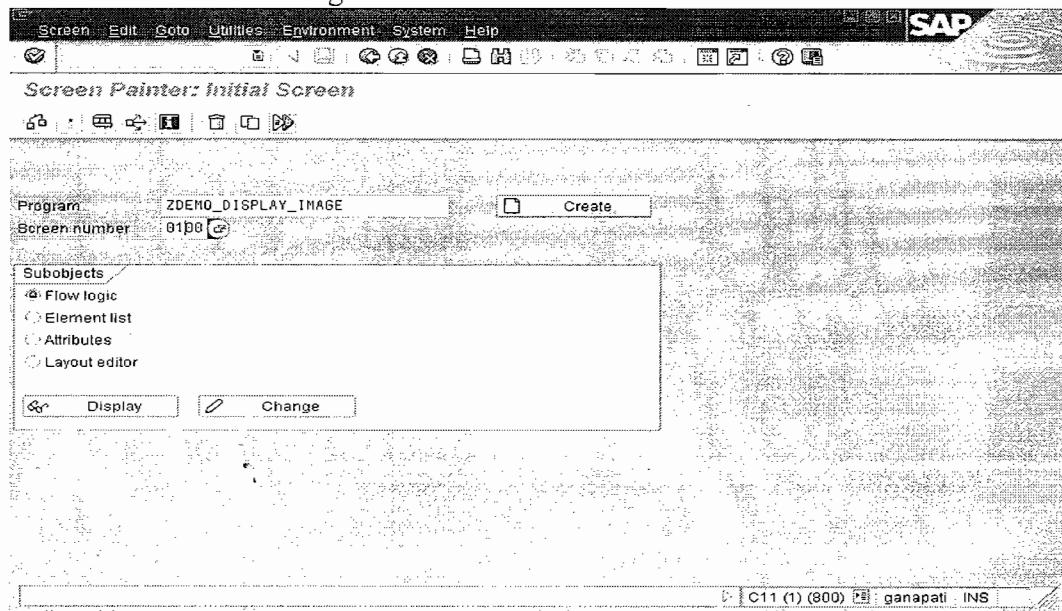
- Graphical mode and
- Alphanumeric mode.

Note : Better Go for Graphical Mode as we can use the drag and drop functionality to design the Screens.

To activate the graphical mode, choose *Utilities* → *Settings* in the Screen Painter, then select the *Graphical layout editor* option.

Creating a Screen: Basics

1. Create a screen in an existing program and define its attributes.
2. Design the screen layout and define the attributes of the elements.
3. Write the flow logic.



Creating Screens

Note : You can create a screen from the Screen Painter(SE51) initial screen or from the object list in the Object Navigator(SE80).

Procedure through SE51 :

Start the Screen Painter.

1. Enter a program name.

The program you specify should be an executable program (type 1), a module pool (type M), or a function group (type F) and must already exist.

2. Enter a screen number.

A screen number must be unique and up to 4 numbers long.

3. Choose *Create*.

The system displays the *Change Screen Attributes* screen.

4. Define the screen attributes.

Screen attributes

Screen attributes enable the system to assign and process a screen. You can set the following screen attributes:

| Attribute | Description and Ergonomic Guidelines |
|--------------------------------|--|
| Program | Name of the module pool to which the screen belongs. |
| Screen number | Identifies a unique name up to 4 numbers long. |
| Short description | Describes a screen's purpose. |
| Package | Identifies the package the screen belongs to. |
| Last changed or Last generated | Date and time when the screen was last changed or generated. |
| Screen type | If you set this option, the screen is flagged as a normal screen. This is the default setting. |
| Normal | |
| Subscreen | Identifies the screen as a subscreen. |

| | | |
|-------------------------|---------------------------------------|--|
| | <i>Modal dialog box</i> | Identifies a specialized interface for display of lists in a dialog box. |
| Settings | | |
| | <i>Hold data</i> | The system only supports the Hold data , Set data , and Delete data functions (under System → User profile) on the screen if this option is selected. The system can hold entries made on the screen at runtime in this way. The system automatically displays this data if the user calls the particular screen again. |
| | <i>Switch off runtime compression</i> | If you set this option, the screen is not compressed at runtime. |
| | <i>Hold scroll position</i> | Ergonomic guideline: You should not use this option, since empty lines may appear on the screen if you hide fields dynamically at runtime. When gaps occur, users typically need longer to process the screen. |
| Other attributes | | |
| | <i>Next screen</i> | Use this option to specify whether the vertical and horizontal scroll positions should be retained for a screen. If you set the attribute, the scroll position is retained when the user returns to the screen after processing another screen. |
| | <i>Cursor position</i> | This also applies if the length or width of the screen changes, if other subscreens are used, or if the cursor is placed outside the visible area. |
| | | This setting is intended for large screens on which the scroll position has previously been lost as a result of certain actions. |
| | | |

blank, the system uses the first screen field that can accept input.

Two Screen Painter Modes

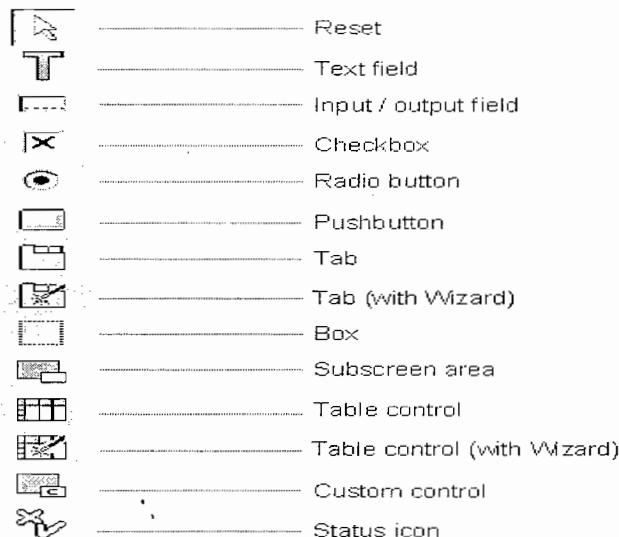
The Screen Painter has a *layout editor* that you use to design your screen layout. It works in two modes:

- **Graphical mode** and
- **Alphanumeric mode.**

Note : Better Go for Graphical Mode as we can use the drag and drop functionality to design the Screens.

To activate the graphical mode, choose *Utilities → Settings* in the Screen Painter, then select the *Graphical layout editor* option.

Elements of screen



All these elements are on the control bar of full screen editor and can be placed on the screen workarea by clicking and placing them wherever needed.

Selecting screen fields

Screen field can be either dictionary objects or program fields. Steps involved in the placing of fields on the screen are as follows:

- Click the pushbutton dict/program field on the full screen editor or goto-
→dict prog fields.
- Enter table name.

- Click get from dictionary.
- Select fields.
- Click copy pushbutton.
- Position the cursor where you want those fields to be placed.

To adjust various screen elements, you can use drag and drop facility for screen elements.

Attributes of Screen Elements

The entire elements of a screen has some attributes. Which determines their behavior.

- General – these attributes are directly managed by the screen painter like name of the element, or text of element or column width and various things associated with the screen.
- Dictionary – Dictionary attributes are applicable to fields, which are from dictionary. Various components of dictionary can be attached to this element like matchcode object, foreign key.
- Program
- Display – Behavior of the element with respect to their display feature.

Field list

This list displays a list of all screen elements together with their screen attributes. One important element of field list is OKCODE. Any pushbutton is associated with function

Code as in menu item in menu painter. When the user clicks the pushbutton this code is stored in OKCODE. This okcode is created by system without a name and is not visible on the screen. In ABAP/4 this field is work field and is nothing but a area wherein system stores the variable and is the last field of the field list and is invisible. Hence user needs to give the name OKCODE. It is not mandatory to give the name OKCODE: developer can give any name to this field.

Screen Flow Logic

You can go to this screen either by

Initial screen of screen painter --> flow logic

Or

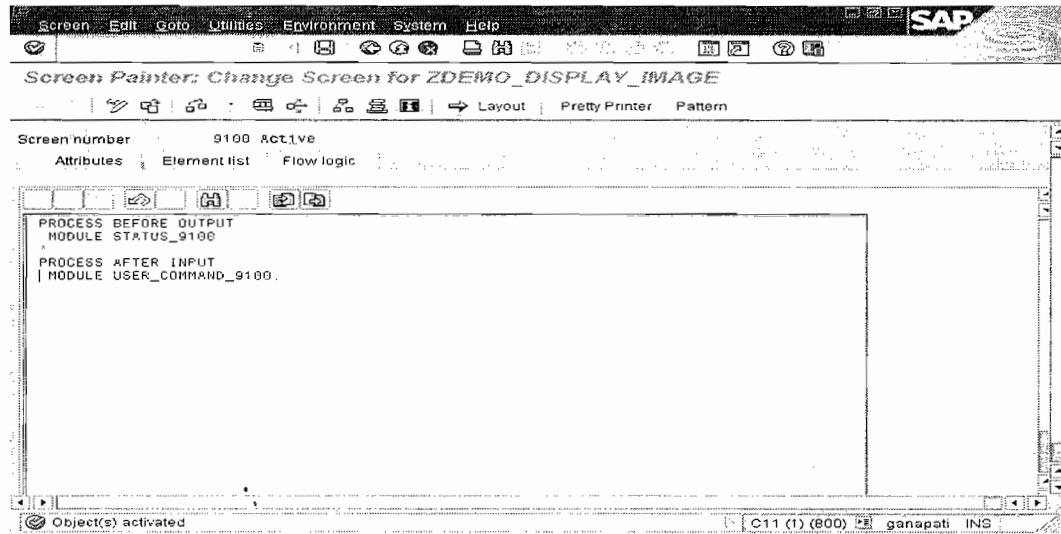
From

Screen attribute screen ---> flow logic

When transaction is executed, the screen is displayed, user enters few fields, selects few functions. Then the screen is processed and processing of screen is done by flow logic editor.

The Flow Logic Editor

To display a screen's flow logic from the Repository Browser, double-click a screen name. The system starts the flow logic editor of the Screen Painter:



Flow Logic Keywords

We can always define the flow logic in the flow logic editor of the Screen Painter, using **only** the following keywords:

| Keyword | Description |
|----------|--|
| CALL | Calls a subscreen. |
| CHAIN | Starts a processing chain. |
| ENDCHAIN | Ends a processing chain. |
| ENDLOOP | Ends loop processing. |
| FIELD | Refers to a field. You can combine this with the MODULE and SELECT keywords. |
| LOOP | Starts loop processing. |
| MODIFY | Modifies a table. |
| MODULE | Identifies a processing module. |
| ON | Used with FIELD assignments. |
| PROCESS | Defines a processing event. |
| SELECT | Checks an entry against a table. |
| VALUES | Defines allowed input values. |

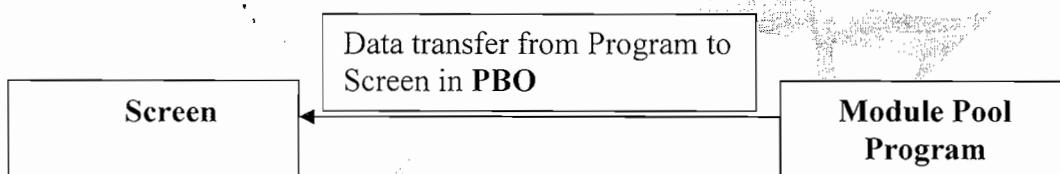
The events that are associated with screen are as follows:

- Process before output(PBO)
- Process after input (PAI)
- Process on value request(POV)
- Process on help request(POH)

The system automatically displays two very important events or modules in flow logic i.e. PAI and PBO

PBO event

This event is triggered before the screen is displayed. The processing of screen before the screen is displayed is done in this event for example filling in default values in the screen fields.



PAI event

This event is responsible for processing of screen after the user enters the data and clicks the pushbutton. The processing of screen can include displaying another screen. Or just displaying list or quitting the transaction itself and many more things. Usually it is displaying another screen. These operations can be carried out in the PAI event. OKCODE plays an important role in this operation.

POV event

Process on value request is triggered when the user clicks F4 function key. You can handle this event when the user press F4 key by writing code for the same in module pool program. Normally when the user press F4 , list of possible values is displayed. The standard list produced by system is adequate for applications you develop yourself. However, you also can have the option of setting up your own documentation and lists of possible values that are more detailed.

POH event

Normally when the user places the cursor on the field and presses F1 function key the system displays its own Help for that particular field. You can add your own functionality to the help button by writing code for the same in the POH event.

Module Pool Program

About Module Pool Program

This component though is not attached to the screen painter plays important role in transaction. Normally, for reports, on line executable programs are written but for transaction Module Pool programs are written. The module pool program contains only modules to handle various events associated with screen and data declaration statements.

System divides the module pool program into several include program. These are global field, PBO modules and PAI modules. It is entirely user's decision whether to use these modules or write directly into main program.

Creation of Module Pool Program:

You can create module pool program either through

Object Browser

System automatically creates the module pool program and for these program which are created through object browser, system creates the include module.

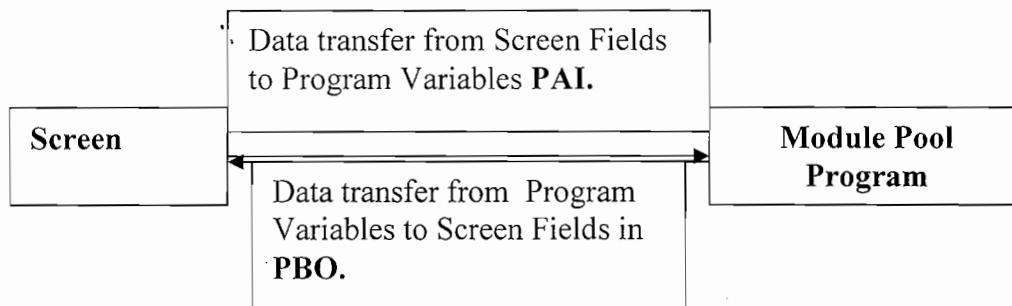
Or

ABAP/4 Editor

In this case it is similar to normal program creation. Type of program should be given 'M' and is not created by system.

Communication between Dynpro and Module Program:

For each screen the system executes the flow logic which contains corresponding events and then the control is passed to Module Pool Program. Module Pool Program handles the code for this events and again passes back control to the flow logic and finally to screen. Unlike on line program, in this case the control remains with flow logic. The switching of control between flow logic and module pool program and back is common process when user executes transaction.



Note: Data will be transferred only for the fields which has the common Names in both Screen and Module Pool Program.

Creation of a Complete Transaction:

Steps involved creating a complete transaction

- Create module pool program
- From screen painter , create screens
- Write flow logic for each screen.
- Write code for all the events in module pool program
- Check for any error in screen and flow logic
- Generate each and every component of screen i.e., flow logic and screen
- Single screen can be tested using Screen Painter
- Create transaction code through object browser
- Generate the transaction code
- User can execute the transaction by entering the transaction code in the command field.

Handling of Function Code:

The function code or OKCODE is the last field of field list. Function code can be handled as follows:

During the designing of the screen, function code is assigned to pushbutton.

- In field list, developer needs to specify OKCODE as last field.
- In Module pool program it is a global field and can be evaluated in the PAI event.
- A function code is treated in the same way, regardless it comes from pushbutton menu item or any other GUI element.

The Field Checks:

About Field Checks

As already mentioned transaction is the only method, which SAP recommends to update the database tables. Data entered in the database table should be valid and correct. Data entered is validated at each and every point. ABAP/4 offers various methods to validate data and those are as follows.

- Automatic field checks
- Checks performed in the flow logic
- Checks performed in the ABAP/4 module pool program

Automatic Field Checks:

These checks are based on the field information stored in the dictionary. These checks are performed by the system automatically when the user enters the data for the screen field. System performs these checks before PAI event is triggered.

Types of field checks performed by system are as follows.

- **Required input**

While designing the screen, for particular screen field if you click the required entry checkbox. Then the field becomes mandatory. When the transaction is executed if user leaves this particular field blank then the system displays error message. User can not process until the user enters some data.

- **Proper Data format**

Each field has its own data format whether it is table field or screen field. Whenever data is entered, system checks for the proper format of the data. For example Date. Each user has its own format for date which is defined in the user master record. If the date defined in the user master record is in the format DD/MM/YYYY, if the user enters the date say YY/DD/MM then the error message is displayed by the user. System also checks for the value of the month or days. For example if month entered is greater than twelve then the error message is displayed.

- **Valid Value for the Field**

In DDIC two tables are related by Primary Key-Foreign Key relationship. Whenever the data is entered by the user the system checks for the check table values. Also in Domain if you have fixed values then the system checks for these values.

Note : Automatic field checks are repeated each time the user enters the data.

About AT EXIT-COMMAND:

Automatic field checks can be avoided by AT EXIT-COMMAND, which works exactly the same way as CANCEL works on application toll bar. In the R/3 screen, if you want to quit the processing of that particular screen without entering the mandatory fields then user can click the CANCEL button, same functionality can be incorporated in the user defined transaction by using AT EXIT-COMMAND. This module can be called before the system executes the automatic field checks and it goes without saying that before PAI event also.

Code for AT EXIT-COMMAND in flow logic and in module pool program can be written as follows.

In flow logic,

Process After Input
Module exit AT EXIT-COMMAND.
In Module pool program.
Module exit.
Case okcode.
When 'EXIT'.
Leave to screen 0.

To achieve this kind of functionality a pushbutton or menu item should be assigned a function type 'E'. It tells the system to process this particular module before carrying out any field checks.

Flow Logic Validations:

Consider the case where you want user to enter only 'LH' and 'SQ' for T001-BUKRS. In this case you are restricting value of a screen field. This can not be achieved by automatic field check. Hence need of additional validation and can be done in flow logic by using following statement.

Field ----- Values.

SYNTAX.

PAI.

Field T001-BUKRS ('1000').

For multiple values

PAI.

Field T001-BUKRS values between 1000 and 2000.

Field T001-BUTXT values ('Emax Technologies' 'Imax Technologies').

In this case when the user enters the value, PAI is triggered and field is checked for that particular value. If entered value is wrong then that field is enabled for user to enter. If you have multiple field statements in your flow logic then it is sequential execution.

Consider the following case.

PAI.

Module 'assign'.

Field T001-BUKRS values ('1000' '2000').

In ABAP/4

Module assign.

Data: BUKRS like T001-BUKRS.

BUKRS = T001-BUKRS.

Endmodule.

In this case T001-BUKRS is used in the flow logic before the field statement. The system will give invalid value or some previous value as the field T001-BUKRS is used in module before it is checked i.e., field statement is after the module in which T001-BUKRS is being used. The field is not available to the system unless it executes the field statement. Field statement transfers the values to the program and is done only once. If you don't have FIELD statement in your flow logic then transfer of values takes place in PAI event.

Consider one more case where you have multiple field statement.

PAI

Field T001-BUKRS values ('1000').

Field T001-BUTXT values ('Emax Technologies' 'Imax Technologies').

In this case if the user enters only BUKRS wrong, then this particular field is enabled and rest of all the fields are disabled for user to input. Many times if the user enters wrong value for one field then you might want to give option to user to enter all the fields; which is not possible by using only Field statement. This functionality can be achieved by **CHAIN-ENDCHAIN**.

SYNTAX

Chain.

Field T001- BUTXT values ('Emax Technologies').

Field T001- BUKRS values between '1000' and '2000'.

Endchain.

In this case if the user enters wrong value only for BUKRS ,both the fields i.e., BUKRS and BUTXT are enabled as they are grouped together in the CHAIN statement.

Note : Usually logically related fields are grouped together with Chain-Endchain statement.

Checking field in ABAP/4

Field statement in flow logic

Module statement in ABAP/4 module pool program.

SYNTAX

PAI.

Field <Field Name> module <name>.

This module can be handled in the main program i.e., module pool program.

For Example

PAI.

Field T001-BUKRS module check..

In ABAP/4 Program.

Module check.

Select single * from T001 where BUKRS = T001-BUKRS.

If sy-subrc <> 0.

Message E001.

Endif.

In this case field T001-BUKRS is checked in the table for its existence.

Dynamically Calling the Screens:

About Displaying Next Screen.

Transaction is nothing but a sequence of screens, which are displayed one after the other. The next screen displayed depends upon the attributes of the first screen. In attributes you need to give Next Screen number i.e., if next screen displayed should be 200 screen, then this number should be given in Next Screen attributes. These are static attributes of the screen. By default if nothing is specified in the program the system branches out to the screen number which is specified in the attribute screen.

Consider a Scenario where we have different Push buttons like MARA,MARD,MARC etc..

In this case, if user selects MARA pushbutton then fields from MARA table are displayed and when the user clicks on the MARD then the fields from MARD table are displayed. From same screen depending upon user's selection the screen is branched out and this has to be done during runtime. This functionality can be achieved by dynamically calling the screen in module pool program.

The screen can branch out to new screen depending upon user selection and can be done by following command in module pool program.

- SET SCREE N

- CALL SCREEN
- LEAVE TO SCREEN <number>

All these commands overwrite the specifications given in the attributes. This overwriting is temporary in the sense that values stored in the attributes are not changed.

SET SCREEN

SYNTAX

Set screen <number>

In module pool program

Case okcode.

When 'DISP':

 Set screen 200.

When 'LIST'.

 Set screen 300.

Endcase.



In this case the entire processing of current screen takes place and then the system branches out to next screen. If you want to branch out to the next screen without processing the current screen then LEAVE SCREEN should be used along with SET SCREEN.

For example :

Case okcode.

When 'DISP':

 Set screen 200.

 Leave screen.

When 'LIST'.

 Set screen 300.

 Leave screen.

Endcase.

Note : When SET SCREEN is used control can not be transferred to the main screen or previous screen , unless you write code for the same.

CALL SCREEN

Usually used for pop-up screens. Many times, there is need for user to enter additional information or secondary information on another screen or pop-up screen. Once the user enters the data, he should be able to go back to main screen

or to the screen where he started. This is not possible by using SET SCREEN. This functionality is achieved by CALL SCREEN.

SYNTAX

CALL SCREEN 200.

Will simply call a screen number 200 from a main screen. Once the screen is displayed the user can enter all the data and return to the main screen by clicking on BACK button.

To call screen as pop-up screen the syntax is

Call Screen starting at <col.no> <line no>
Ending at <col.no> <line no> .

In this case window will be popped as window and user can close it by using BACK button.

LEAVE TO SCREEN

To SET a new screen without processing current screen you need to use the following two statements together.

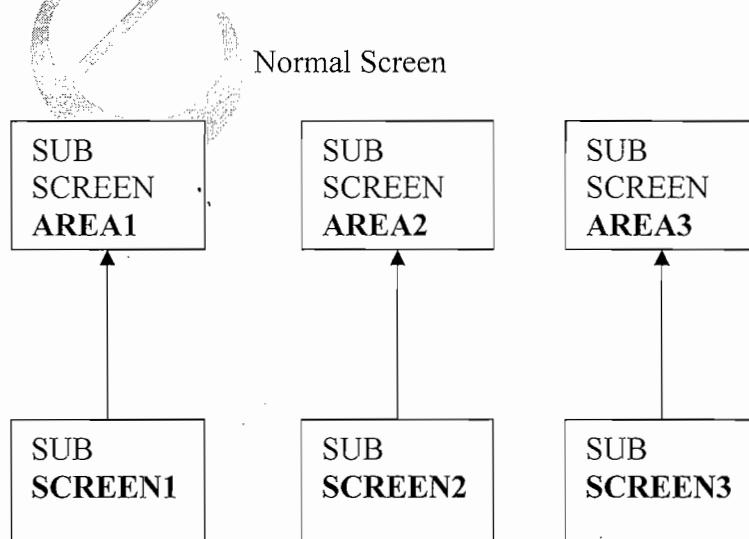
SET SCREEN <no>
LEAVE SCREEN.

Or a single statement

LEAVE TO SCREEN <no>.

SUBSCREENS & SUBSCREEN AREAS:

A subscreen is a screen within the screen and which can be called in the Subscreen Area of the Normal Screen.



Steps to create a subscreen are as follows.

- Create a subscreen area on MAIN screen and name it.
- Create a separate screen of subscreen type.
- Arrange the fields on this screen so that they fit in subscreen area exactly.
If it is larger, then only that part of the screen will be visible that fits in the main area.
- Write code for calling the subscreen in flow logic.

To call subscreen, from your flow logic you need to include the statement both in PAI and PBO.

SYNTAX.

PBO.

Call subscreen <area> including <prg name> <screen number>.

PAI.

Call subscreen <area>.

Area is the name of the subscreen are on main screen.

Prg name is the name of the module pool program.

Screen number is the subscreen screen number.

There are many Don'ts with subscreen and these are

- GUI status can not be set to the screen.
- Okcode is not applicable to the screen.
- Subscreen can not call another screen.
- It can not contain AT EXIT-COMMAND.

Note : You can call multiple subscreen in the same area (at any given point of time only one subscreen can be called in the subscreen area) and is done dynamically during runtime by using variable screen number.

TABLE CONTROLS:

About Screen Tables

A table can be created in transaction. These tables when designed on the screen are called as SCREEN TABLES. These screen tables are of two types viz.

- Table controls
- Step loops

Though these are tables when code is written to handle them the tables are treated as loops.

Feature of Table Controls

- Data is displayed in the form of table when many records match the criteria.
- Table control gives user the feeling of an actual table.
- You can scroll through the table vertically and horizontally (the way you do it for actual table)
- You can select rows and columns
- Re-size the width of a column.
- You can have separator lines in between rows and columns
- Automatic resizing of the table when the user resizes the window.

In general table control includes all the features of an actual table and user gets the feeling that he is actually working with tables. You can update information in table control and it can be updated in the database table by writing code for it.

Steps associated for creating complete screen table is as follows.

- Declaration of table control in module pool program.
- Designing of table controls on the screen.
- Passing data to table in flow logic.

Declaration of Table Control in the Module Pool Program

SYNTAX

Controls : <name> type TABLEVIEW using screen <number>

When you use table control in a screen you must declare the structure in module pool program.

Important fields of tableview are as follows:

- Lines – number of displayable rows in a table.
- Top_line – the row of table where the screen displays starts.

- Current_line – the row currently being processed inside a loop.

When you process the table control in flow logic depending upon where you want to start display of rows you need to use these variables.

Designing Table Control on Screen:

- To design table control on the screen , you need to click on TABLE in control bar and place it on the screen. You can adjust the length and width of table control.
- Name the table control. (Here you need to use same name which you have used for declaration of table control in module pool program).
- From Dictionary object, select table fields and place them in the table control.

Passing data to Table Control:

As already mentioned table controls are tables but are treated like loops. **Usually transfer of data from program to screen is automatic. But in case of table control , transfer of data is not automatic.** You need to explicitly transfer the data to table control. ABAP/4 provides LOOP statement , which is associated with flow logic to transfer the data. Because table control is treated like a loop, data from where it is transferred should be a loop. You can not transfer the data by only select statement , you need to put the data into internal table. ABAP/4 provides the LOOP statement , which is associated with the flow logic and allows you to loop through the table control and internal tables. In between LOOP – ENDLOOP, you can use most of the flow logic keywords like FIELD, VALUES, MODULES,etc

You need to code a LOOP statement in both PBO and PAI event of the screen. **With LOOP statement, you can transfer the data from program to table control and vice-versa.** That is if user updates the value in the table control, then you can update database table with this value. And this can be done in PAI event. **So even if you are not updating database table through the table control, you need to put the LOOP statement in the PAI event also.**

SYNTAX**PBO**

LOOP at <itab> with CONTROL <table control name> CURSOR <scroll variable>.

ENDLOOP.

PAI.

LOOP at <itab>.

ENDLOOP>

Proper usage of table control is as follows.

In Flow Logic

```
PBO  
LOOP AT <ITAB> WITH CONTROL TC1 CURSOR TC1-TOP_LINE.  
MODULE ASSIGN.  
ENDLOOP.  
  
PAI.  
LOOP AT <ITAB>  
ENDLOOP.
```

The transfer of the data from program to table control takes place in steps and these steps are as follows :

- With LOOP AT statement the first row is picked up and placed in the header of the internal table.
- Whatever statements you have in between LOOP-ENDLOOP are executed : In this case you have Module statement. In module statement value of internal table are assigned to table control field.
- The row in internal table is transferred to the first line of the table control as stated in the LOOP AT statement.
- The system encounters the ENDLOOP statement and Control is passed to the next line of the internal table.
- In the same way all the records of the internal table are passed to the table control.

Branching to List Processing:

Switching to List Mode

You can display a list with in a transaction. You can produce a list from module pool program by using the command **LEAVE TO LIST PROCESSING**. This statement switches the system from dialog mode to list mode and from this point onwards until you return to dialog mod , you can use all the normal report statements like write , select or any other event.

Returning back from LIST mode:

You can return back to dialog mode by clicking on the BACK button.

You can have your GUI status and write code for the same. In this you can include the command LEAVE LIST-PROCESSING. When the system reaches this command it leave the list mode and returns to the dialog mode.

HELP and VALUE Request:**About HELP:**

In any transaction, when the user press F1 or ? on field , system provides the help facility for that particular field. In dialog program, when the F1 is pressed, Help provided by R/3 system is sourced from data element documentations. If this documentation is not present for that particular field or if user needs to display additional information for that particular field, then user defined help can be provided through PROCESS ON HELP REQUEST.

In ABAP/4 help can be provided to the user by :

- Data element documentation. The F1 help can be enhanced by adding an additional text for the data element in the ABAP/4 dictionary.

And can be done by following steps:

- Place cursor on the screen field.
- GOTO → DOCUMENTATION --> DATA ELEMENT DOCUMENT.
- You can now extend the existing help.

Using the **PROCESS ON HELP-REQUEST**.

If you don't have this event in a program, then the documentation of the field in the ABAP/4 dictionary is taken into consideration. If this event exists in the program then it is executed.

PROCESS ON HELP-REQUEST

This event is triggered when user presses F1 on a screen field. You need to handle this event in flow logic by specifying the fields and attaching the module to it.

SYNTAX

PROCESS ON HELP-REQUEST.
FIELD <NAME> MODULE <NAME>.

In module pool program.

MODULE HELP.

Write : / 'This is field is from T001 table'.

Write : / 'It is 4 characters'.

ENDMODULE.

When the user press F1 on this particular field then this message will be displayed on the screen.

VALUE REQUEST.

Whenever the user presses F4 on the screen field , list of possible values for that particular fields are displayed. If the standard value-help is inadequate or if you want to display additional fields or with different combination of fields, then developer can program this in PROCESS ON VALUE_REQUEST event in the flow logic and subsequent module in the module pool program. When the user presses F4 , list of possible values are displayed either from matchcode objects or check table or help view or domain.

Each of it is explained briefly:

- **Matchcode Objects:** Are aggregated dictionary objects and detailed procedure to create these objects is explained in the later part of the material.
- **Check Table:** If a check table is assigned to the table field and if the user pressed F4 for that particular field, then all the key fields are displayed.
- **Domain Values:** The values defined in the domain are displayed. These values are set in domain when the domain is created in the dictionary.
- **Help Views:** In cases where the check table is not sufficient, you can create a help view with this check table, which gives additional information like explanatory text for the fields of the check table.
- **User defined Help :** By Providing the list of values through one internal table .

Changing the Screen during the runtime

The attributes are assigned to the screen field when the screen is designed in full screen editor. Such kind of assignment is static in the sense that these attributes are fixed, but many times the need to change the attributes of the screen arises and has to be done during runtime.

Need to Change Screen

These can be requirement in the transaction, that certain field on the screen.

- Appear only in certain conditions.
- Are in Change/Display mode according to user inputs.
- Becomes mandatory subject to specific inputs.
- Changes its format depending upon certain conditions.

Modify the Screen

At the runtime, attributes for each screen field is stored in a system defined internal table with header line called as **SCREEN TABLE**. It contains name of field and its attributes. This table can be modified during the runtime i.e., through module pool program. Screen table has following fields.

| FIELD NAME | LENGTH | DESCRIPTION |
|--------------------|---------------|------------------------------------|
| NAME | 30 | Name of screen field |
| GROUP1 | 3 | Field belongs to field group1 |
| GROUP2 | 3 | Fld-grp-2 |
| GROUP3 | 3 | Fld-grp-3 |
| GROUP4 | 3 | Fld-grp-4 |
| ACTIVATE input | 1 | Field is visible & ready for input |
| REQUIRED | 1 | Field input is mandatory |
| INPUT | 1 | Field ready for input |
| OUTPUT | 1 | Field for display only |
| INTENSIFIED | 1 | Field is highlighted |
| INVISIBLE | 1 | Field is suppressed |
| LENGTH | 1 | Field output length is reduced |
| DISPLAY 3D | 1 | Field is displayed with 3D frame |
| VALUE_HELP help | 1 | Field is displayed with value help |

Ex: SCREEN-ACTIVE = 0 has the same effect as the following statements.

SCREEN-INPUT = 0.
 SCREEN-OUTPUT = 0.
 SCREEN-INVISIBLE = 1.

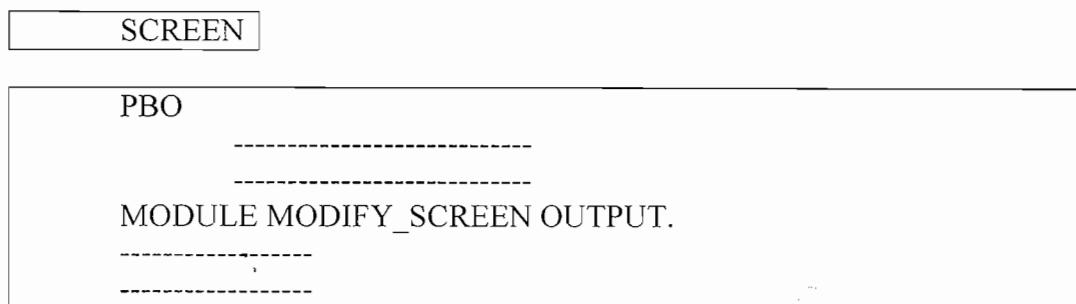
- The fields SCREEN-NAME and SCREEN-GROUP1 through SCREEN-GROUP4 tell you which field and /or field group has the attributes.
- You can assign up to 4 groups to a field.
- You need to program screen modifications in module which is processed during the event PBO.

‘SCREEN’ is an internal table and , in order to change the field values, LOOP statement has to be used so that the headerline can be populated with the new values, changing the earlier values the SCREEN table had for that screen. Finally the changed record in the headerline is NOT APPENDED , but is MODIFIED to

Module Pool Programming

We Never Compromise in Quality, Would You?

the SCREEN table . That is , we first use ‘LOOP AT SCREEN’ then assign the values and finally PRIOR TO ENDLOOP give MODIFY SCREEN.



ABAP/4
PROGRAM

MODULE MODIFY_SCREEN.

LOOP AT SCREEN.

IF SCREEN-NAME = T001-BUKRS.

SCREEN-INPUT = 1.

ENDIF.

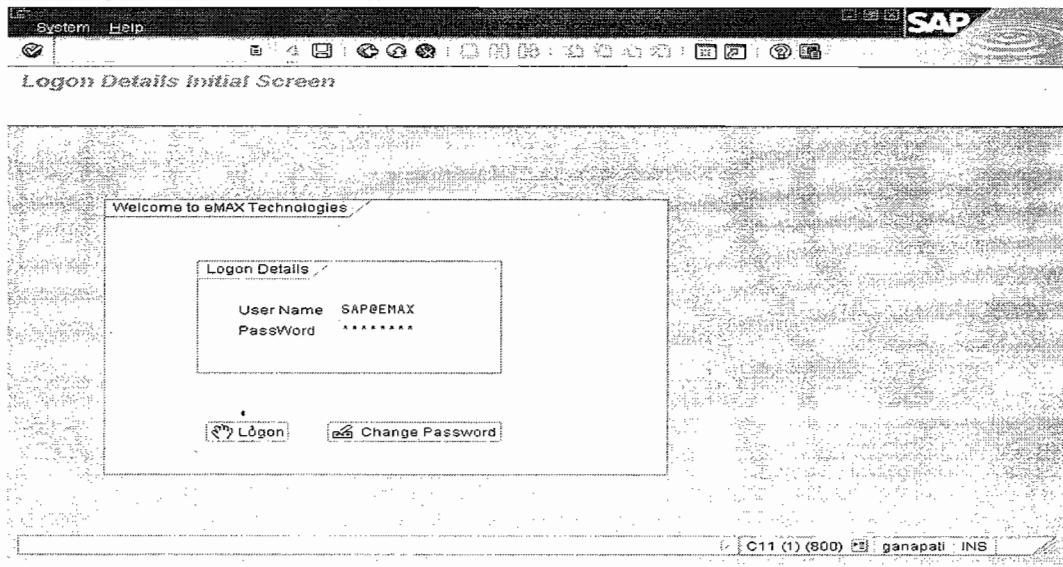
MODIFY SCREEN.

ENDLOOP.

Overview of the Application,

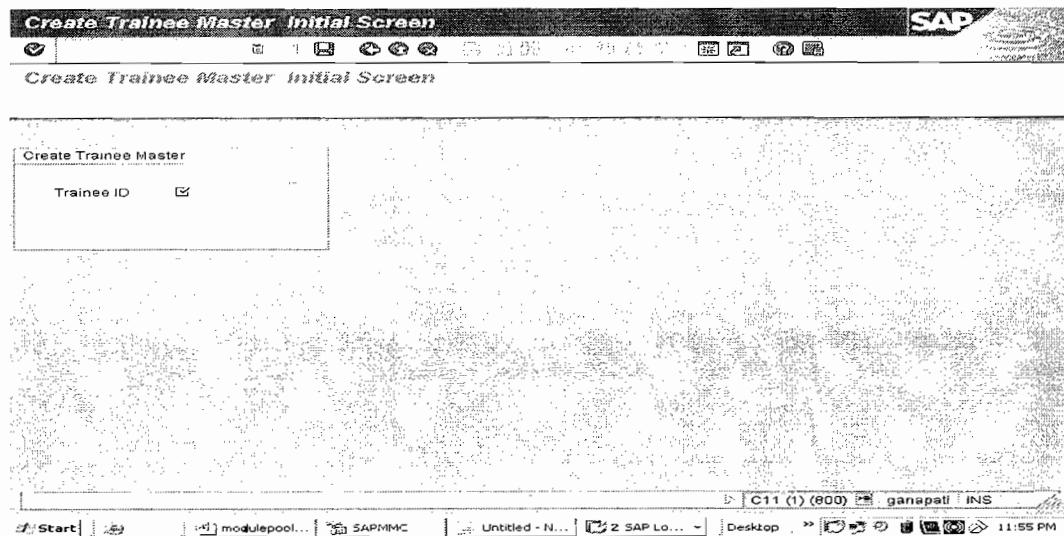
Create Trainee Master:

Initial Screen :

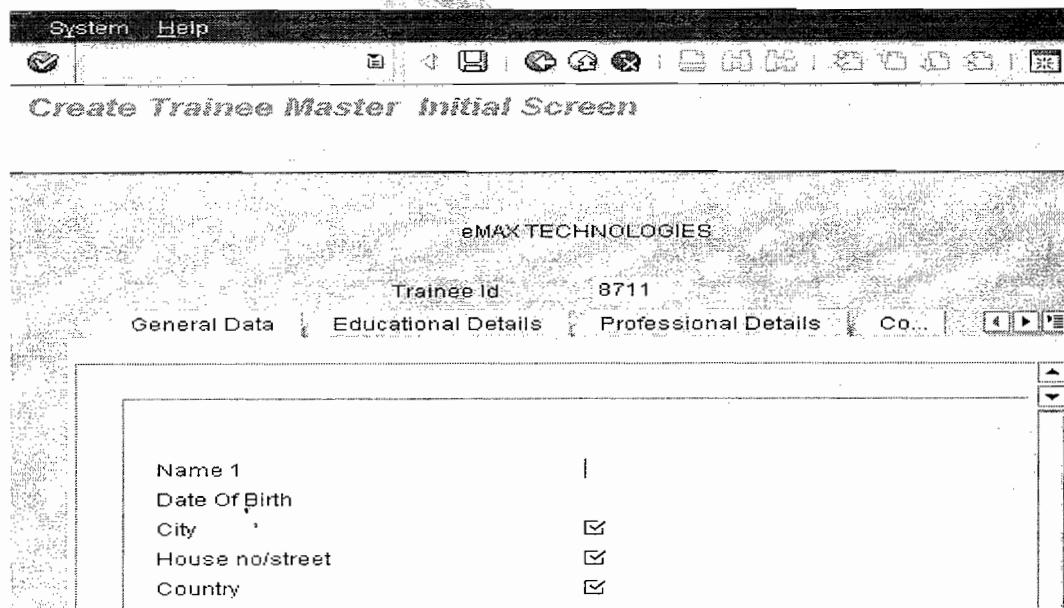


We Can Logon and also can change the Password.

