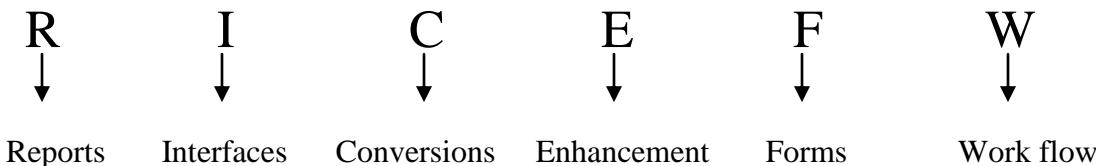


**SAP:** - Systems applications and Products in Data Processing

**ABAP:** - Advanced Business Application Programming language

ABAP isn't case sensitive.

Role of an ABAPer in real time:-



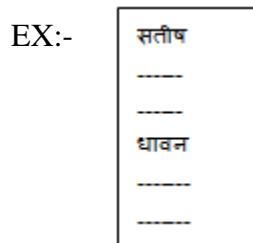
In SAP ABAP each task is called one object.

**Report:** - Based on the given input we'll fetch [read or get] data from data base and display in a predefined format.

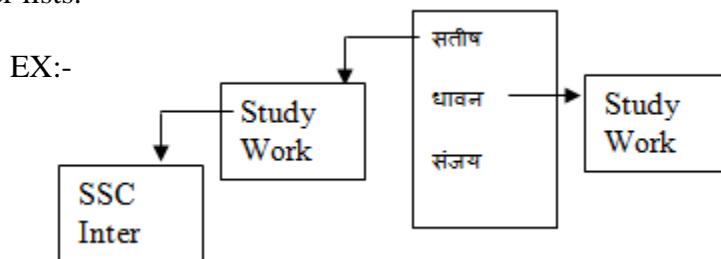
**Types of Reports:-**

1. Classical Reports
2. Interactive Reports
3. ALV Reports

**Classical Reports:** - It's nothing but to display the entire information in a single list.



**Interactive Reports:** - It's nothing but to display the summarized information in the basic list and detailed information in further lists.

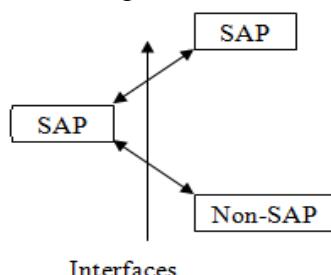


**ALV Reports:** - This is used to display the data with additional features i.e. Borders, colors, shades, lines etc.

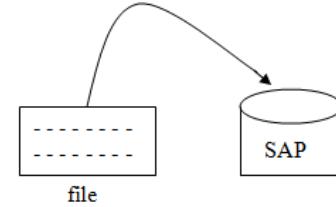
The real time reports are internal purpose.

**Interfaces:** - Interfaces are used to connecting from SAP to SAP as well as SAP to non-SAP.

Interfaces are ALE/IDOCs, BAPI



**Conversion:** - Conversion programs are used to transverse the data from file to SAP system.  
Conversion programs are BDC,LSMW

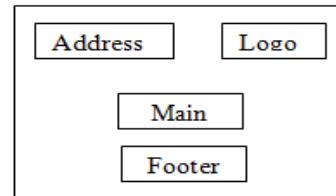


**Enhancements:** - Enhancements are used to adding some additional functionality to the standard functionality with out disturbing the standard functionality.

Enhancements are BADI, user exists, customer exits, enhancement frame work, enhancement spot

**Forms:** - Forms are used to design the business documents such as offer letters, experience letters, pay bills etc.

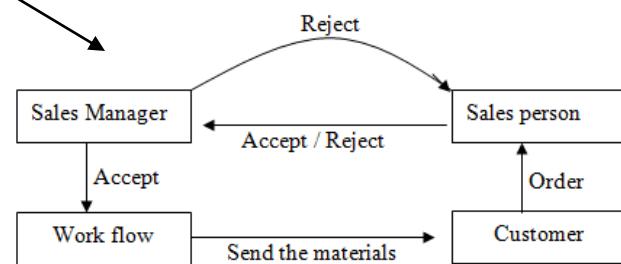
Forms are SAP Script, SMART forms.



**Work flow:** - Work flow is used to automate the exist business process & used to identify the work load of employee as well as performance of the employee.

#### Types of project:-

1. Implementation project
2. Up gradation project
3. Maintenance / Support project
4. Roll out project



**Implementation project:** - When ever we want to develop the SAP from fundamental on wards then we go for implementation project. Implementation project takes 1 year to 3 years.

#### Phases of implementation project:-

1. Project preparation phase
2. Business Blue Print phase
3. Realization phase
4. Final preparation phase
5. Go-live & support phase

In the implementation project the ABAPer must start from Realization phase.

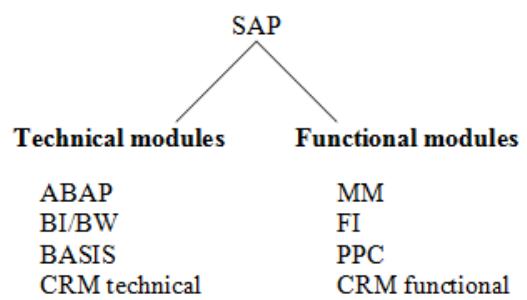
**Up gradation project:** - When ever the version is changed if you want to implement the new technologies into existing system then you go for up gradation project. Up gradation project takes 30 days to 3 months.

**Maintenance / Support project:** - After go level support phase each and every company requires 24\*7 supports. Then they go for support project.

**Roll out project:** - whenever we purchase a new company if you want to extend the same SAP to new company, then we go for roll out project.

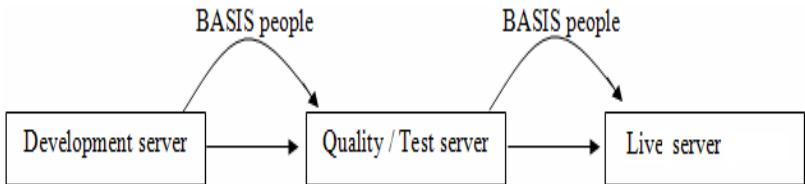
If you want to implement the new module in the existing SAP system then we go for roll out project.

#### Real time system landscape:-



The source code of an ABAP consists of either a statement or comment.

A statement is the collection of operators, operands, variables and keywords.



**Operators:** - In ABAP we've 3 types of operators.

## Mathematical Operator:-

<b>Operator</b>	<b>Description</b>	<b>Example</b>
+	Addition	$3 + 2 = 5$
-	Subtraction	$3 - 2 = 1$
*	Multiplication	$3 * 2 = 6$
**	Exponent ional	$3 ** 3 = 3^2 = 9$
/	Division	$4 / 2 = 2$
Mod	Reminder of division Of two numbers	$5 \text{ mod } 2 = 1$

## **Comparative Operator:-**

<b>Operator</b>	<b>Description</b>	<b>Example</b>
< or LT	Less than	$a < b$ or a LT b
$\leq$ or LE	Less than or equal	$a \leq b$ or a LE b
> or GT	Greater than	$a > b$ or a GT b
$\geq$ or GE	Greater than or equal	$a \geq b$ or a GE b
= or EQ	Equal	$a = b$ or a EQ b
$\neq$ or NE	Not equal	$a \neq b$ or a NE b

### **Relational or Logical Operator:-**

<b>Operator</b>	<b>Description</b>	<b>Example</b>
And	AND	$a < b$ and $a < c$
Or	OR	$a < b$ or $a < c$
Not	NOT	-----

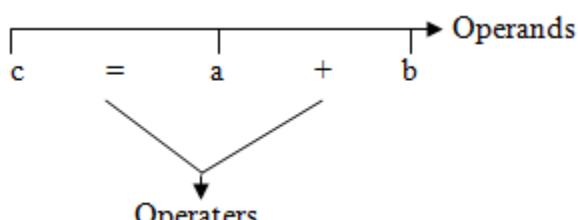
**Operands:** - Operands are the variables which we need to perform the particular operation.

WE HOPE

**Variables:** – Variable is the name given to the memory location

**Keywords:** - Keywords are used to identify the type of the statement. Keywords are ( $C^2D^3EQ$ )

- 1. Calling keyword
  - 2. Controlling keyword
  - 3. Declaration keyword
  - 4. Definition keyword
  - 5. Data base keyword
  - 6. Event keyword
  - 7. Operational keyword



**Declaration keyword:** - This keyword is used to declare the variables in ABAP. Some of the declarative keywords are Data, Parameters, Tables, and Types etc.

**Comment:** - Comments are non executable statements. These are used to improve the readability of the program. If you want to comment the entire line then you must place '\*' in first column of the line.

Ex: - \* a b c ---

If you want to comment the part of the line then you must place in “(double quotations) from double quotation onwards it treats as a comment.

## Data types

### Numeric Data types

Integer → I

Float → F

Packed Decimal → P

### Character data types

char → C

numeric char → N

Date → D

Time → T

**Note:** - Char is alpha numeric. That means it accept the integer and character.

**Note:** - Here Integer, Float, Date and Time are fixed length data types & rest of the data types are variable length data types.

### Syntax of Integer (I):-

Data <variable name> type I.

Ex: - data A type I.

The initial / default value of integer is ‘0’.

### Syntax of Float (F):-

Data <variable name> type F.

Ex: - data A type F.

The initial value of Float is ‘.00’.

### Syntax of Packed Decimal (P):-

Data <variable name> (<length>) type P decimals <no>.

Ex: - data A(10) type P decimal 3.

The initial value of packed decimal is ‘.000’ (number of decimals)

### Syntax of Char (C):-

Data <variable name> (<length>) type C.

Ex: - data A(5) type C.

The initial value of char is ‘space / empty’.

### Syntax of Date (D):-

Data <variable name> type D.

Ex: - data A type D.

The initial format is ‘YYYYMMDD’. ➔ (20140425)

### Syntax of Numeric Char (N):-

Data <variable name> (<length>) type N.

Ex: - data A(10) type N.

The initial value of Numeric Char is ‘000’. ➔ (Based on length)

## Syntax of Time (T):-

Data <variable name> type T.

Ex: - data A type T.

The initial value of time is '000000'. ➔ format is 'HHMMSS'

### In 'C' language

```
Int a, b, c;  
a=10;  
b=10;  
c=a+b;  
Printf("%d",c);
```

### In ABAP

```
Data A type I.  
Data B type I.  
Data C type I.  
A = 10.  
B = 20.  
C = A + B.  
Write C.
```

In ABAP each statement ends with dot (.). 'WRITE' is the operational keyword to display the output in ABAP.

If more than one variable having the same keyword list of same keyword every time, we use chain operators & variables are separated by commas (,). And statements end with dot (.)

```
Data: A type I, B type I, C type I.  
A = 10.  
B = 20.  
C = A + B.  
Write C.
```

## Technical requirements to create the program

1. Name of the program

In ABAP the name of the program must be starts with 'y' or 'z' because a→x reserved for SAP.

2. Provide the place / folder where we save the program. The place / folder is called development class up to 4.6c version. Now it's calling as package.

**Note:** - \$TMP is the default package which is provided by SAP. The latest version is ECC6.0 (Enterprise Central Component).

## Steps to up the server in our own PC:-

Double click on SAP management console on desktop. Select the server. Right click all tasks → start. Provide the password 'india123'. Click on 'As ABAP WP Table'. Continuously click on refresh button until the statistics wait. Minimize that one.

## Steps to log on to SAP:-

Double click on SAP logon on the desktop. Select server → click on logon. Provide client is 800. User is 'sapuser'. Password is 'india123' and click on enter.

## Steps to create the program: -

Execute SE38 (ABAP editor). Provide the program name. Click on create. Provide title. Select type is executable program. Click on save. Click on local object.

**Note:** - Click on local object means our program will be saved on \$TMP package.

## Steps to execute the program:-

Save the program CTRL + S.

Check the program CTRL + F2.

Active the program CTRL + F3.

Execute the program F8.

Parameter is the keyword which accepts the input at runtime.

### In 'C' Language

```
Int a, b, c;
Scanf("%d%d", &a, &b);
c=a+b;
Printf("%d", c);
```

### In ABAP

Parameter: A type I, B type I.  
 Data C type I.  
 $C = A + B.$   
 Write C.

**Note:** - Default is the keyword to provide the default value to the input variable.

Syntax: -

Parameter <name> type <data type> **default** <value>

Ex: - parameter A type I **default** 10.

A

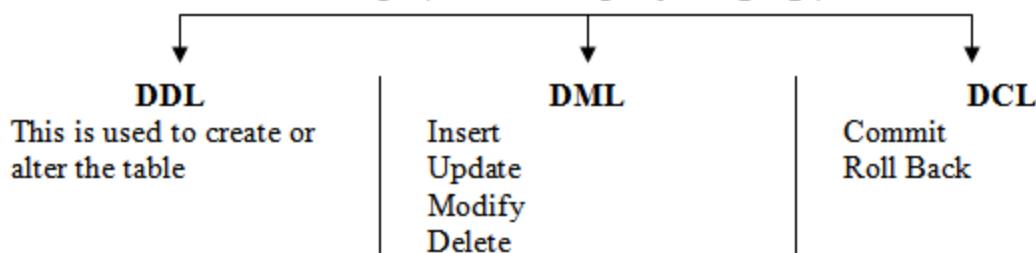
**Note:** - Obligatory is the keyword to provide input field is as mandatory.

Syntax: - parameter <name> type <data type> **obligatory**.

Ex: - parameter A type I **obligatory**.

**Note:** - Parameter can't accept float data type. Only it accept packed decimal.

### SQL (Structured Query Language)

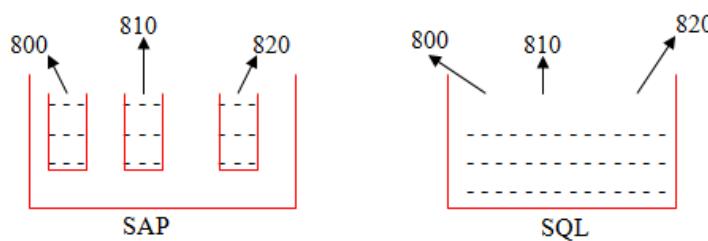


→ SQL is the database dependent & SAP is the database independent. So SAP doesn't support SQL.

→ Open SQL support SAP, because open SQL is the database independent.

→ Open SQL doesn't support DDL. So we can't create the tables by using simple statements like oracle.

→ Open SQL support **DDIC** (Data Dictionary) which is used to create or alter the tables.



## DATA DICTIONARY

Data dictionary is the central source of the database management system. The main functionality of the DDIC is to create or alter the tables.

### **Creating the db table by using DDIC in two ways: -**

1. Direct method / Pre defined method / Built in method
2. Data element method.

### **Technical requirement to create the table: -**

1. Name of the table starts with 'y' or 'z' because a→x reserved for SAP.
2. Provide list of fields, data types and lengths.
3. Provide the **delivery class**.
4. Provide the **Technical settings**.

**Delivery class:** - It defines the owner of the table as well as it controls the transport of the data from one table to another table.

Technical setting is nothing but combination of DATA CLASS & SIZE OF THE CATEGORY.

**Data class:** - Data class defines the physical area of the database in where our table is logically stored. Some of the important data classes are

- APPL 0 → Master data class  
 APPL 1 → Transaction data class  
 APPL 2 → Organization data class

**Master data class:** - Master data class is the data, which data we can access frequently as well as update rarely.

Ex: - customer master data, employee master data

**Transaction data:** - It's the data, which data we can access frequently as well as updated frequently.

Ex: - Sales order data, purchase order data

Transactional data is dependent data.

**Organizational data:** - It's the data, which data we can access frequently & updated rarely. Organizational data is created at the time system configures.

Ex: - Company data, branches data

**Size category:** - It determines the space require for the table.

Each table must have at least one field as primary field that should be the character data type that should be in the first field in the table.

C \ 10	C \ 25	C \ 35
Eid	Ename	Eadd

### **Steps to create the table by using direct method: -**

Execute **SE11** in command prompt. Select the data base table. Provide the table name. Click on create. Provide short description name. Provide delivery class 'A'. Select the **display / maintenance allowed**. Click on fields tab. Click on predefined type button. Provide the field names, data types, lengths and short description.