After Logon :



Provide the Trainee ID and Continue.



Which accepts various details of the Trainee through the respective TABS
and also

Updates the relevant databases tables When SAVEed.

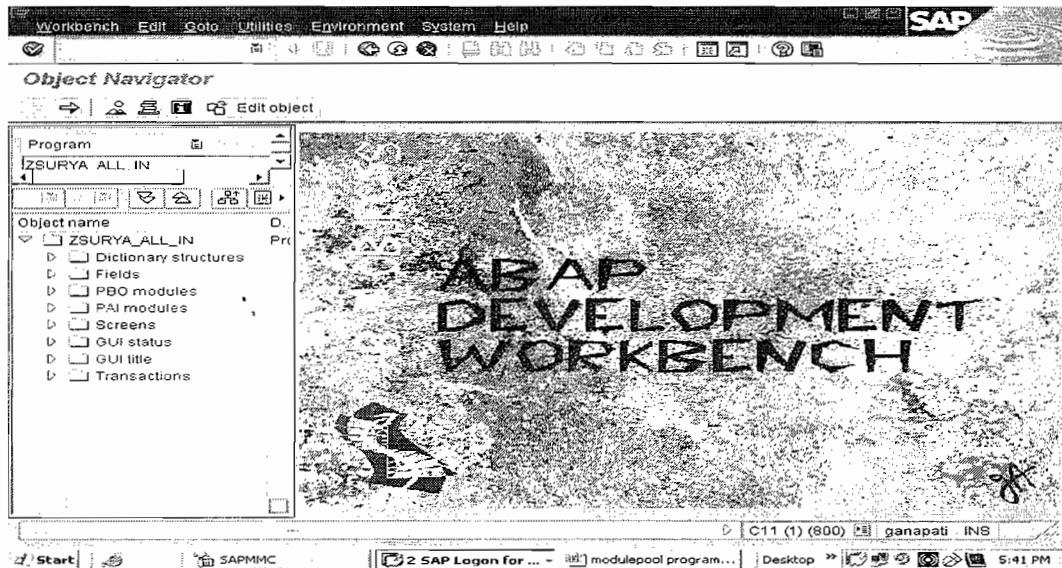
Design Steps For the Application:

Step 1 : Create the Module Program.

Create and Design Screen Nos:

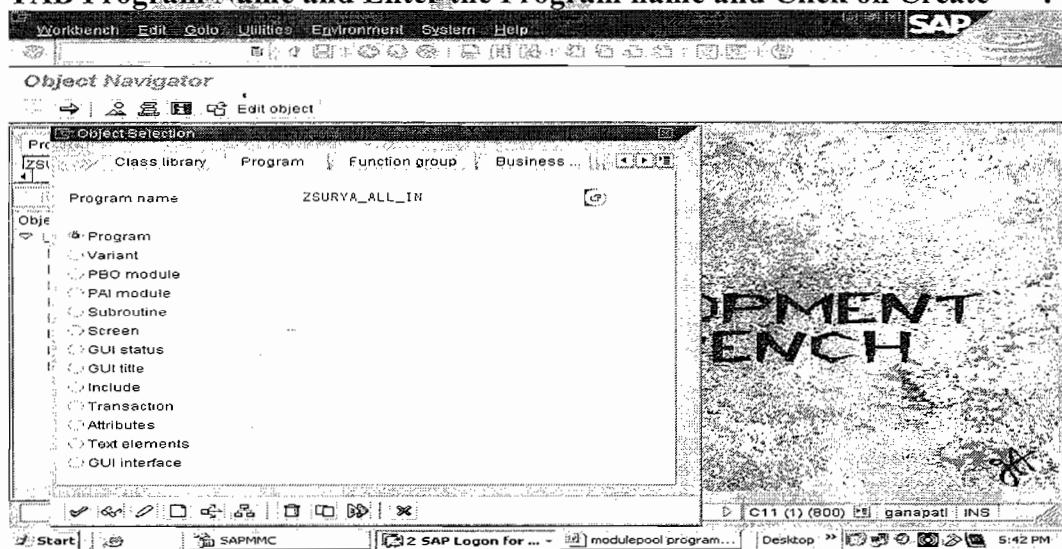
Write the Flow Logic for each Screen.

Create the Transaction Code.



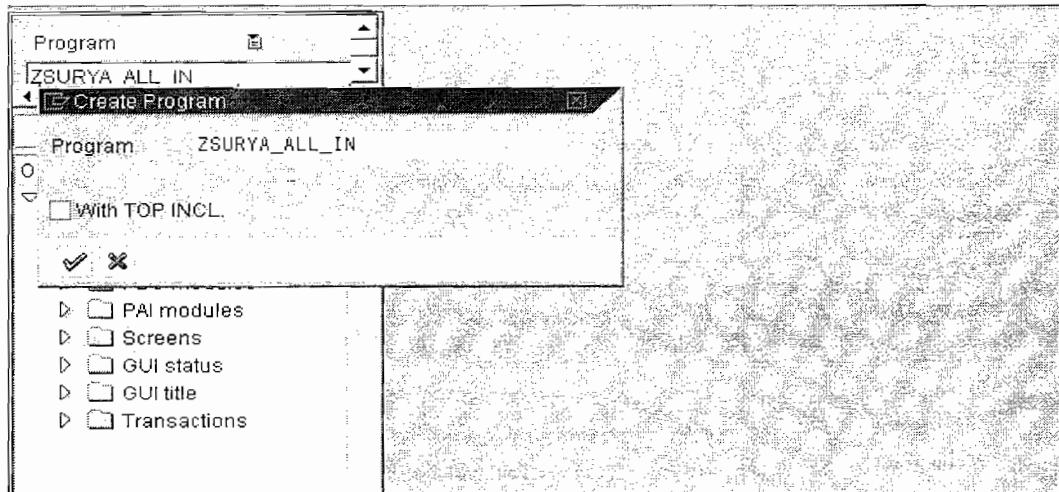
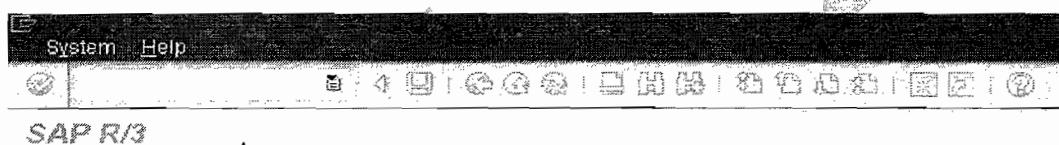
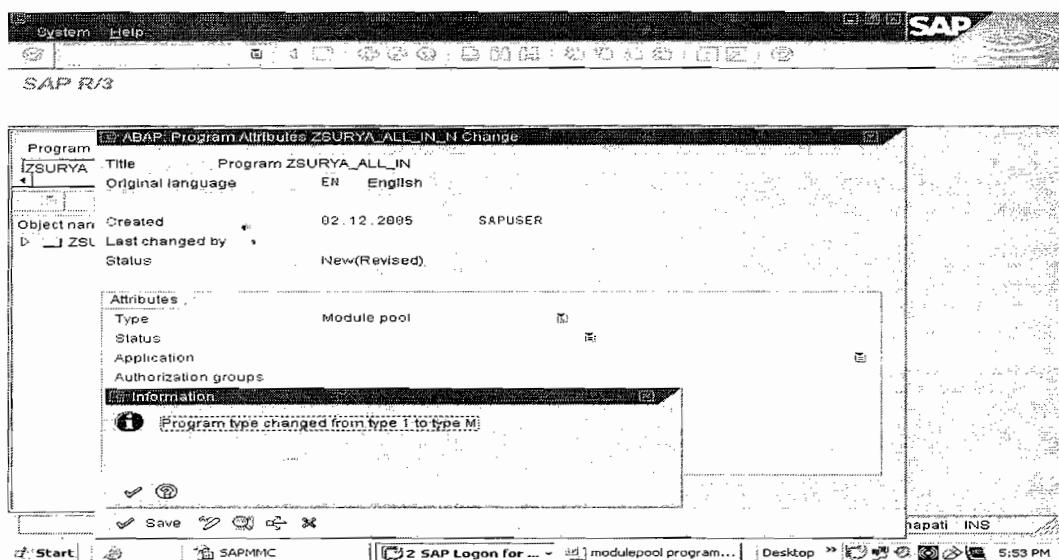
Steps to Create Module Program :

Execute SE80(Object Navigator) and Click On Edit object and Select the TAB Program Name and Enter the Program name and Click on Create .



Module Pool Programming

We Never Compromise in Quality, Would You?



Enter.

Now Module Program is created.

Step 2 : Creating , Design the Screen and their Flow Logic.

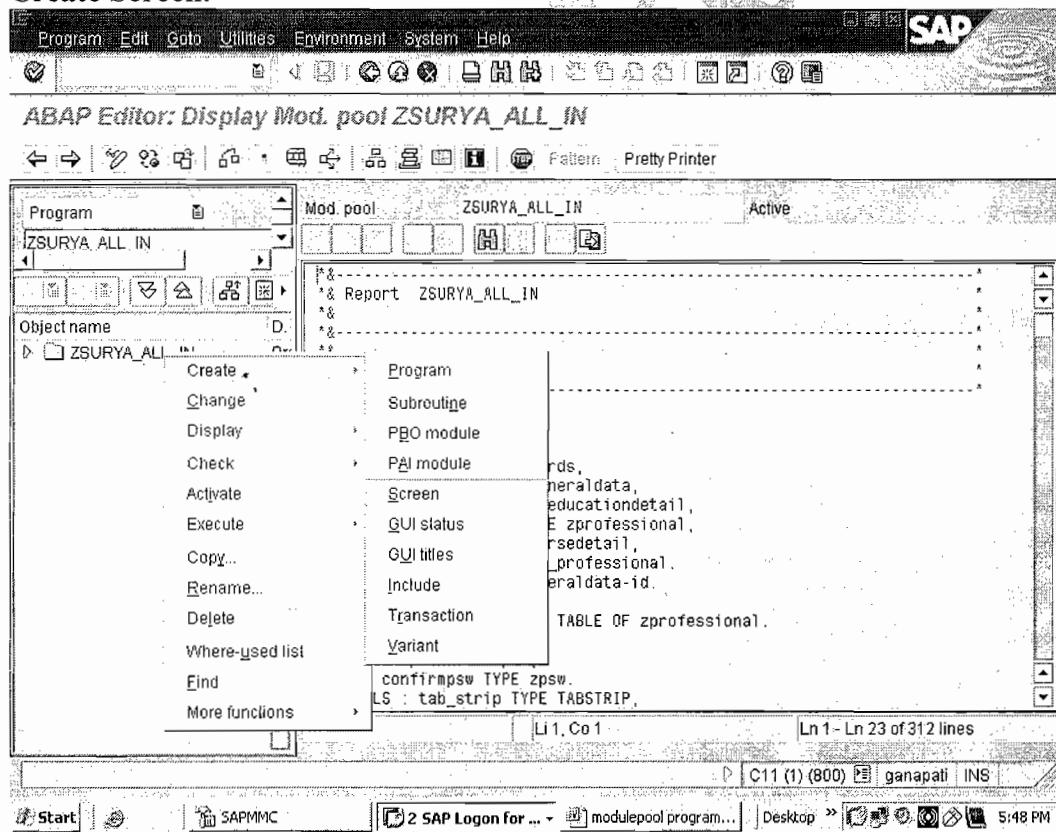
Check the below list of screens to be created and the type of screens and purpose of the screens.

Module Pool Programming

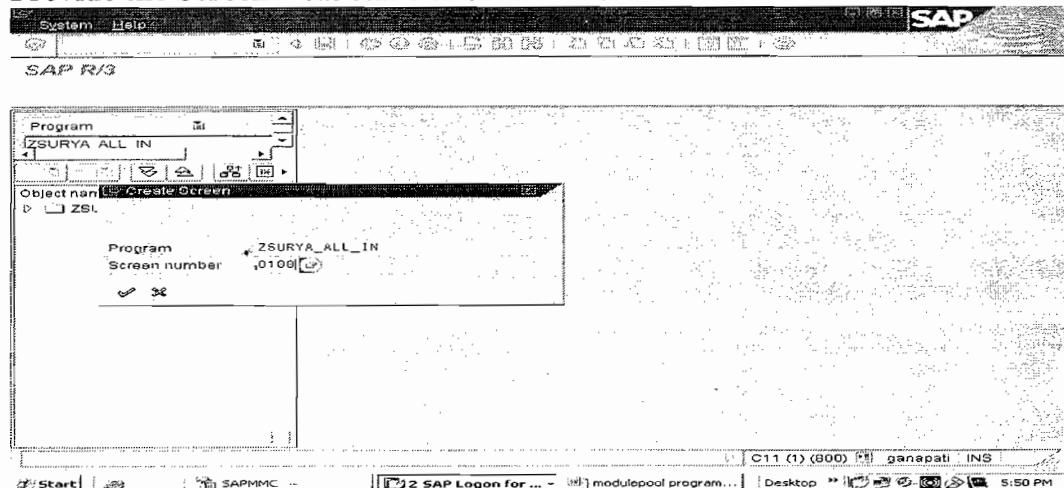
We Never Compromise in Quality, Would You?

| Screen No | Purpose | Type |
|-----------|-------------------------------|--|
| 0100 | Logon Details | <input checked="" type="checkbox"/> Normal |
| 0200 | Change Password | <input checked="" type="checkbox"/> Normal |
| 0300 | Create Trainee Initial Screen | <input checked="" type="checkbox"/> Normal |
| 0400 | Create Trainee Main Screen | <input checked="" type="checkbox"/> Normal |
| 0500 | Trainee General Details | <input checked="" type="radio"/> Subscreen |
| 0600 | Trainee Education Details | <input checked="" type="radio"/> Subscreen |
| 0700 | Trainee Professional Details | <input checked="" type="radio"/> Subscreen |
| 0800 | Trainee Course Details | <input checked="" type="radio"/> Subscreen |

To Create the Screen , Select the Module Pool Program and Right Click -> Create Screen.



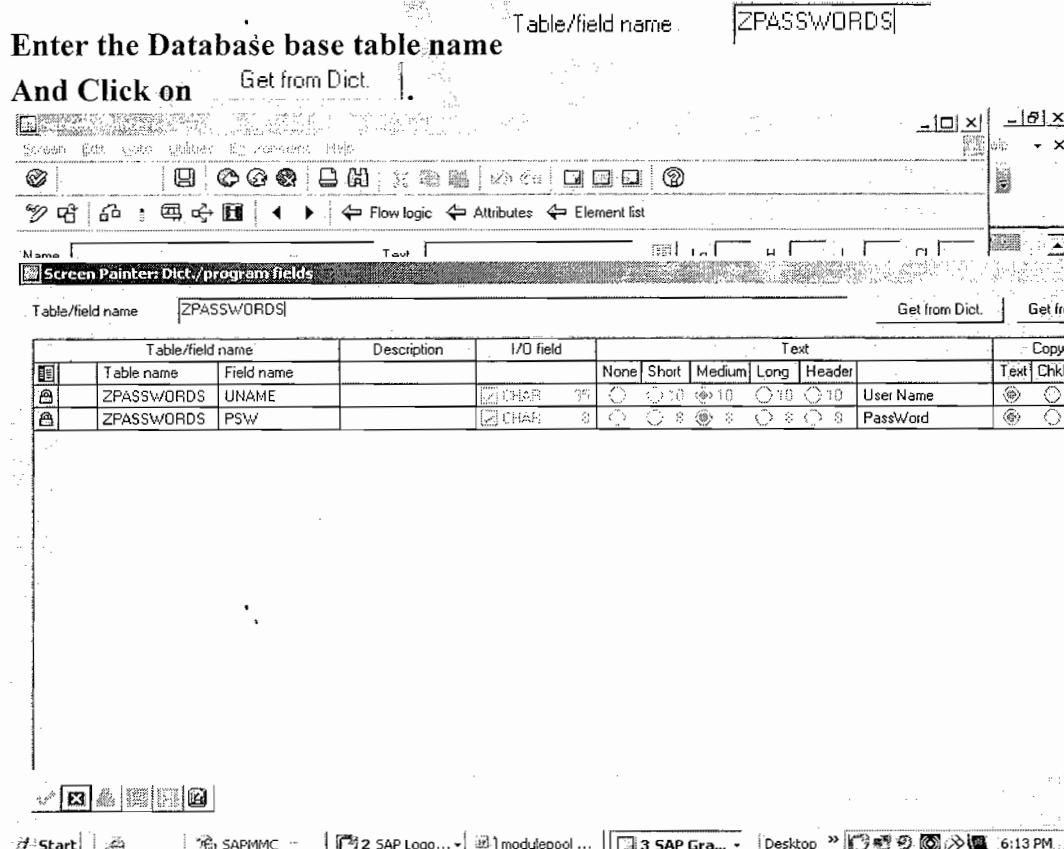
Provide the Screen No.Details of Scene 0100



Enter the Screen Attributes i.e Type of the Screen , Next Screen Details, etc.

Click On Layout to Design the Screen Layout.

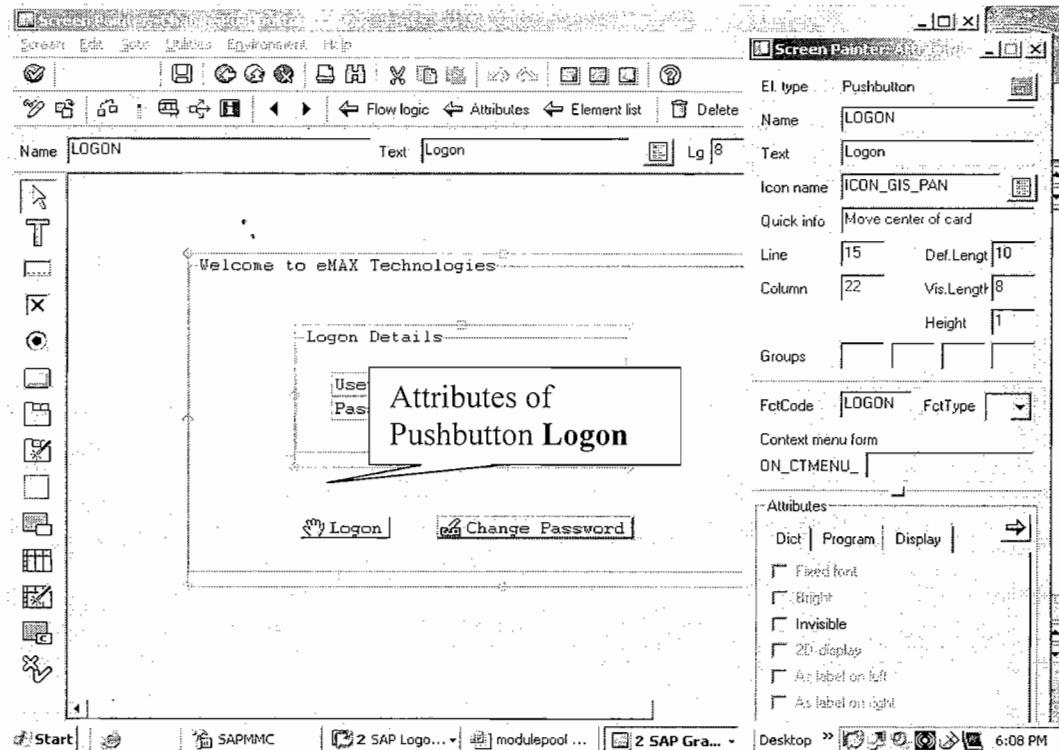
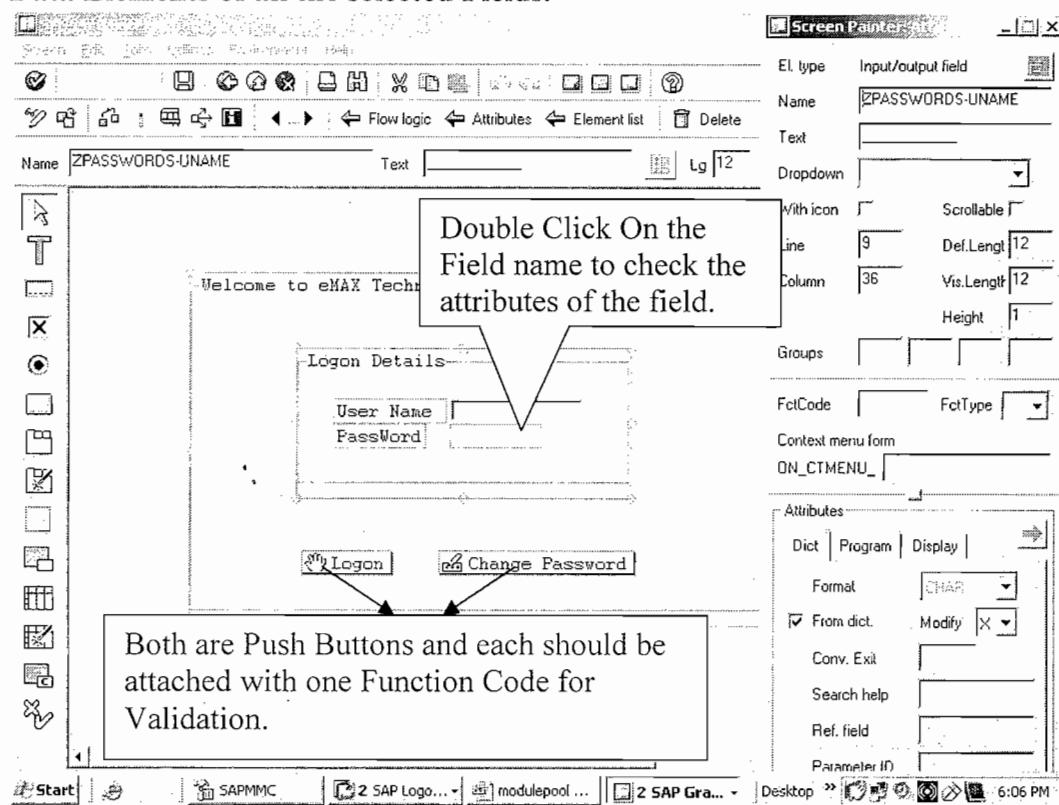
Click (Dictionary/Program Fields F6 Window).



Module Pool Programming

We Never Compromise in Quality, Would You?

Select the required fields and Enter.
So that the Fields and Documentation is copied from the Corresponding
Data Elements of all the selected Fields.



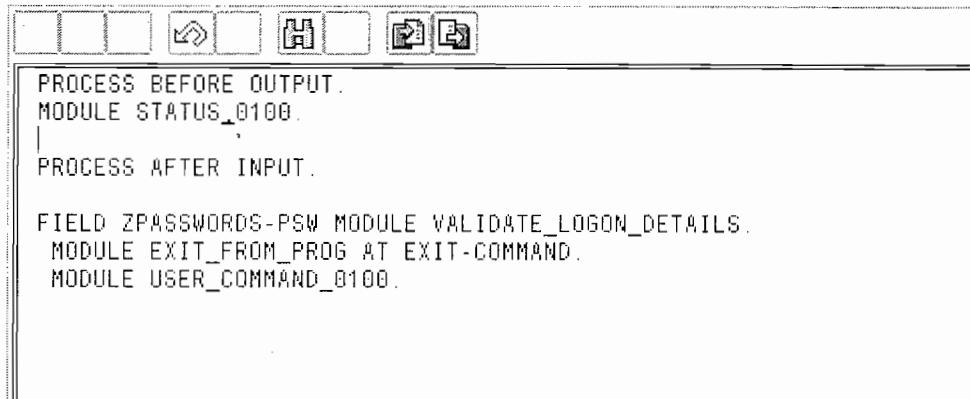
Module Pool Programming

We Never Compromise in Quality, Would You?

Once the Design of the layout is Finished, Click on **Flow logic** to write the Flow logic related to Screen 100.

Screen number 100 Active

Attributes Element list Flow logic

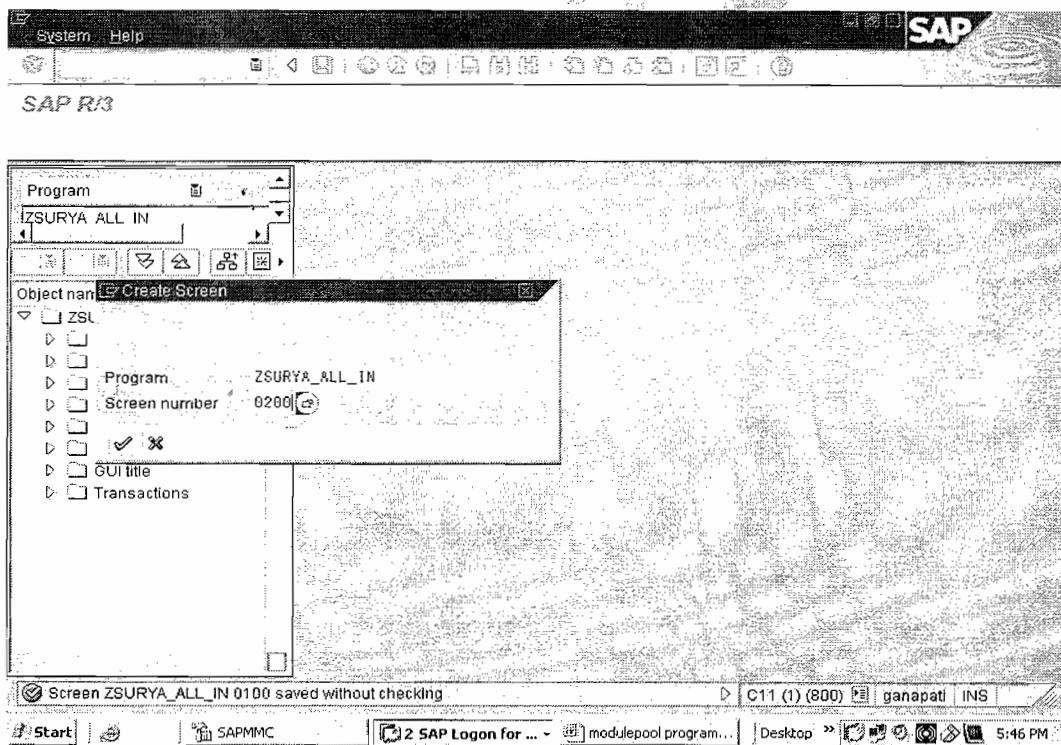


PROCESS BEFORE OUTPUT.
MODULE STATUS_0100.

PROCESS AFTER INPUT.

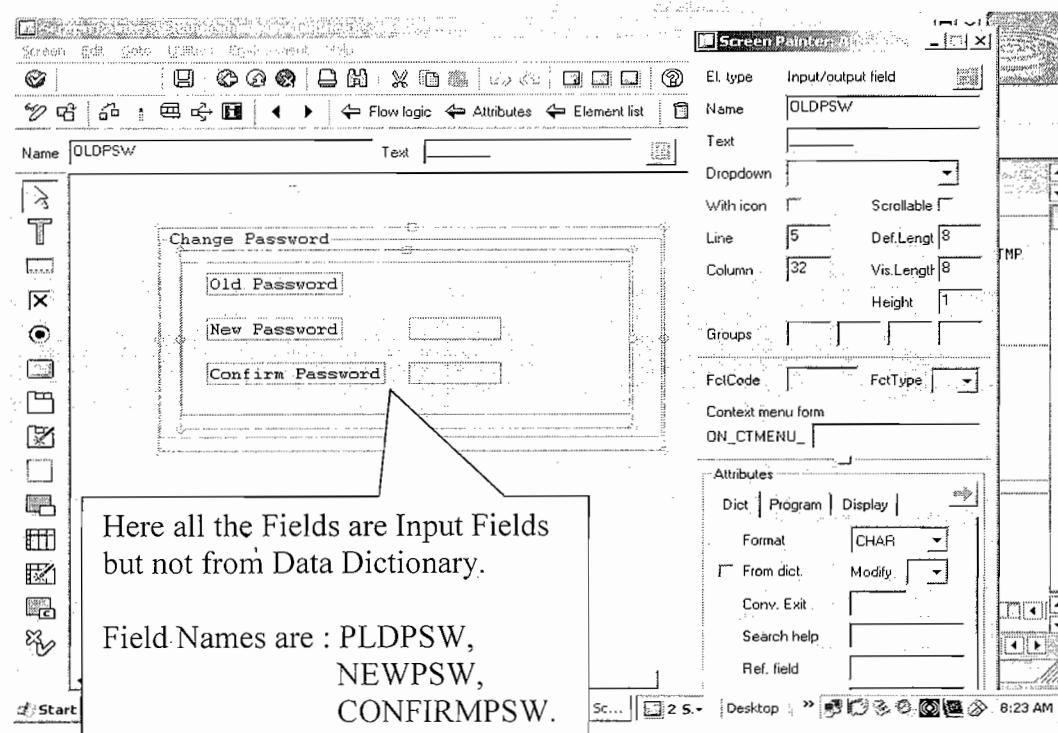
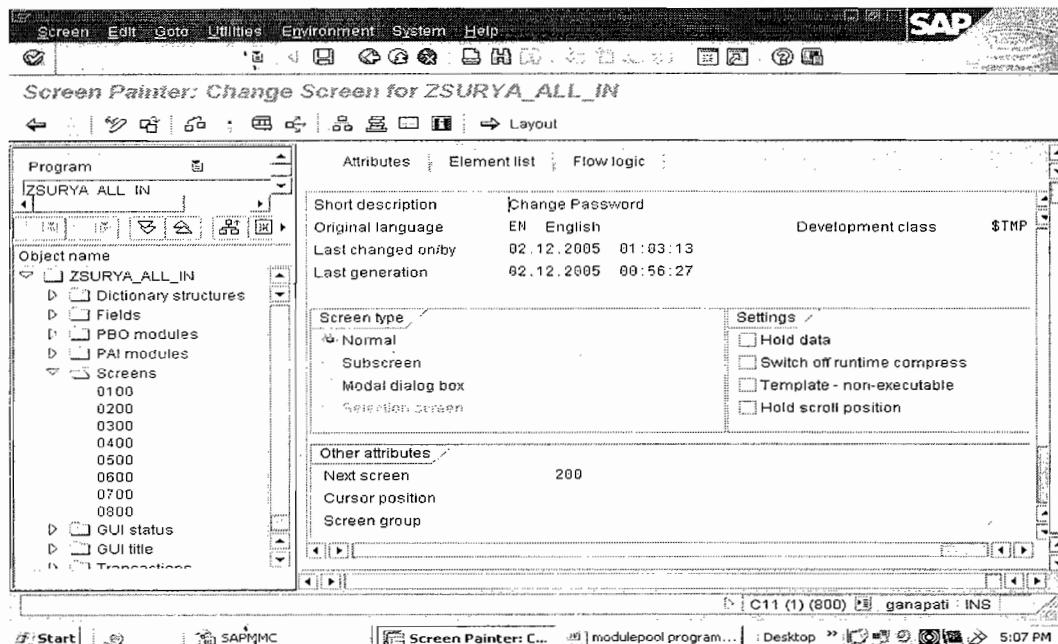
FIELD ZPASSWORDS-PSW MODULE VALIDATE_LOGON_DETAILS.
MODULE EXIT_FROM_PROG AT EXIT-COMMAND.
MODULE USER_COMMAND_0100.

Details of Screen 200:

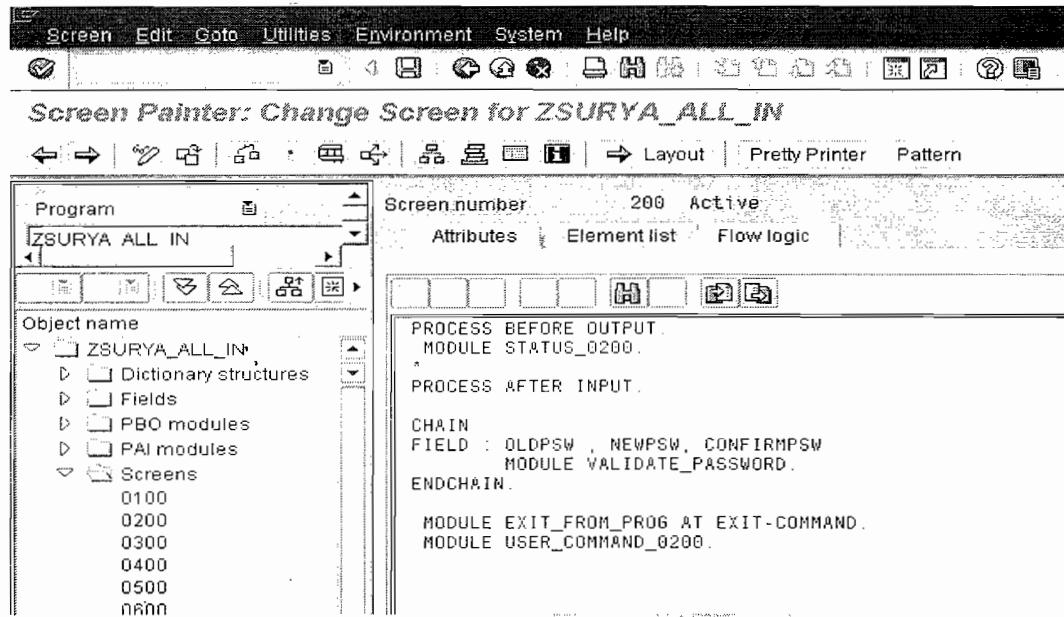


Module Pool Programming

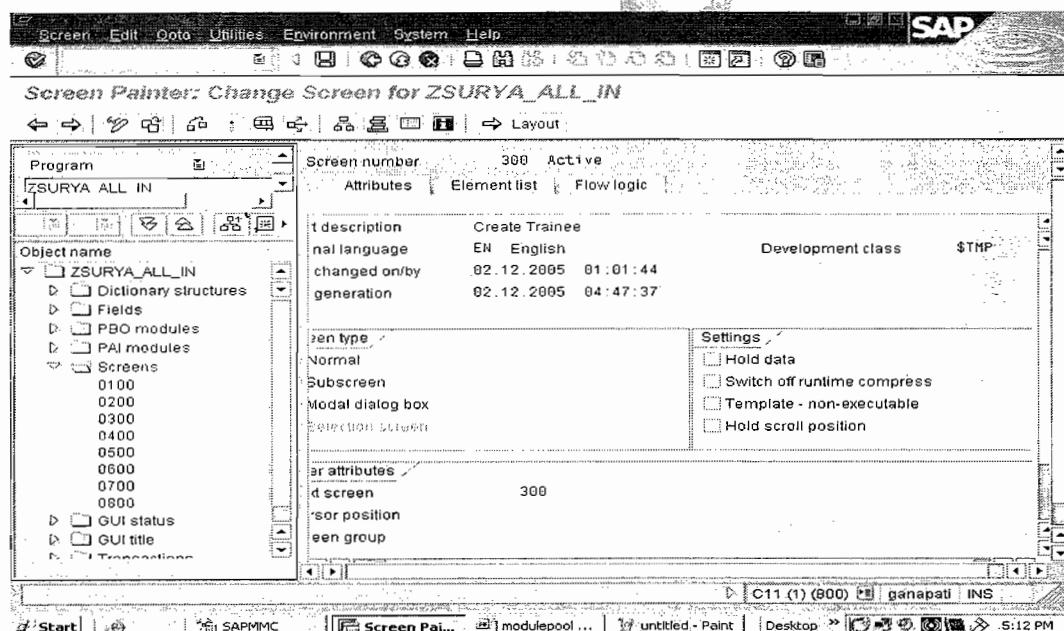
We Never Compromise in Quality, Would You?



Flow Logic :



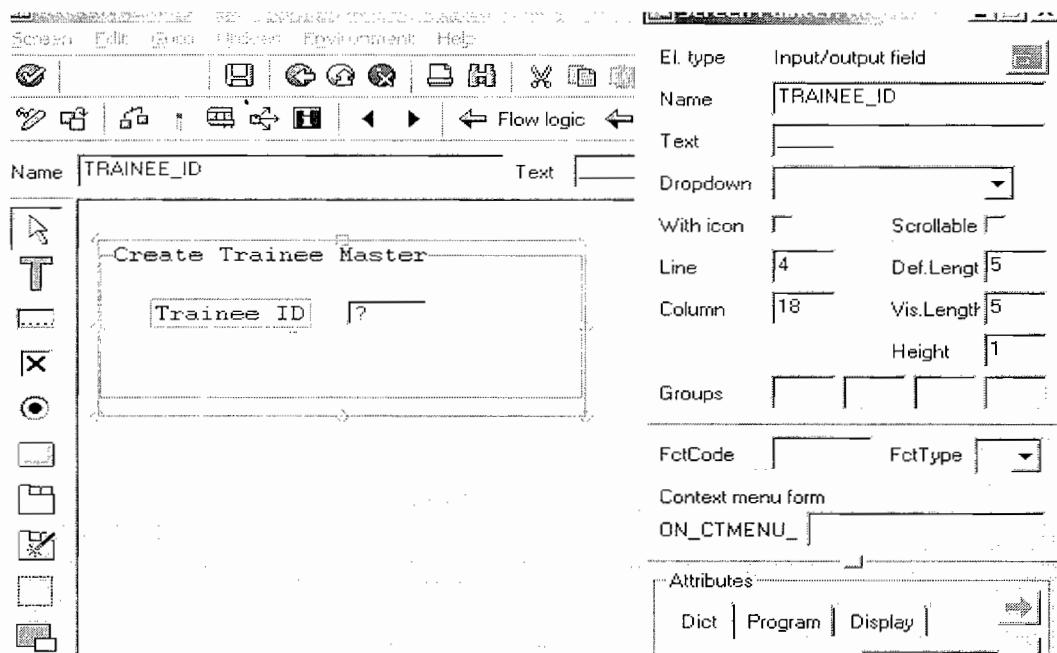
Screen 300 Details :



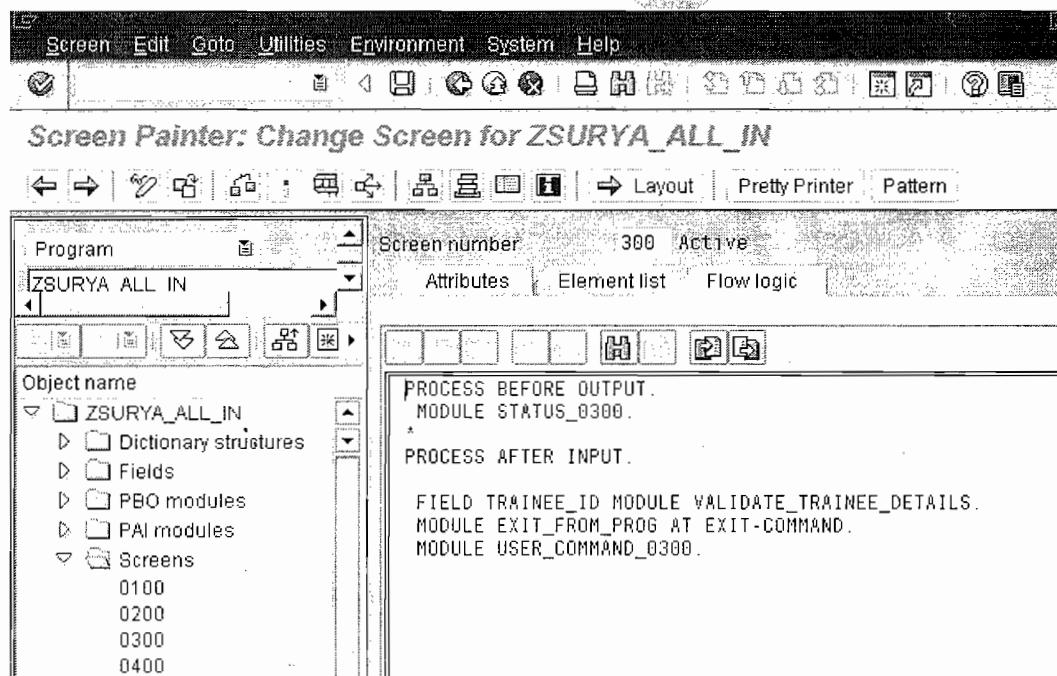
Layout :

Module Pool Programming

We Never Compromise in Quality, Would You?