<u>Field</u>	<u>Key</u>	<u>Data type</u>	<u>Length</u>	<u>Short Description</u>
Eid	✓	CHAR	10	Employee ID
Ename		CHAR	25	Name of the employee
Eadd		CHAR	35	Address of employee

Save the table (CTRL + S). Check the table (CTRL + F2). Click on technical settings. Select the data class as **APPL 0** & size category as **zero**. Save the technical settings. Click on back. Then activate table (CTRL + F3).

### Steps to provide the data to the table directly: -

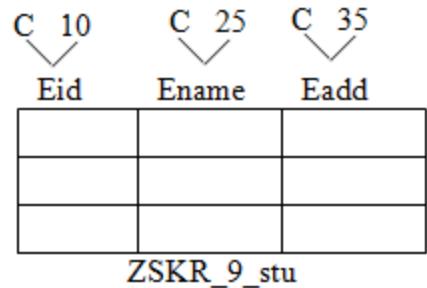
In the menu bar click on utilities → table contents → provide the data. Click on save. Repeat same steps for all employees.

### Steps to display the data from table: -

In the menu bar click on utilities → Table contents display. Click on execute.

#### → Create the following table by using direct method

**Note:** - In the real time, we never create the data base table by using direct method or predefined type. Because, if you want to establish the relation between any two tables, then you must maintain the same domain name in both the tables. In direct method there is no domain concept. So we go for data element method.

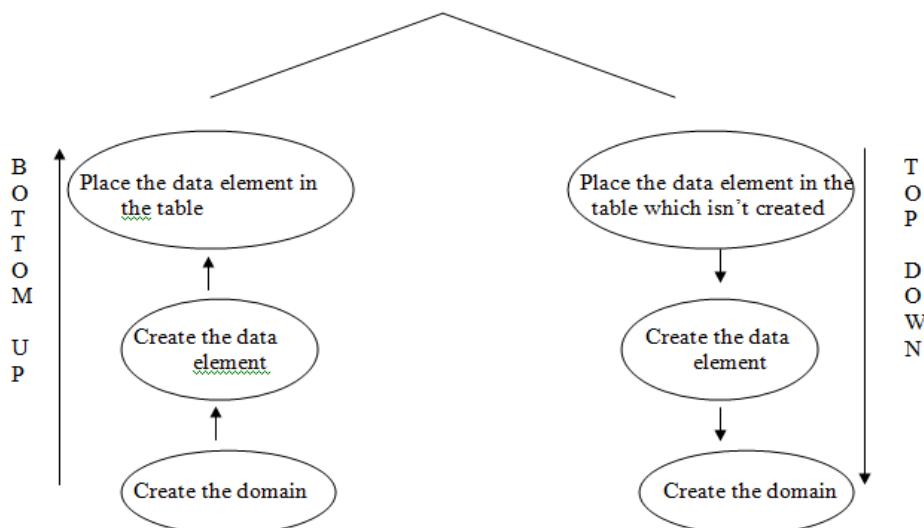


<u>Field</u>	<u>Key</u>	<u>Init</u>	<u>Data Element</u>
Sid	✓	✓	<div style="border: 1px solid black; padding: 10px; width: fit-content;"> <div style="border: 1px solid black; border-radius: 50%; padding: 10px; margin-bottom: 10px;"> <u>Data Type</u>      <u>Length</u>            Char                10         </div> <div style="border: 1px solid black; border-radius: 5px; padding: 5px; margin-bottom: 10px;"> <u>Short description</u>            Student id         </div> <div style="border: 1px solid black; border-radius: 5px; padding: 5px; text-align: center;"> <u>Domain</u> <div style="border: 1px solid black; border-radius: 5px; padding: 5px; text-align: center;"> <u>Data Element</u> </div> </div></div>

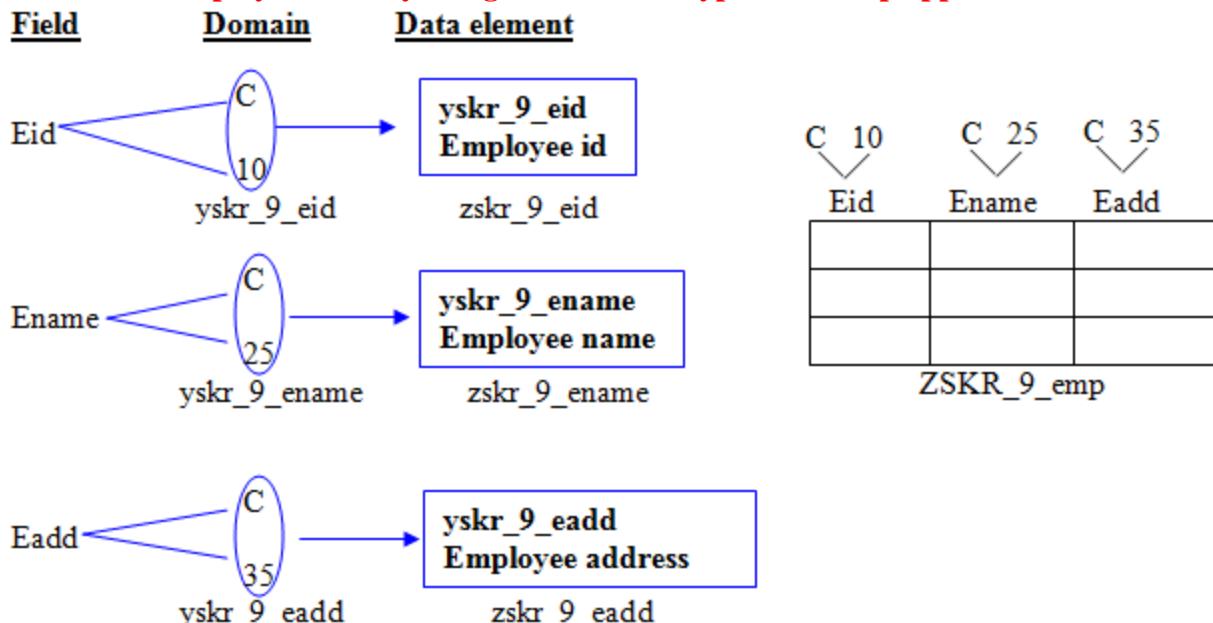
**Domain:** - Domain is the collection of data types & lengths.

**Data Element:** - Data element is the collection of domain with short description

**Create the table by using  
data element type**



→ Create the employee table by using data element type bottom up approach.



**Steps to create the domain:** - Execute SE11. Select the radio button domain. Provide the domain name by click on create. Provide short description. Provide the data type & length. Save the domain. Check the domain. Activate the domain. Repeat the same steps for all domains.

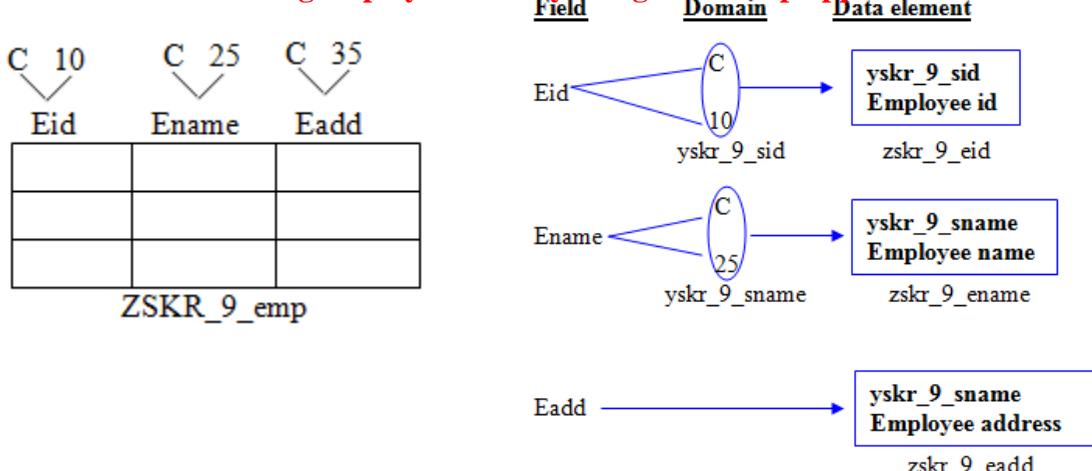
**Steps to create the data element:** - Execute the **SE11**. Select the radio button data type. Provide the data element name. Click on create. Click on enter. Provide short description. Provide domain which is already created. Save the data element. Check the data element. Activate the data element. Repeat the same steps for all the data elements.

**Steps to create the table by using data element type (bottom up approach):** - Execute **SE11**. Select the radio button database table. Provide the table name. Click on create. Provide short description. Provide delivery class is 'A'. Select **Display / Maintenance allowed**. Click on field tab. Provide the field name, data elements.

<u>Field</u>	<u>Key</u>	<u>Data Element</u>
Eid	✓	zskr_9_eid
Ename		zskr_9_ename
Eadd		zskr_9_eadd

Save the table. Check the table. Before activate, click on technical settings. Select the data class, size category. Save the technical settings. Click on back. Activate the table.

→ Create the following employee table by using bottom up approach.



**Note:** - If you get the warning message enhancement category for table missing then in menu bar click on extras → enhancement category → enter → select the radio button can be enhancement (char type or numeric) → enter.

**Note:** - If you want to provide the data directly to the table then you must select the display maintenance allowed.

**Note:** - If you select the display maintenance allow with restrictions, then display only possible. Create records not possible.

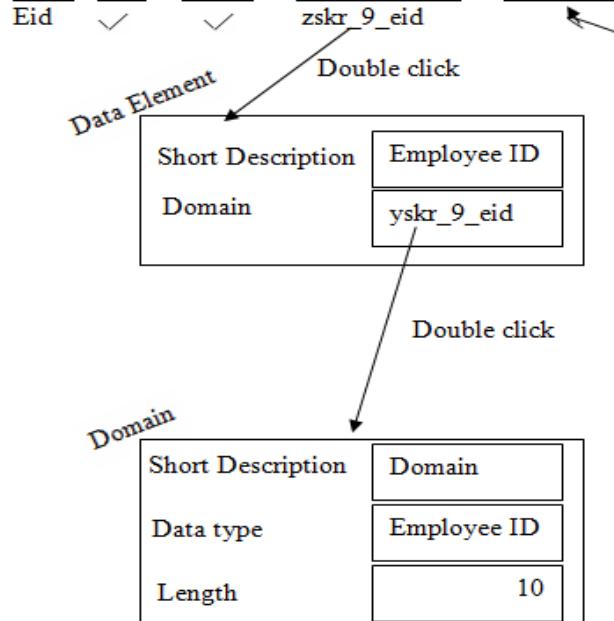
**Note:** - If you select the display maintenance not allowed then display & create integer not possible.

→ **Create the following employee table by using top down approach.**

C 10	C 25	C 35
Eid	Ename	Eadd

ZSKR\_9\_emp1

Field   Key   Initial   Data Element   Data type   Length   Short Description



### Steps to create the table by using top down approach: -

Execute **SE11**. Select the radio button **Data base table**. Provide the table name. Click on create. Provide short description. Provide delivery class is 'A'. Select the **display maintenance allowed**. Click on fields tab. Provide the field name data element which is not created. Double click on data element. Click on yes. Click on local object. Enter. Click on yes. Provide short description. Provide domain name which isn't created. Double click on it. Click on yes. Click on local object. Create the domain. Click on yes. Provide short description, data type, length. Save, check, activate the domain. Click on back. Save, check, activate data element. Click on back. Repeat the same steps for all the other things. Save the table. Check the table. Click on technical settings. Select the data class, size category. Save the technical settings. Click on back. Activate the table.

**Note:** - If you want to display the particular field information, click on utilities on menu bar. Select the table contents, display. In the menu bar click on settings → format list → choose list. Select our required field check box. Enter. Execute.

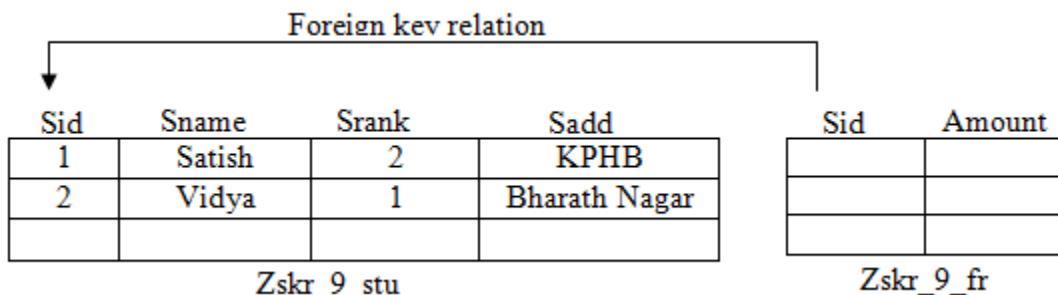
### Foreign key: -

Foreign key is a field in one table. This is connected to another table via foreign key relationship. The purpose is to validate the data being entered in one table (foreign key table) by checking against list of possible values in another table.

### Technical requirement to establish the foreign key relationship: -

1. The domain of the both fields in both the tables must be the same.
2. The check table field must be the primary.

### → Establish the foreign key relation between employee & salary tables.



### Steps to establish the foreign key relation: -

Execute **SE11**. Select the radio button **Data base object**. Provide the foreign key table name. Click on **change**. Select the **fields** for which field we want establish the foreign key relation. Click on **foreign key** icon under fields tab. Provide the check table name. Click on **generate proposal**. Then the system automatically provides the relation between these two tables. Enter. Save, check, activate the table.

### Working with reference fields: -

In the real time when ever we are working with amount field then you must provide reference as currency field.

Amount	Currency
250	Dollars
251	Rupees

In real time when ever we are working with quantity field then you must provide reference as unit of measurement field.

Quantity	Unit of measurement
10	LT
20	KG

Field	Data type
Amount	CURR
Currency	CUKY
Quantity	QUAN
Units	UNIT

→ Create the following material table.

C 3	C 10	C 15	QUAN 10	UNIT 3	CURR 10	CUKY 5
MANDT	MNO	MDES	MQTY	MUOM	Mprice	Mcurrency

In the real time when ever we create the data base table then you must provide the MANDT client field. The table contains MANDT fields then the table is client dependent. That is the data in the table can not reflected to all other clients in the same server. If the table doesn't contain MANDT field then the table is client independent that is the data in data base table automatically reflected to all other clients in the same server.

**Steps to provide the reference fields: -**

Select the quantity field. Click on currency / quantity fields tab. Provide the reference table and reference field.

Field	Data element	Data type	Reference table	Reference field
MQTY	ZSKR_9_MQTY	QUAN	ZSKR_9_MAT	MUOM

Select the amount / price field. Click as currency / quantity fields tab. Provide the reference table, reference field.

Field	Data element	Data type	Reference table	Reference field
Mprice	ZSKR_9_Mprice	CURR	ZSKR_9_MAT	MCUKY

**Note:** - T006 is the standard data base table which contains all the unit of measurements.

**Note:** - In the real time when ever we are working with unit of measurement field then we must establish the foreign key relations with **T006** as a check table. If you want to establish the foreign key relation then you must maintain the SAP domain “**MEINS**” into our field domain.

**Steps to establish the foreign key relation: -**

Select the unit of measurement in our table. Click on foreign key icon under the fields tab. Then the system will give a proposal with values table T006 as check table. Then click on yes. Enter. Save, check, activate the table.

**Note:** - “**TCURC**” is the standard data base table which contains all the currencies.

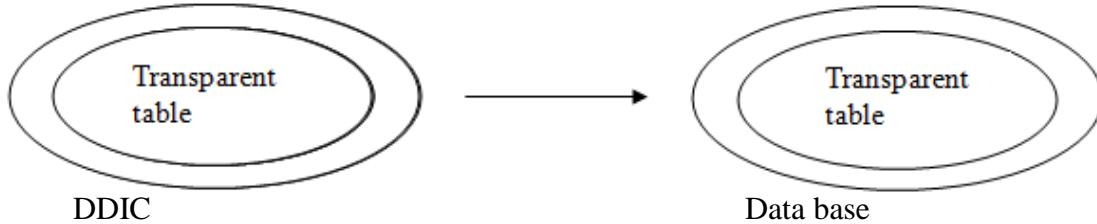
In the real time when ever we are working with currency field then we must establish the foreign key relation with TCURC as check table. If you want to establish the foreign key relation then we must maintain the SAP domain “**WAERS**” into our table field domain.