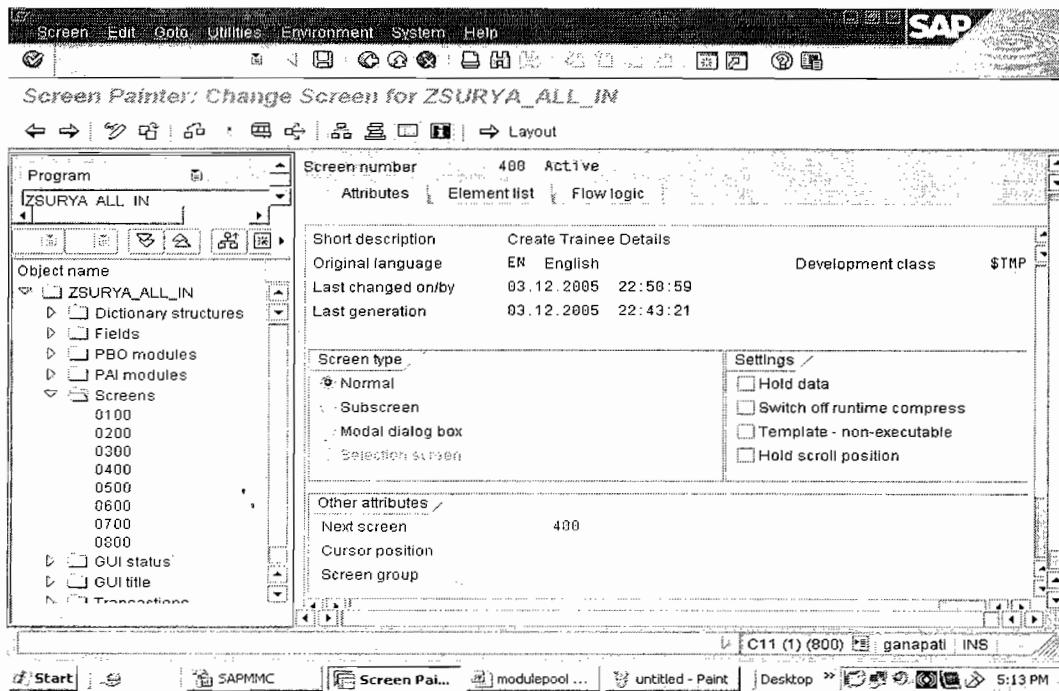
Flow Logic :



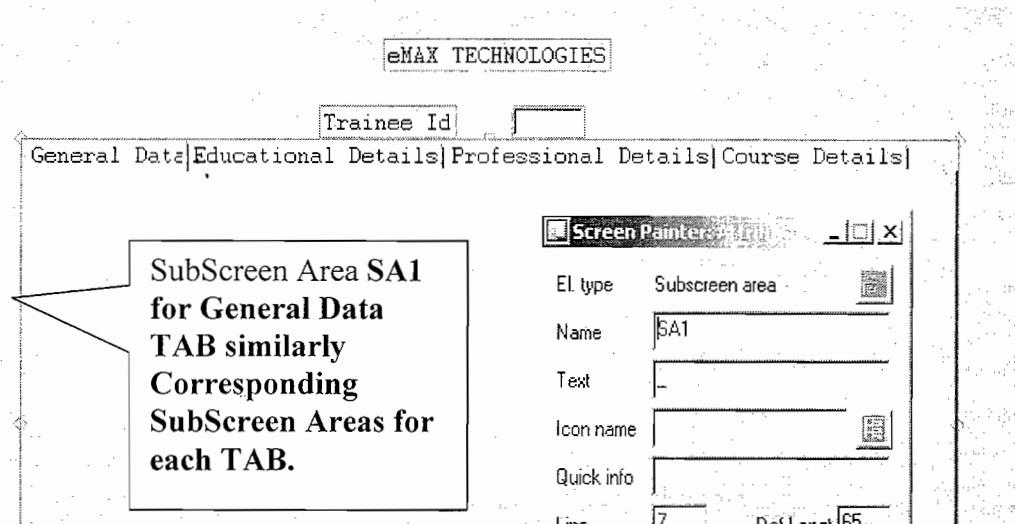
Screen 400 Details :

Module Pool Programming

We Never Compromise in Quality, Would You?



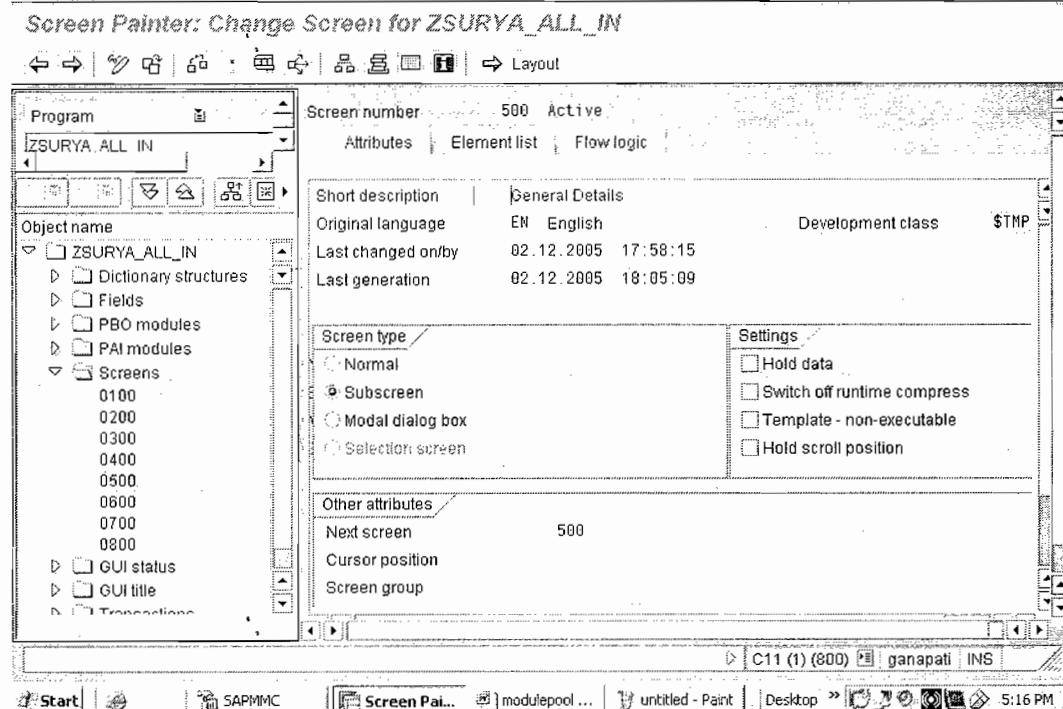
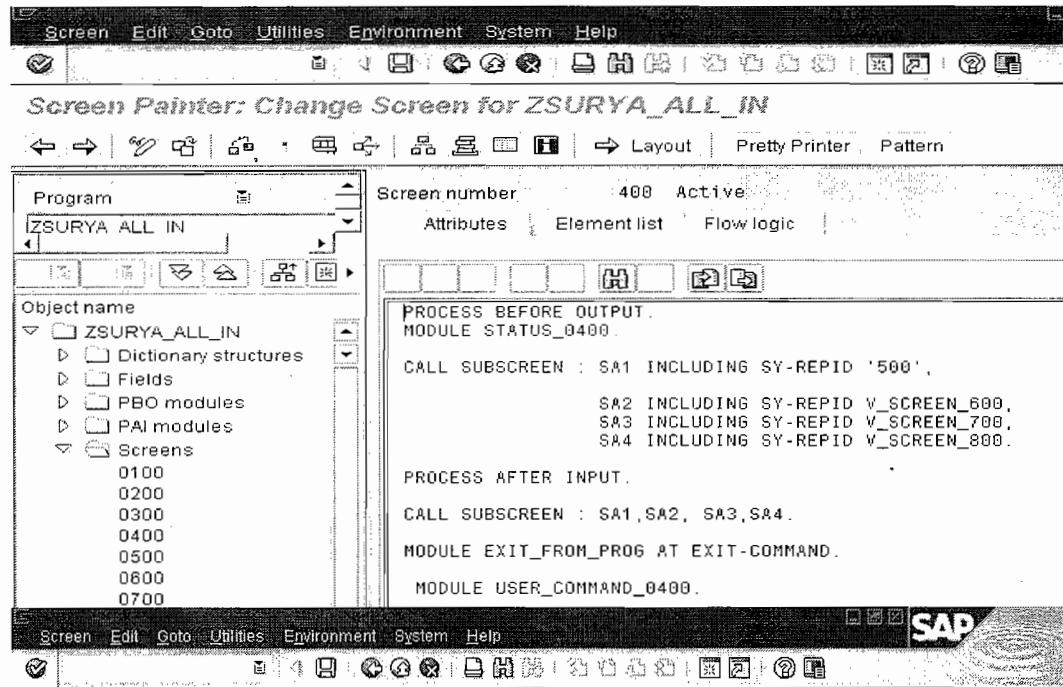
Layout Details :



Flow Logic :

Module Pool Programming

We Never Compromise in Quality, Would You?

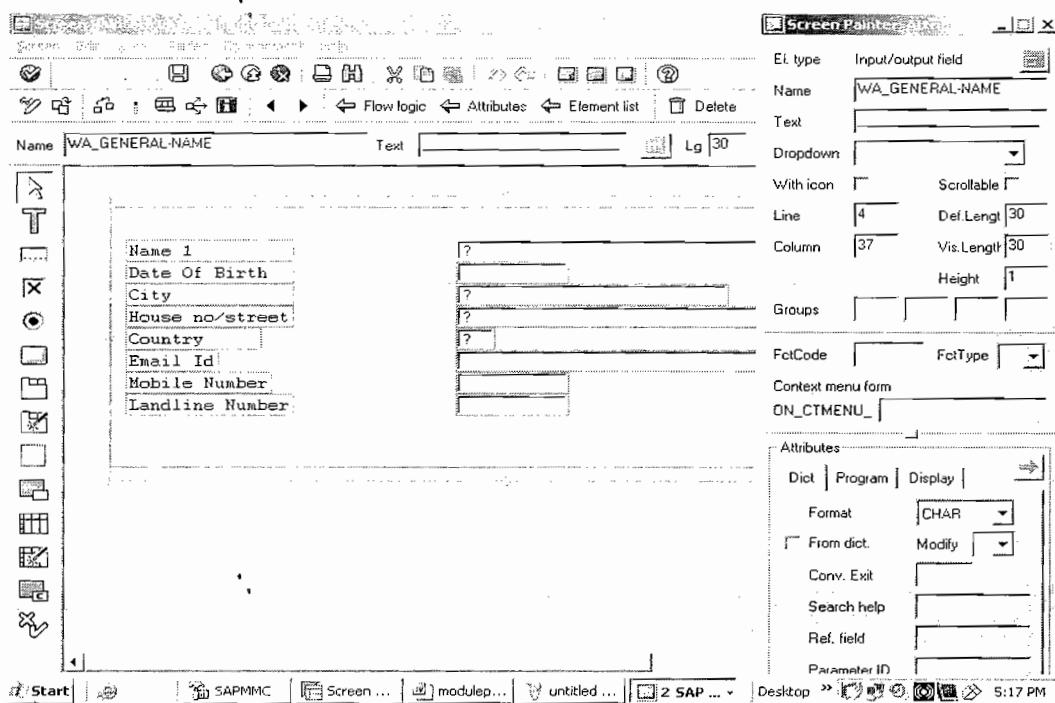


Screen 0500 Details :

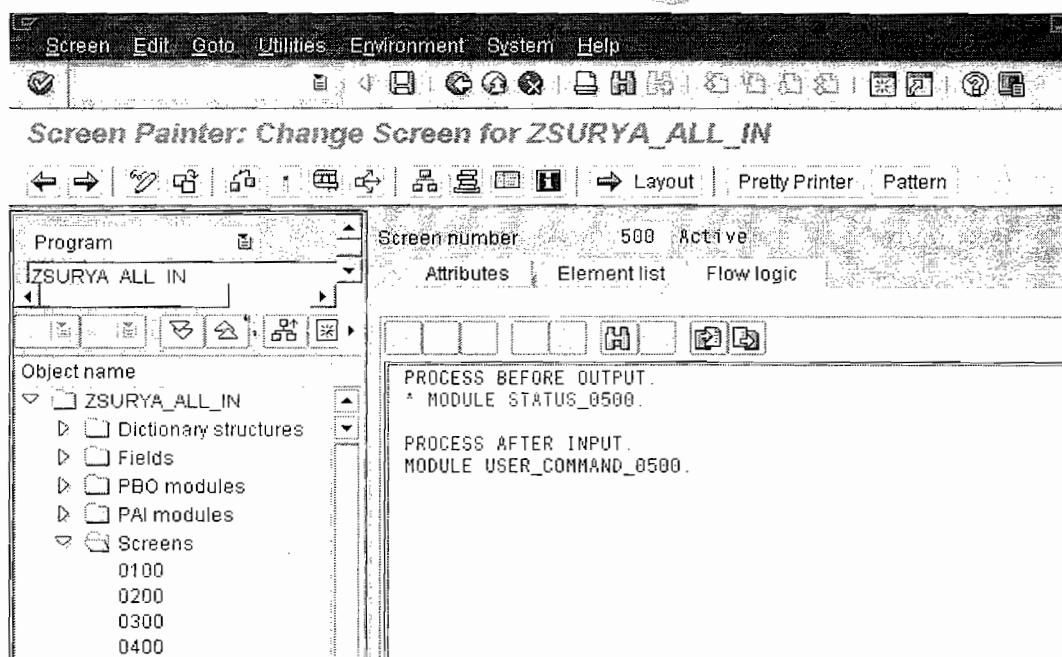
Layout for Screen 500 :

Module Pool Programming

We Never Compromise in Quality, Would You?



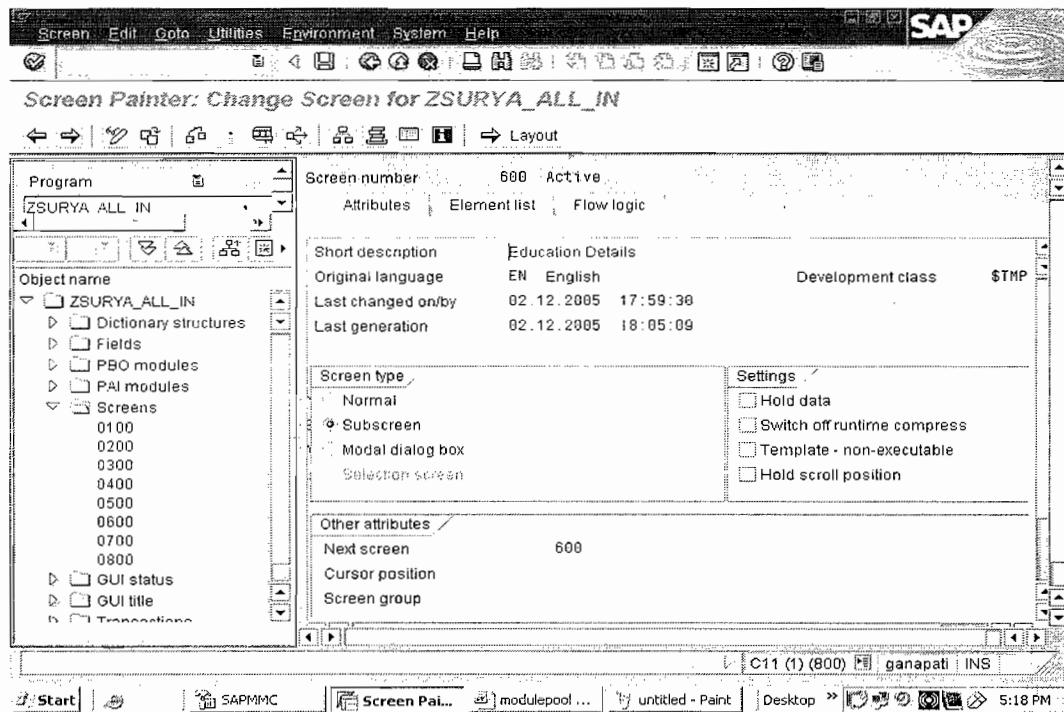
Flow Logic for Screen 500 :



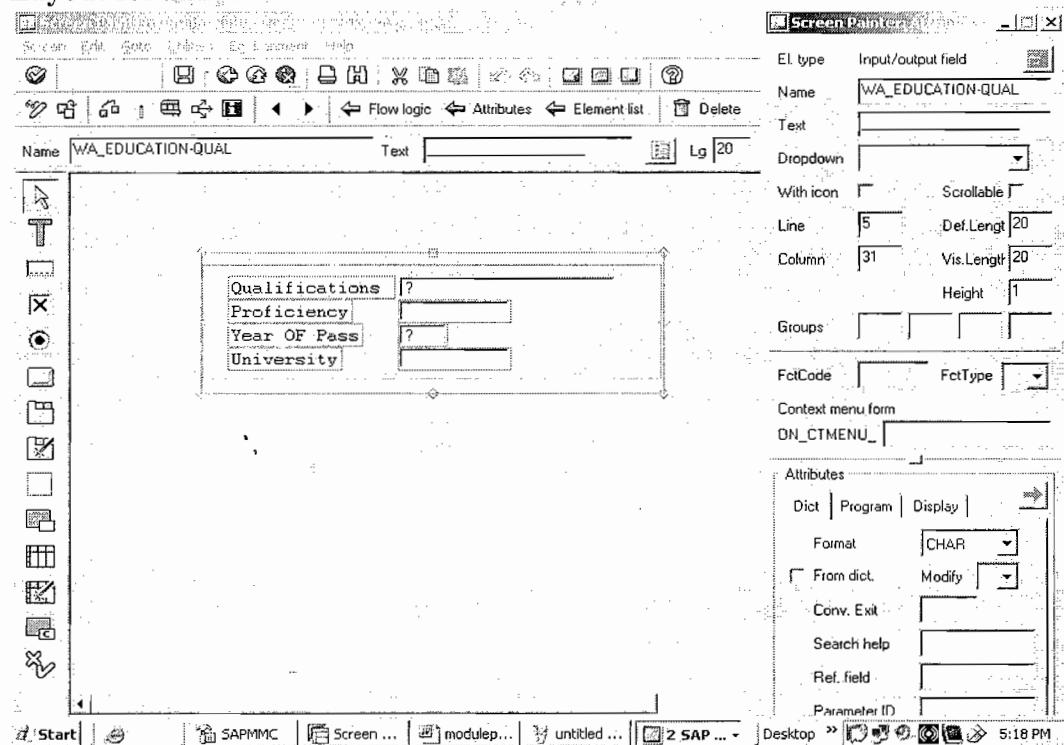
Screen 600 Details :

Module Pool Programming

We Never Compromise in Quality, Would You?



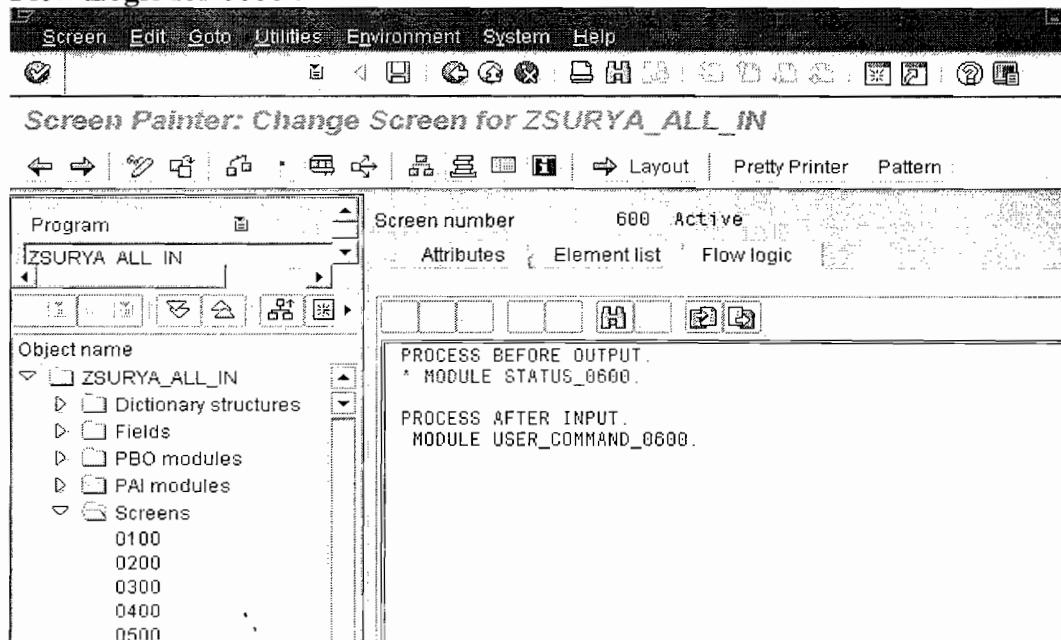
Layout for Screen 0600 :



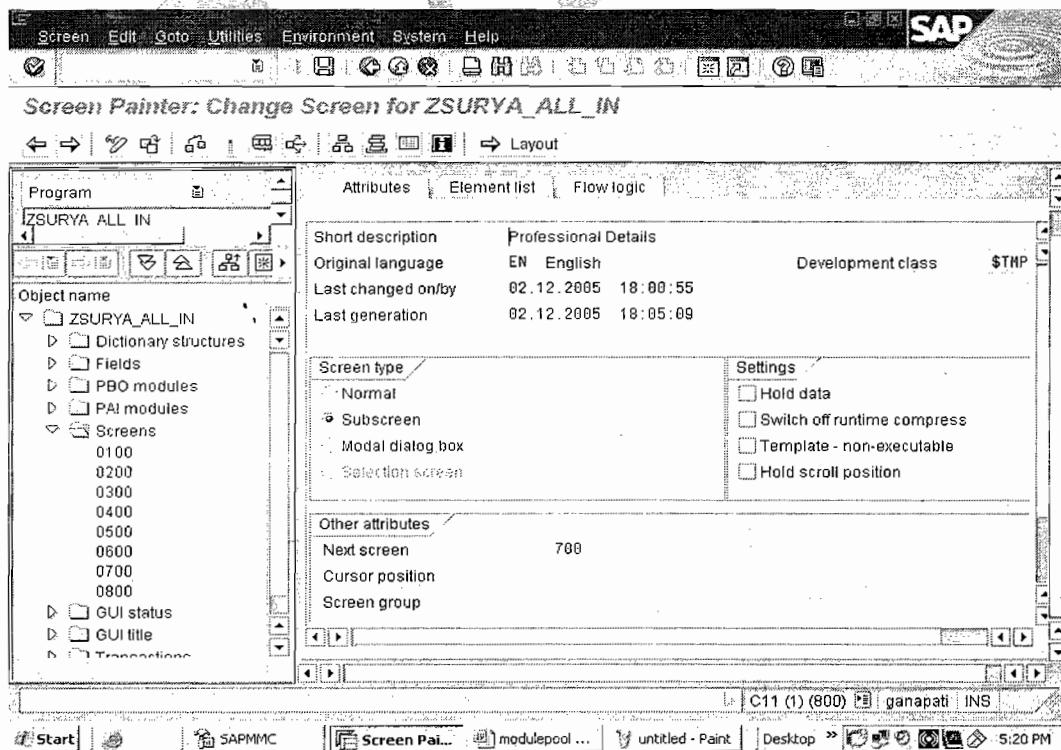
Module Pool Programming

We Never Compromise in Quality, Would You?

Flow Logic for 0600 :



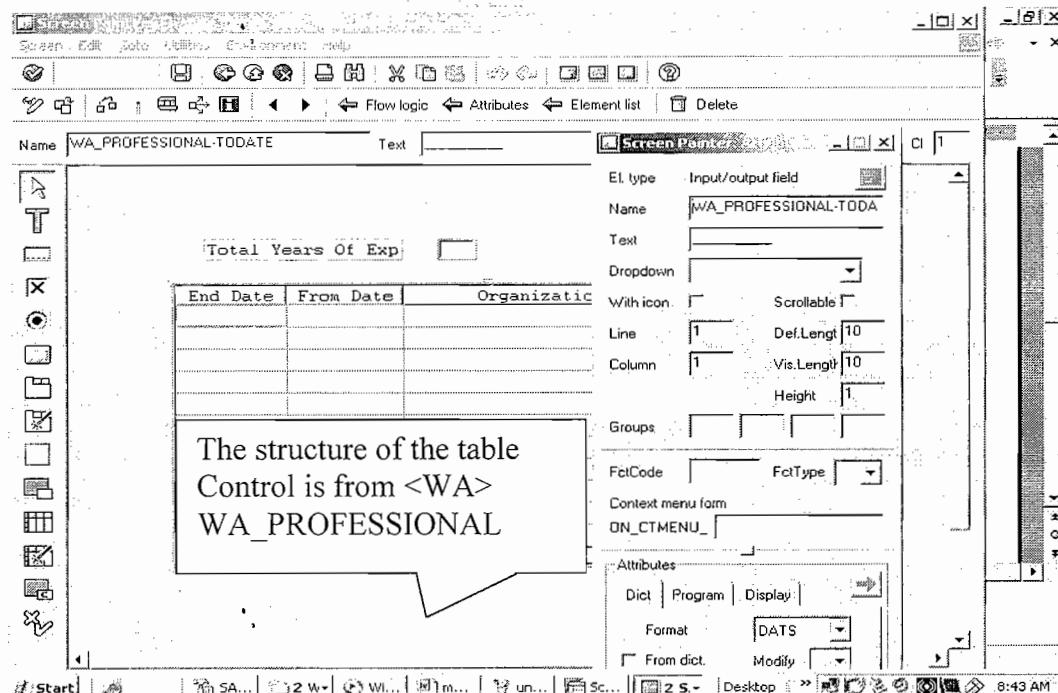
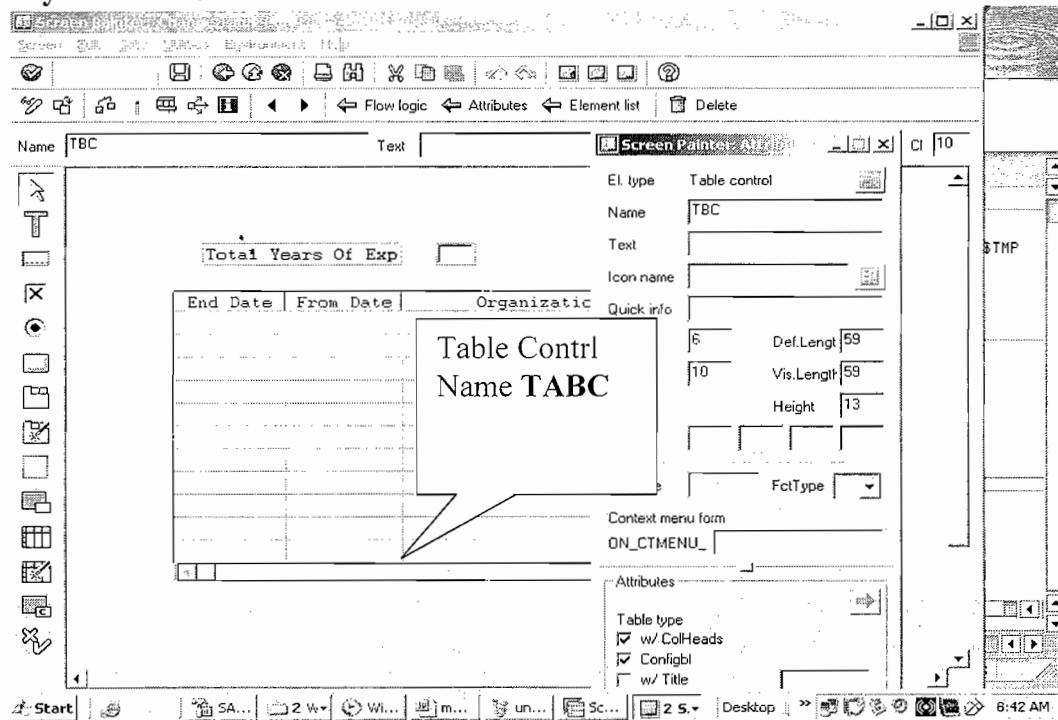
Screen 700 Details :



Module Pool Programming

We Never Compromise in Quality, Would You?

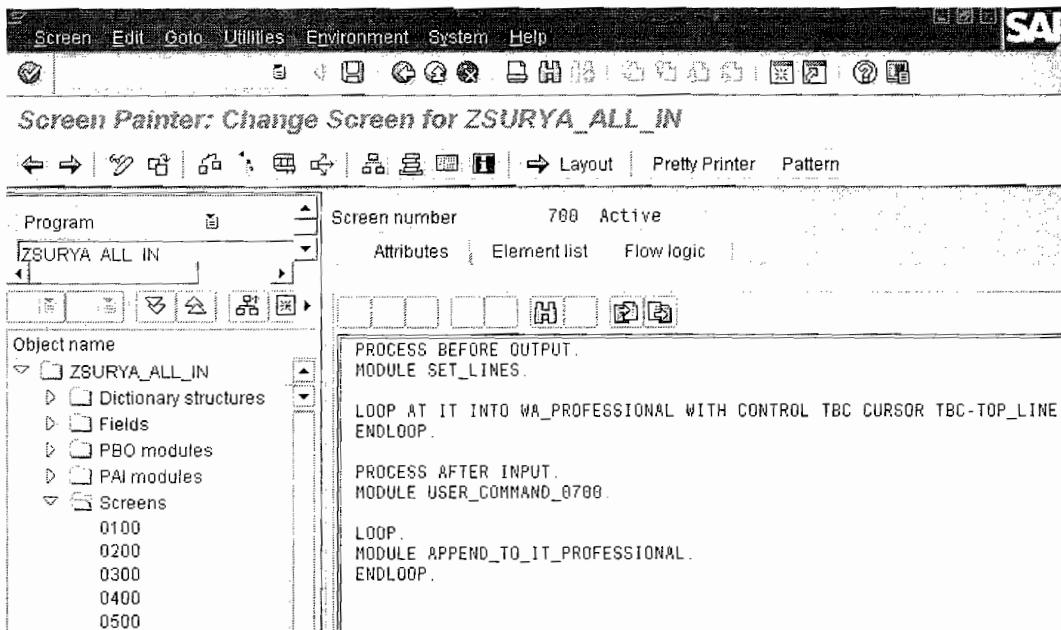
Layout Details:



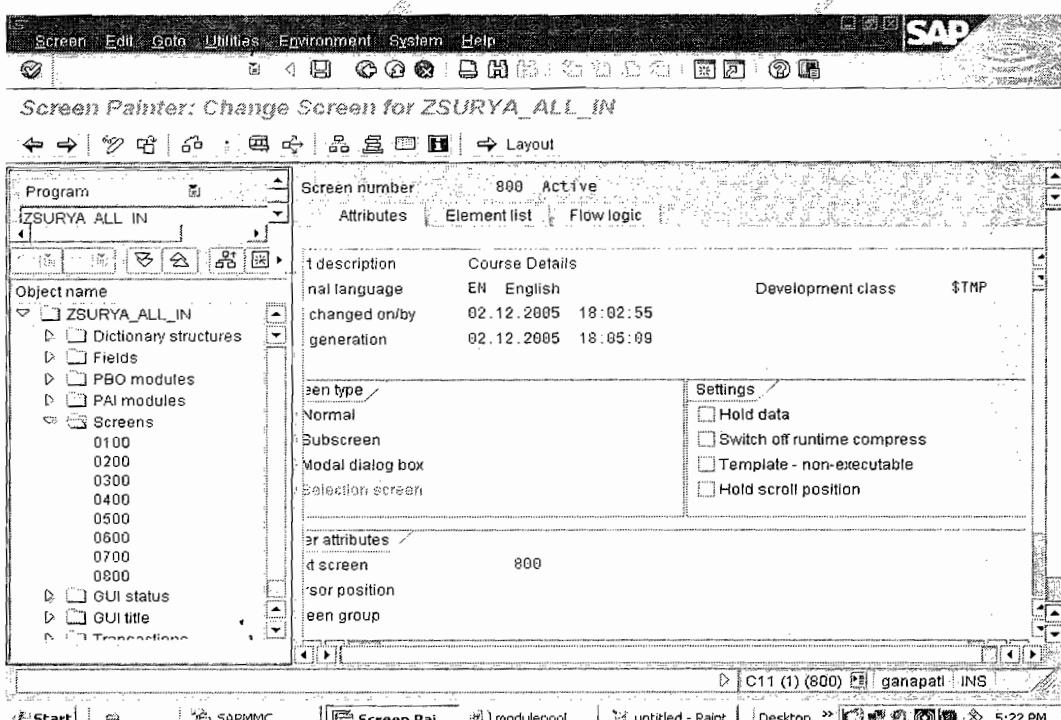
Flow Logic:

Module Pool Programming

We Never Compromise in Quality, Would You?



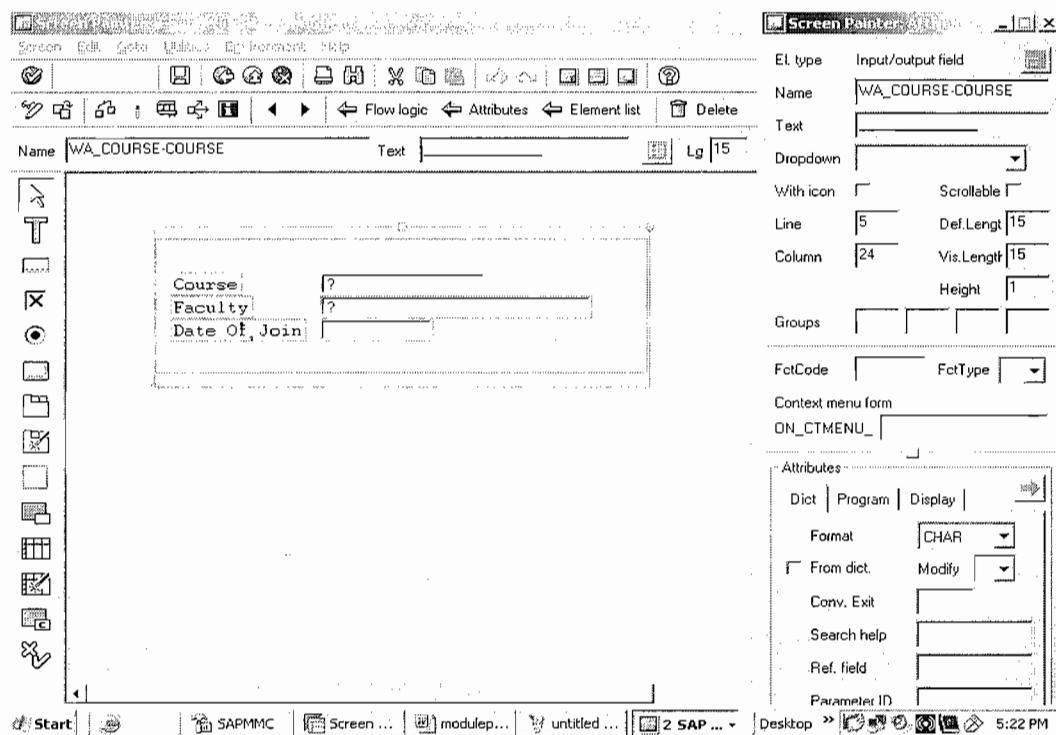
Screen 800 Details :



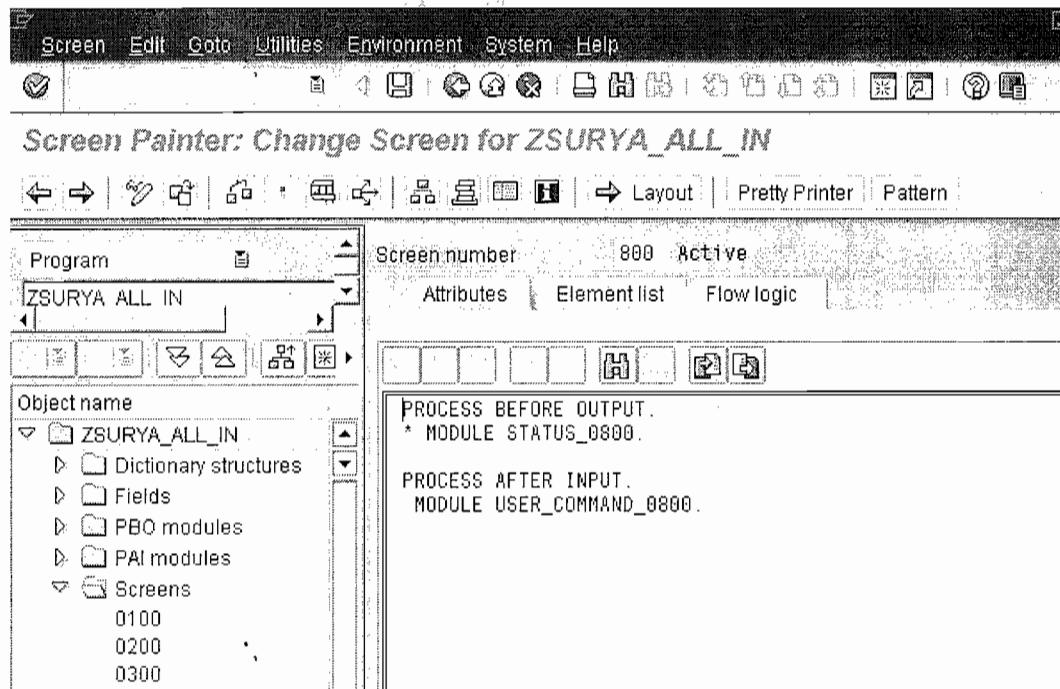
Layout Details :

Module Pool Programming

We Never Compromise in Quality, Would You?