Select the currency field in our table. Click on foreign key icon under the fields tab. Then the system automatically gives a proposal with TCURC as check table. Click on yes. Enter. Save, check, activate the table.

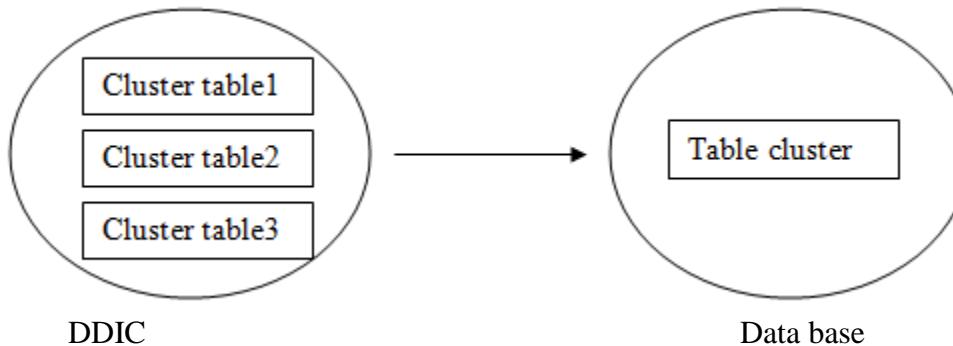
### Types of tables: -

1. Transparent tables
2. Cluster tables
3. Pool tables

**Transparent tables:** - Transparent tables are one to one relationship. That is if you create one transparent table in the data dictionary, then it'll store like only one data base table in the data base.



**Cluster table:** - This tables are many – one relationship. That is if you create the many clustered tables in DDIC then they will form like a table cluster & store in the data base.

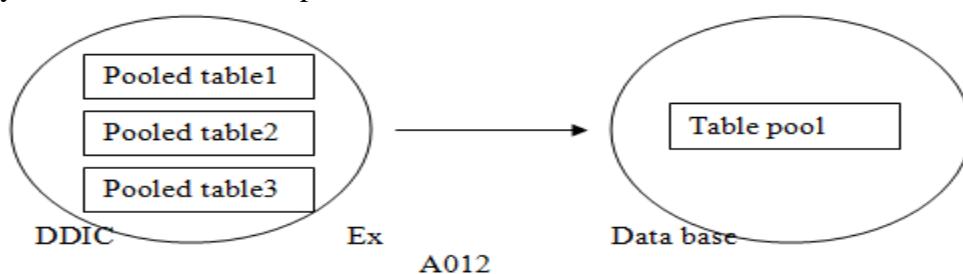


Buffering isn't possible for clustered table. From this reason only fetching the data from clustered table take more time.

Clustered table is suitable when you fetch the fewer amount of data from more fields.

### Pooled table: -

Pooled tables are many – one relationship. That means if you create many pooled tables in DDIC, then they will form like a table pool & stored in the data base.



Pooled tables are suitable when we fetch the large amount of data from fewer fields.

**Note:** - Joins aren't possible for Pooled tables.

**→ Create the table pool. Create pooled table & attached to the table pool.**

### Steps to create table pool: -

Execute SE11. In the menu bar click on utilities → other dictionary objects. Select the radio button table pool. Provide the table pool name. Click on create. Select the radio button table pool. Enter. Provide short description. Click on save. In the menu bar click on go to → technical settings. Select the size category. Save. Back. Check & activate the table pool.

### **Steps to create pooled table: -**

Execute SE11. Select the radio button data base table. Provide the pooled table name. Click on create. Provide short description. Provide delivery class is 'A'. Click on fields tab. Provide the field names, data element. Save the table. In the menu bar, click on extras → change table category. Select the radio button pooled table. Enter. Click on delivery & maintenance tab. Provide the table pool. Save, check the table. Click on technical settings. Select the data class size category. Save the technical settings. Click on back. Activate the table.

**→ Create the table cluster. Create clustered table & attached to the table clustered.**

**When ever we create the clustered table then we must maintain the primary fields in the table clustered. When ever we create the clustered table then we must maintain at least one non primary key.**

### **Steps to create table cluster: -**

Execute SE11. In the menu bar click on utilities → other dictionary objects. Select the radio button table pool / cluster. Provide the table cluster. Click on create. Select the radio button table cluster. Enter. Provide short description. Click on save. Select the VARKEY & click on minus. Click on + button. Provide the mandatory field of clustered table.

Field name	key	Data type	Length
MANDT	✓	CLNT	3
LIFNR	✓	CHAR	10

Click on save. Click on go to on menu bar → technical settings. Click on back. Check, activate.

### **Steps to create clustered table: -**

Execute SE11. Select the radio button data base table. Provide the table name. Click on create. Provide short description. Provide deliver class is 'A'. Click on field tab. Provide field names, data element.

Field	Key	Initial	Data element
MANDT	✓	✓	MANDT
LIFNR	✓	✓	LIFNR
NAME1			NAME1

Click on save. In the menu bar click on extras → change category. Select the radio button clustered table. Enter. Click on delivery & maintenance tab. Provide the table clustered. Save, check the table. Click on technical settings. Provide data class, size category, save the technical settings. Click on back. Activate the table.

**Note:** - In the real time we never create cluster & pool tables. Only we can create transparent tables.

**Note:** - In the real time when ever we're altering the table we get the error structure change of field error. Then we must adjust the data base.

### **Steps to adjust the data base: -**

Click on utilities in menu bar → data base object → data base utility. Click on activate & adjust data base. Click on yes. Now the table is activated.

### **Index: -**

Indexes are used to improve the performance of the select query. There are two types of indexes.

1. Primary index
2. Secondary index

**Primary index:** - Primary index is the primary fields. Without a primary index we can't create the data base table. We can place up to 16 primary indexes per table. We can create the primary index only for custom tables. Not for standard tables.

**Secondary index:** - Secondary index is possible for other than primary fields. Without a secondary index we can create the data base table. We can create up to 9 secondary indexes per a table. We can create secondary index for both standard & custom tables.

**Note:** - We can't delete the domain which is already assigned to data element. We can't delete the data element which is already assigned to data base table. We can create the data base table with out a data element by using direct method or predefined type.

### **Steps to create secondary index: -**

Execute SE11. Select the radio button data base table. Provide the table name for which table we want to display. Click on display. Click on index beside technical settings. Click on create index. Provide the index name. Enter. Provide short description. Click on table fields. Select our require fields. Enter. Save, check, activate.

```
select customer name ort01 from ZTEST099 %_HINTS ORACLE 'INDEX("ZTEST099" "ZTEST099~123")' into table it_t001.
```

### **Some of field names in T001: -**

BUKRS → Company code

BUTXT → Company name

ORT01 → City

LAND1 → Country

### **Some of field names in KNA1: -**

KUNNR → Customer number

NAME1 → Name

ORT01 → City

LAND1 → Country

SPRAS → Language

### **Some of the field names in LFA1 : -**

LIFNR → Vendor number

NAME1 → Name

ORT01 → City

LAND1 → Country

SPRAS → Language

Differences between data base table and structure

#### **Data base table**

1. Data base table must contain at least one field as primary field.
2. Data base table contains the permanent data.
3. We must provide the size of the data base table.
4. We must provide the delivery class to the table.

Adding some additional fields to the data base table

#### **Custom Table ('Z' table) Ex: ZSKR**

1. By using include structure, we can add additional fields to the custom table.
2. The same structure can be included in any number of custom tables.
3. All steps of include structure saved either in our own package or in \$TMP package.

#### **Standard table Ex: - T001**

1. By using Append structure we can add additional fields to the standard data base table.
2. The same structure can be appended to only one standard data base table.
3. All steps of append structure must be saved in our own package only. Because \$TMP is non transportable.

#### **Structure**

1. Structure doesn't contain any primary fields.
2. Structure doesn't contain any data.
3. We only to provide the size of the structure.
4. We no need to provide the delivery class to the structure.