Flow Logic :



Code For the above program.

```
*****  
* PROGRAM : ZSURYA_ALL_IN *  
* AUTHOR : SURYA PRAKASH . BODDU *  
* PURPOSE : MODULE POOL PROGRAM TO CREATE THE *  
* TRAINEE MASTER DATA *  
* REFERENCE : NA *  
* COPIED FROM : NA *  
* TRAPORT REQUEST NO : C11D2K9001 *  
*****  
  
REPORT ZSURYA_ALL_IN  
TABLES : ZPASSWORDS.  
DATA: WA_PSW LIKE ZPASSWORDS,  
      WA_GENERAL LIKE ZGENERALDATA,  
      WA_EDUCATION LIKE ZEDUCATIONDETAIL,  
      WA_PROFESSIONAL LIKE ZPROFESSIONAL,  
      WA_COURSE LIKE ZCOURSEDETAIL,  
      IT LIKE TABLE OF WA_PROFESSIONAL.  
DATA TRAINEE_ID LIKE ZGENERALDATA-ID.  
  
DATA IT_PROFESSIONAL LIKE TABLE OF ZPROFESSIONAL.  
DATA : OLDPSW TYPE ZPSW,  
      NEWPSW TYPE ZPSW,  
      CONFIRMPSW TYPE ZPSW.  
CONTROLS : TAB_STRIP TYPE TABSTRIP,  
            TBC TYPE TABLEVIEW USING SCREEN '700'.  
  
DATA : V_SCREEN_600 TYPE DYNPRO VALUE '0600',  
      V_SCREEN_700 TYPE DYNPRO VALUE '0700',  
      V_SCREEN_800 TYPE DYNPRO VALUE '0800'.  
*&-----*  
*& Module USER_COMMAND_0100 INPUT  
*&-----*  
* text  
*-----*  
MODULE USER_COMMAND_0100 INPUT.  
  
CASE SY-UCOMM.  
  WHEN 'LOGON' OR SPACE.  
    SELECT * FROM ZPASSWORDS INTO WA_PSW WHERE PSW =  
      ZPASSWORDS-PSW.  
  ENDSELECT.  
  IF SY-SUBRC = 0.  
    CALL SCREEN 300.
```

```
ENDIF.  
WHEN 'PSW'.  
  SELECT * FROM ZPASSWORDS INTO WA_PSW WHERE PSW =  
    ZPASSWORDS-PSW.  
ENDSELECT.  
IF SY-SUBRC = 0.  
  CALL SCREEN 200.  
ELSE.  
  MESSAGE E000(ZMSG).  
ENDIF.  
  CLEAR SY-UCOMM.  
ENDCASE.  
ENDMODULE.      " USER_COMMAND_0100 INPUT  
*&-----*  
*&  Module USER_COMMAND_0200 INPUT  
*&-----*  
*  text  
*-----*  
MODULE USER_COMMAND_0200 INPUT.  
  
CASE SY-UCOMM.  
WHEN 'SAVE'.  
  UPDATE ZPASSWORDS SET PSW = NEWPSW WHERE PSW =  
    OLDPWSW.  
  IF SY-SUBRC = 0.  
    MESSAGE S000(ZMSG1).  
  ENDIF.  
WHEN 'BACK'.  
  LEAVE TO SCREEN 0.  
  
ENDCASE.  
CLEAR SY-UCOMM.  
ENDMODULE.      " USER_COMMAND_0200 INPUT  
*&-----*  
*&  Module USER_COMMAND_0300 INPUT  
*&-----*  
*  text  
*-----*  
MODULE USER_COMMAND_0300 INPUT.  
CASE SY-UCOMM.  
WHEN SPACE.  
  CALL SCREEN 400.  
  EXIT.  
WHEN 'BACK'.  
  LEAVE TO SCREEN 0.  
ENDCASE.
```

```
CLEAR SY-UCOMM.  
ENDMODULE.          " USER_COMMAND_0300 INPUT  
*&-----*  
*&  Module USER_COMMAND_0400 INPUT  
*&-----*  
*  text  
*-----*  
MODULE USER_COMMAND_0400 INPUT.  
  
CASE SY-UCOMM.  
  WHEN 'BACK'.  
    LEAVE TO SCREEN 0.  
    CLEAR SY-UCOMM.  
    WHEN 'SAVE'.  
*GENERAL DATA  
  WA_GENERAL-ID = TRAINEE_ID.  
  
*EDUCATION DETAILS  
  WA_EDUCATION-ID = TRAINEE_ID.  
*PROFESSIONAL  
  WA_PROFESSIONAL-ID = TRAINEE_ID.  
  
MODIFY ZGENERALDATA FROM WA_GENERAL.  
IF SY-SUBRC = 0.  
  MESSAGE S002(ZMSG1) WITH TRAINEE_ID.  
ENDIF.  
CLEAR WA_GENERAL.  
MODIFY ZEDUCATIONDETAIL FROM WA_EDUCATION.  
CLEAR WA_EDUCATION.  
MODIFY ZPROFESSIONAL FROM TABLE IT_PROFESIONAL.  
REFRESH IT_PROFESIONAL.  
MODIFY ZCOURSEDETAIL FROM WA_COURSE.  
CLEAR WA_COURSE.  
  
CLEAR SY-UCOMM.  
  
ENDCASE.  
  
CHECK SY-UCOMM = 'TAB1'  
OR SY-UCOMM = 'TAB2'  
OR SY-UCOMM = 'TAB3'  
OR SY-UCOMM = 'TAB4'.  
  
TAB_STRIP-ACTIVETAB = SY-UCOMM.  
CLEAR SY-UCOMM.  
ENDMODULE.
```

```
" USER_COMMAND_0400 INPUT
*&-----*
*&   Module STATUS_0100 OUTPUT
*&-----*
*   text
*-----*

MODULE STATUS_0100 OUTPUT.
ZPASSWORDS-UNAME = 'SAP@EMAX'.
SET PF-STATUS 'STATUS'.
SET TITLEBAR 'TITLE_100'.

ENDMODULE.      " STATUS_0100 OUTPUT
*&-----*
*&   Module VALIDATE_LOGON_DETAILS INPUT
*&-----*
*   text
*-----*

MODULE VALIDATE_LOGON_DETAILS INPUT.
SELECT * FROM ZPASSWORDS INTO WA_PSW WHERE PSW =
ZPASSWORDS-PSW.
ENDSELECT.
IF SY-SUBRC <> 0.
MESSAGE E000(ZMSG).
ENDIF.

ENDMODULE.      " VALIDATE_LOGON_DETAILS INPUT
*&-----*
*&   Module EXIT_FROM_PROG INPUT
*&-----*
*   text
*-----*

MODULE EXIT_FROM_PROG INPUT.
IF SY-UCOMM = 'BACK' OR
SY-UCOMM = 'EXIT' OR
SY-UCOMM = 'CANCEL'.
LEAVE TO SCREEN 0.
ENDIF.

ENDMODULE.      " EXIT_FROM_PROG INPUT
*&-----*
*&   Module STATUS_0300 OUTPUT
*&-----*
*   text
*-----*

MODULE STATUS_0300 OUTPUT.
SET PF-STATUS 'STATUS'.
```

```
SET TITLEBAR 'ZTITLE1'.
ENDMODULE.          " STATUS_0300 OUTPUT
*&-----*
*&   Module STATUS_0200 OUTPUT
*&-----*
*   text
*-----*

MODULE STATUS_0200 OUTPUT.
SET PF-STATUS 'STATUS'.
SET TITLEBAR 'TITLE_200'.
OLDPSW = ZPASSWORDS-PSW.

ENDMODULE.          " STATUS_0200 OUTPUT
*&-----*
*&   Module VALIDATE_PASSWORD INPUT
*&-----*
*   text
*-----*

MODULE VALIDATE_PASSWORD INPUT.

SELECT * FROM ZPASSWORDS INTO WA_PSW WHERE PSW =
OLDPSW.
ENDSELECT.

IF SY-SUBRC = 0.
IF NEWPSW <> CONFIRMPSW.
MESSAGE E001(ZMSG).
ENDIF.
ELSE.
MESSAGE E002(ZMSG).
ENDIF.
ENDMODULE.          " VALIDATE_PASSWORD INPUT
*&-----*
*&   Module STATUS_0400 OUTPUT
*&-----*
*   text
*-----*

MODULE STATUS_0400 OUTPUT.
CLEAR SY-UCOMM.
SET PF-STATUS 'STATUS'.
ENDMODULE.          " STATUS_0400 OUTPUT
*&-----*
*&   Module APPEND_TO_IT_PROFESSIONAL INPUT
*&-----*
*   text
*-----*
```

```
MODULE APPEND_TO_IT_PROFESSIONAL INPUT.  
  WA_PROFESSIONAL-ID = TRAINEE_ID.  
  APPEND WA_PROFESSIONAL TO IT_PROFESSIONAL.  
ENDMODULE.      " APPEND_TO_IT_PROFESSIONAL INPUT  
*&-----*  
*&   Module VALIDATE_TRAINEE_DETAILS INPUT  
*&-----*  
*   text  
*-----*  
MODULE VALIDATE_TRAINEE_DETAILS INPUT.  
  SELECT * FROM ZPROFESSIONAL INTO WA_PROFESSIONAL WHERE  
ID =  
  TRAINEE_ID.  
ENDSELECT.  
IF SY-SUBRC = 0.  
  MESSAGE E003(ZMSG).  
ENDIF.  
  
ENDMODULE.      " VALIDATE_TRAINEE_DETAILS INPUT  
*&-----*  
*&   Module USER_COMMAND_0500 INPUT  
*&-----*  
*   text  
*-----*  
MODULE USER_COMMAND_0500 INPUT.  
  
ENDMODULE.      " USER_COMMAND_0500 INPUT  
*&-----*  
*&   Module USER_COMMAND_0600 INPUT  
*&-----*  
*   text  
*-----*  
MODULE USER_COMMAND_0600 INPUT.  
  
ENDMODULE.      " USER_COMMAND_0600 INPUT  
*&-----*  
*&   Module USER_COMMAND_0800 INPUT  
*&-----*  
*   text  
*-----*  
MODULE USER_COMMAND_0800 INPUT.  
  
ENDMODULE.      " USER_COMMAND_0800 INPUT  
*&-----*  
*&   Module SET_LINES OUTPUT
```

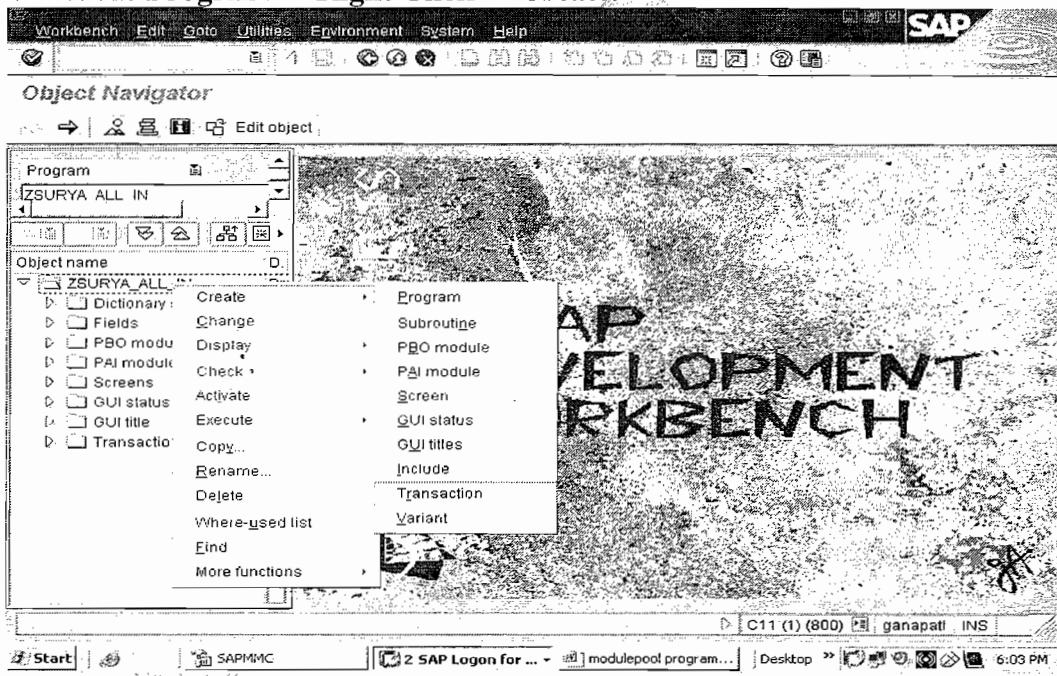
Module Pool Programming

We Never Compromise in Quality, Would You?

```
*&-----*
*   text
*-----*
MODULE SET_LINES OUTPUT.
TBC-LINES = 20.
ENDMODULE.          " SET_LINES OUTPUT
*&-----*
*&  Module USER_COMMAND_0700 INPUT
*&-----*
*   text
*-----*
MODULE USER_COMMAND_0700 INPUT.

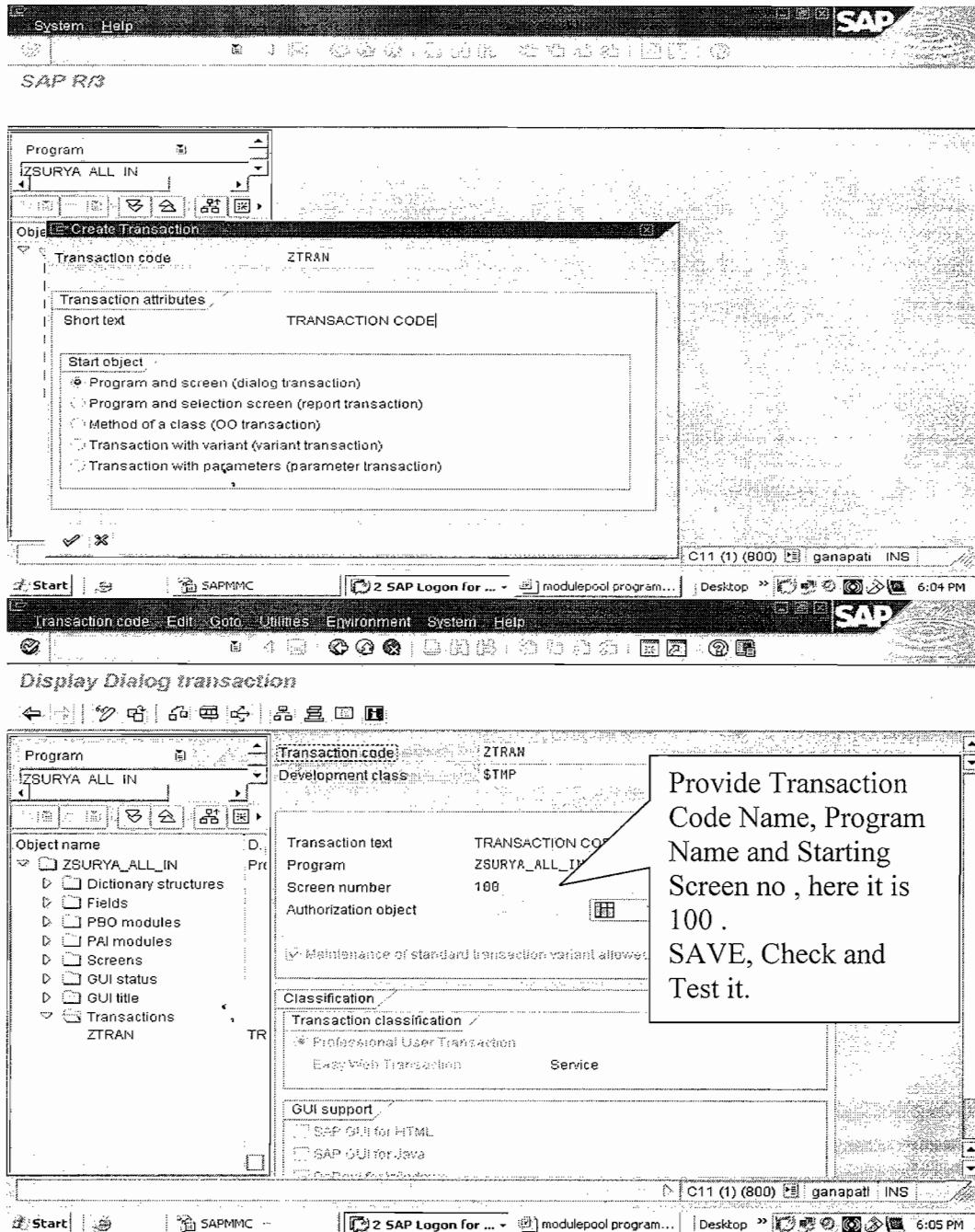
ENDMODULE.          " USER_COMMAND_0700 INPUT
```

Create Transaction code for the program :
Select the Program -> Right Click -> Create.



Module Pool Programming

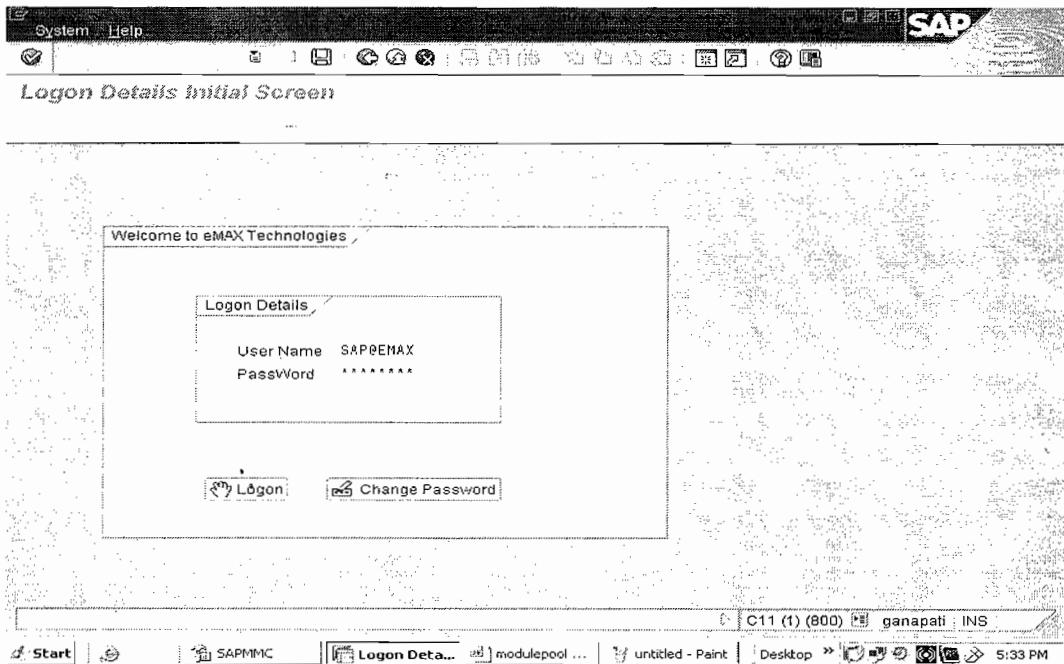
We Never Compromise in Quality, Would You?



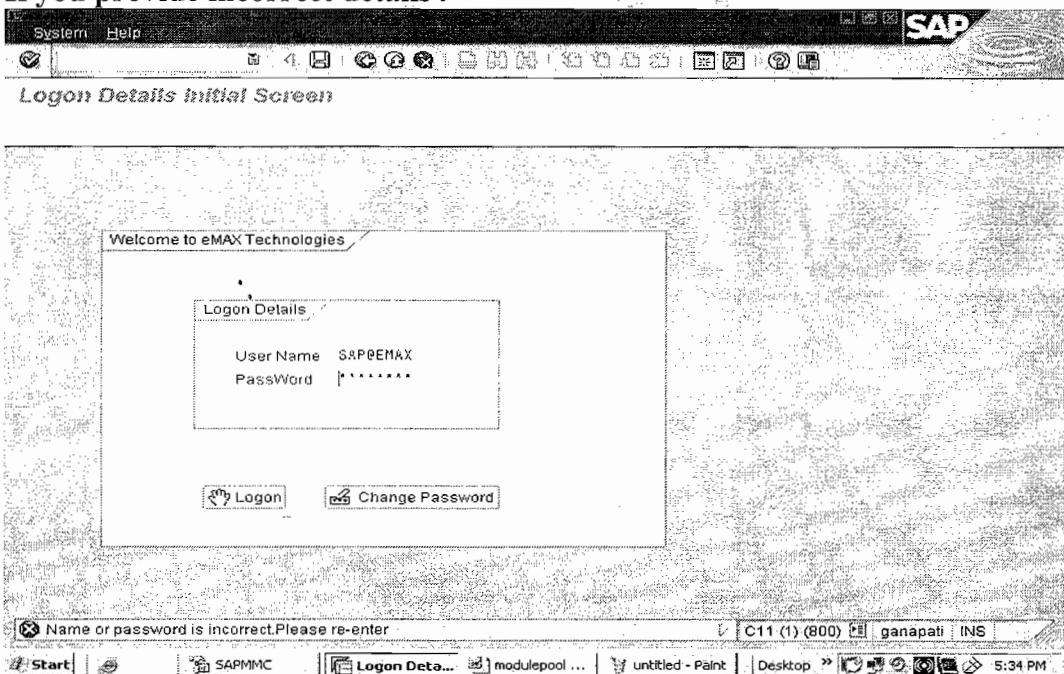
Execution Steps and Output Screens :
Execute the transaction ZTRAN.
Initial Screen

Module Pool Programming

We Never Compromise in Quality, Would You?



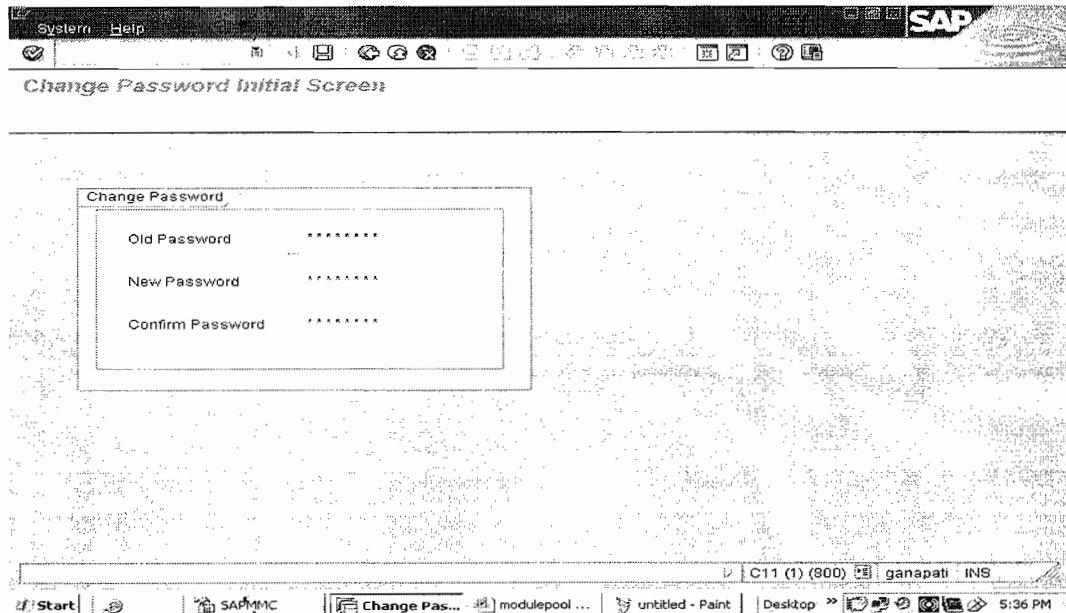
**Provide the logon details
If you provide incorrect details .**



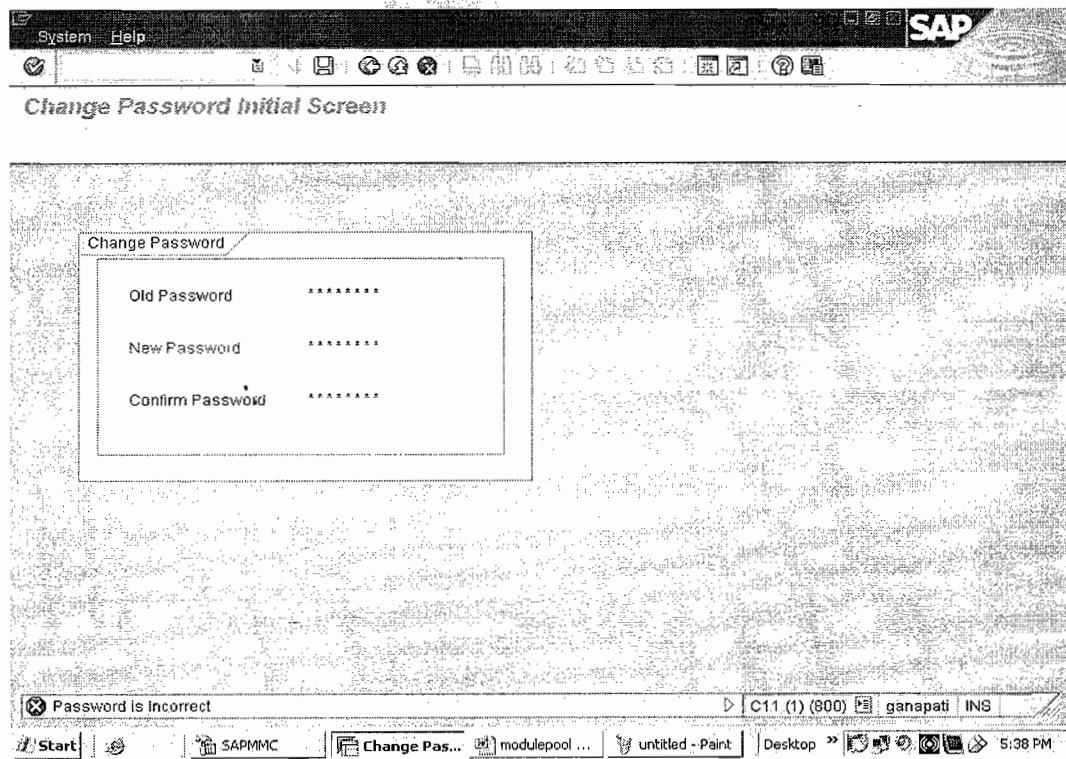
**If you provide correct logon details and press change password button then
you get change password screen**

Module Pool Programming

We Never Compromise in Quality, Would You?



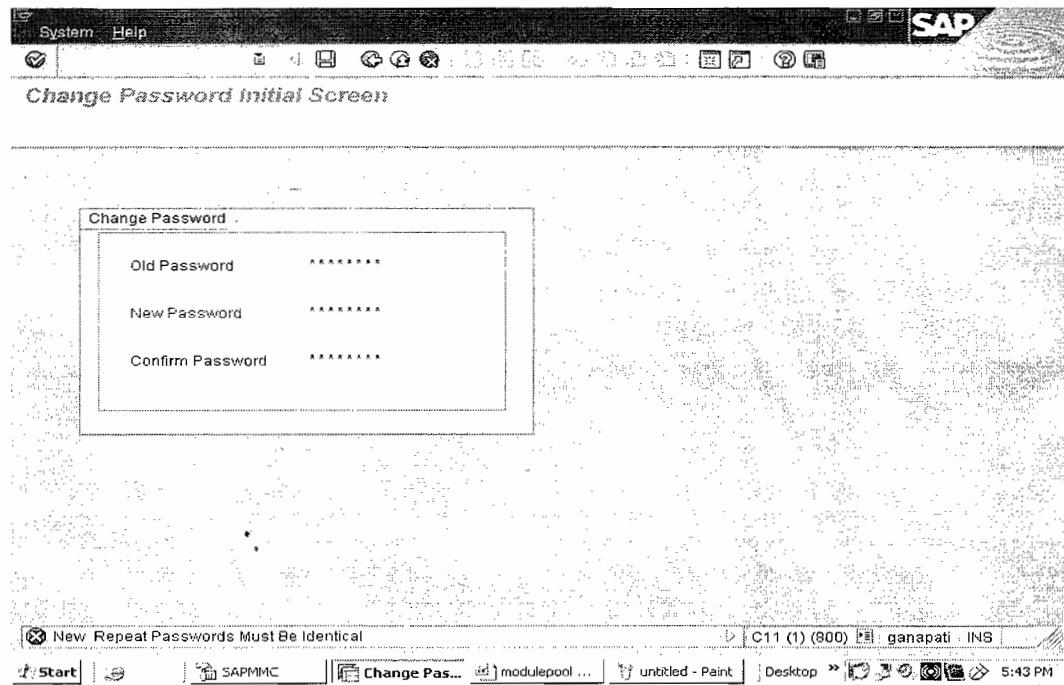
If the Old Pass word is wrongly entered then it through the error for the SAVE.



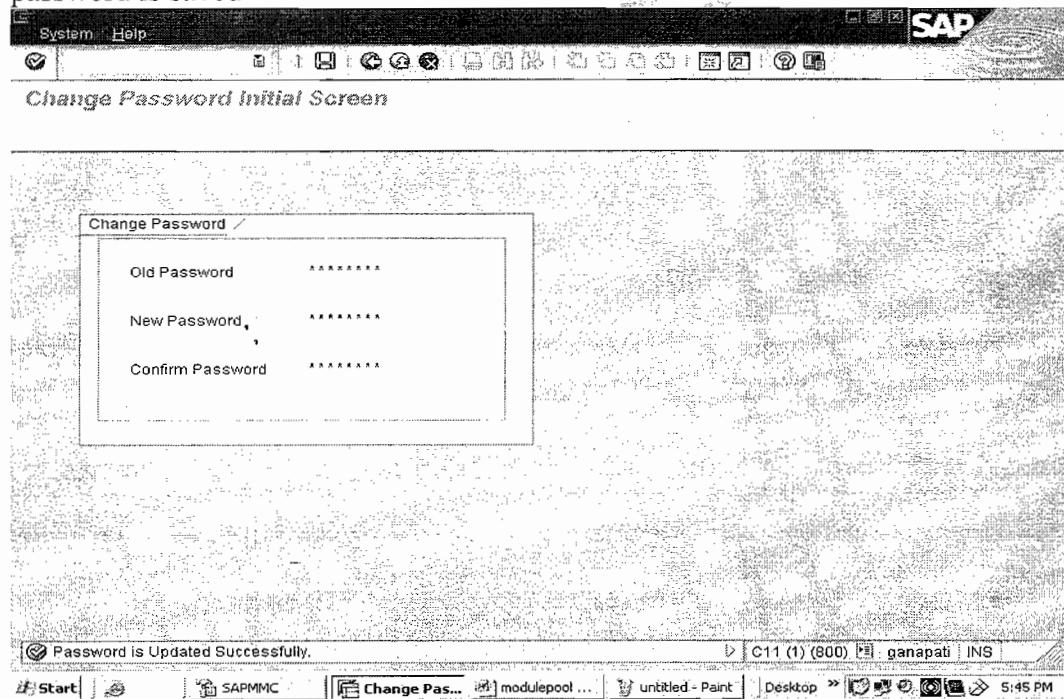
When both New Password and Confirm Passwords are not Same.

Module Pool Programming

We Never Compromise in Quality, Would You?



If you provide all the details correct and press **SAVE** then only your new password is saved

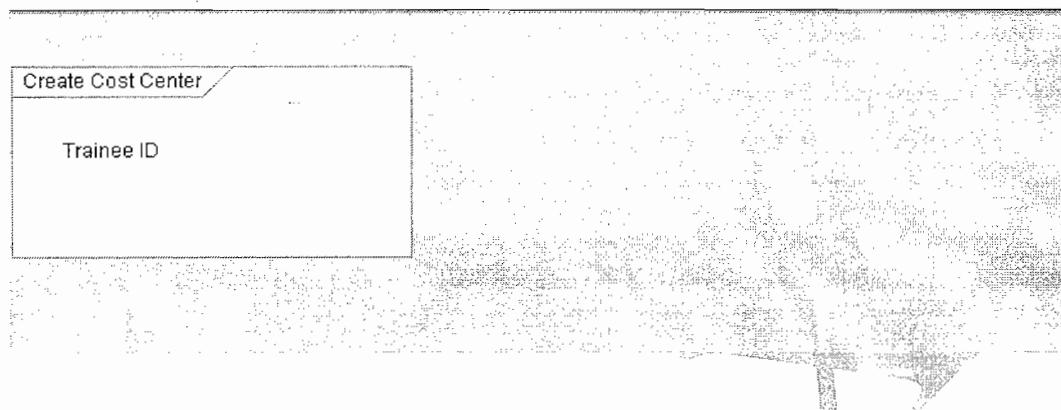
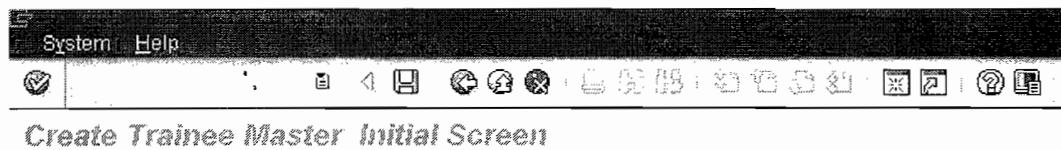


And press **BACK** and provide correct logon details and press **LOGON** or **ENTER**

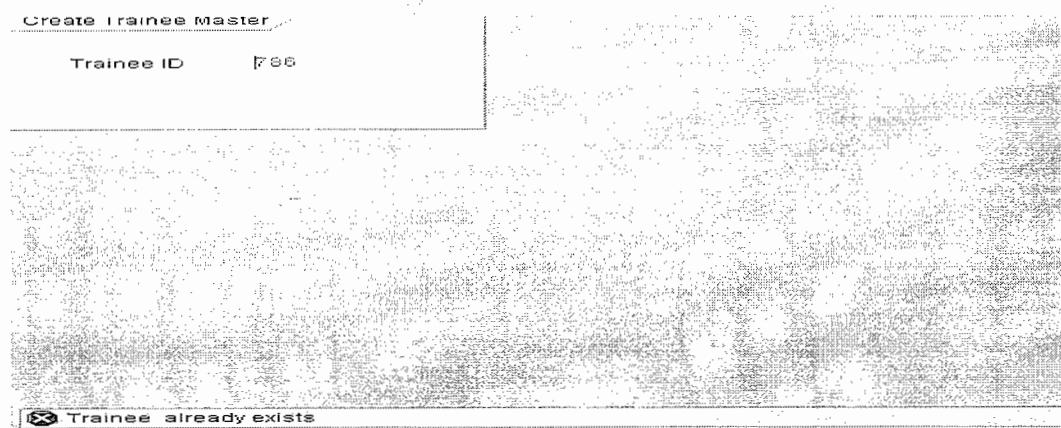
Then you get trainee master initial screen

Module Pool Programming

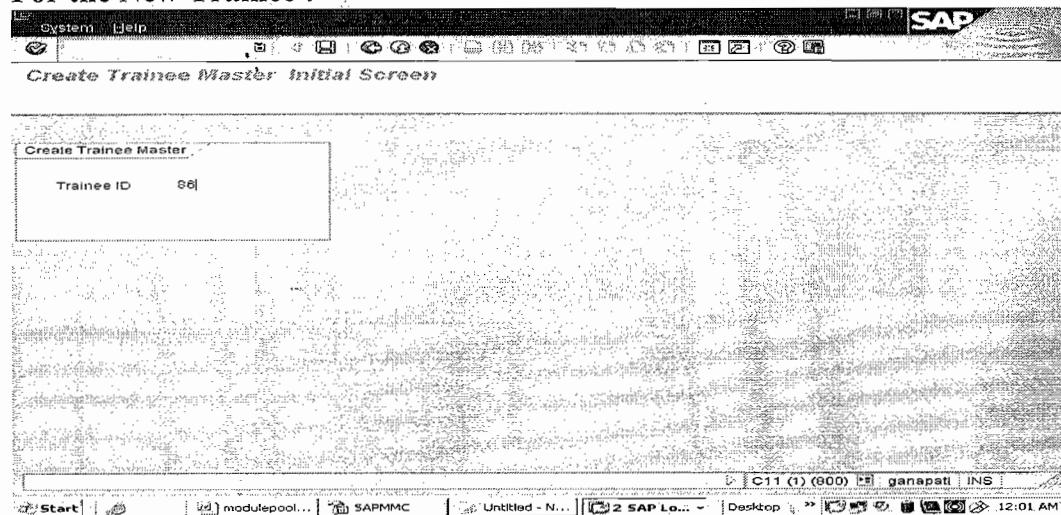
We Never Compromise in Quality, Would You?



Provide the New Trainee ID

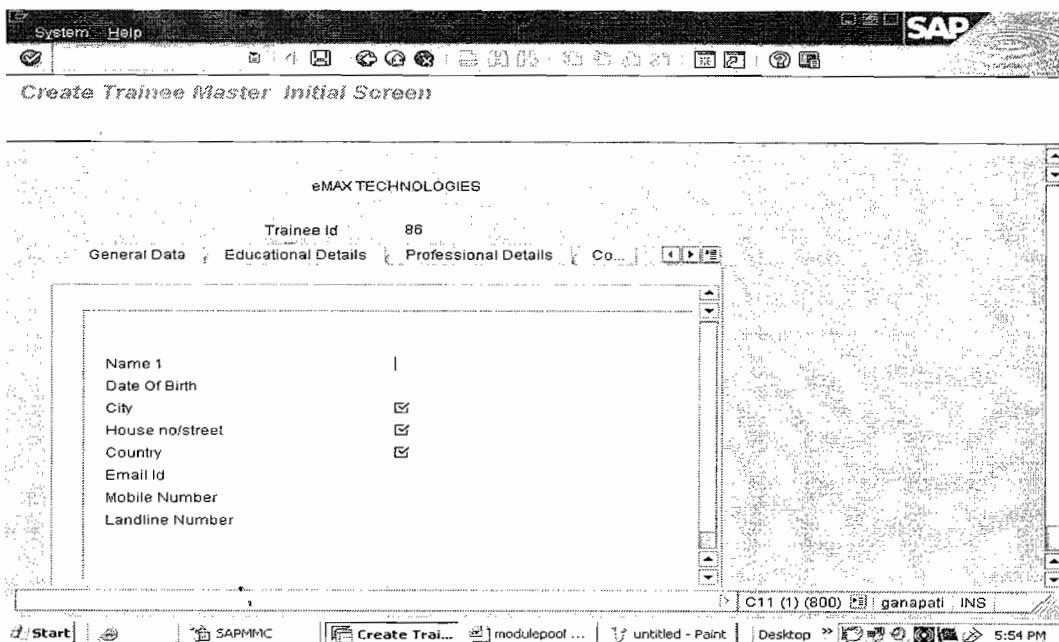


For the New Trainee :

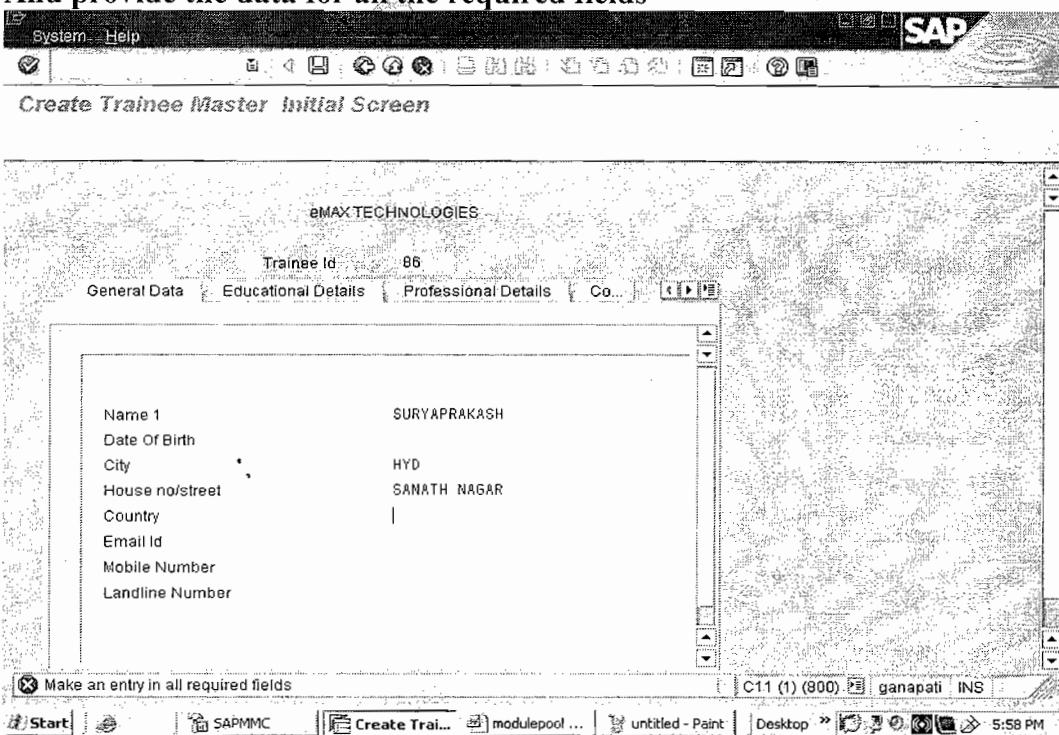


Module Pool Programming

We Never Compromise in Quality, Would You?



And provide the data for all the required fields

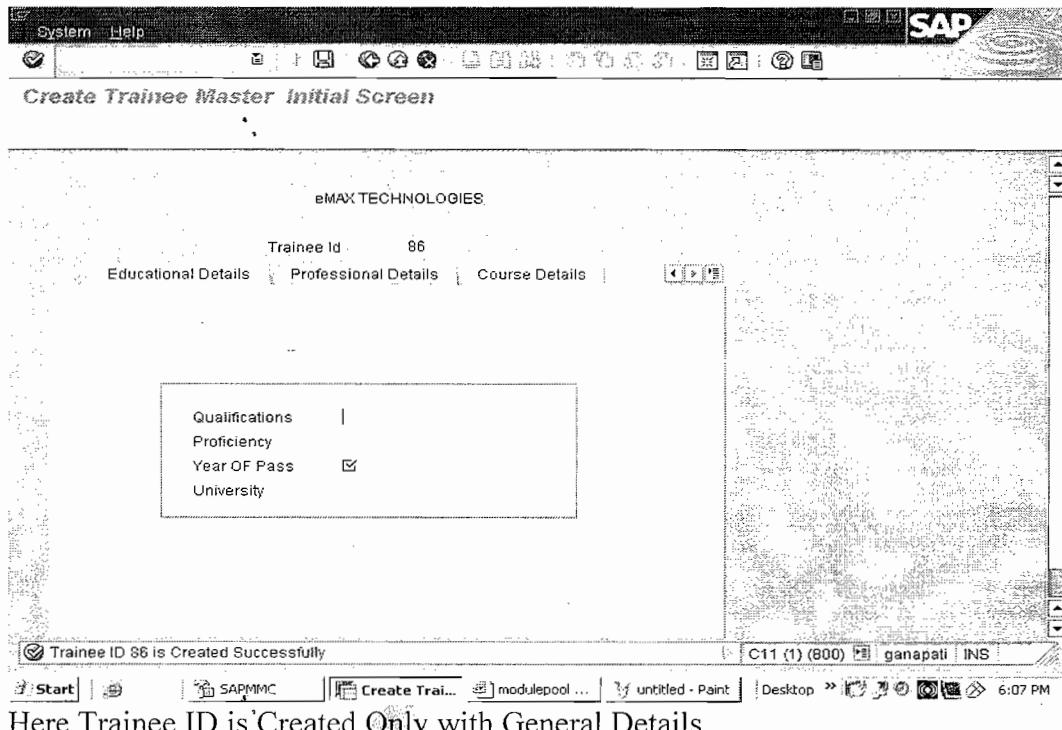


After you provide all the required fields and press **SAVE** then this data will be saved into related database table. When you Click on any tab in the tab strip then you go to related screen.

When you press the **SAVE** whatever the data available up to that point (TAB)will be saved into relevant data base tables

Module Pool Programming

We Never Compromise in Quality, Would You?



Here Trainee ID is Created Only with General Details.

Observe the Data from the Corresponding Data base Table .

| Table ZGENERALDATA Display | |
|----------------------------|--------------|
| Trainee ID | 86 |
| Name 1 | SURYAPRAKASH |
| Date Of Birth | |
| City | HYD |
| House no/street | SANATH NAGAR |
| Country | IN |
| eMAIL ID | |
| MOBILENO | 0 |
| LANDLINE | |

Tables Used in this Program :

Table : ZPASSWORDS

Module Pool Programming

We Never Compromise in Quality, Would You?

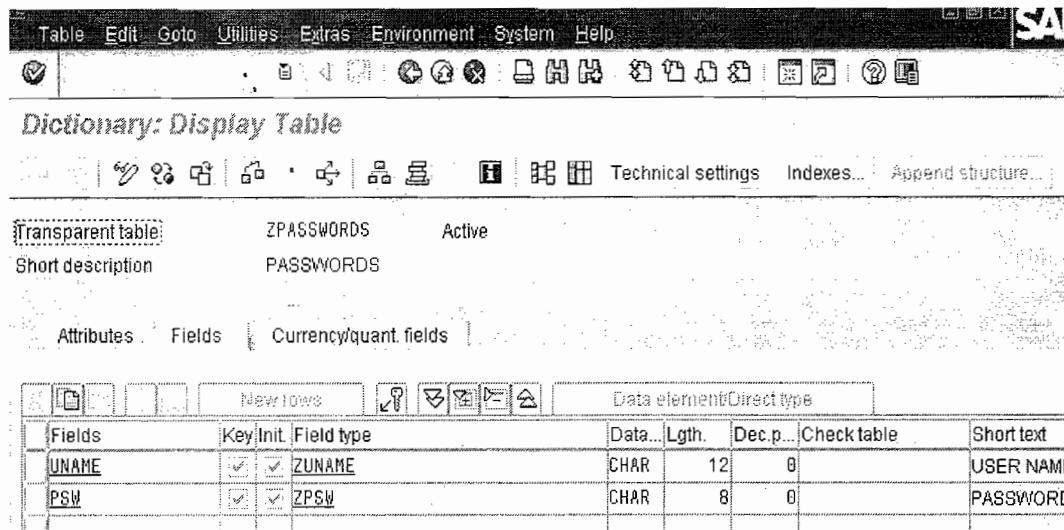


Table : ZGENERALDATA

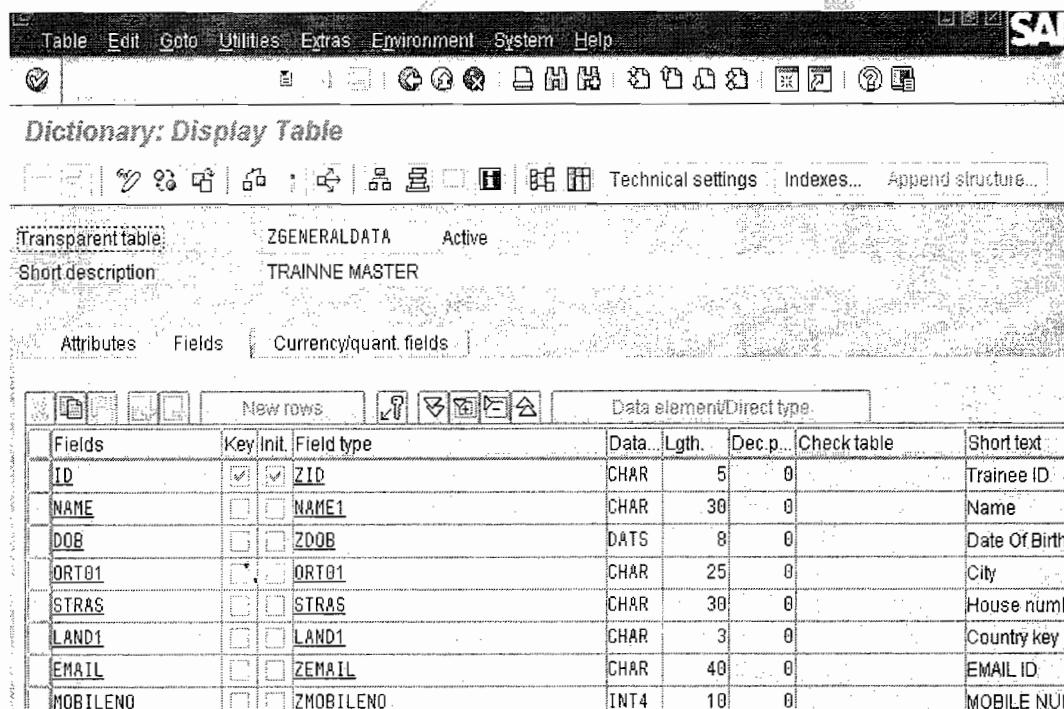


Table : ZPROFESSIONAL

The screenshot shows the SAP Dictionary: Display Table interface for the ZPROFESSIONAL table. The table is active and has a short description of 'PROFISHINAL DETAILS'. The Fields tab is selected, showing the following data elements:

| Fields | Key | Init. | Field type | Data... | Lgth. | Dec.p... | Check table | Short text |
|--------------|-------------------------------------|-------------------------------------|------------|---------|-------|----------|-------------|-------------|
| ID | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | ZID | CHAR | 5 | 0 | | Trainee ID |
| TOTALEXP | <input type="checkbox"/> | <input type="checkbox"/> | ZEXP | CHAR | 3 | 0 | | TOTAL YEAR |
| FROMDATE | <input type="checkbox"/> | <input type="checkbox"/> | ZDATE | DATS | 8 | 0 | | DATE |
| TODATE | <input type="checkbox"/> | <input type="checkbox"/> | ZDATE1 | DATs | 8 | 0 | | TO DATE |
| ORGANIZATION | <input type="checkbox"/> | <input type="checkbox"/> | ZORG | CHAR | 25 | 0 | | ORGANIZATI |
| ORT01 | <input type="checkbox"/> | <input type="checkbox"/> | ORT01 | CHAR | 25 | 0 | | City |
| LAND1 | <input type="checkbox"/> | <input type="checkbox"/> | LAND1 | CHAR | 3 | 0 | | Country key |

Table : ZCOURSEDETAIL

The screenshot shows the SAP Dictionary: Display Table interface for the ZCOURSEDETAIL table. The table is active and has a short description of 'COURSEDETAILS'. The Fields tab is selected, showing the following data elements:

| Fields | Key | Init. | Field type | Data... | Lgth. | Dec.p... | Check table | Short text |
|---------|-------------------------------------|-------------------------------------|------------|---------|-------|----------|-------------|---------------|
| UD | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | ZID | CHAR | 5 | 0 | | Trainee ID |
| COURSE | <input type="checkbox"/> | <input type="checkbox"/> | ZCOURSE | CHAR | 15 | 0 | | COURSES |
| FACULTY | <input type="checkbox"/> | <input type="checkbox"/> | ZFACULTY | CHAR | 25 | 0 | | FACULTY |
| DOJ | <input type="checkbox"/> | <input type="checkbox"/> | ZDOJ | DATs | 8 | 0 | | Date Of Joini |

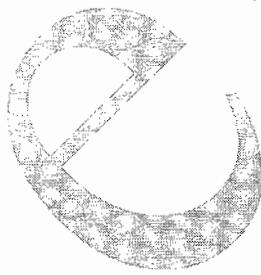
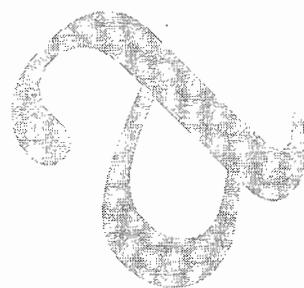
4 .LSMW (Legacy system Migration Workbench)

1. Introduction to LSMW

2. Working with Batch Input Recording

3. Working With Direct Input

4. Working With IDOC



Step-by-Step Guide for Using LSMW to Update Customer Master Records***LSM Workbench: What is it?***

The LSM Workbench is an R/3-based tool that supports You when transferring data from non-SAP systems ("Legacy Systems") to R/3 once or periodically. The tool supports conversion of data of the legacy system in a convenient way. The data can then be imported into the R/3 system via batch input, direct input, BAPIs or IDocs.

Business Case:

As a part of reorganization and to better serve the customer needs, you are regrouping many of the customers. In SAP terms, you are changing the Sales Office, Sales Group and Customer Groups for specific Customer Master Records. Typically, you would maintain customer records with transaction XD02 to update 'Sales View'. You would enter Customer Key (Customer No, Sales Organization, Distribution Channel, and Division) and update relevant fields on Sales View screen.

This document contains Step-by-step instructions to use LSMW to update Customer Master Records. It has two demonstration examples - one using Batch Recording and another using standard SAP Object.

Note! The screen prints in this article are from IDES Release 4.6. They may differ slightly in other versions.

Demo Example 1**LSMW to Update Customer Master Records with Transaction Recording**

Call Legacy System Migration Workbench by entering transaction code LSMW. Every conversion task is grouped together as Project / Subproject / Object structure. Create a **Project** called **LSMW_DEMO** and a **Subproject** as **CUSTOMERS** and Object as **CUST_REC** as shown in **Figure 1**.

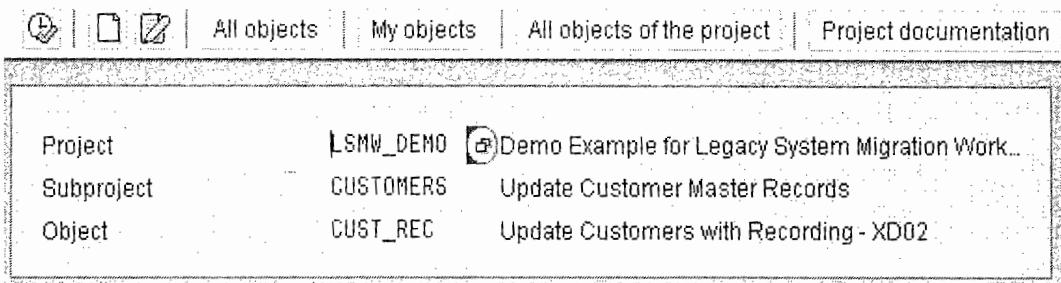
Legacy System Migration Workbench

Figure 1 Conversion Task with Project, Subproject and Object

The main screen of LSMW provides wizard-like step-by-step tasks, as shown in **Figure 2**. To complete your data conversion, you need to execute these steps in sequence. Once a step is executed, the cursor is automatically positioned to the next step.

Note that these steps may look different depending upon your **Personal menu** settings. You could make step numbers visible by ‘Numbers on’ icon or hidden by ‘Numbers off’ icon. You can execute a step by double-clicking on the row. Toggle icon ‘Doubleclick=Display’ or ‘Doubleclick=Edit’, makes the step in ‘display’ mode or ‘change’ mode.

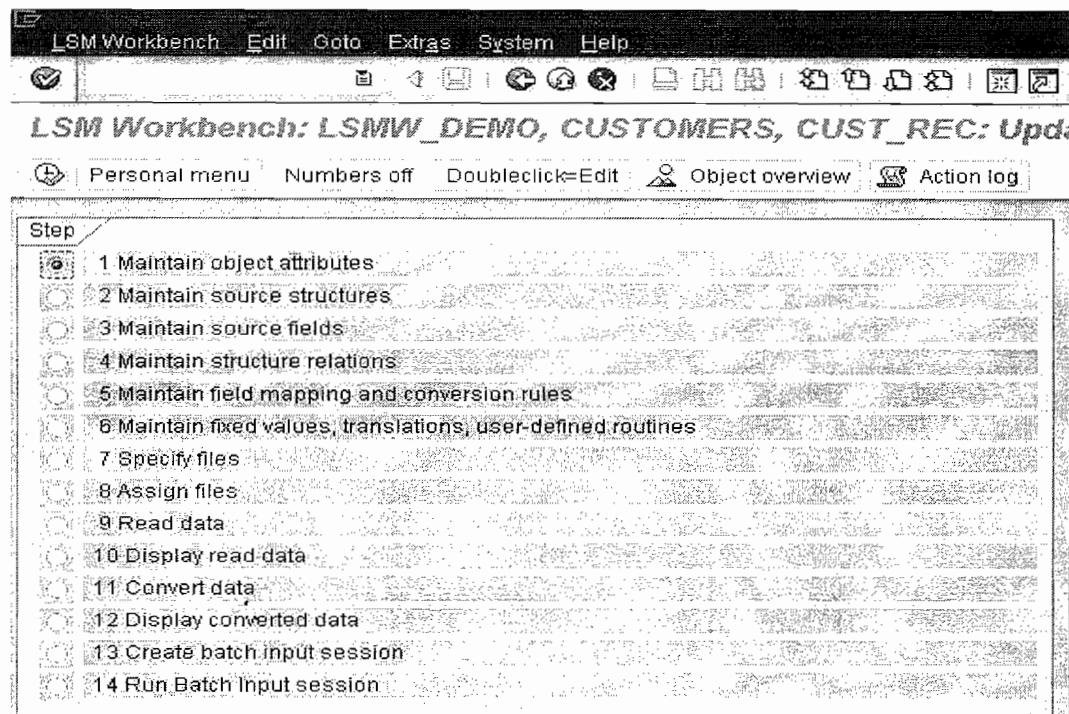
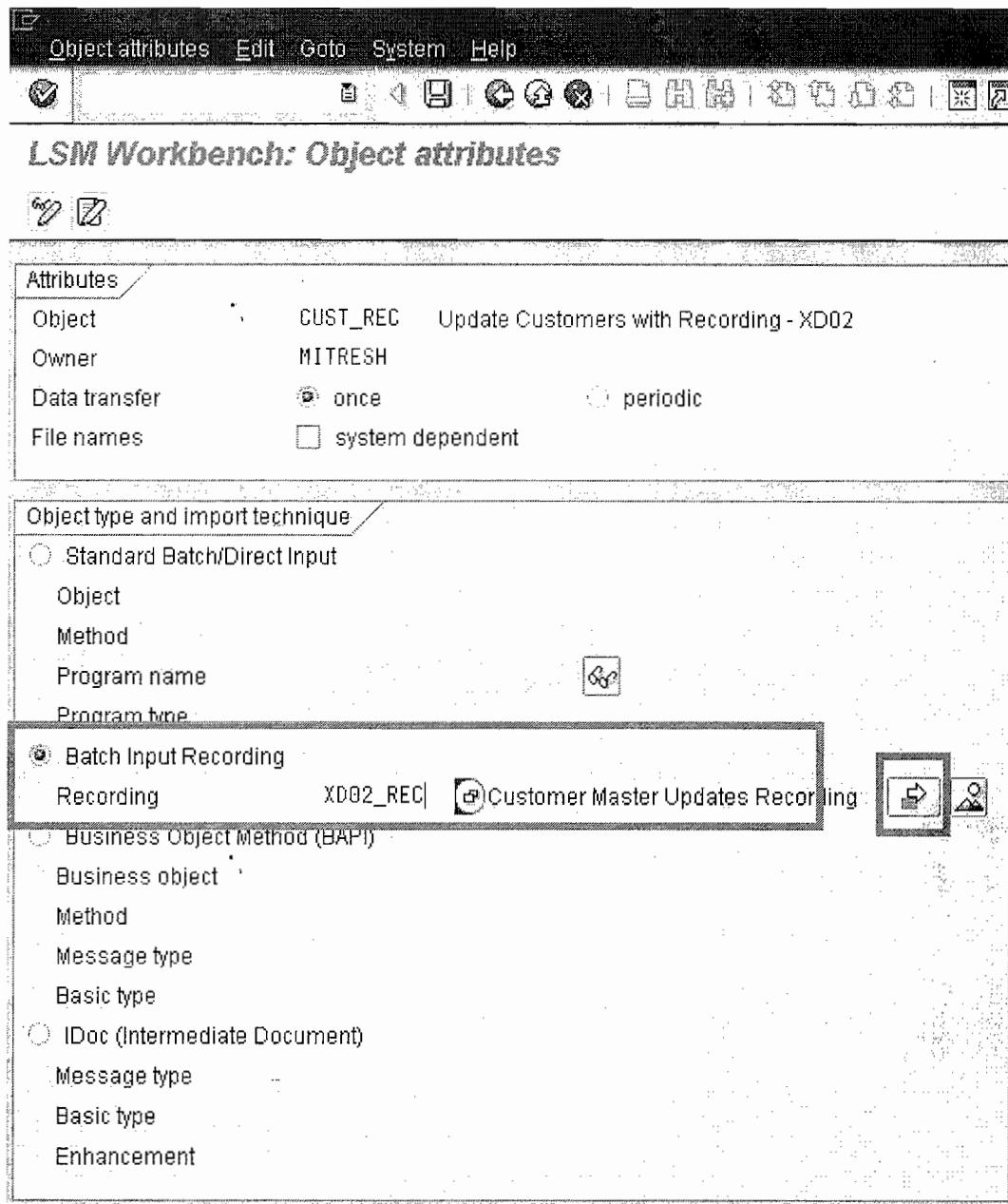


Figure 2 LSMW Wizard – initial screen

Step 1: Maintain Object attributes

In this example, you will be updating the customer master records with the help of recording a transaction (XD02). Choose radio button **Batch Input Recording** and click on the recording overview icon to record the R/3 transaction. Enter the **Recording name** as XD02_REC, the **description** as **Customer Master Updates Recording**, and the **transaction code** as XD02.

**Figure 3** Object type ‘Transaction Recording’

The system calls the transaction code XD02 and prompts you to complete the **Change**

Customer transaction, as shown in Figure 4. Enter the key customer information (I entered customer number 1000, sales organization 1000, distribution channel 01, and division 00) and choose ‘Sales’ view within ‘Sales area data’. Make changes to these three fields (I entered, sales office 1010, sales group 110, and customer group 01) and save the transaction.

Change Customer: Initial Screen

The screenshot shows the SAP Change Customer: Initial Screen (XD02). At the top, there are fields for Customer (1000), Company code (1000), Sales organization (10), Distribution channel (00), and Division (00). Below these are two large boxes of checkboxes:

- General data:**
 - Address
 - Control data
 - Marketing
 - Payment transactions
 - Tax categories
 - Unloading points
 - Foreign trade
 - Contact persons
- Company code data:**
 - Accounting info
 - Payment transactions
 - Correspondence
 - Insurance
 - Withholding tax

At the bottom left is a checkbox for "Use central address management".

Figure 4 Transaction recording for Transaction Code 'XD02'

Once the transaction is completed, R/3 records the flow of screens and fields and saves the information, as shown in **Figure 5**.

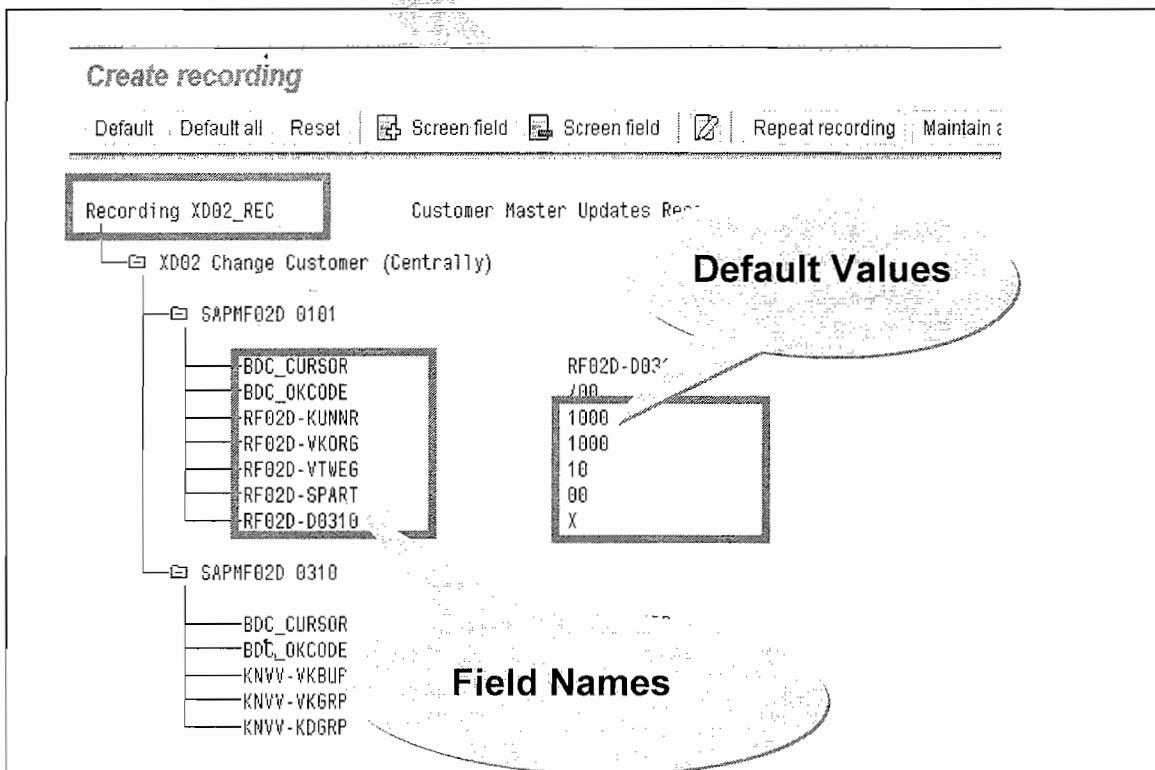
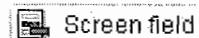


Figure 5 Transaction recording overview

Note that the fields are populated with default values. The values you entered when you recorded the transaction are set by default.

Note that if you have more fields in the recording than needed, you can remove them by clicking 'Remove Screen field' icon.



Observe that the transaction-recording process stores field names in a technical format. By pressing the F1 key on individual screen fields and then pressing the F9 key, the system displays technical names. You then can replace the technical names with descriptive names. Double-click on the field **RF02D-KUNNR** and enter the name as **KUNNR** and the description as **Customer Account Number** and remove the default value. (See **Figure 6**.)

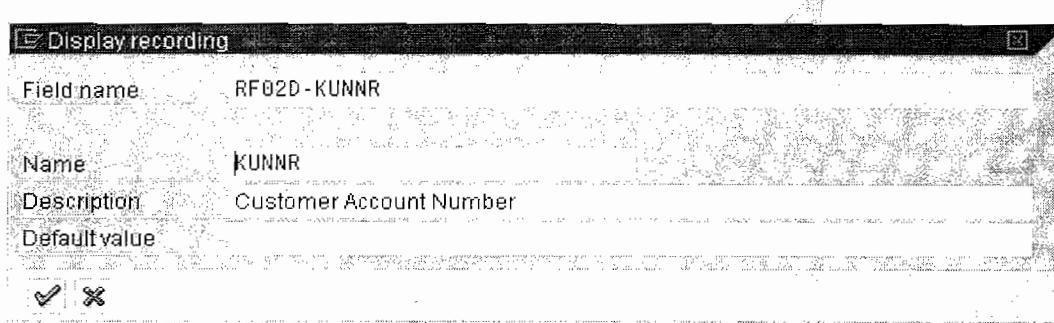


Figure 6 Field attributes

Similarly, double-click on all other fields with default values and make appropriate changes. Once you have made changes, the recording overview screen looks like what you see in **Figure 7**.

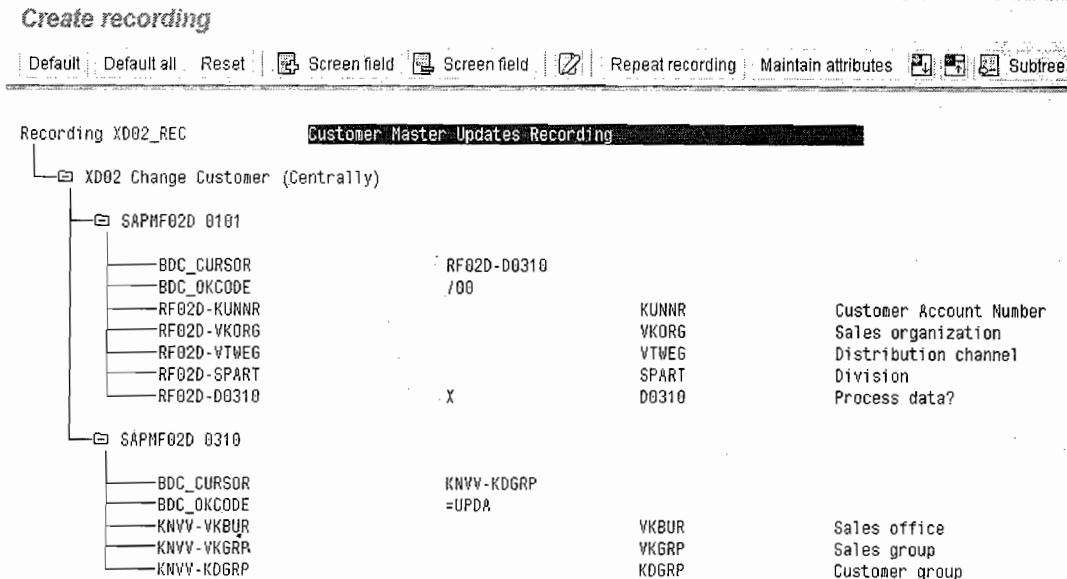


Figure 7 Transaction Recording Overview – with screen field attributes

Save your changes. When you go back to the initial screen, you will see that the initial screen steps have changed. Since you want to import data via the BDC method, the **Direct Input** and **IDoc**-related steps are hidden, as they are not relevant.

Step 2. Maintain Source Structures

Give a name and a description to the source structure (**Figure 8**).

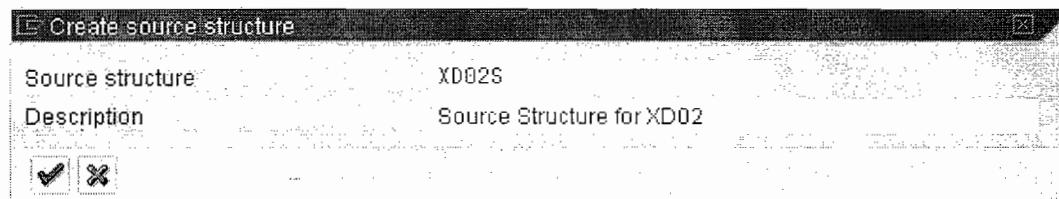


Figure 8 Source Structure

Step 3. Maintain Source Fields

In this step, you need to list what fields are present in the source structure. The easiest way is to click on 'Table Maintenance' icon to enter Fieldname, Type and Length for each field as shown in **Figure 9**.

| Source fields of source structure XD02S | | | | |
|---|------|------|-------------------|--|
| Field name | Type | L... | Field description | |
| CUSTOMER | C | 10 | CUSTOMER | |
| SALESORG | C | 4 | SALESORG | |
| DISTCHANNEL | C | 2 | DISTCHANNEL | |
| DIVISION | C | 2 | DIVISION | |
| SALESOFFICE | C | 4 | SALESOFFICE | |
| SALESGROUP | C | 3 | SALESGROUP | |
| CUSTOMERGROUP | C | 2 | CUSTOMERGROUP | |

Figure 9 Source fields of source Structure

Note that your input file will have four fields as key fields and you need to update three fields in the system.

Step 4: Maintain Structure Relations

Execute a step to 'Maintain Structure Relations'. (See **Figure 10**.) Since, there is only one Source and Target Structure, the relationship is defaulted automatically.

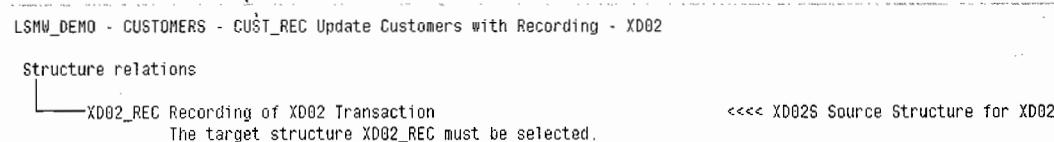


Figure 10 Structure Relation

Step 5: Maintain field mapping and conversion rules

Field **RF02D-D0310** represents that you chose ‘Sales view’ for the customer Master screen accordingly its value should be set to **X**. Keep your cursor on field RF02D-D0310 and click on **Constant** rule icon to choose the constant value of ‘**X**’.

If your source file already has the field value, you choose rule ‘Source Field’.

Keep cursor on field ‘KUNNR’ and click on ‘Assign Source field’ icon to choose source field CUSTOMER from structure XD02S as shown in **Figure 11**.

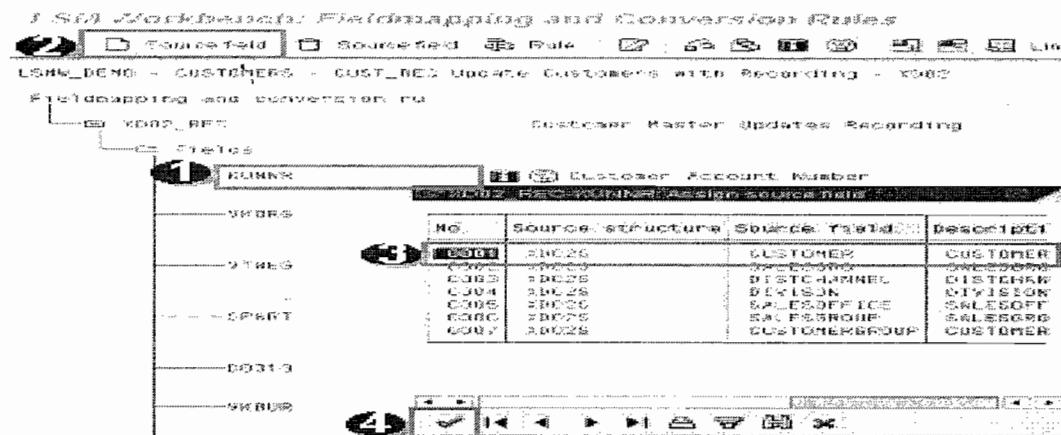


Figure 11 Assign source fields

Similarly, assign ‘Source Field’ rules to the remaining fields.

Once all the fields are mapped, you should have an overview screen as shown in **Figure 12**.

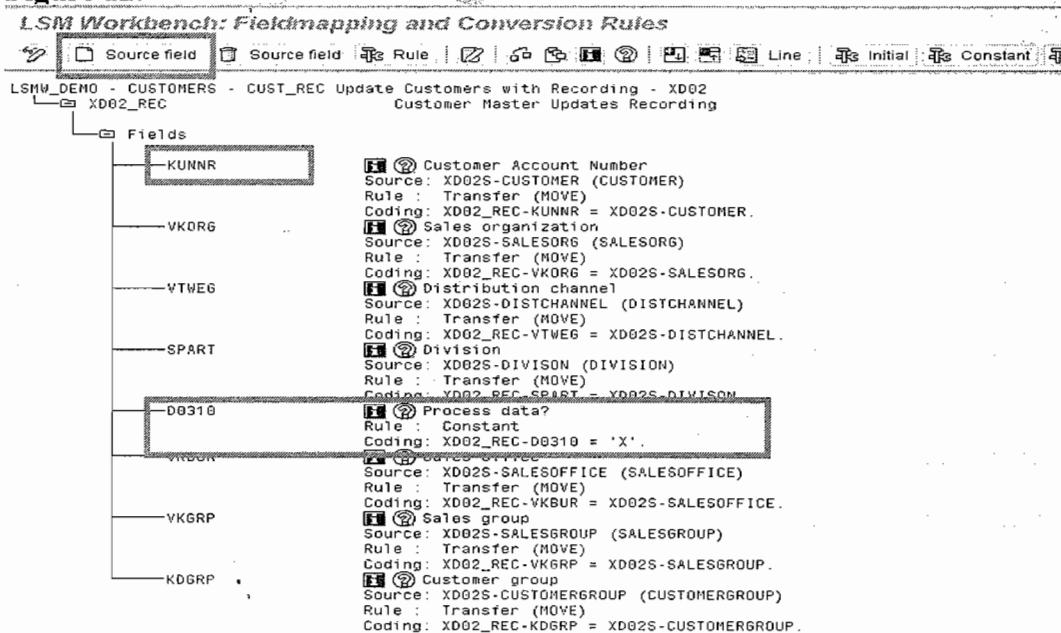


Figure 12 Field mapping and Conversion rules overview

Step 6: Maintain fixed values, translations, user-defined routines

You can also maintain re-usable translations and user-defined routines, which can be used across conversion tasks. In this case, that step is not required.

Step 7: Specify files

In this step, we define how the layout of the input file is. The input file is a [Tab] delimited with the first row as field names. It is present on my PC (local drive) as C:\XD02.txt. (See **Figure 13.**)

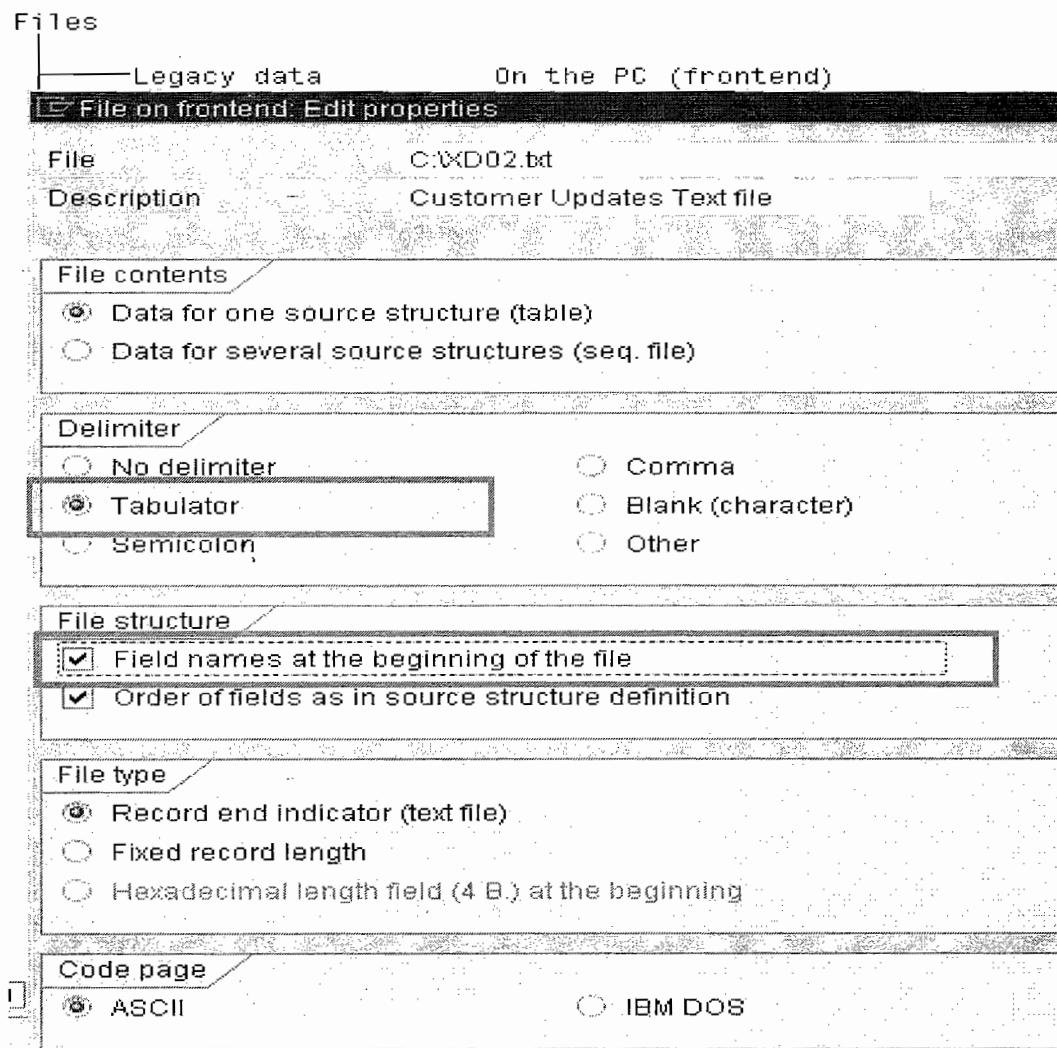


Figure 13 File attributes

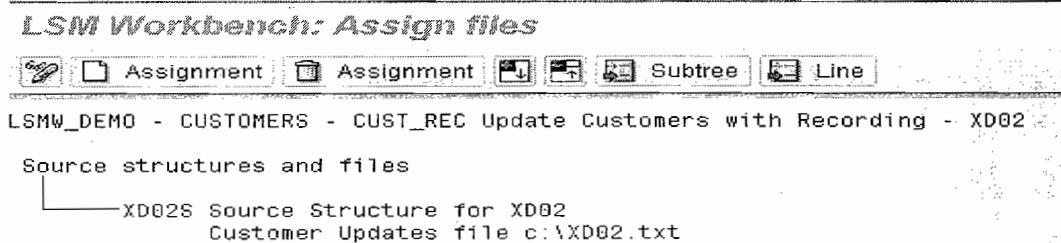
Create an Excel file (**Figure 14**) with your data and save it as a Tab-delimited text file on your local drive (C:\) and name it XD02.txt.

| A | B | C | D | E | F | G | |
|----|----------|----------|----------|----------|----------|----------|---------|
| 1 | Customer | SalesOrg | DistChnl | Division | SalesOff | SalesGrp | CustGrp |
| 2 | 1000 | 1000 | 10 00 | | 1010 | 110 01 | |
| 3 | 1007 | 1000 | 10 00 | | 1010 | 110 01 | |
| 4 | 1008 | 1000 | 10 00 | | 1010 | 110 01 | |
| 5 | 1010 | 1000 | 10 00 | | 1010 | 110 01 | |
| 6 | 1020 | 1000 | 10 00 | | 1010 | 110 01 | |
| 7 | 1025 | 1000 | 10 00 | | 1010 | 110 01 | |
| 8 | 1026 | 1000 | 10 00 | | 1010 | 110 01 | |
| 9 | 1030 | 1000 | 10 00 | | 1010 | 110 01 | |
| 10 | 1040 | 1000 | 10 00 | | 1010 | 110 01 | |
| 11 | 1051 | 1000 | 10 00 | | 1010 | 110 01 | |
| 12 | | | | | | | |

Figure 14 Source data in Excel file (saved as Tab delimited file)

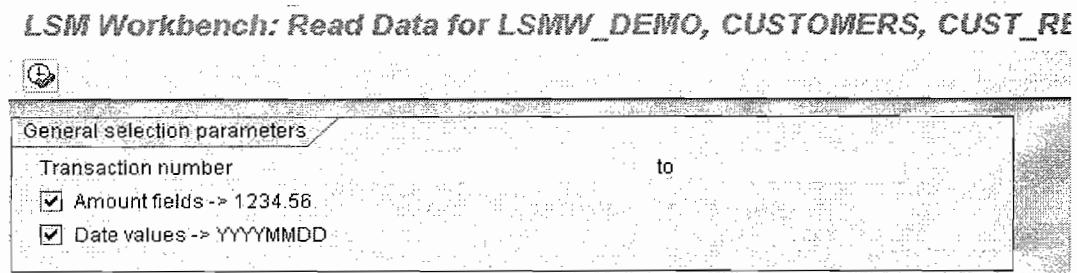
Step 8: Assign files

Execute step ‘Assign Files’ (**Figure 15**) and the system automatically defaults the filename to the source structure,

**Figure 15** Assign file to Source Structure

Step 9: Read data

In this step, LSMW reads the data (**Figure 16**) from the source file (from your PC’s local drive). You have the option to read only selected rows and convert data values to Internal format.

**Figure 16** Read Data

Step 10: Display read data

This step (**Figure 17**) is optional. If required, you can review the field contents for the rows of data read.

| File name | LSMW_DEMO_CUSTOMERS_CUST_REC.lsmw.read | |
|---------------|--|-------------|
| Structure | XDB02S | |
| Field name | Field text | Field value |
| CUSTOMER | CUSTOMER | 1000 |
| SALESORG | SALESORG | 1000 |
| DISTCHANNEL | DISTCHANNEL | 10 |
| DIVISION | DIVISION | 00 |
| SALESOFFICE | SALESOFFICE | 1010 |
| SALESGROUP | SALESGROUP | 110 |
| CUSTOMERGROUP | CUSTOMERGROUP | 01 |

Figure 17 Display Read Data**Step 11: Convert data**

This is the step that actually converts the source data (in source format) to a target format. Based on the conversion rules defined, source fields are mapped to target fields.