**Note:** - Adding the following fields to the custom tables ZSKR\_9\_EMP1 and ZSKR\_9\_STU by using include structure.

ORT01 → City
LAND1 → Country
SPRAS → Language

If you want to add block of fields or set of fields into more than one custom table instead of maintaining those fields in each and every table it's better to create one structure with those fields & later we include the same structure in all tables.

#### **ZSKR\_9\_EMP**

Eid  
Ename  
Eadd

ORT01
LAND1
SPRAS

#### **ZSKR\_9\_STU**

Sid  
Sname  
Sadd

ORT01
LAND1
SPRAS

In this object or task we must create one structure with ORT01, LAND1, SPRAS & later we include this structure in the ZSKR\_9\_EMP & ZSKR\_9\_STU custom data base tables.

**Note:**

'/O' is used to opens a new session without terminating the current session.  
'/N' is used to opens a new session with terminating the current session.

#### **Steps to create the structure: -**

Execute SE11. Select the radio button data type. Provide the structure name. Click on create. Select the radio button structure. Enter. Provide short description. Provide component, component type.

<b>Component</b>	<b>Component type</b>
ORT01	ORT01
LAND1	LAND1
SPRAS	SPRAS

Save the structure, check, execute the structure.

#### **Steps to include the same structure in any number of custom tables: -**

Execute SE11. Select the radio button data base table. Provide the table name. Click on change. Place the cursor where you want to include the structure. Provide the field name ".include". Provide the data element as structure name. Enter. Save, check, activate the table.

#### **Steps to create our own package: -**

Execute **SE80**. Click on edit object. Select development coordination. Click on package radio button. Write your package name. Press F5 button or click on create. Provide short description. Click on enter or Save. Click on enter.

**Note:** In the real time packages are created by BASIS people. Based on the module or based on the object category & request number is created by ABAPer.

→ Add the CMM level field which is CHAR & length '1' to the T001 data base table through APPEND structure.

### Steps to add some additional fields to the standard data base table: -

Execute SE11. Select the radio button data base table. Provide the table name for which table we want to add some additional fields. Click on display. Click on append structure on application tool bar. Click on create append. Provide the append structure name. Enter. Provide short description. Provide component, component type, double click on component type or data element. Click on yes. Click on yes. Save in our package. Enter. Enter. Provide short description. Provide the domain which isn't created. Double click on domain. Click on yes. Save in our own package. Enter. Click on yes. Provide the short description. Provide data type, size. Save, check, activate domain. Click on back. Save, check, activate the data element. Repeat the same steps for all components. Save the structure, check, activate.

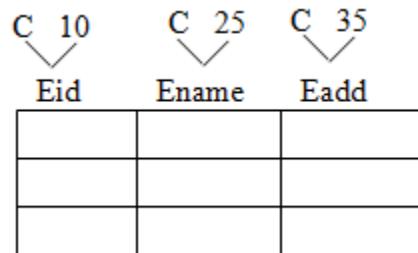
**Note:** In the real time when ever we develop any task then we must save in our own package & request number not in \$TMP.

### Syntax of declaring the structure / table in ABAP editor: -

Data : Begin of <structure / table>,

-----  
----- } List of fields  
-----

End of <structure / table>.



**Ex: -**

**Data:** Begin of emp,  
    Eid(10) type c,  
    Ename(25) type c,  
    Eadd(35) type c,  
End of emp.

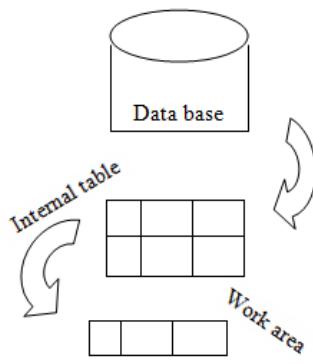
} Work area

Work area always holds only one record at a time. So we go for internal tables.

### Internal table: -

1. Internal table is the collection of records.
2. Internal tables are temporary tables. That means the data in internal table won't save anywhere in SAP.
3. Internal tables are dynamic memory location. That means we no need to provide the size of the internal table.
4. The scope of the internal table is up to that program.
5. Placing the data to the internal table as well as getting the data from internal table is always record by record.

### Backend picture of SAP ABAP: -



## **Differences between data base table & internal tables: -**

### **Data base table**

1. Data base tables are permanent storage location.
2. We can access the data base table from anywhere in SAP.
3. We must provide the size of the data base tables.

### **Internal table**

1. Internal tables are temporary storage location.
2. We can access the internal table with in the program only.
3. Internal tables are dynamic. So we no need to provide the size.

## **Syntax of declaring the internal table : -**

Data <internal table> like table of <work area>.

## **Syntax of accessing the fields from work area :-**

<work area name> - <file name>

Ex: WA – Eid.

**Note:** - Append is the key word to transfer the data from work area to internal table.

```

Data: Begin of WA,
      Eid(10) type c,
      Ename(25) type c,
      Eadd(35) type c,
      End of WA.

Data IT like table of WA.
WA-Eid = '1'.
WA-Ename = 'HARSHA'.
WA-Eadd = 'TIRUPATHI'.
Append WA to IT.
WA-Eid = '2'.
WA-Ename = 'SATISH'.
WA-Eadd = 'HYDERABAD'.
Append WA to IT.
Loop at IT into WA.
Write: / WA-Eid, WA-Ename, WA-Eadd.
Endloop.

```

1	HARSHA	TIRUPATHI
2	SATISH	HYDERABAD

## **T001 (company codes)**

ORT01 → City  
 LAND1 → Country  
 BUKRS → Company code  
 BUTXT → Company name

## **KNA1 (customer master table)**

KUNNR → Customer number  
 NAME1 → Name  
 SPRAS → Language

## **Syntax to select query: -**

Select <field1> <field2> .... from <data base table> into table <internal table> where <condition>

**→Display the company codes company names & cities**

```

Data: Begin of WA,
      BUKRS(4) type C,
      BUTXT(25) type C,

```

```

ORT01(25) type C,
End of WA.
Data IT like table of WA.
Select BUKRS BUTXT ORT01 from T001 into table IT.
Loop at IT into WA.
Write:/ WA-BUKRS, WA-BUTXT, WA-ORT01.
Endloop.

```

**→ Display the customer numbers, customer names, cities & countries.**

```

Data: Begin of WA,
      KUNNR(10) type C,
      NAME1(35) type C,
      ORT01(35) type C,
      LAND1(3) type C,
      END of WA.
Data IT like table of WA.
Select KUNNR NAME1 ORT01 LAND1 from KNA1 into table IT.
Loop at IT into WA.
Write:/ WA-KUNNR, WA-NAME1, WA-ORT01, WA-LAND1.
Endloop.

```

**Syntax of declaring the field:** -

```

Data <field name>(<length>) type <data type>
      (OR)
Data <field name> type <data elements>
      (OR)
Data <field name> type <data base table> - <field name>
Ex: - Data BUKRS(4) type C.
      (OR)
      Data BUKRS type BUKRS
      (OR)
      Data Bukrs type T001-Bukrs

```

**→ Display the company codes, company names & cities.**

```

Data: Begin of WA,
      BUKRS type T001-BUKRS,
      BUTXT type T001-BUTXT,
      ORT01 type T001-ORT01,
      End of WA.
Data IT like table of WA.
Select BUKRS BUTXT ORT01 from T001 into table IT.
Loop at IT into WA.
Write:/ WA-BUKRS, WA-BUTXT, WA-ORT01.
Endloop.

```

Select-options is the keyword which accepts the single value, multiple single values, single range & multiple ranges.

**Syntax of select-options:** -

Select-options <name of the select-options> for <variable name>.

**Ex:** -

Data v1 type T001-BUKRS.

Select-options S\_BUKRS for v1.



S\_BUKRS  to   
(c,4)

Data v1 type KNA1-KUNNR.

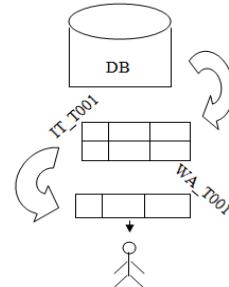
Select-options S\_KUNNR for v1.

### **Syntax of select-options in select query: -**

Select <field1> <field2> ---- from <database table> into table <Internal Table> where <field> in <select-options>.

### **→ Based on given company codes, display the company codes, company names & cities.**

```
Data v1 type T001-Bukrs.  
Select-options S_BUKRS for v1.  
Data: Begin of WA_T001,  
      BUKRS type T001-BUKRS,  
      BUTXT type T001-BUTXT,  
      ORT01 type T001-ORT01,  
      End of WA_T001.  
Data IT_T001 like table of WA_T001.  
Select BUKRS BUTXT ORT01 from T001 into table IT_T001 where BUKRS in  
S_BUKRS.  
Loop at IT_T001 into WA_T001.  
  Write: / WA_T001-BUKRS, WA_T001-BUTXT, WA_T001-ORT01.  
Endloop.
```



### **→ Based on given country display the customer numbers, customer names & countries.**

```
Data v1 type KNA1-LAND1.  
Select-options S_LAND1 for v1.  
Data: Begin of WA_KNA1,  
      KUNNR type KNA1-KUNNR,  
      NAME1 type KNA1-NAME1,  
      LAND1 type KNA1-LAND1,  
      End of WA_KNA1.  
Data IT_KNA1 like table of WA_KNA1.  
Select KUNNR NAME1 LAND1 from KNA1 into table IT_KNA1 where LAND1 in  
S_LAND1.  
Loop at IT_KNA1 into WA_KNA1.  
  Write: / WA_KNA1-KUNNR, WA_KNA1-NAME1, WA_KNA1-LAND1.  
Endloop.
```

### **Parameter: -**

It's the keyword which accepts the single value at run time.

### **Syntax:-**

Parameter <name of the parameter> type <data type>

Ex: - parameter P\_BUKRS type T001-BUKRS.



P\_BUKRS

### **Syntax of parameter in select query: -**

Select <field1> <field2> ---- from <database table> into table <Internal Table> where field = <parameter>

Ex: -

Select BUKRS BUTXT LAND1 from T001 into table IT\_T001 where BUKRS = P\_BUKRS.

### Differences between parameter & select-options: -

#### Parameter

1. Parameter is the keyword which accepts the single value at run time.
2. Without provide input parameter we can't get data.

#### Select-options

1. Select-options is the keyword which accept the single value, multiple single values, single range & multiple ranges.
2. Without providing input select-options we can get the entire data from database.

### Types of Internal Table: -

- 1 Indexed
  - Standard
  - Sorted
- 2 Hashed

### Standard Internal Table: -

1. It accepts the duplicate records.
2. Here all fields are non-unique.
3. Passing data from work area to internal table is always through **Append** keyword.
4. Searching of a record is **linear** search.

### Syntax: -

Data <Internal Table> like standard table of <work area>.

### Sorted Internal Table: -

1. It may or mayn't accept the duplicate.
2. Here, we must specify at least one field as unique \ non-unique.
3. Pushing data from work area to internal table is always through **Insert** keyword.
4. Searching of a record is **Binary search**

### Syntax: -

Data <Internal Table> like sorted table of <work area> with unique / non-unique key <field> ---- .

Ex: - data IT like sorted table of WA with unique key Eid.

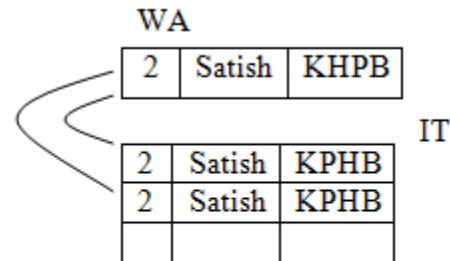
### Hashed internal table: -

1. It won't accept the duplicate records
2. Here we must specify at least one field as unique
3. Pushing data from work area to internal table is always through **Collect** keyword.
4. Searching of a record is by using Hash algorithm.

### Syntax: -

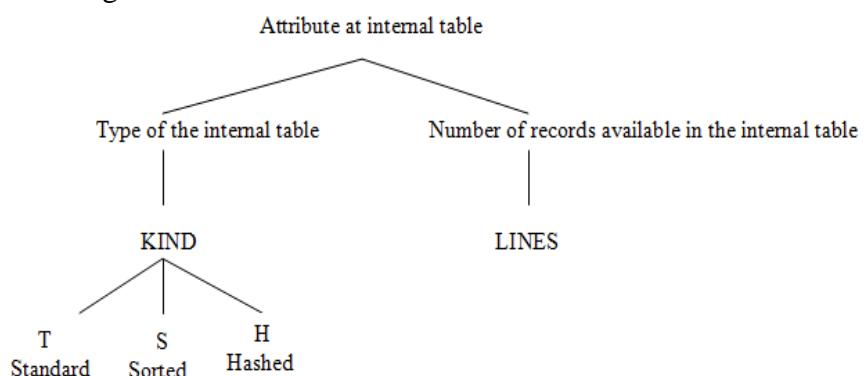
Data <internal table> like Hashed table of <work area> with unique / non-unique key <field1> <field2> --.

In the real time most of the times we use standard internal table because we are working with data of data base table. In the data base there is no duplicate data. Some times we use Hashed internal tables.



Append WA to IT

Append WA to IT



Kind is the keyword which returns the type of internal table. If the internal table type is standard then writes T. Sorted writes S. Hashed writes H. Lines is the keyword which returns the number of records in internal table.

#### **Syntax:-**

Describe table <internal table> KIND <variable1> lines <variable2>  
                                    ↓                            ↓  
                                    Optional                  Optional

By default variable is the character data type & length is 1.

```
Data v1.  
Data v2 type i.  
Data: Begin of WA_KNA1,  
      KUNNR type KNA1-KUNNR,  
      NAME1 type KNA1-NAME1,  
      ORT01 type KNA1-ORT01,  
      End of WA_KNA1.  
Data IT_KNA1 like table of WA_KNA1.  
Select KUNNR NAME1 ORT01 from KNA1 into table IT_KNA1.  
Describe table IT_KNA1 KIND v1 LINES v2.  
Write: / v1, v2.
```

By default internal table is standard internal table.

#### **MARA (Material Master table)**

MATNR → Material number

MTART → Material type

MATKL → Material group

MEINS → Unit of measurement

#### **Types of declaring Internal tables:-**

##### **1. Declaring the internal table with some of the fields from any one of the database table:-**

#### **Syntax:-**

Data: begin of <work area>,  
-----  
-----

End of <work area>.

Data <IT> like table of <work area>.

#### **Ex:-**

```
Data: Begin of wa_t001,  
      Bukrs type t001-bukrs,  
      Ort01 type t001-ort01,  
      end of wa_t001.  
Data IT like table of wa_t001.
```

##### **2. Declaring the internal table with all fields from any one of data base tables:**

#### **Syntax:-**

Data begin of <work area>.

Include structure <data base table>.

Data end of <work area>.

Data <internal table> like table of <work area>.

Ex:-

```

Data begin of wa_t001.
    Include structure t001.
Data end of wa_t001.
Data IT like table of wa_t001.

```

**3. Declaring the internal table by referring database table: -**

Syntax:-

Data <work area> like <data base table>.  
Data <Internal Table> like table of <work area / database table>

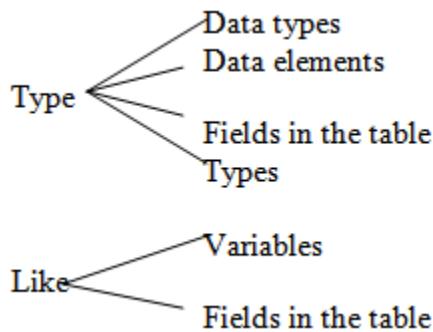
Ex:-

```

Data wa like t001.
Data IT like table of wa.
(OR)
Data IT like table of t001.
Data wa like line of IT.

```

**Note:** – Type is used to refer the data types, data elements, fields in the table & types. Where as like is used to refer the variable or fields in the table.



**Types** is used to assign the data type to the data object directly while declaration. **Like** is used to assign the data type of another object to declare the data object. The data type is referred indirectly.

**4. Declaring the internal tables by using types keyword: -**

Syntax:-

Types: Begin of <type name>,  
-----  
-----  
-----

End of <type name>.

Data <wa> type <table name / type name>.  
Data <internal table> like table of <work area>.  
(or)  
Data <internal table> type table of <type name>.

Ex: -

```

Types: begin of ty_t001,
    Bukrs type t001-bukrs,
    Ort01 type t001-ort01,
    End of ty_t001.
Data wa_t001 type ty_t001.
Data IT type table of ty_t001.
(Or)
Data IT like table of wa_t001.

```

**Note:** - If you want to comment the multiple lines then select the all lines. Click on CTRL + Less than. If you want to uncomment then select all lines, click on CTRL + Grater than.

### → Display the all companies & their entire fields information

```
* Data wa_t001 like t001.  
* Data IT_t001 like table of wa_t001.
```

```
-----  
* Data IT_t001 like table of t001.  
* Data wa_t001 like line of IT_t001.
```

```
Types begin of ty_t001.  
    Include structure t001.  
Types end of ty_t001.  
Data wa_t001 type ty_t001.  
Data IT_t001 type table of ty_t001.  
Select * from t001 into table it_t001.  
Loop at IT_t001 into wa_t001.  
    Write: / wa_t001-bukrs, wa_t001-butxt.  
Endloop.
```

### Internal table with header line: -

By default with header line creates one work area with the name of internal table. That means the name of the work area as well as name of the internal table as similarly.

### Declaring the IT with header by using occurs keyword: -

#### Syntax: -

Data: begin of <internal table> occurs 0,

```
-----  
-----  
-----
```

End of <internal table>.

Ex: -

```
Data: begin of IT_t001 occurs 0,  
      Bukrs type t001-bukrs,  
      Ort01 type t001-ort01,  
      End of IT_t001.
```

Here occurs 0 by default allocates 8 kb of memory for the internal table. If the data in the internal table exceeds 8 kb, then it'll provide one more 8 kb of memory ... up to 2 GB.

**Occurs 'n': -** By default it allocates 'n' records of memory for internal table. If the data in the internal table exceeds 10 records. Then it'll be allocates one more 'n' records of memory ---- up to 2 GB.

IT_t001	Bukrs	ort01

IT_t001	Bukrs	ort01

### Declaring the internal table with header line by using types keyword: -

Types: begin of <type name>.

```
-----  
-----
```

End of <type name>.

Data <internal table> type table of <type name> with header line.

IT_t001	Bukrs	ort01

IT_t001	Bukrs	ort01

Ex:

```
Types: begin of ty_t001,
        Bukrs type t001-bukrs,
        Ort01 type t001-ort01,
        End of ty_t001.
```

Data it\_t001 type table of ty\_t001 with header line.

**Note:** - By default tables keyword creates one **work area** with the name of **database table name** and also it contains all the fields of data base table.

Syntax: -

Tables<database table name>.

Tables t001.

Bukrs	butxt	ort01	-----

1 Append t001 to IT\_t001.

2. Loop at it\_t001 into t001.

**Initializing techniques:** -

1. Clear
2. Refresh
3. Free

**Clear:** - clear clears the contents of the work area & also clear is used to clear the contents of the internal table.

Syntax: -

Clear <work area>

Clear <internal table>

Ex:-

Clear WA\_t001.

Clear IT\_t001.

Bukrs	butxt	Ort01
100	TCS	Hyd

After clear ↓

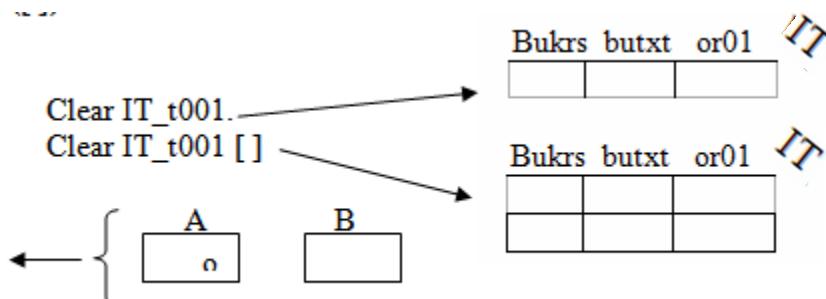
Bukrs	butxt	Ort01

If we are working with internal table with header line then the name of the work area as well as name of the internal table is similar name. In this case also clear clears the contents of the work area only. If you want to clear the contents of internal table then we place open & close ([ ]) brackets to the internal table.

Bukrs	butxt	or01

Ex:-

```
Data A type I.
Data B(5) type C.
A = 10.
B = 20.
Clear: A, B.
Write:/ A, B.
```



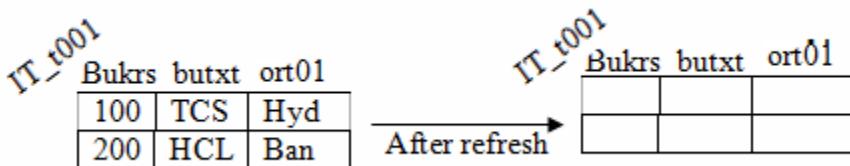
### **Refresh:-**

Refresh always clears the contents at the internal table only.

### **Syntax: -**

Refresh <internal table>.

**Ex:** - Refresh IT\_t001.



**Note:** - If you're working with internal table with header line also refresh always clear the content of the internal table.

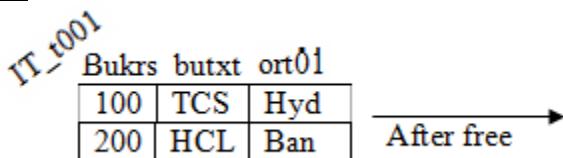
### **Free: -**

Free acts like refresh.

1. Refresh clears the contents of internal table only. Not the memory which is allocated for that.
2. Free clears the contents of the internal table as well as memory which is allocated for that.

**Syntax: -** Free <internal table>.

**Ex:** - Free IT\_t001.



### **Operations on Internal Table: -**

1. Pushing data from work area to internal table by using
  - a. Append → Standard table
  - b. Insert → Sorted table
  - c. Collect → Hashed table
2. Reading the data from internal table
 

Single Record
Multiple Records (Loop at)

  3. Modify the data in an internal table by using **modify** keyword.
  4. Delete the data in an internal table by using **delete** keyword.
  5. Sort the data in an internal table by using **sort** keyword.

**Append:** - Append is the keyword to transfer the data from work area to at the last record of internal table.

### **Syntax: -**

Append <work area> to <internal table>.

**Ex:** - Append WA\_t001 to IT\_t001.

**Insert:** - Insert is the keyword to transfer the data from work area to internal table based on the key field.

**Syntax:** -

Insert <work area> into table <internal table>.

**Ex:** -

Insert WA into table IT.

**Note:** - If we use insert keyword to standard internal table then it acts like append.

**Collect:** - Collect checks the internal table whether the record is available or not based on the key field. If not it acts like append (record adds last). Other wise it adds the numeric fields from work area to number field in the internal table.

**Syntax:** -

Collect <WA> into <IT>.

**Ex:** - Collect WA into IT.

Whether we are working with collect keyword then we must declare **other than numeric** (data types) fields are unique.

Sno	Esal
3	5000

Sno	Esal
1	4000
2	6000
3	2000
4	3000
7	5000

→ 2000+5000 = 7000

**Reading a single record from internal table based on index**

**Syntax:** -

Read table <internal table> into <work area> index <no>.

**Ex:** -

Read table IT into WA index 2.

We never use 1<sup>st</sup> condition in real time.

**Reading a single record from the internal table based on condition**

**Syntax:** -

Read table <internal table> into <work area> with key condition.

**Ex:** -

Read table IT into WA with key BUKRS = '2000'.

**OP:** - 1000 IBM Chennai

**Reading multiple records from the internal table based on index.**

**Syntax:** -

Loop at <internal table> into <work area> from <index no> to <index no>.

optional

-----

-----

-----

Endloop.

**Ex:** -

Loop at IT into WA from 2 to 3.

Write: / WA\_t001-BUKRS, WA\_t001-BUTXT, WA\_t001-ORT01.

Endloop.

**OP:** - 200 IBM Chennai

एम एन सतीष कुमार रेडि

BUKRS	BUTXT	ORT01
100	TCS	