Step 12: Display Converted data

Again this is an optional step to view how the source data is converted to internal SAP format (**Figure 18**).

| Field name | LSMW_DEMO_CUSTOMERS_CUST_REC.lsmw.conv | |
|-------------|--|-------------|
| Structure | XDB02_REC | |
| Field name | Field text | Field value |
| TABNAME | Table name | XDB02_REC |
| TCODE | Transaction code | XDB02 |
| RF02D-KUNNR | Customer Number | 1000 |
| RF02D-VKORG | Sales Organization | 1000 |
| RF02D-VTWEG | Distribution Channel | 10 |
| RF02D-SPART | Division | 00 |
| RF02D-D0310 | SalesView | X |
| KNVV-VKBUR | Sales Office | 1010 |
| KNVV-VKGRP | Sales Group | 110 |
| KNVV-KDGRP | Customer Group | 01 |

Figure 18 Display Converted Data**Step 13: Create batch input session**

Once the source data is converted in an internal format, you can create a batch session to process updates (**Figure 19**).

LSM Workbench: Create Batch Input Session

File name (with path) LSMMW_DEMO_CUSTOMERS_CUST_REC.lsmwconv
transactions per BI session 1
Name of the BI session(s) CUST_REC
User id MITRESH
 Keep Batch input session(s)?

Figure 19 Create Batch Input Session**Step 14: Run Batch Input Session**

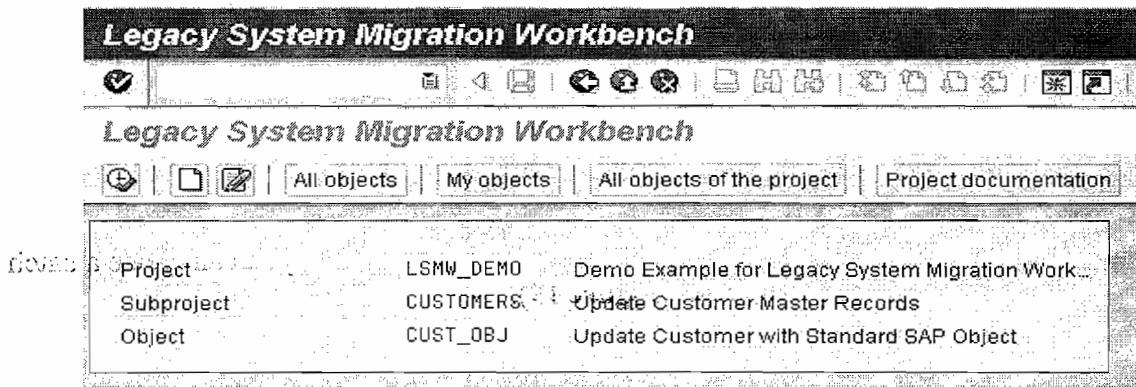
You can execute the BDC session by **Run Batch input session**. Executing a batch input session is a standard SM35 transaction for managing BDC sessions. Once you have successfully executed the batch input session, the customer master records are updated in the system. You can confirm this by viewing the customer master records (XD03).

Note! Browsing thru these 14 steps, you may get a feeling that this is a very lengthy and time-consuming activity. However, for the purposes of demonstration, I have made it detailed. Although it looks lengthy, actually it takes hardly few hours from start-to-finish! After playing around with few simple LSMW scripts, you will find it so easy to change and create more complex ones.

Demo Example 2**LSMW to Update Customer Master Records with Standard Object**

As an alternative to using ‘Transaction Recording’, you could also use a standard SAP object to update Customer Master Records. Business Object ‘0050’ is already pre-defined in the system with standard Batch Input Interface Program ‘RFBIDE00’.

Create an Object **CUST_OBJ** within Project as **LSMW_DEMO** and Subproject as **CUSTOMERS** as shown in **Figure 20**.

**Figure 20** LSMW Object with Standard SAP Object

Note! For the Demo example 2, I will list only those steps that are different from the first demo example.

Step 1: Maintain Object attributes

You will be updating the customer master records with the help of Standard Batch Input; therefore, choose radio-button **Standard Batch/Direct Input** as shown in **Figure 21**. Enter Object ‘0050’ for Customer Master records and default method ‘0000’ and click on Save.

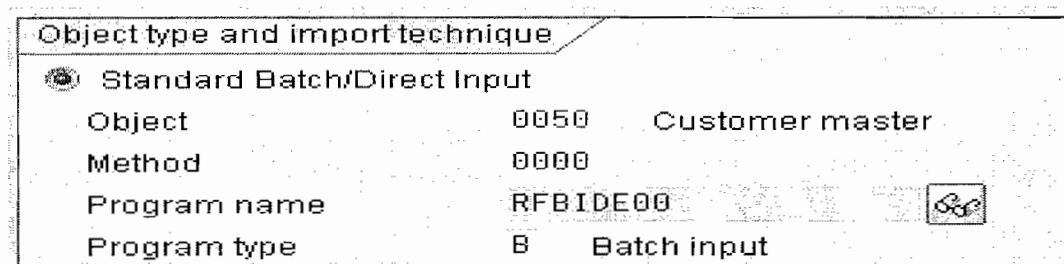
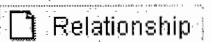


Figure 21 Standard Batch/Direct Input Object Attributes

Step 4: Maintain Structure Relations

Sales view of Customer Master is stored in table KNNV. Accordingly, you need to update structure BKNVV. However, in addition, the Standard Object ‘0050’ also requires updates to BGR00, BKN00 and BKNA1 structures. (If you do not maintain Structure relations for mandatory entries, you might get a message such as ‘Target structure BKNA1 needs a relation to a source structure’.)

Even though you don't want to update any fields in these structures, you need to create a relationship with source structures. In all, you need to create relationship for four target structures.

Create relationship between source structures XD02S with these target structures with icon ‘Create Relationship’ 

Keep Cursor on these four target structures and click on icon ‘Create Relation’ and structure relations are maintained as shown in **Figure 22**.

LSMW (Legacy System Migration Workbench)

We Never Compromise in Quality, Would You?



Figure 22 Structure Relation



Step 5: Maintain field mapping and conversion rules

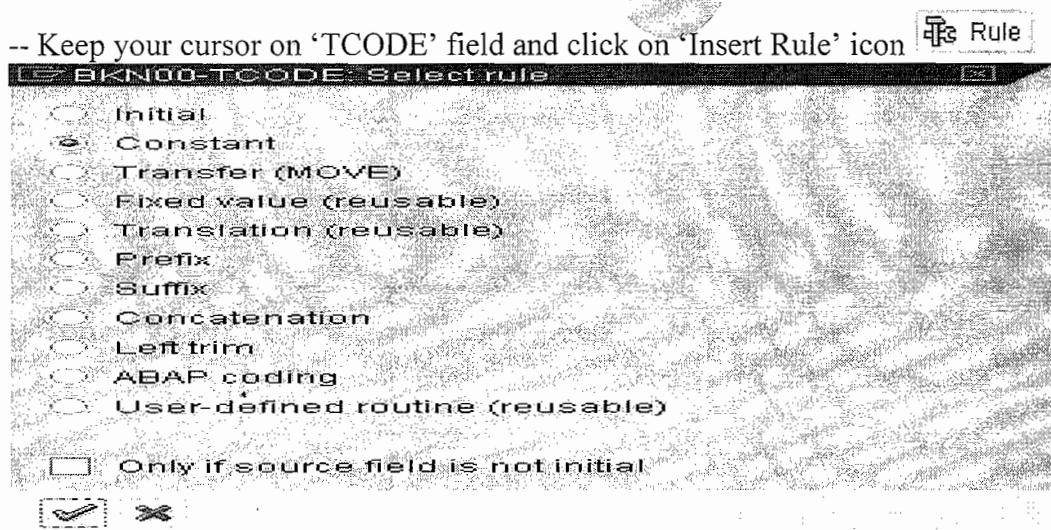


Figure 23 LSMW Conversion Rules

Choose radio button 'Constant' (Figure 23) to enter value 'XD02' transaction code.

-- Keep your cursor on field 'KUNNR' and click on 'Assign source field' icon



Choose source field 'Customer' from source structure 'XD02S'. (See Figure 24.)

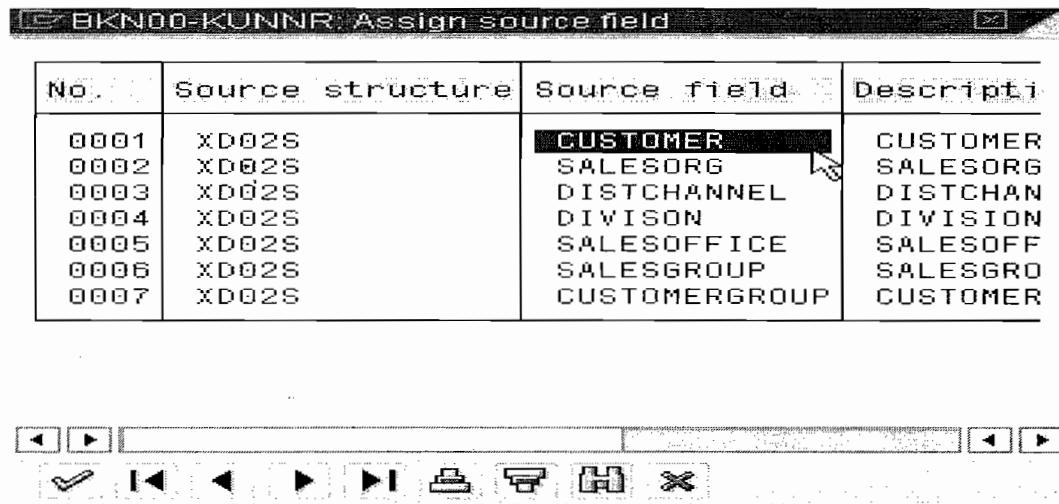
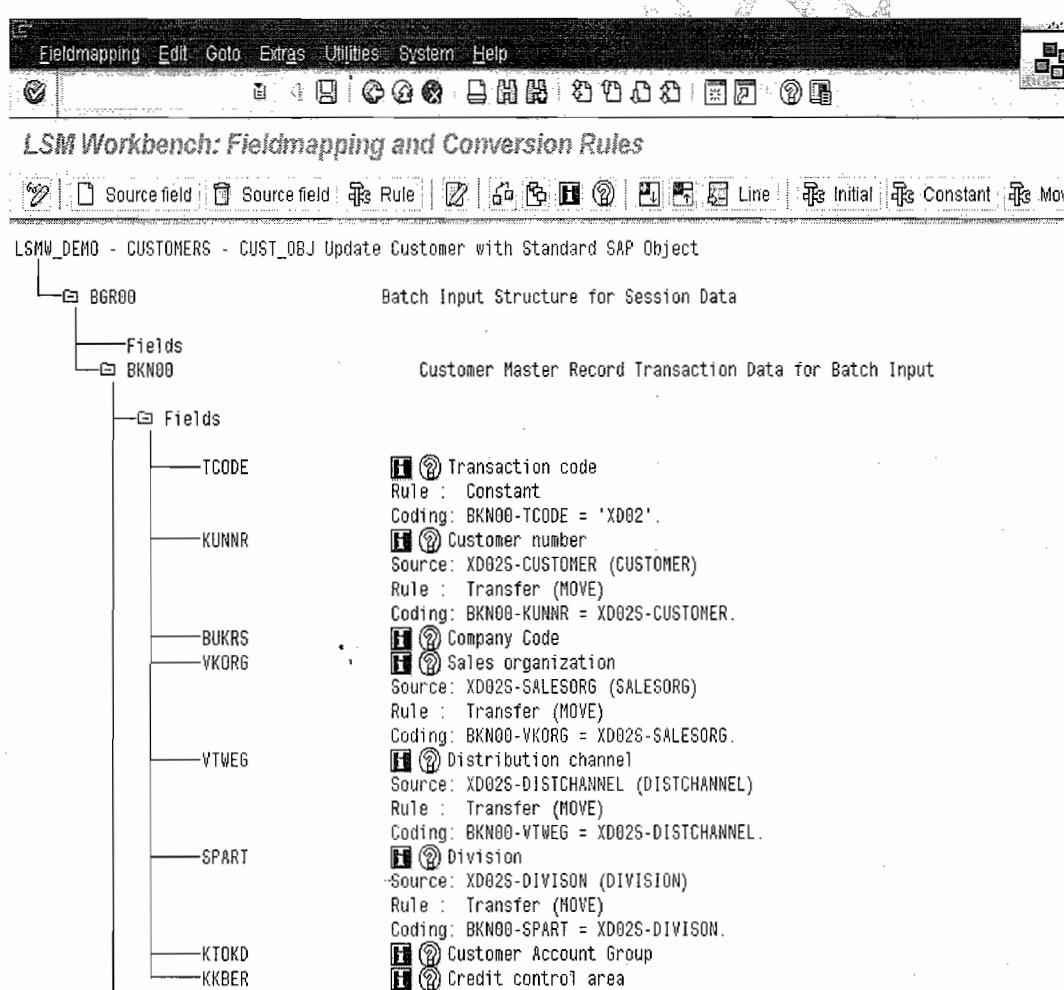


Figure 24 Assign Source fields

-- Similarly, choose source fields for Sales Organization, Distribution Channel, and Division. (See **Figure 25.**)



LSMW (Legacy System Migration Workbench)

We Never Compromise in Quality, Would You?

Figure 25 Field Mapping and Conversion Rules

-- Scroll down to structure BKNVV fields and assign source fields to three fields Sales Office, Sales Group, and Customer Group (**Figure 26**).

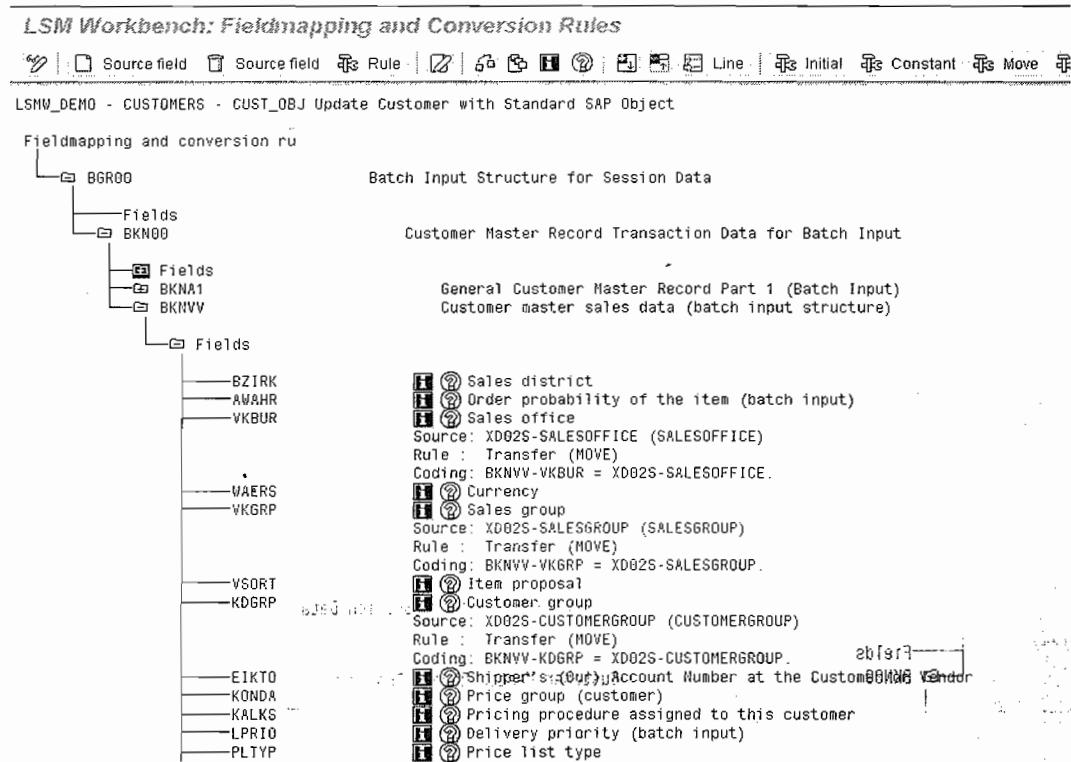


Figure 26 Field Mapping and Conversion Rules

Save and go back to main screen.

Step 12: Display Converted data

When you convert data, LSMW automatically converts into the appropriate structure layouts, as required by Standard program (RFBIDE00). (See **Figure 27**).

LSM Workbench: Display Converted Data

Field contents | Change display | Display color legend

| File name LSMW_DEMO_CUSTOMERS_CUST_0BJ.lsmw.conv | | |
|--|----------|--------------|
| Line | Struktur | Conts. |
| 1 | BGR00 | 0GUST_0BJ 80 |
| 2 | BKN00 | 1XD02 |
| 3 | BKNA1 | 2BKNA1 |
| 4 | BKNVV | 2BKNVV |

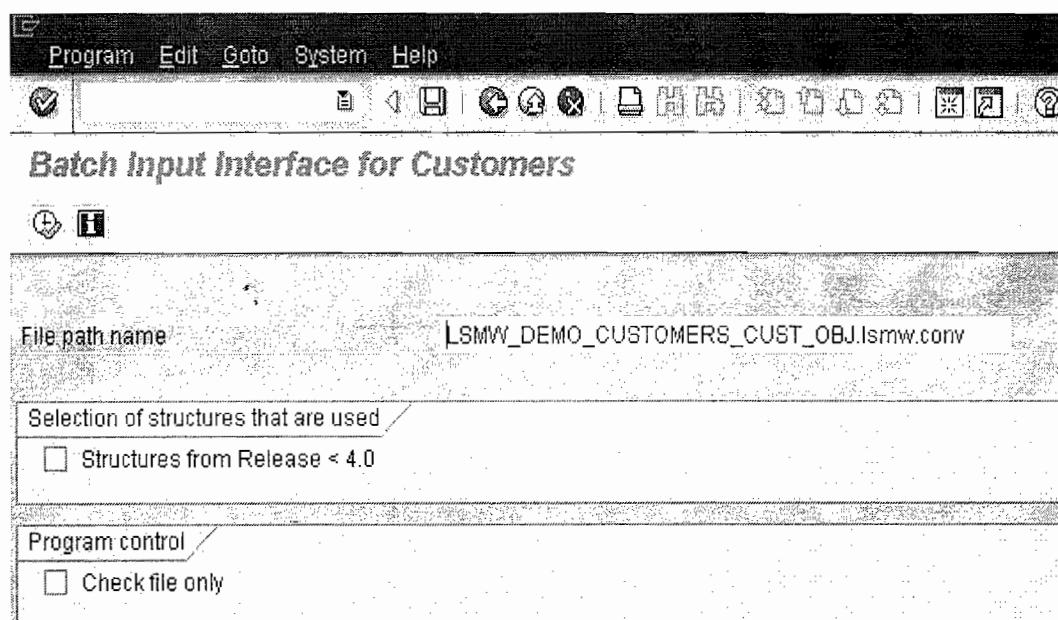
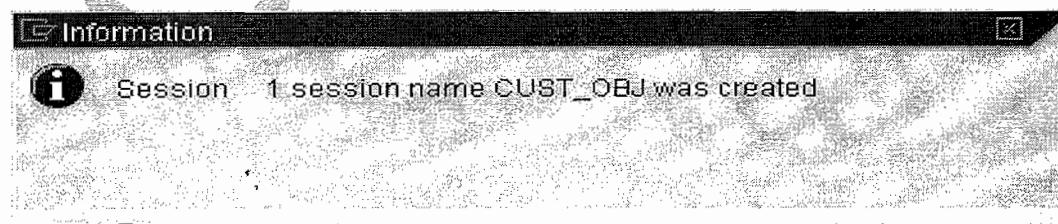
Figure 27 Converted data into multiple structures

Note that if you had only one record in source file, the converted file has four records.

Earlier, creating this input file, so that the standard interface program can read it, was a big nightmare, the primary reason being that it could have multiple record layouts. Even for a simple conversion with one input record, you would have to create this complex file with many record layouts. The advantage of LSMW is that it prepares these multi-layout files automatically.

Step 13: Create batch input session

Once source data is converted in internal format, you can create a BDC session to process the updates (**Figures 28 and 29**).

**Figure 28** Create BDC Session**Figure 29** BDC Session 'CUST_OBJ' created

Summary

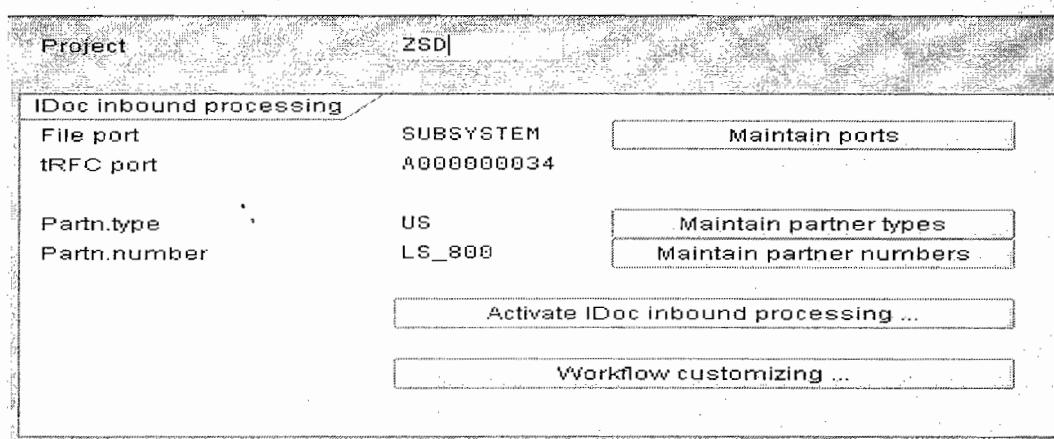
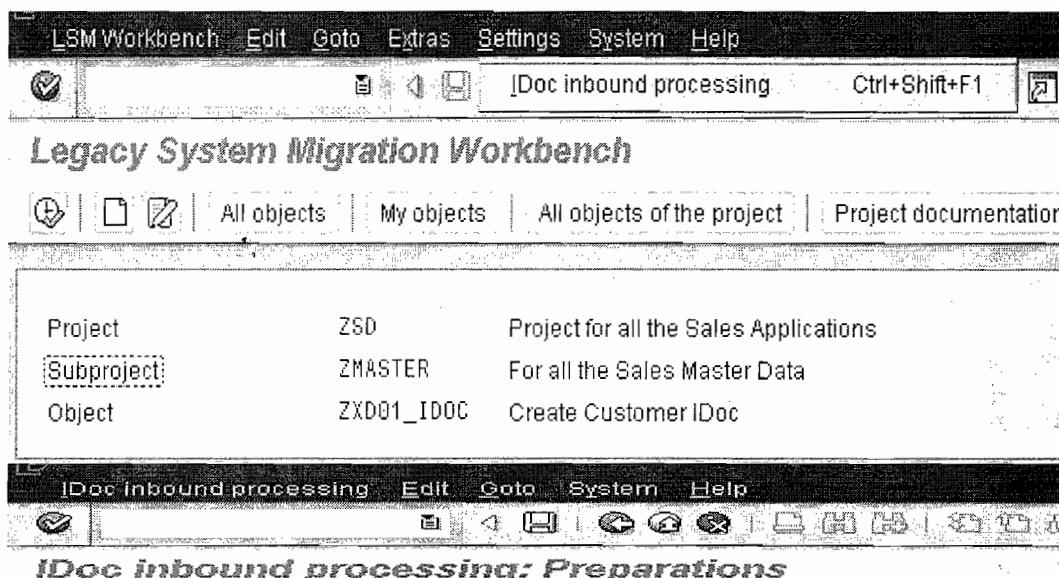
Once BDC session is processed successfully, SAP updates the customer master records with relevant changes. Review these specific customers (transaction code XD03) and confirm that the changes are correctly reflected in the master records.

Demo Example 3:**Create Customer Master Record through IDoc:**

Note: Preparations for IDOC inbound processing:

From the Initial Screen of LSMW:

- Choose settings -> IDOC inbound processing in LSMW

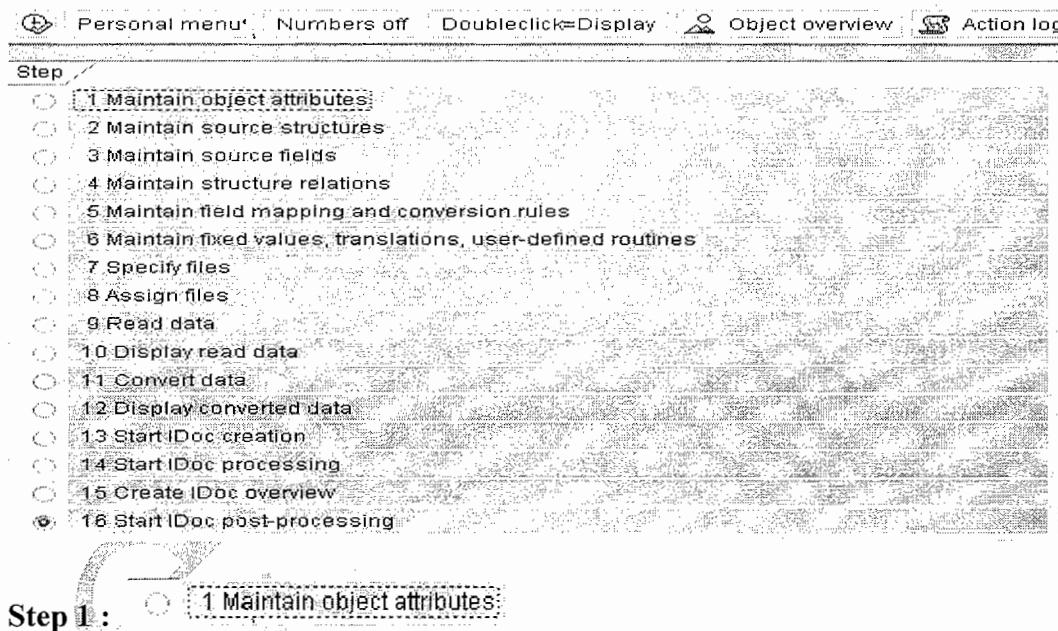


- Set up File port for file transfer, create port using WE21 or through [Maintain ports](#)

- Additionally set up RFC port for submitting data packages directly to function module IDoc_Inbound_Asyncronous, without creating a file during data conversion using WE21 or through Maintain ports.
- Setup partner type (SAP recommended 'US').
- Maintain partner number using WE20(Inbound Partner Profile). Where we Maintain the Message Type + Process Code.
- Activate IDOC inbound processing.
- Verify workflow customizing if exists.

Once the IDoc inbound Processing is Activated , SAVE it and Back to Initial Screen and Execute it.

LSM Workbench: ZSD, ZMASTER, ZXDO1_IDOC: Create Custom...



Step 1 : 1 Maintain object attributes

Provide the IDoc and Message Type Details.

IDoc (Intermediate Document)

Message type DEBMAS

Basic type DEBMAS05

Enhancement

SAVE it.

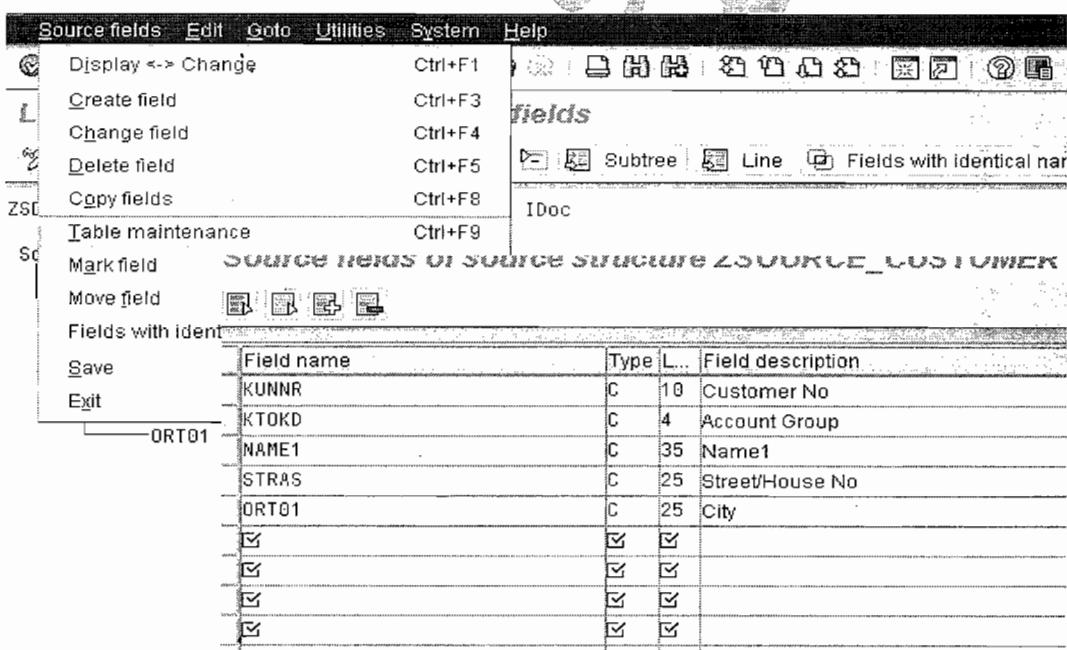
Step 2 : 2 Maintain source structures

Go in Change Mode , and Create a Structure Name and SAVE it. BACK to Initial Screen.



Step : 3 Maintain source fields

Select the Structure Name and Source Fields -> Table Maintenance and Provide the list of fields .

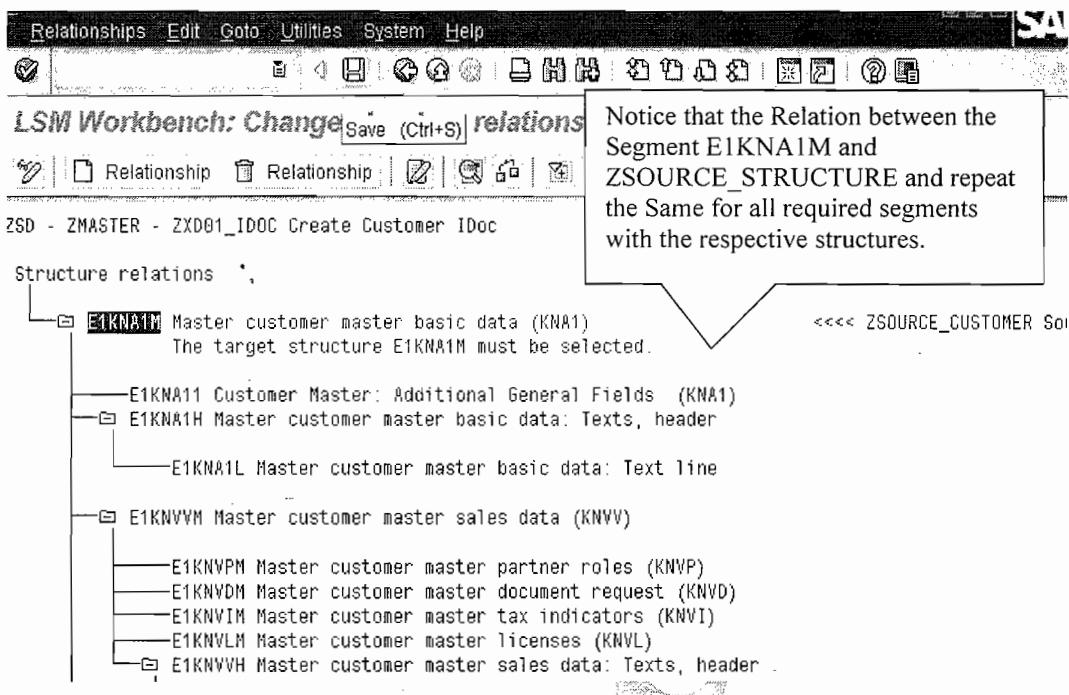


SAVE and BACK to Initial Screen.

LSMW (Legacy System Migration Workbench)

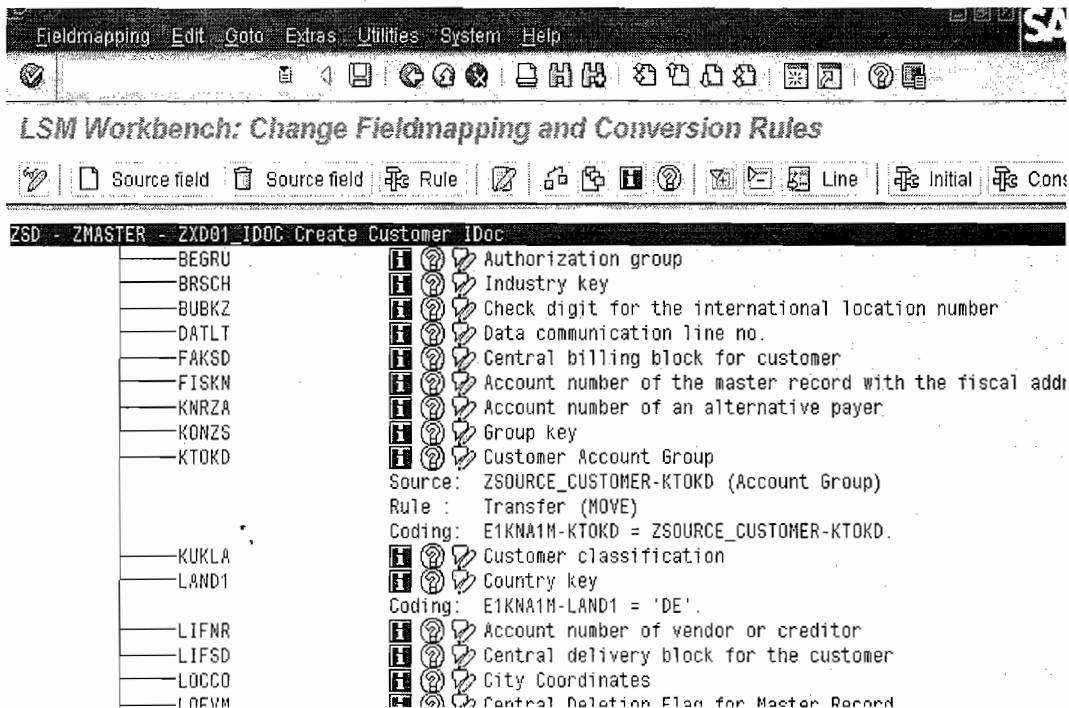
We Never Compromise in Quality, Would You?

Step : 4 Maintain structure relations

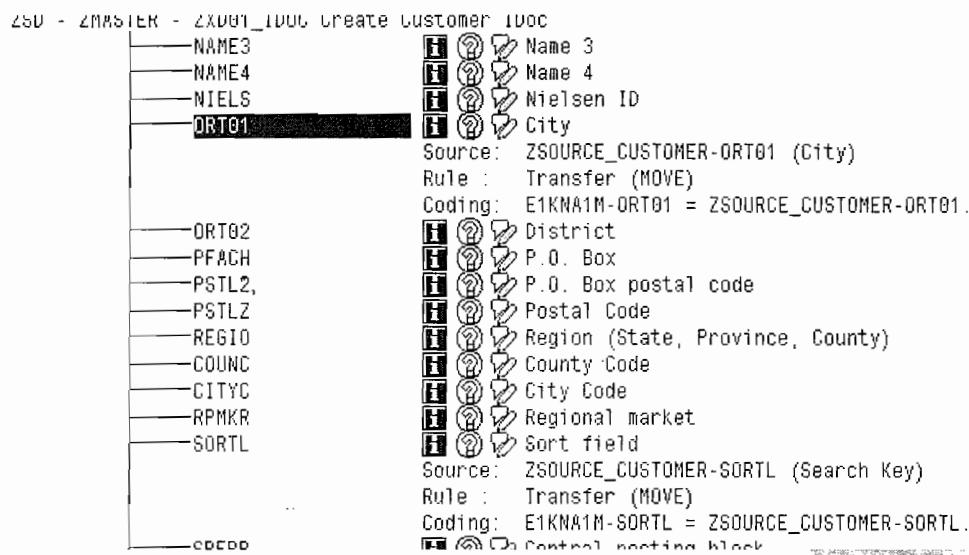


SAVE it and BACK to Initial Screen.

Step : 5 Maintain field mapping and conversion rules



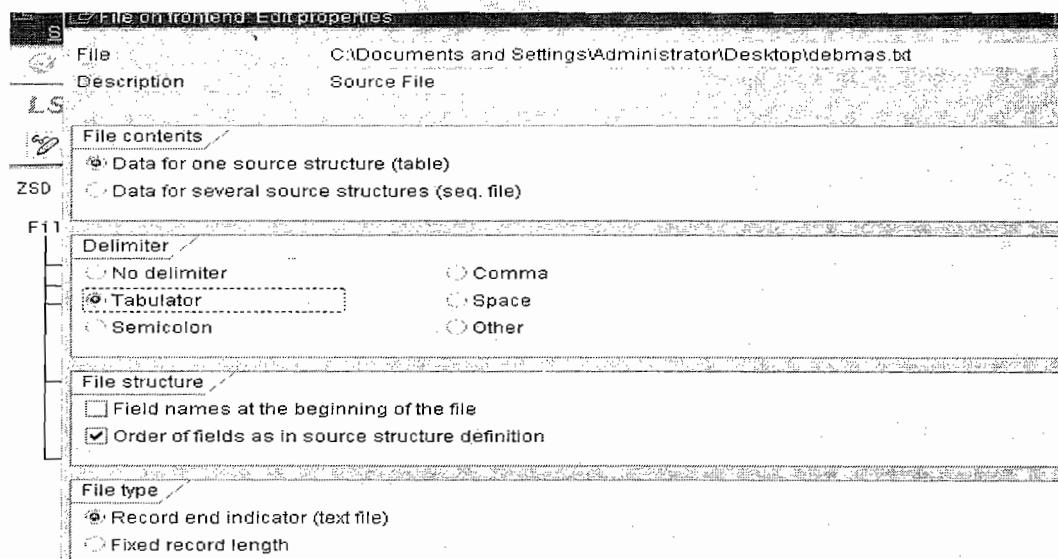
LSMW (Legacy System Migration Workbench)
We Never Compromise in Quality, Would You?



Select the Filed and Source Field if data comes from Source Structure or Click On Rule and Provide the Fixed Values.

Save and Back .

Step 7 Specify files :

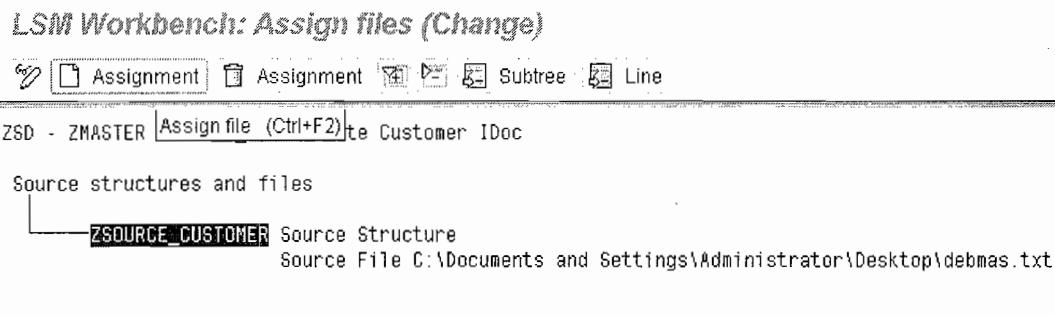


Enter , SAVE and BACK.

Step 8 Assign files :

Select the Source Structure and Create and Provide the File Name.

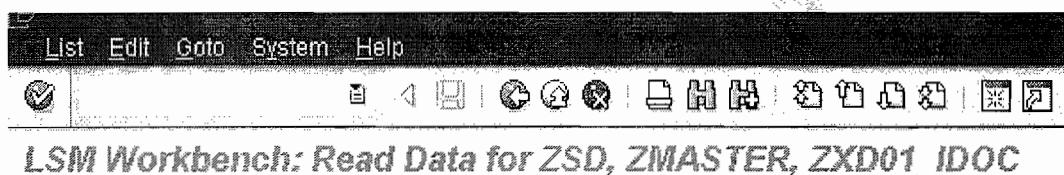
LSMW (Legacy System Migration Workbench)
We Never Compromise in Quality, Would You?



SAVE and BACK.

Step : 9 Read data:

Execute it,



| LSM Workbench: Read Data for ZSD, ZMASTER, ZXDO1_IDOC | | | |
|---|--|---------|-------------|
| 16.12.2005 - 23:37:40 | | | |
| Read file(s): | C:\Documents and Settings\Administrator\Desktop\debmas.txt | | |
| Written file: | ZSD_ZMASTER_ZXDO1_IDOC.lsmw.read | | |
| Source structure | Read | Written | Not written |
| ZSOURCE_CUSTOMER | 1 | 1 | 0 |
| Transactions read: | 1 | | |
| Records read: | 1 | | |
| Transactions written: | 1 | | |
| Records written: | 1 | | |

Step : 10 Display read data:

| File | ZSD_ZMASTER_ZXDO1_IDOC.lsmw.read | | | |
|------|----------------------------------|---------|---------------|--------------------------|
| Line | Struktur | Conts. | | |
| 1 | ZSMWZSD | ZMASTER | ZXDO1_IDOCC11 | 80020051216233739SAPUSER |
| 1 | ZSOURCE_CUSTOMER | 96778 | 0004emax | eMAX Technologies Begum |

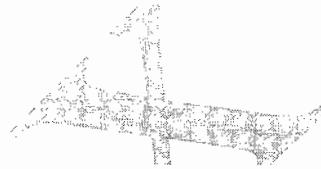
LSMW (Legacy System Migration Workbench)

We Never Compromise in Quality, Would You?

Double Click On it to Check the Contents

| File ZSD_ZMASTER_ZXD01_IDOC.lsmw.read | | |
|---------------------------------------|-----------------|-------------------|
| Structure ZSOURCE_CUSTOMER | | |
| Field name | Field text | Field value |
| KUNNR | Customer No | 96778 |
| KTOKD | Account Group | 0004 |
| SORTL | Search Key | emax |
| NAME1 | Name1 | eMAX Technologies |
| STRAS | Street/House No | Begumpet |
| ORT01 | City | HYD |

Step 11: Convert data Execute it.



LSM Workbench: Convert Data for ZSD, ZMASTER, ZXD01_IDOC

(1)

| | |
|--|-----------------------|
| General selection parameters | |
| Transaction number | to |
| Output type | |
| <input checked="" type="radio"/> Create file | <input type="radio"/> |
| <input type="radio"/> Create IDocs directly. | <input type="radio"/> |
| Number of IDocs per package | 50 |

Execute it.

LSM Workbench: Convert Data for ZSD, ZMASTER, ZXD01_IDOC

LSM Workbench: Convert Data for ZSD, ZMASTER, ZXD01_IDOC

16.12.2005 - 23:42:14

Read file: ZSD_ZMASTER_ZXD01_IDOC.lsmw.read
Written file: ZSD_ZMASTER_ZXD01_IDOC.lsmw.conv

Transactions read: 1
Records read: 1
Transactions written: 1
Records written: 2

LSMW (Legacy System Migration Workbench)

We Never Compromise in Quality, Would You?

Step 12 Display converted data

Execute it :

The screenshot shows the LSM Workbench interface. The title bar reads "LSM Workbench: Display Converted Data". The menu bar includes "Read Data", "Edit", "Goto", "System", and "Help". Below the menu is a toolbar with various icons. The main area displays a table titled "File ZSD_ZMASTER_ZXD01_IDOC.lsmw.conv". The table has two columns: "Line" and "Struktur". The "Line" column contains rows 1 and 2, which map to the "Struktur" column. Row 1 maps EDI_DC40 to EDI_DC40 with value 800. Row 2 maps E1KNA1M to E2KNA1M005 with value 800. The "Struktur" column also lists DEBMAS05 and 1 00000100000001.

Click on the row to check the Converted Contents into IDoc Format.

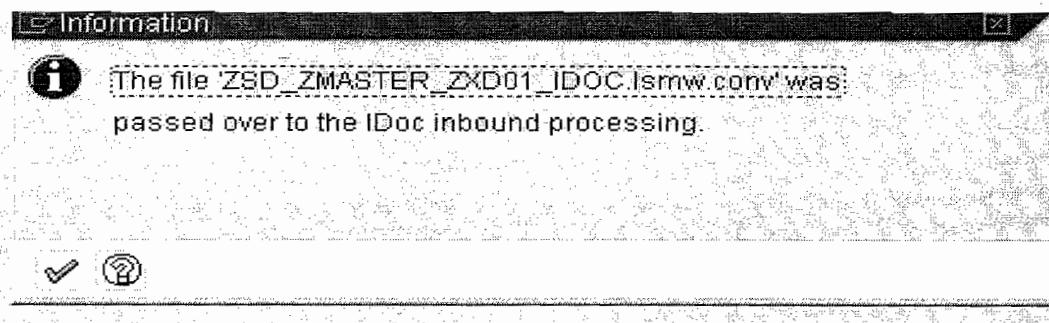
| File ZSD_ZMASTER_ZXD01_IDOC.lsmw.conv | | |
|---------------------------------------|-----------------------------------|--------------------------|
| External struct. E2KNA1M005 | | Internal struct. E1KNA1M |
| Field name | Field text | Field value |
| MANDT | Client | 800 |
| DOCNUM | IDoc number | 1 |
| SEGNUM | Segment number | 000001 |
| PSGNUM | Number of superior parent segment | 000000 |
| HLEVEL | Hierarchy level of SAP segment | 01 |
| MSGFN | Function | 009 |
| KUNNR | Customer number | 96778 |

Step 13 Start IDoc creation Execute it.

LSM Workbench: Create IDocs

The screenshot shows the LSM Workbench interface. The title bar reads "LSM Workbench: Create IDocs". The main area displays a table with two columns: "File with converted data" and "ZSD_ZMASTER_ZXD01_IDOC.lsmw.conv".

Execute it.



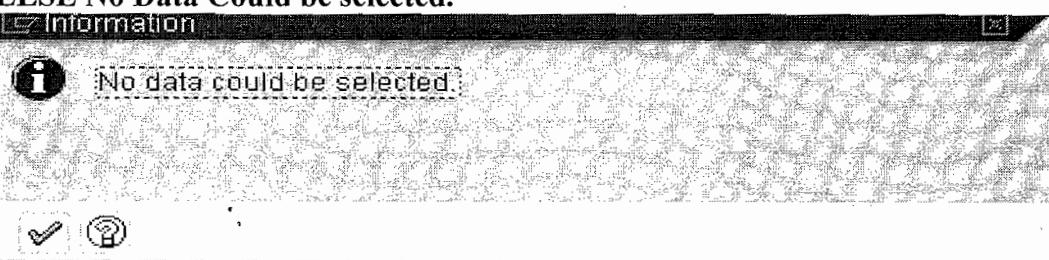
Step 14 Start IDoc processing :

Inbound Processing of IDocs Ready for Transfer

| IDoc selection | Parallel proc. | |
|---------------------|----------------|--|
| IDoc number | | to <input type="button" value="→"/> |
| Date created | 16.12.2005 | to <input type="button" value="→"/> |
| Time created | 23:48:09 | to <input type="button" value="→"/> <input type="button" value="→"/> |
| IDoc status | | to <input type="button" value="→"/> |
| Message type | DEBMAS | to <input type="button" value="→"/> |
| Message code | | to <input type="button" value="→"/> |
| Message function | | to <input type="button" value="→"/> |
| Sender partner type | US | to <input type="button" value="→"/> |
| | DEBMAS | |

Execute it.

Which Process all the Pending IDocs which are ready to transfer if any ELSE No Data Could be selected.



Step 15 Create IDoc overview**IDoc lists**

| | | | | |
|----------------------|------------|----|------------|--|
| Date created | 16.12.2005 | to | 16.12.2005 | |
| Time created | 23:55:42 | to | 24:00:00 | |
| Date last changed | | to | | |
| Time last changed | 00:00:00 | to | 24:00:00 | |
| Logical message type | DEBMAS | to | | |
| Message code | | to | | |
| Message function | | to | | |
| Test option | | to | | |
| Partner type | US | to | | |
| Partner function | | to | | |
| Partner number | LS_800 | to | | |
| Partner port | | to | | |

IDoc Lists

Direction-> Status Number
Selected IDocs 1
Inbox 1
53 Application document posted:

Combined list

| Status | Mess.type | Var. | Fct. | No. of IDocs | Message description |
|--------|-----------|------|------|--------------|-----------------------------|
| 53 | DEBMAS | | | 1 | Customer master data distrl |

Step 16 Start IDoc post-processing

This is helpful to process the failed IDocs again.

Processing inbound IDocs



- Try posting again:
51: Error: Application document not posted
- Resubmit after ALE/EDI error:
56: IDoc with errors added
61: Processing despite syntax error (inbound)
63: Error passing IDoc to application
65: Error in ALE service
- Ignore syntax error:
60: Error during syntax check of IDoc (inbound)
- Reset status:
62: IDoc passed to application

Select the Corresponding IDoc Status and Re-process it.

Theory :

- 1) Explain Different Importing Techniques in LSMW?
- 2) Explain the Steps required in each type of Importing Technique?

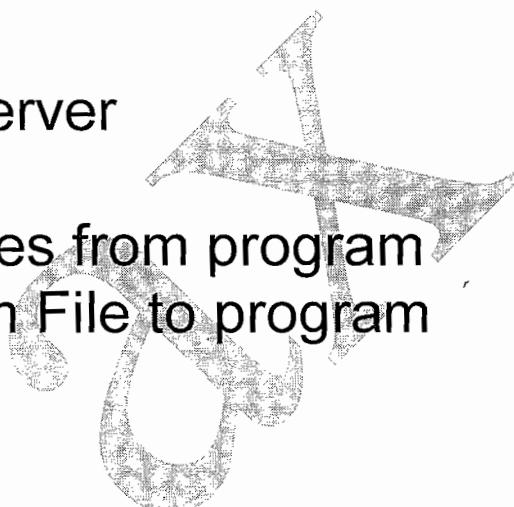
Exercises:

- 1) Use LSMW for the migration of Bank Master Data from legacy system to R/3.
- 2) Use LSMW for migration of G/L Account Master data from legacy system to R/3.
- 3) Use LSMW upload the Customer Master Data using IDoc ?
- 4) Use BAPI in LSMM to Upload any Master Data?
- 5) Use Direct Input in LSMW to Upload any Master Data?

05.working with Files

1. Files on presentation Server

- a. Writing Data to Files from program
- b. Reading Data from File to program
- i. Text Files
- ii. Excel Files
- iii. **Files with different record structures**



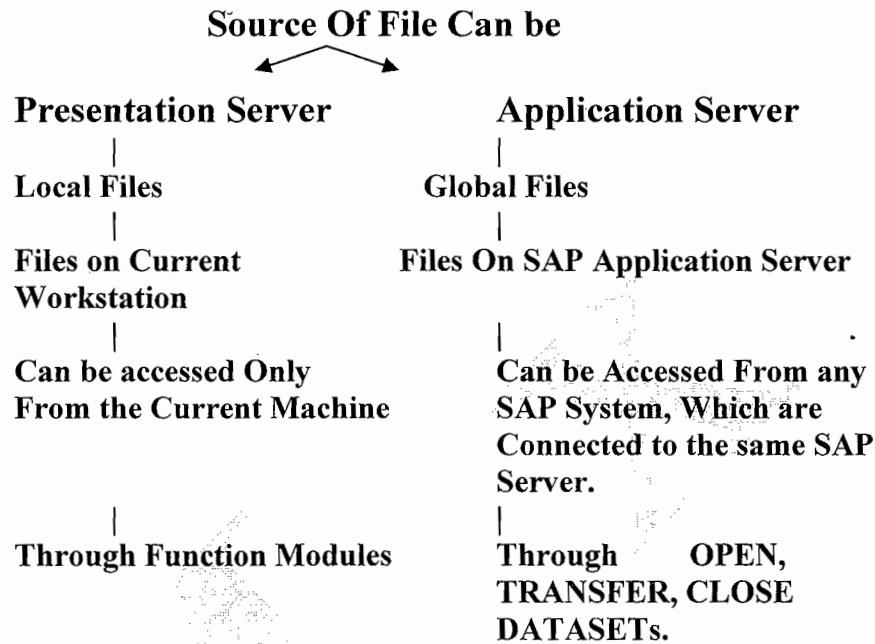
2. Files On application Server

- a. Writing Data to Files from Program
- b. Reading Data from File to Program

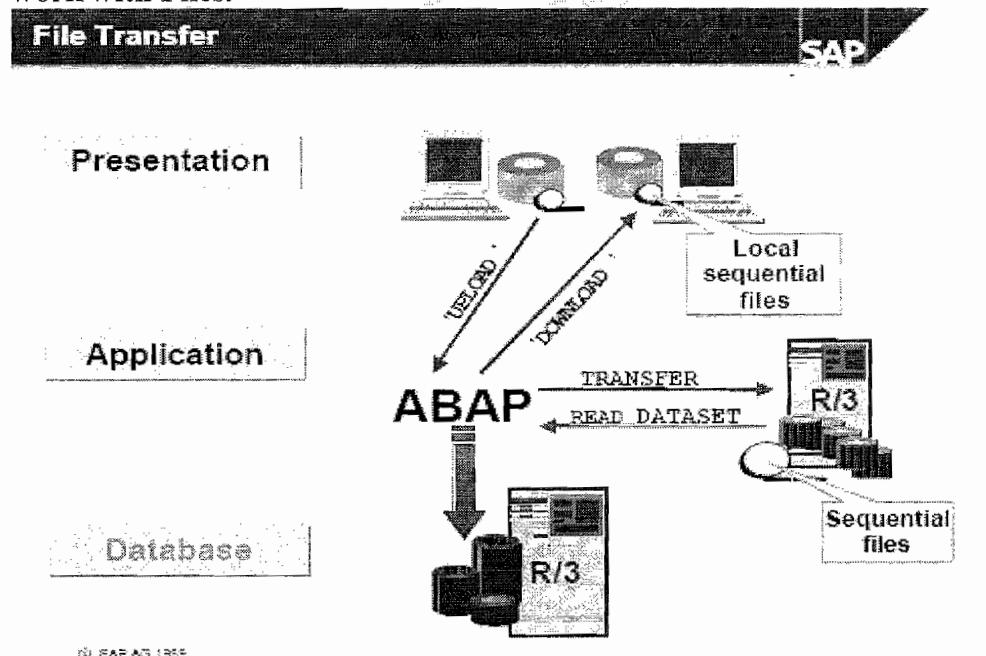


Working With Files :**DAY-1**

Note: The Procedure to work with files depends on the Source Of the FILE.



Note: Go through this diagram , Only after Completion Of the Procedure to work with Files.



Working with Presentation Server Files :

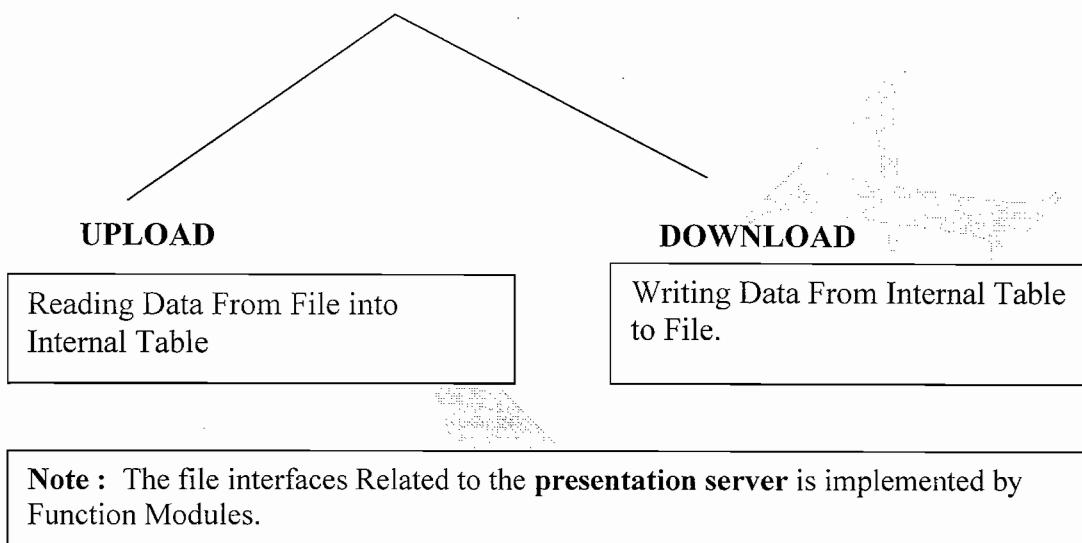
Files on Presentation server / workstation are LOCAL FILES.

HANDLING OF LOCAL FILES:

Note: Either Writing Data to File

OR

Reading Data from File is Always through **Internal Table**.



Note : The file interfaces Related to the **presentation server** is implemented by Function Modules.

UPLOAD (Upload Data from file to Internal Table) :

Note : We Can use the below Function Modules for the Same

UPLOAD
Or
WS_UPLOAD
OR
GUI_UPLOAD.

Note :

Both UPLOAD and WS_UPLOAD are Out Dated and currently we Use GUI_UPLOAD.

Local files are processed using GUI_UPLOAD and GUI_DOWNLOAD functions. The local files are brought into ABAP/4 memory using these functions. All the records of the file are UPLOADED into an internal table in one shot. Similarly, all records are DOWNLOADED in one shot from an internal table to a local file.

Requirement :

Upload the Profit Center Data from the TEXT File and Display the Same.

```
*****  
* PROGRAM : ZDEMO_UPLOAD_PROFIT_CENTER_DATA *  
* AUTHOR : GANPATI . ADIMULAM *  
* PURPOSE : BDC PROGRAM TO UPLOAD ALL THE PROFIT *  
* CENTER MASTER DATA INTO SAP *  
* REFERENCE : NA *  
* COPIED FROM : NA *  
* TRAPORT REQUEST NO : C11D2K9001 *  
*****
```

```
REPORT ZDEMO_UPLOAD_PROFIT_CENTER_DATA .
```

```
DATA : BEGIN OF WA_DATA,  
        PRCTR TYPE PRCTR, "Profit Center  
        DATAB TYPE DATAB, "START DATE  
        DATBI TYPE DATBI, "END DATE  
        KTEXT TYPE KTEXT, "NAME  
        VERAK TYPE VERAK, "PERSON RESPONSIBLE  
        KHINR TYPE KHINR, "HIERARCHY AREA  
      END OF WA_DATA.
```

```
DATA : IT_DATA LIKE TABLE OF WA_DATA.
```

```
*****  
* S T A R T - O F - S E L E C T I O N *  
*****
```

```
START-OF-SELECTION
```

```
CALL FUNCTION 'GUI_UPLOAD'  
  EXPORTING  
    FILENAME           = 'C:\KE51.TXT'  
    HAS_FIELD_SEPARATOR = 'X' "TAB DELIMITER  
  TABLES  
    DATA_TAB          = IT_DATA
```

```
IF SY-SUBRC = 0.  
MESSAGE S009(ZDEMO) WITH 'C:\KE51.TXT'.  
ENDIF.
```

```
*****  
* E N D - O F - S E L E C T I O N *  
*****
```

```
END-OF-SELECTION.
```

```
IF NOT IT_DATA IS INITIAL.  
  WRITE : /5 'PROFIT CENTER', 15 'START DATE', 25 'END DATE' ,  
         35 'NAME', 50 'PERSON.RESP', 70 'HIER.AREA'.  
  ULINE.  
  LOOP AT IT_DATA INTO WA_DATA.  
  WRITE : /5 WA_DATA-PRCTR, "Profit Center  
         15 WA_DATA-DATAB, "START DATE
```

File Handling in SAP

We Never Compromise Quality, Would You?

```
25 WA_DATA-DATBI, "END DATE
35 WA_DATA-KTEXT, "NAME
50 WA_DATA-VERAK, "PERSON RESPONSIBLE
70 WA_DATA-KHINR. "HIERARCHY AREA
CLEAR WA_DATA.
ENDLOOP.
ELSE.
WRITE : / 'NO DATA IS uploaded FROM THE FILE'.
ENDIF.
```

Program Execution :

Execute the Program : ZDEMO_UPLOAD_PROFITCENTER_DATA

Input File :

| File is with Field Separator TAB Space . | | | |
|--|----------|----------|-----------|
| 16397 | 13022001 | 30122007 | Revenues |
| 14567 | 13022005 | 30122008 | Furniture |

OutPut :

| Demo to Upload the Profit Center Data | | | | |
|---------------------------------------|----------------------|-----------|--------------|--------------|
| PROFIT CEN | START DATE | END DATE | NAME | PERSON. RESP |
| 16397 | 01.20.130207.20.3012 | Revenues | Mr. Ganapati | 1 |
| 14567 | 05.20.130208.20.3012 | Furniture | Mr. Prakash | 1 |

File is Successfully Uploaded and it is displayed the same contents from the File. Check the Contents with the Input File.

File Is Successfully Downloaded to C:\KE51.TXT

Working with Files Other than TAB Separator & with Multiple Record Types:

Many times the input files will have records with different structures. In all such cases each record will have a field (usually the first) which identifies the record (ex: 'H' for header, 'D' for detail or '10' & '20' etc). Text file with multiple record types would be like this

```
10 : 21879 : rel : fresh : rel : Ameerpet, HYD, IN, EN
20 : 21879 : DE : 20050000 : 01122334
20 : 21879 : DE : 23022200 : 1122334455
20 : 21879 : DE : 23984899 : 2233445566
```

Processing Text file with multiple record types :

Note 1 : To Process such files, it is necessary to first read the record into a character field that is at least of minimum of the lengths of the different structure in the file. If 'H' type record is 30 char long (including the record identifier) and 'D' type is 40 long (including the record identifier), then this character field should be at least 40 char long.

Note: The Function Module GUI_DOWNLOAD can identify the Field separator TAB through the Importing Parameter HAS_FIELD_SEPARATOR = 'X' but not for other Separators.

Steps :

1. Since the structure(Length) of each record is different ,
UPLOAD the Data into an ITAB with Generic structure i.e
Each record as a Continuous TEXT Field Using GUI_UPLOAD.
2. Since we need to identify the data Uniquely Field by Field,data , again we have to split the above internal table according the file structure by considering the SEPARATOR.

Input File :

```

X:\D:\Kunbktxt\Notepad
File Edit Format View Help
10.21879,rel,fresh,rel,Ameerpet,HYD,IN,EN
20.21879,DE,20050000,01122334
20.21879,DE,23022200,1122334455
20.21879,DE,23984899,2233445566

```

The file contains both Customer Master General Data and Bank Details.

In each record , the first two characters are to identify the type of the Record.

i.e 10 for Customer Master General and
20 for Customer Bank Details.

Customer General Data File Structure

| Field Position | SAP Field Name | Length | Description | Example |
|----------------|----------------|--------|----------------|------------|
| 1 | ID | 2 | Identification | 10(Header) |
| 2 | KUNNR | 10 | Customer No | 21169 |
| 3 | NAME1 | 35 | Name | Rel.Fresh |
| 4 | SORTL | 10 | Search Key | Rel |
| 5 | STRAS | 35 | Street | Ameerpet |
| 6 | ORT01 | 35 | City | HYD |
| 7 | LAND1 | 4 | Country Key | IN |
| 8 | SPRAS | 2 | Language Key | EN |

Customer Master Bank Details:

| Field Position | SAP Field Name | Length | Description | Example |
|----------------|----------------|--------|------------------|------------------|
| 1 | ID | 2 | Identification | 20(Bank Details) |
| 2 | KUNNR | 10 | Customer No | 21169 |
| 3 | BANKS | 3 | Bank Country Key | DE |
| 4 | BANKL | 15 | Bank Key | 50013050 |

File Handling in SAP

We Never Compromise Quality, Would You?

| | | | | |
|---|-------|----|--------------------|-----------|
| 5 | BANKN | 18 | Bank Account No | 123456789 |
|---|-------|----|--------------------|-----------|

Note : This File structure is helpful to declare the corresponding Internal table and also to SPLIT the file according to the file structure.

Program to Upload the above file data into Corresponding ITABS and Display :

```
*****
* PROGRAM   : ZDEMO_UPLOAD_DATA_FROM_FILE          *
* AUTHOR    : GANAPATI . ADIMULAM                  *
* PURPOSE   : BDC PROGRAM TO UPLOAD ALL            *
*             THE CUSTOMER MASTER DATA INTO SAP      *
* REFERENCE : NA                                    *
* COPIED FROM : NA                                *
* TRAPORT REQUEST NO : C11D2K9001                 *
*****
```

```
REPORT ZDEMO_UPLOAD_DATA_FROM_FILE.
```

```
DATA : BEGIN OF WA_FILE,
      TEXT(150) TYPE C,
      END OF WA_FILE.
```

```
DATA IT_FILE LIKE TABLE OF WA_FILE. "ITAB with Generic File Structure
```

```
DATA : BEGIN OF WA_KNA1,
      KUNNR TYPE KUNNR, "CUSTOMER
      NAME1 TYPE NAME1, "NAME
      SORTL TYPE SORTL, "SEARCH TERM
      STRAS TYPE STRAS, "STREET
      ORT01 TYPE ORT01, "CITY
      LAND1 TYPE LAND1, "COUNTRY
      SPRAS TYPE SPRAS, "LANGU
      END OF WA_KNA1.
```

```
DATA IT_KNA1 LIKE TABLE OF WA_KNA1.
```

```
DATA : BEGIN OF WA_KNBK,
      KUNNR TYPE KUNNR, "CUSTOMER
      BANKS LIKE KNBK-BANKS, "BANK COUNTRY
      BANKL LIKE KNBK-BANKL, "BANK KEY
      BANKN LIKE KNBK-BANKN, "ACCNO
      END OF WA_KNBK.
```

```
DATA IT_KNBK LIKE TABLE OF WA_KNBK.
```

```
DATA V_ID(2).
```

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

START-OF-SELECTION.

```
*UPLOAD
CALL FUNCTION 'GUI_UPLOAD'
  EXPORTING
    FILENAME           = 'C:\XD01_KNBK.TXT'
    TABLES             =
      DATA_TAB          = IT_FILE
```

1) Upload the File into ITAB with Generic File Structure.

```
*SPLIT THE FILE ITAB AND MOVE TO CORRESPONDING ITABS.
LOOP AT IT_FILE INTO WA_FILE.
```

```
IF WA_FILE+0(2) = '10'. "GENERAL DATA
```

```
  SPLIT WA_FILE AT ','  
    INTO V_ID  
      WA_KNA1-KUNNR  
      WA_KNA1-NAME1  
      WA_KNA1-SORTL  
      WA_KNA1-STRAS  
      WA_KNA1-ORT01  
      WA_KNA1-LAND1  
      WA_KNA1-SPRAS .
```

2) SPLIT the Internal Table According the File Structure.

```
APPEND WA_KNA1 TO IT_KNA1.  
ELSEIF WA_FILE+0(2) = '20'. "BANK DATA  
  SPLIT WA_FILE AT ','  
    INTO V_ID  
      WA_KNBK-KUNNR  
      WA_KNBK-BANKS  
      WA_KNBK-BANKL  
      WA_KNBK-BANKN.
```

```
APPEND WA_KNBK TO IT_KNBK.
```

```
ENDIF.
```

```
ENDLOOP.
```

```
WRITE : / 'CUSTOMER MASTER GENERAL DATA'.
ULINE.
```

```
LOOP AT IT_KNA1 INTO WA_KNA1.
```

```
  WRITE : / WA_KNA1-KUNNR,  
        WA_KNA1-NAME1,  
        WA_KNA1-SORTL,  
        WA_KNA1-STRAS,  
        WA_KNA1-ORT01,  
        WA_KNA1-LAND1,  
        WA_KNA1-SPRAS .
```

```
CLEAR WA_KNA1.  
ENDLOOP.
```

```
SKIP 2.
```

```
WRITE : / 'CUSTOMER MASTER BANK DATA'.
ULINE.
```

```
LOOP AT IT_KNBK INTO WA_KNBK.
```

```
  WRITE : / WA_KNBK-KUNNR,  
        WA_KNBK-BANKS,  
        WA_KNBK-BANKL,  
        WA_KNBK-BANKN.
```

File Handling in SAP

We Never Compromise Quality, Would You?

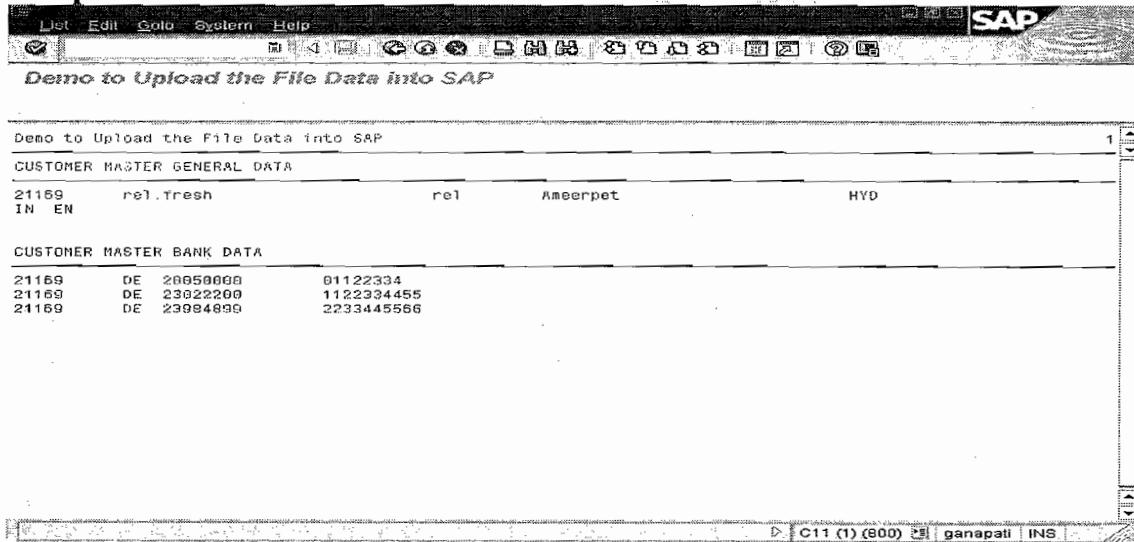
CLEAR WA_KNBK.
ENDLOOP.

Execute the Program :

Input File :

XDO1_knbk.txt - Notepad
File Edit Format View Help
10,21169,rel,fresh,rel,Ameerpet,HYD,IN,EN
20,21169,DE,20050000,01122334
20,21169,DE,23022200,1122334455
20,21169,DE,23984899,2233445566

Output :



Working with Uploading Excel File :

DAY-2

Note1 :

Uploading Excel file is not through the Function Module
GUI_UPLOAD, We have another function Module for the Same i.e
ALSM_EXCEL_TO_INTERNAL_TABLE

**Note 2 : The output of the Function Module is not the Direct Data , it is
With ROW,COL and Value.**

Example File :

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|----|-------|-------|-----|-----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A1000 | AGTCH | HYD | IND | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 2000 | EMAX | HYD | IND | | | | | | | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 13 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 14 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 15 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 17 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 18 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 19 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 21 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 22 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 23 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 24 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 25 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 26 | | | | | | | | | | | | | | | | | | | | | | | | | | |

The Output of the Function Module

ALSM_EXCEL_TO_INTERNAL_TABLE is

An an Internal Table with fields ROW ,COL and VALUE ie

| ROW | COL | VALUE | |
|-----|-----|-------|--|
| 1 | 1 | 1000 | |
| 1 | 2 | AGTCH | |
| 1 | 3 | HYD | |
| 1 | 4 | IND | |
| 2 | 1 | 2000 | |
| 2 | 2 | EMAX | |
| 2 | 3 | HYD | |
| 2 | 4 | IND | |

We need to Move all the Details of the First row as First Record into One More Internal Table

We need Move all the Details of the 2nd row as 2nd Record into One More Internal Table

Note : So Next Step is Move the Data to another Internal table according to the file Structure i.e

Col1 BUKRS

Col2 BUTXT

Col3 ORT01

Col4 LAND1

Requirement:

Develop a Program to accept the File according to the File Format selected

AND Upload the data into ITAB and Display the Same

Selection Criteria

File Name & Path

DownLoad to Presentation Server

Text File

Excel File

Source Code of the Program:

```
REPORT ZDEMO_DOWNLOAD_TO_PRE_SERVER .  
  
*****  
* PROGRAM Name : ZDEMO_UPLOAD_FROM_PRE_SERVER *  
* AUTHOR : GANAPATI . ADIMULAM *  
* PURPOSE : Program to Upload the data from*  
* EXCEL/TXT Files based on the *  
* User Selection *  
* REFERENCE OBJ : NA *  
* TRANSPORT REQUEST: C11K900004 *  
*****  
*MODIFICATION LOG : *  
*****  
* MOD - 0001 *  
* MOD - 0002 *  
  
*****  
REPORT ZDEMO_UPLOAD_FROM_PRE_SERVER.  
DATA : BEGIN OF WA_T001_TXT,  
        BUKRS TYPE BUKRS, "Company Code"  
        BUTXT TYPE BUTXT, "Company Name"  
        ORT01 TYPE ORT01, "City"  
        LAND1 TYPE LAND1, "Country Key"  
        WAERS TYPE WAERS, "Currency Key"  
    END OF WA_T001_TXT.  
  
DATA : IT_T001_TXT LIKE STANDARD TABLE OF WA_T001_TXT.  
DATA WA_T001_EXCEL LIKE WA_T001_TXT.  
  
DATA : IT_T001_EXCEL LIKE STANDARD TABLE OF WA_T001_EXCEL.  
  
DATA : IT_RETURN LIKE STANDARD TABLE OF ALSMEX_TABLINE,  
      WA_RETURN LIKE ALSMEX_TABLINE,  
      WA RETURNS LIKE ALSMEX_TABLINE.  
  
DATA : V_FILE TYPE STRING,  
      V_MASK(10) TYPE C.  
  
SELECTION-SCREEN BEGIN OF BLOCK B1 WITH FRAME TITLE TEXT-001.  
PARAMETER : P_FILE LIKE RLGRAP-Filename OBLIGATORY.  
SELECTION-SCREEN END OF BLOCK B1.  
  
SELECTION-SCREEN BEGIN OF BLOCK B2 WITH FRAME TITLE TEXT-002.  
PARAMETER : R_TXT RADIobutton GROUP G1 DEFAULT 'X' USER-COMMAND C1,  
            R_EXCEL RADIobutton GROUP G1.  
SELECTION-SCREEN END OF BLOCK B2.  
  
AT SELECTION-SCREEN ON VALUE-REQUEST FOR P_FILE.  
  
*Fill the Mask based On the Selection  
IF R_TXT = 'X'.  
  V_MASK = '*TXT'.  
ELSEIF R_EXCEL = 'X'.  
  V_MASK = '*XLS'.
```

```
ENDIF.  
  
CALL FUNCTION 'KD_GET_FILENAME_ON_F4'  
  EXPORTING  
    PROGRAM_NAME = SY-CPROG  
    MASK         = V_MASK  
  CHANGING  
    FILE_NAME   = P_FILE.
```

AT SELECTION-SCREEN.

```
TRANSLATE P_FILE TO UPPER CASE.  
IF R_TXT = 'X'.
```

```
*Check the File ends with TXT or Not  
  SEARCH P_FILE FOR '*TXT'.  
  IF SY-SUBRC <> 0.  
    MESSAGE E004(ZDEMO).  
  ENDIF.
```

```
ELSEIF R_EXCEL = 'X'.  
*Check the File ends with XLS or Not  
  SEARCH P_FILE FOR '*XLS'.
```

```
  IF SY-SUBRC <> 0.  
    MESSAGE E005(ZDEMO).  
  ENDIF.
```

ENDIF.

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

```
*DOWN LOAD BASED ON THE BUTTON SELECTED  
V_FILE = P_FILE.
```

```
IF R_TXT = 'X'.
```

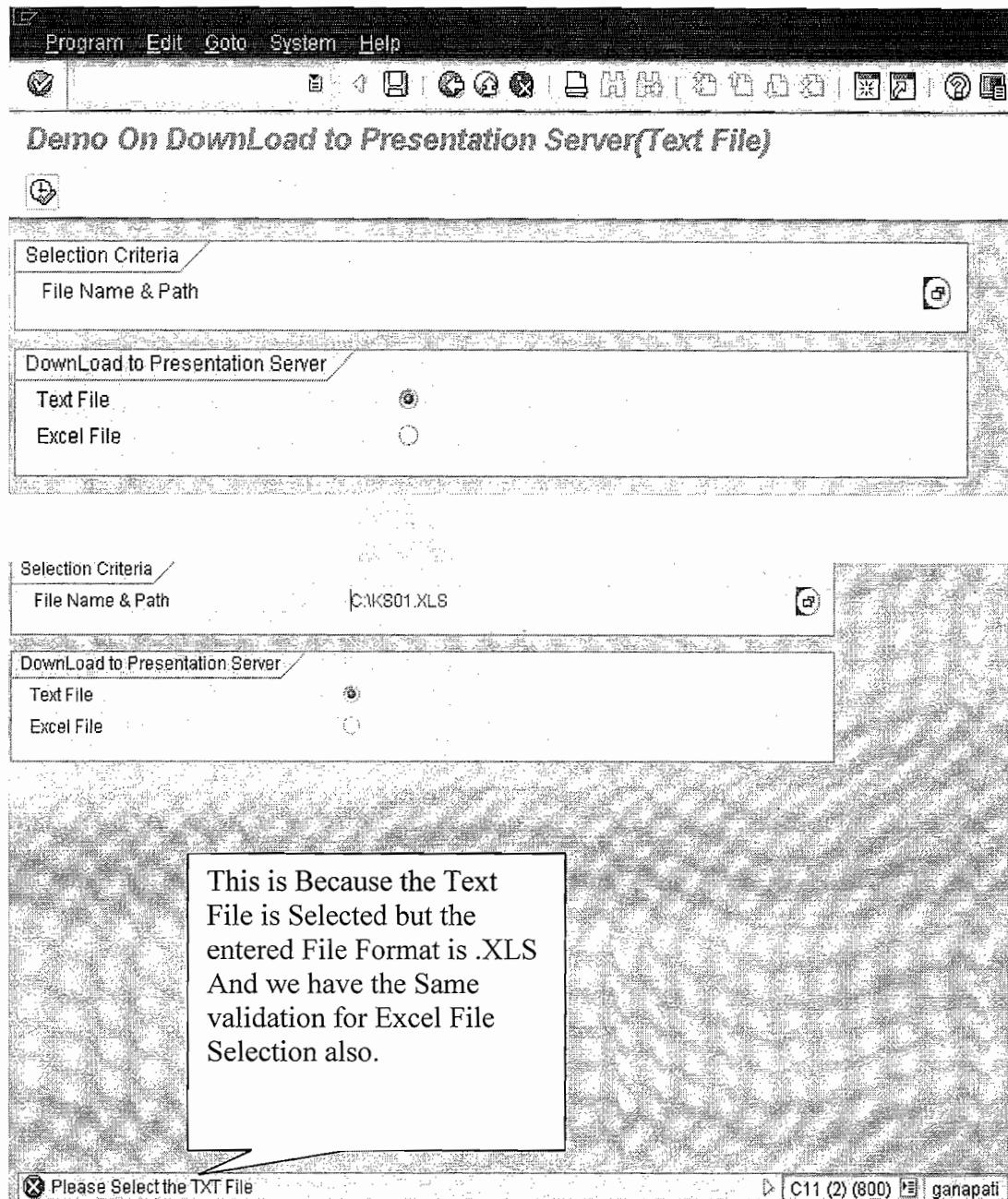
```
  CALL FUNCTION 'GUI_UPLOAD'  
    EXPORTING  
      FILENAME        = V_FILE  
      FILETYPE        = 'ASC'  
      HAS_FIELD_SEPARATOR = 'X'  
    TABLES  
      DATA_TAB        = IT_T001_TXT.
```

```
  IF SY-SUBRC = 0.  
    MESSAGE S002(ZDEMO) WITH V_FILE.  
    LOOP AT IT_T001_TXT INTO WA_T001_TXT.  
      WRITE : / WA_T001_TXT-BUKRS,  
              WA_T001_TXT-BUTXT,  
              WA_T001_TXT-ORT01,  
              WA_T001_TXT-LAND1.  
      CLEAR WA_T001_TXT.  
    ENDLOOP.
```

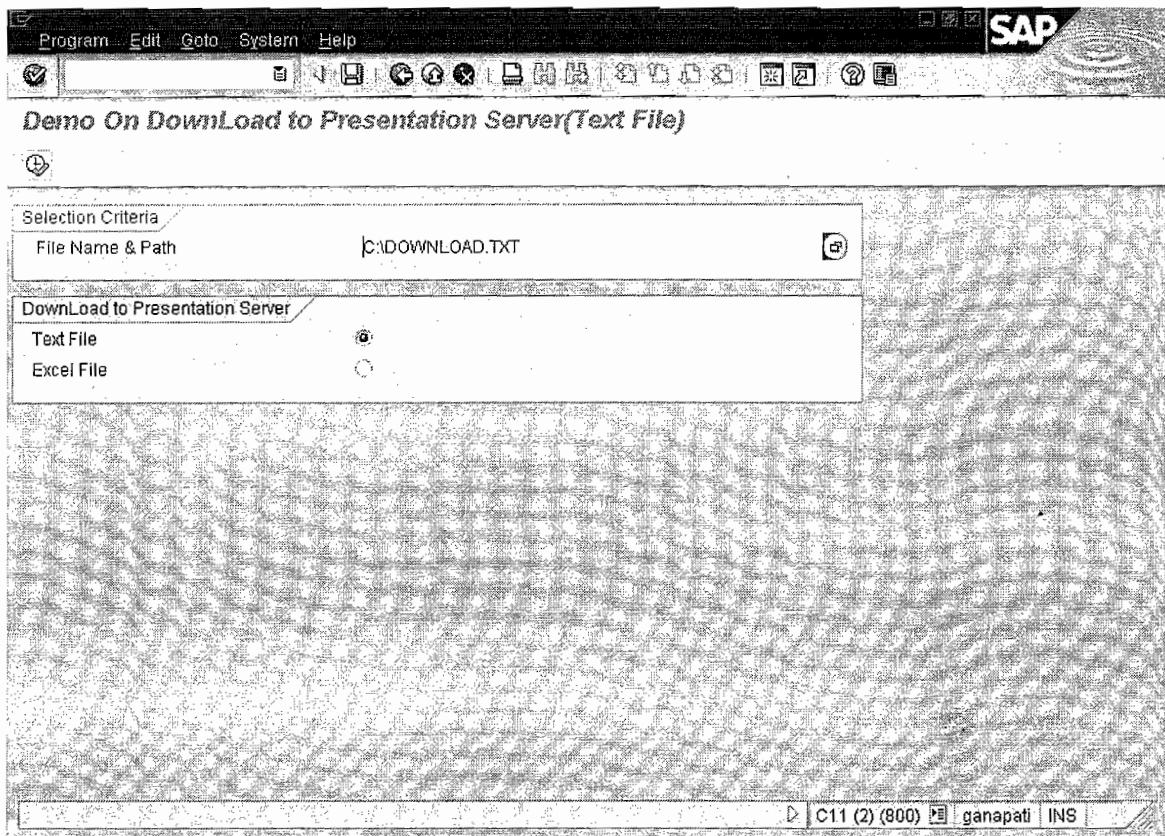
```
ENDIF.  
  
ELSEIF R_EXCEL = 'X'.  
  
    CALL FUNCTION 'ALSM_EXCEL_TO_INTERNAL_TABLE'  
        EXPORTING  
            FILENAME           = P_FILE  
            I_BEGIN_COL       = 1  
            I_BEGIN_ROW       = 1  
            I_END_COL        = 4  
            I_END_ROW        = 2  
        TABLES  
            INTERN           = IT_RETURN  
* EXCEPTIONS  
*     INCONSISTENT_PARAMETERS      = 1  
*     UPLOAD_OLE                 = 2  
*     OTHERS                      = 3  
  
    IF SY-SUBRC = 0.  
        MESSAGE S000(ZDEMO) WITH P_FILE.  
  
        SORT IT_RETURN BY ROW COL.  
  
*MOVE ALL THE ROWS AND COLUMN DATA TO THE ITAB  
LOOP AT IT_RETURN INTO WA RETURNS.  
  
    WA_RETURN = WA_RETURNS.  
    CASE WA_RETURN-COL .  
        WHEN 1.  
            WA_T001_EXCEL-BUKRS = WA_RETURN-VALUE.  
  
        WHEN 2.  
            WA_T001_EXCEL-BUTXT = WA_RETURN-VALUE.  
        WHEN 3.  
            WA_T001_EXCEL-ORT01 = WA_RETURN-VALUE.  
        WHEN 4.  
            WA_T001_EXCEL-LAND1 = WA_RETURN-VALUE.  
    ENDCASE.  
  
*AT END OF EACH ROW I.E AFTER MOVING ALL THE COLUMN DATA  
*APPEND THAT DATA AS ONE RECORD  
AT END OF ROW.  
  
    APPEND WA_T001_EXCEL TO IT_T001_EXCEL.  
    CLEAR : WA_RETURN, WA_T001_EXCEL.  
  
    ENDDAT.  
  
    ENDOOP.  
ENDIF.  
  
ENDIF.  
  
*DISPLAY DATA FROM IT_T001_EXCEL.  
  
LOOP AT IT_T001_EXCEL INTO WA_T001_EXCEL.  
  
WRITE : / WA_T001_EXCEL-BUKRS,WA_T001_EXCEL-BUTXT,WA_T001_EXCEL-ORT01,
```

```
WA_T001_EXCEL-LAND1.  
ENDLOOP.
```

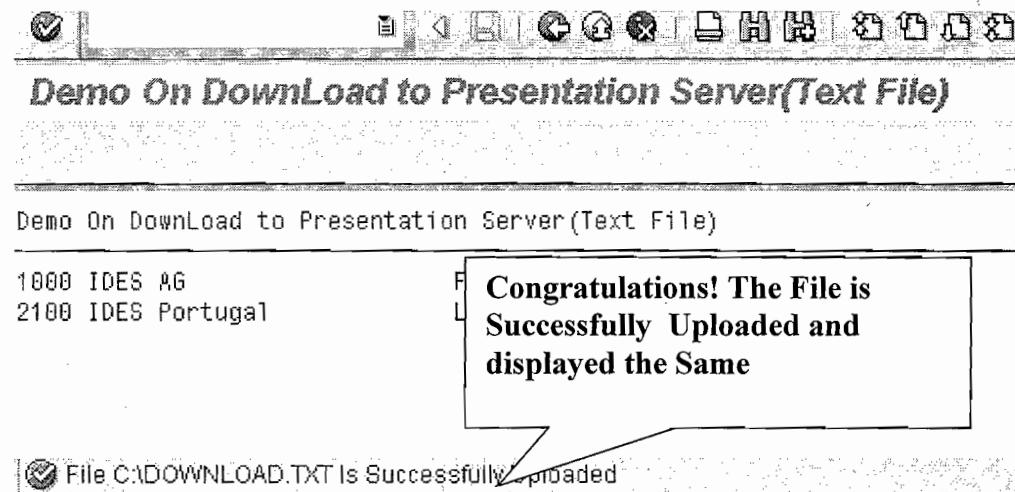
Execute the Program :



Execute the Program for the below Input:

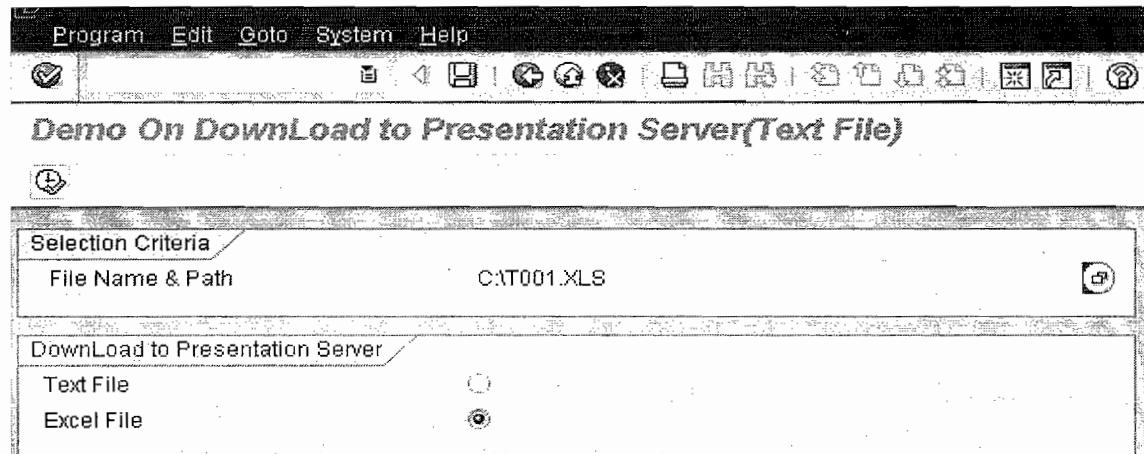


Output of the Program :



Test Case for EXCEL File:

Execute the Same Program for EXCEL File:



Execute the Same

Output :

| A | B | C | D | E |
|----|------|--------|-----|-----|
| 1 | 1000 | AGTECH | HYD | IND |
| 2 | 2000 | EMAX | HYD | IND |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |
| 11 | | | | |

File C:\T001.XLS Is Successfully Uploaded

C11 (2) (800) ganapati INS

Working With DOWNLOAD (Download Data from Internal Table to File):

Note: We Can use the below Function Modules for the same

**DOWNLOAD
OR
WS_DOWNLOAD
OR
GUI_DOWNLOAD.**

Note :

Both DOWNLOAD and WS_DOWNLOAD are Out Dated and currently we Use GUI_DOWNLOAD.

Important EXPORTING parameters for this function are:

Filename = Name of the local file to which the internal table is to be downloaded.

File type = file type, default values are ASC, DAT, BIN

Mode = Write mode, overwrite (' ') or Append ('A')

Tables to be passed to the function:

Data_tab = the internal table that is to be downloaded.

Program to Download the Data into XL Sheet :

```
*****
* PROGRAM : ZDEMO_GUI_DOWNLOAD
* AUTHOR  : GANPATI . ADIMULAM
* PURPOSE  : PROGRAM TO DOWNLOAD ALL
*           THE SALES ORDER CONTRACTS
* REFERENCE : NA
* COPIED FROM : NA
* TRAPORT REQUEST NO : C11D2K9001
*****
```

REPORT ZDEMO_GUI_DOWNLOAD .

DATA : BEGIN OF WA_VEDA,

VBELN TYPE VBELN_VA, " Sales document

VPOSN TYPE POSNR_VA, " Item

VLAUFZ TYPE VLAUF_VEDA, " Validity of Contract

VABNDAT TYPE VADAT_VEDA, " Agreement acceptance date

VBGEDAT TYPE VBDAT_VEDA, " Contract start date

VUNTDAT TYPE VUDAT_VEDA, " Contract Sign Date

END OF WA_VEDA.

DATA : IT_VEDA LIKE TABLE OF WA_VEDA.

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

```
SELECT VBELN  
      VPOSN  
      VLAUFZ  
      VABNDAT  
      VBEGDAT  
      VUNTDAT  
INTO TABLE IT_VEDA  
FROM VEDA  
UP TO 20 ROWS.
```

```
*****  
*      END-OF-SELECTION.      *  
*****
```

END-OF-SELECTION.

```
IF NOT IT_VEDA IS INITIAL.  
LOOP AT IT_VEDA INTO WA_VEDA.  
WRITE: /  WA_VEDA-VBELN,  
          WA_VEDA-VPOSN,  
          WA_VEDA-VABNDAT,  
          WA_VEDA-VBEGDAT,  
          WA_VEDA-VUNTDAT.  
CLEAR WA_VEDA.  
ENDLOOP.  
ELSE.  
  WRITE : / 'NO RECORDS'  
ENDIF.
```

*DOWNLOAD THE RECORDS

```
CALL FUNCTION 'GUI_DOWNLOAD'  
  EXPORTING  
    BIN_FILESIZE           =  
    FILENAME               = 'C:\DOWNLOAD.XLS'  
    FILETYPE                = 'ASC'  
    APPEND                  = ''  
    WRITE_FIELD_SEPARATOR   = 'X'  
  TABLES  
    DATA_TAB                = IT_VEDA
```

GUI DOWNLOAD Downloads the Data into the file based on the Extension, .XLS as XL File, .TXT as TEXT File .DOC as Word Document

```
IF SY-SUBRC <> 0.  
* MESSAGE ID SY-MSCID TYPE SY-MSGTY NUMBER SY-MSGNO  
*           WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
```

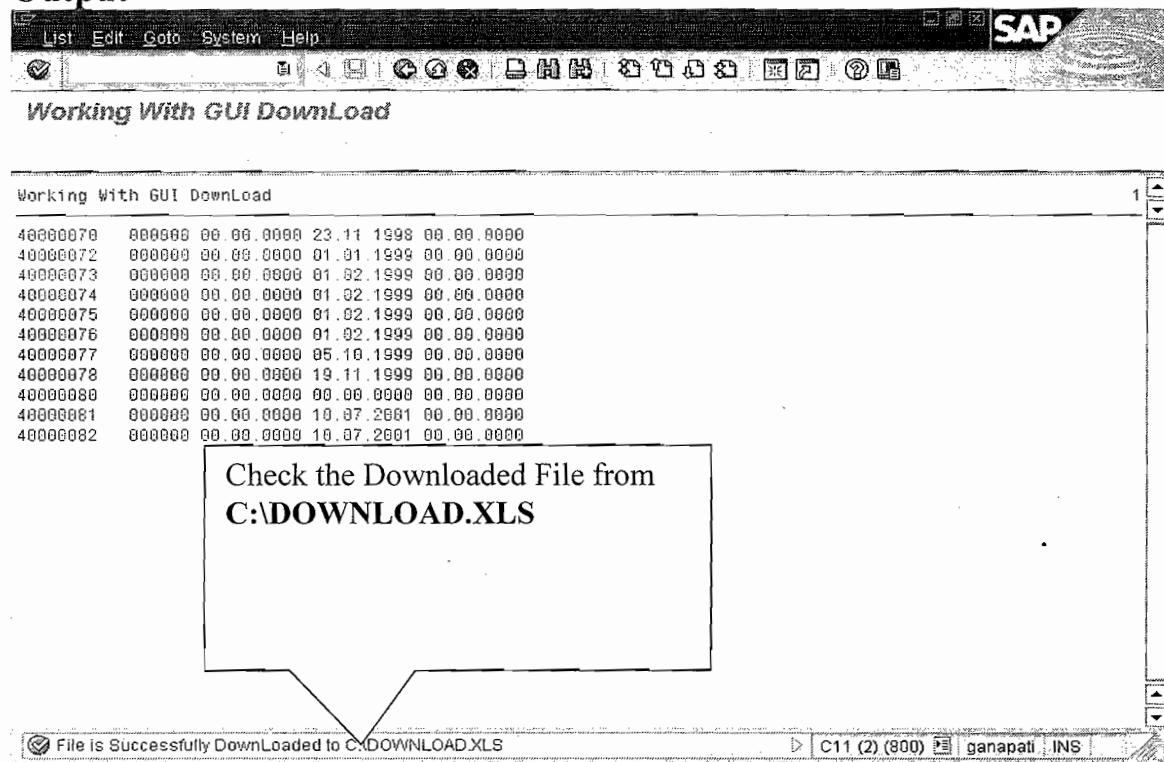
```
ELSE.  
MESSAGE S009(ZDEMO) WITH 'C:\DOWNLOAD.XLS'.  
ENDIF.
```

'X' for TAB Separator as XL File is TAB Separator

File Handling in SAP

We Never Compromise Quality, Would You?

Output



Contents from the file that is Generated:

| Microsoft Excel - DOWNLOAD.XLS | | | | | | | | | | | | |
|--------------------------------|----------|---|---|---|----------|---|---|---|---|---|---|---|
| Type a question for help | | | | | | | | | | | | |
| C12 | A | B | C | D | E | F | G | H | I | J | K | L |
| 1 | 40000070 | 0 | 1 | 0 | 19981123 | 0 | | | | | | |
| 2 | 40000072 | 0 | 0 | 0 | 19990101 | 0 | | | | | | |
| 3 | 40000073 | 0 | 1 | 0 | 19990201 | 0 | | | | | | |
| 4 | 40000074 | 0 | 1 | 0 | 19990201 | 0 | | | | | | |
| 5 | 40000075 | 0 | 1 | 0 | 19990201 | 0 | | | | | | |
| 6 | 40000076 | 0 | 1 | 0 | 19990201 | 0 | | | | | | |
| 7 | 40000077 | 0 | 1 | 0 | 19991005 | 0 | | | | | | |
| 8 | 40000078 | 0 | 1 | 0 | 19991119 | 0 | | | | | | |
| 9 | 40000080 | 0 | 0 | 0 | 0 | 0 | | | | | | |
| 10 | 40000081 | 0 | 1 | 0 | 20010710 | 0 | | | | | | |
| 11 | 40000082 | 0 | 1 | 0 | 20010710 | 0 | | | | | | |
| 12 | | | | | | | | | | | | |
| 13 | | | | | | | | | | | | |
| 14 | | | | | | | | | | | | |
| 15 | | | | | | | | | | | | |
| 16 | | | | | | | | | | | | |
| 17 | | | | | | | | | | | | |
| 18 | | | | | | | | | | | | |
| 19 | | | | | | | | | | | | |
| 20 | | | | | | | | | | | | |
| 21 | | | | | | | | | | | | |
| 22 | | | | | | | | | | | | |
| 23 | | | | | | | | | | | | |
| 24 | | | | | | | | | | | | |

Working with Files On Application Server :

Introduction:

Files on application server are sequential files. A sequential file is also called as **DATASET**.

Handling of Sequential file:

Three steps are involved in sequential file handling.

- 1. OPEN**
- 2. PROCESS**
- 3. CLOSE**

Here processing of file can be READING a file or WRITING on to a file.

1. OPEN FILE: Before processing the Data, a file needs to be opened.
After processing, file should be closed.

SYNTAX:

```
OPEN DATASET <file name >
FOR { OUTPUT / INPUT /APPENDING}
IN {TEXT MODE /BINARY MODE} ENCODING DEFAULT.
```

This statement returns **SY-SUBRC** as 0 for successful opening of file or 8, if unsuccessful.

FOR OUTPUT: Opens the file for writing.

FOR INPUT: Opens a file for READ and place the cursor at the beginning of the file.

FOR APPENDING: Opens the File for writing and places the cursor at the end of file. IF file doesn't exists, then it is generated other wise the new contents will be added at the end of the existing contents.

IN BINARY MODE: The READ or TRANSFER will be character wise. Each time 'n' characters are Read or Transferred .The next READ or TRANSFER will start from the next character position and not on the next line.

IN TEXT MODE: The READ or TRANSFER will start at the beginning of a new line each time. If for READ, the destination is shorter than the source, and then gets truncated. If destination is longer , then it is padded with spaces.

Defaults: If nothing is mentioned, then defaults are FOR INPUT and in BINARY MODE.

2 . PROCESS FILE:

**Processing a file involves Reading the file or Writing on to the file i.e,
TRANSFER.**

A) TRANSFER Statement Syntax:

TRANSFER <WA> TO <Dataset Name >.

Each transfer statement, In Binary mode, writes the length of the field to the dataset. In Text mode, writes one line to the dataset.

If the file is not already open, TRANSFER tries to OPEN file FOR OUTPUT (IN BINARY MODE) or using the last OPEN DATASET statement for this file.

IN FILE HANDLING, TRANSFER IS THE ONLY STATEMENT WHICH DOES NOT RETURNS SY-SUBRC.

B) READ Statement

SYNTAX: READ DATASET <Dataset Name > INTO <WA>

Each READ will get one record from the dataset. In Binary mode it reads the length of the field and in Text mode it reads each line.

3. CLOSE FILE

All sequential files, which are still open at the end of the program, will be closed by the program. However, it is a good programming practice to explicitly close all the datasets that were opened.

SYNTAX: CLOSE DATASET <Dataset Name>

SY-SUBRC will be set to 0 or 8 depending on whether the CLOSE is Successful or Not.

DELETE FILE

SYNTAX

```
DELETE DATASET <Dataset Name>
```

SY-SUBRC will be set to 0 or 8 depending on whether the DELETE is Successful or Not.

Writing Data From Program(<ITAB>) to Sequential Files :

Pseudo Logic for Writing Data to the sequential files.

```
Open dataset for output / Appending in a particular mode.
```

For Each Records from ITAB.

```
i.e LOOP AT ITAB INTO WA.  
      TRANSFER the WA to a dataset.  
      ENDLOOP.
```

```
Close the dataset.
```

Example Program to write the Sales orders against Contracts Data to Application Server:

```
*****  
* PROGRAM : ZDEMO_WRITE_DATA_TO_APP_SERVER *  
* AUTHOR : GANPATI . ADIMULAM *  
* PURPOSE : PROGRAM TO DOWNLOAD ALL *  
*           THE SALES ORDER CONTRACTS *  
*           TO APPLICATION SERVER *  
* REFERENCE : NA *  
* COPIED FROM : NA *  
* TRAPORT REQUEST NO : C11D2K9001 *  
*****  
  
REPORT ZDEMO_WRITE_DATA_TO_APP_SERVER.  
  
DATA : BEGIN OF WA_VEDA,  
        VBELN      TYPE    VBELN_VA,      " Sales document  
        VPOSN      TYPE    POSNR_VA,      " Item  
        VLAUFZ    TYPE    VLAUF_VEDA,    " Validity of Contract  
        VABNDAT   TYPE    VADAT_VEDA,    " Agreement acceptance date  
        VBGEDAT   TYPE    VBDAT_VEDA,    " Contract start date
```

File Handling in SAP

We Never Compromise Quality, Would You?

```
VUNTDAT TYPE VUDAT_VEDA, " Contract Sign Date
END OF WA_VEDA.

DATA : IT_VEDA LIKE TABLE OF WA_VEDA.

*****
*      START-OF-SELECTION. *
*****
START-OF-SELECTION.

SELECT VBELN
      VPOSN
      VLAUFZ
      VABNDAT
      VBEGDAT
      VUNTDAT
INTO TABLE IT_VEDA
FROM VEDA
UP TO 20 ROWS.

*****
*      END-OF-SELECTION. *
*****
END-OF-SELECTION.

IF NOT IT_VEDA IS INITIAL.
LOOP AT IT_VEDA INTO WA_VEDA.
WRITE : / WA_VEDA-VBELN,
         WA_VEDA-VPOSN,
         WA_VEDA-VABNDAT,
         WA_VEDA-VBEGDAT,
         WA_VEDA-VUNTDAT.
CLEAR WA_VEDA.
ENDLOOP.
ELSE.

WRITE : / 'NO RECORDS FOUND'.
ENDIF.

SKIP 2.

*OPEN THE FILE FOR WRITING THE DATA

IF NOT IT_VEDA IS INITIAL.

OPEN DATASET 'MY_NEW_FILE' IN TEXT MODE FOR OUTPUT ENCODING DEFAULT.

IF SY-SUBRC = 0.

LOOP AT IT_VEDA INTO WA_VEDA.

TRANSFER WA_VEDA TO 'MY_NEW_FILE'.
CLEAR WA_VEDA.

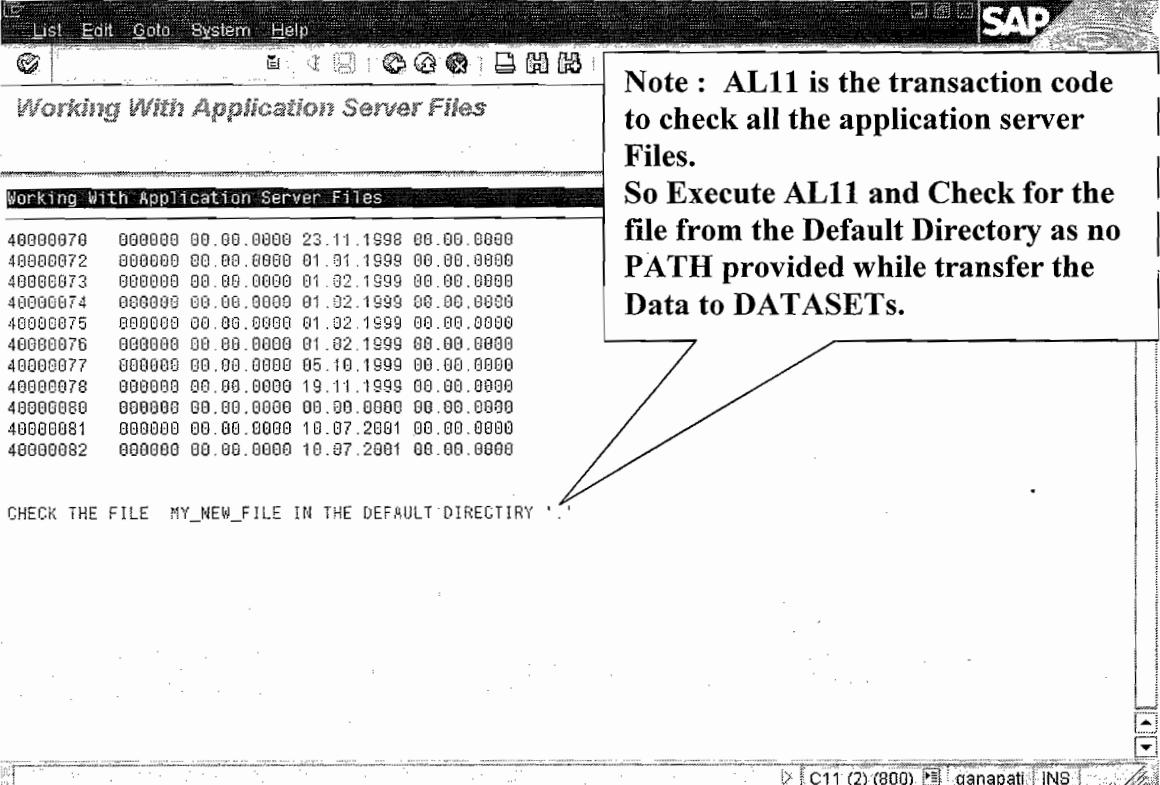
ENDLOOP.

WRITE : / 'CHECK THE FILE MY_NEW_FILE IN THE DEFAULT DIRECTIRY '''''.
ENDIF.
```

```
*close the File  
CLOSE DATASET 'MY_NEW_FILE'.  
  
ELSE.  
WRITE : / 'NO FILE GENERATED AS THERE IS NO RECORDS TO DOWNLOAD'.  
ENDIF.
```

Execute the Program : ZDEMO_WRITE_DATA_TO_APP_SERVER.

Output:



| ID | Date | Time |
|----------|-----------------------|------------|
| 40000070 | 00.00.0000 23.11.1998 | 00.00.0000 |
| 40000072 | 00.00.0000 01.01.1999 | 00.00.0000 |
| 40000073 | 00.00.0000 01.02.1999 | 00.00.0000 |
| 40000074 | 00.00.0000 01.02.1999 | 00.00.0000 |
| 40000075 | 00.00.0000 01.02.1999 | 00.00.0000 |
| 40000076 | 00.00.0000 01.02.1999 | 00.00.0000 |
| 40000077 | 00.00.0000 05.10.1999 | 00.00.0000 |
| 40000078 | 00.00.0000 19.11.1999 | 00.00.0000 |
| 40000080 | 00.00.0000 00.00.0000 | 00.00.0000 |
| 40000081 | 00.00.0000 10.07.2001 | 00.00.0000 |
| 40000082 | 00.00.0000 10.07.2001 | 00.00.0000 |

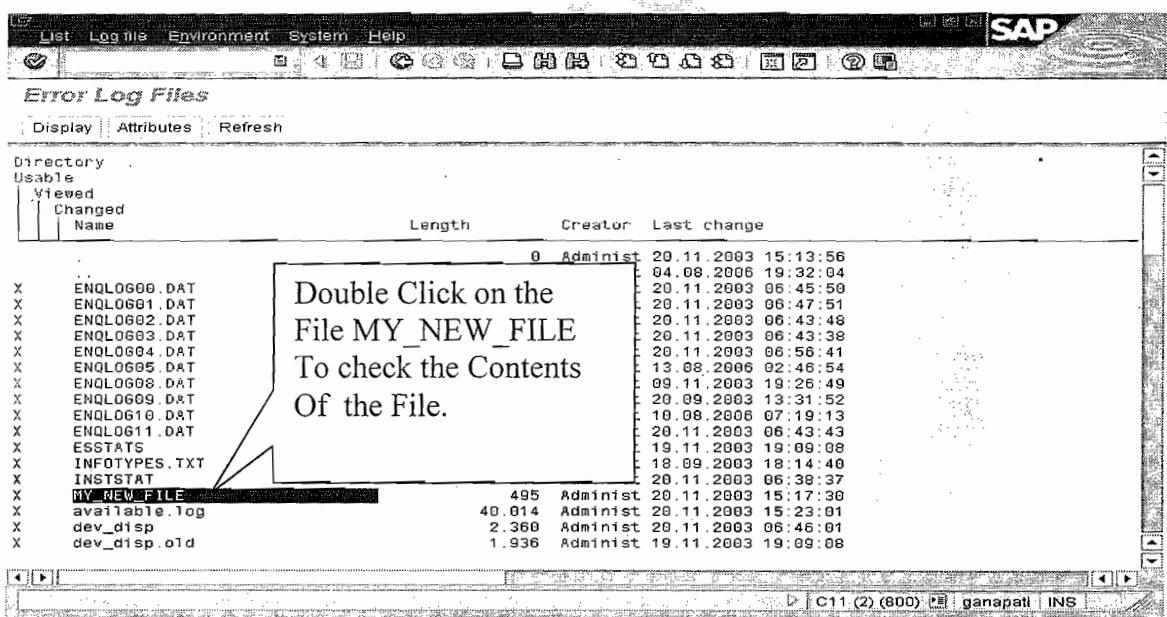
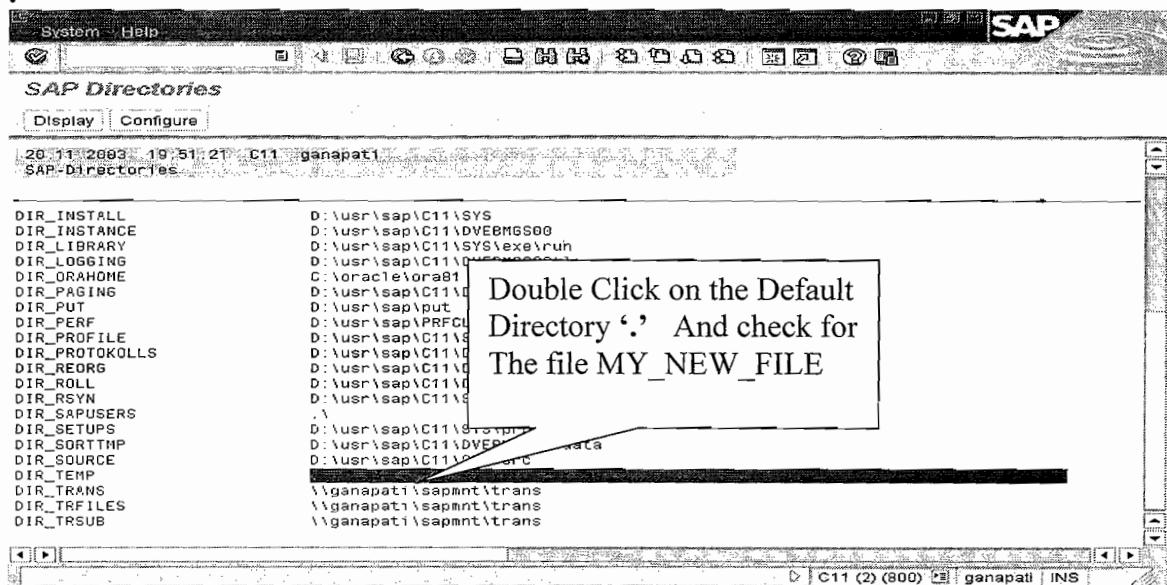
CHECK THE FILE MY_NEW_FILE IN THE DEFAULT DIRECTIRY

Note : AL11 is the transaction code to check all the application server Files.
So Execute AL11 and Check for the file from the Default Directory as no PATH provided while transfer the Data to DATASETS.

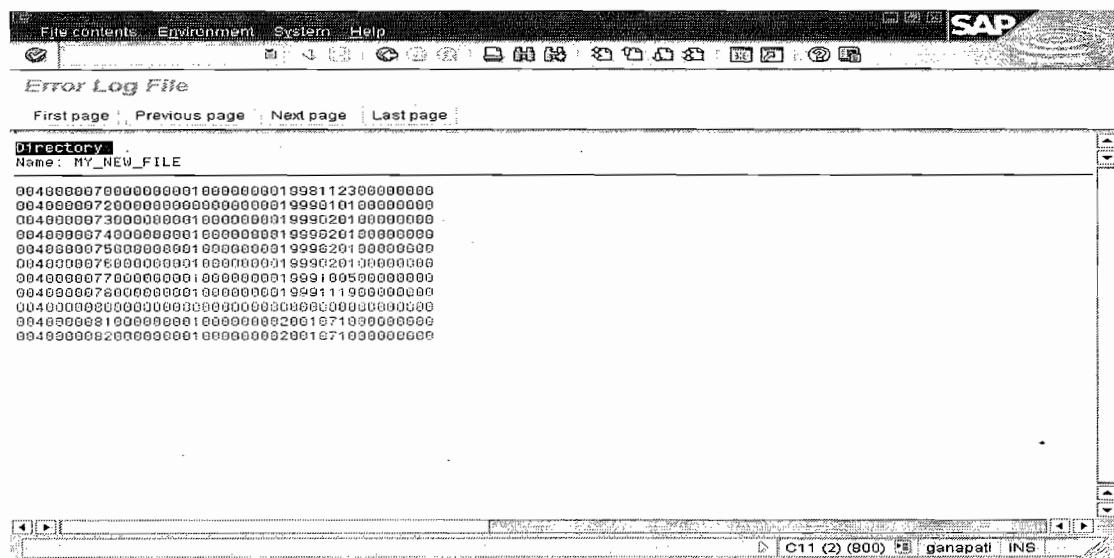
Execute AL11 to check for the file MY_NEW_FILE

File Handling in SAP

We Never Compromise Quality, Would You?



File Contents:



The screenshot shows the SAP GUI interface with the title bar "File contents Environment System Help" and the SAP logo. Below the title bar is a toolbar with various icons. The main area is titled "Error Log File". It displays a directory listing with one item: "Name: MY_NEW_FILE". The file content is shown as a large block of binary data consisting of numerous zeros and ones. At the bottom of the SAP window, there is a status bar with the text "C11 (2) (800)" and "ganapati INS".

```
0040000070000000001000000001998112300000000  
00400000720000000000000000001999010100000000  
0040000073000000001000000001999020100000000  
0040000074000000001000000001999020100000000  
0040000075000000001000000001999020100000000  
0040000076000000001000000001999020100000000  
0040000077000000001000000001999100500000000  
0040000078000000001000000001999111900000000  
00400000800000000010000000030000000000000000  
00400000810000000010000000020015710300000000  
00400000820000000010000000020015710300000000
```

Steps to READ Data From DATASETS :

Pseudo Code For Reading:

OPEN DATASET for INPUT(Read) MODE.

Starts DO Loop. "Since we don't know the exact no of records in a file

READ DATASET into WA.

If READ is not successful.

EXIT the loop.

ELSE. "Record Found

Do relevant processing for that record.

ENDIF.

ENDDO.

CLOSE DATASET.

Program to read the Data from the Application Server File :

```
*****  
* PROGRAM : ZDEMO_READ_DATA_FROM_APP_SERVR *  
* AUTHOR : GANAPATI . ADIMULAM *  
* PURPOSE : PROGRAM TO READ THE DATA FROM *  
* APPLICATION SERVER *  
* TO APPLICATION SERVER *  
* REFERENCE : NA *  
* COPIED FROM : NA *  
* TRAPORT REQUEST NO : C11D2K9001 *  
*****
```

REPORT ZDEMO_READ_DATA_FROM_APP_SERVR.

```
DATA : BEGIN OF WA_VEDA,
        VBELN      TYPE  VBELN_VA,      " Sales document
        VPOSN      TYPE  POSNR_VA,     " Item
        VLAUFZ     TYPE  VLAUF_VEDA,   " Validity of Contract
        VABNDAT    TYPE  VADAT_VEDA,   " Agreement acceptance date
        VBEGDAT    TYPE  VBDAT_VEDA,   " Contract start date
        VUNTDAT    TYPE  VUDAT_VEDA,   " Contract Sign Date
      END   OF WA_VEDA.

DATA : IT_VEDA LIKE TABLE OF WA_VEDA.

***** START-OF-SELECTION. *****
START-OF-SELECTION.

*OPEN THE FILE FOR READIGN THE FILE DATA
OPEN DATASET 'MY_NEW_FILE' IN TEXT MODE FOR INPUT ENCODING DEFAULT.

DO .

  READ DATASET 'MY_NEW_FILE' INTO WA_VEDA.

  IF SY-SUBRC = 0.
    APPEND WA_VEDA TO IT_VEDA.
  ELSE.
    EXIT. "AS THERE IS NO FURTHER RECORD TO READ
  ENDIF.

  ENDDO.

*close the File
CLOSE DATASET 'MY_NEW_FILE'.

***** END-OF-SELECTION. *****
END-OF-SELECTION.

IF NOT IT_VEDA IS INITIAL.
  LOOP AT IT_VEDA INTO WA_VEDA.
    WRITE : / WA_VEDA-VBELN,
              WA_VEDA-VPOSN,
              WA_VEDA-VABNDAT,
              WA_VEDA-VBEGDAT,
              WA_VEDA-VUNTDAT.
    CLEAR WA_VEDA.
  ENDLOOP.
ELSE.
  WRITE : / 'NO RECORDS FOUND'.
ENDIF.
```

Input File :

The screenshot shows the SAP interface with the title bar "Error Log File". Below it, there are links for "First page", "Previous page", "Next page", and "Last page". A "Directory" section shows the path "Name: MY_NEW_FILE". The main area displays a large amount of binary or hex data starting with "0040000070000000000010000000000199911230000000000".

Output:

The screenshot shows the SAP interface with the title bar "Working With Application Server Files". Below it, there are links for "List", "Edit", "Code", "System", and "Help". The main area displays a table titled "Working With Application Server Files" with several rows of data. One row is highlighted in yellow.

| | 00000070 | 00000000 | 00.00.0000 | 23.11.1999 | 00.00.00000000 |
|----------|----------|------------|------------|------------|----------------|
| 40000071 | 00000000 | 00.00.0000 | 01.01.1999 | 00.00.0000 | 00000000 |
| 40000072 | 00000000 | 00.00.0000 | 01.02.1999 | 00.00.0000 | 00000000 |
| 40000073 | 00000000 | 00.00.0000 | 01.03.1999 | 00.00.0000 | 00000000 |
| 40000074 | 00000000 | 00.00.0000 | 01.02.1999 | 00.00.0000 | 00000000 |
| 40000075 | 00000000 | 00.00.0000 | 01.02.1999 | 00.00.0000 | 00000000 |
| 40000076 | 00000000 | 00.00.0000 | 01.03.1999 | 00.00.0000 | 00000000 |
| 40000077 | 00000000 | 00.00.0000 | 05.10.1999 | 00.00.0000 | 00000000 |
| 40000078 | 00000000 | 00.00.0000 | 19.11.1999 | 00.00.0000 | 00000000 |
| 40000079 | 00000000 | 00.00.0000 | 00.00.0000 | 00.00.0000 | 00000000 |
| 40000080 | 00000000 | 00.00.0000 | 10.07.2001 | 00.00.0000 | 00000000 |
| 40000081 | 00000000 | 00.00.0000 | 10.07.2001 | 00.00.0000 | 00000000 |
| 40000082 | 00000000 | 00.00.0000 | 10.07.2001 | 00.00.0000 | 00000000 |

Exercises

- 1) Develop a Program to Down Load the List of Purchase Orders, Items which Costs More than 1 lakh INR into Presentation Server with COMMA (,) Separator?
- 2) Develop a Program to Down Load the List of Sales Orders, Items which Costs More than 1 lakh INR into Application Server with '/' Separator?
- 3) Write a Program to Upload the data from the File and Update the Database tables KNA1 with Record Type 10 and KNBK with Record type 20 and Process the File From Presentation Server?

Maintain the below File in Both Presentation Server and Application Server and provide Two Radio button

- . Presentation Server
- . Application Server

Based on the radio button Selected ,Read the Data the from File and Process it.

File Details:

10,1111,eMAX technologies, eMAX, Hyderabad, Ameerpet,IN,EN
20,1111,DE,1212121212,018301004245
20,1111,DE,232121212,018301004250
20,1111,DE,3412121212,01830100460
10,2222,Clarion park Technologies, Clarion,Hyderabad,S.R.nagar,IN,EN
20, 2222,DE,1212121212,018301004245
20, 2222,DE,232121212,018301004250
20, 2222,DE,3412121212,01830100460

Structure of File with Record Type 10

| Value | Name |
|-------------------|----------------------|
| 10 | Record Type |
| 1111 | Customer No |
| EMAX technologies | Name of the Customer |
| EMAX | Hyderabad |
| Hyderabad | City |
| Ameerpet | Street |
| IN | Country |
| EN | Language |

Note: For Data types and lengths of the Fields refer table KNA1.

Structure of File with Record Type 20

| Value | Name |
|--------------|------------------|
| 20 | Record Type |
| 1111 | Customer no |
| DE | Bank Country Key |
| 1212121212 | Bank key |
| 018301004245 | Account Number |

Note: For Data types and lengths of the Fields refer table KNBK (Customer Bank Details).

4. Develop a program to Upload the Data from the XLS file and Display the Same.

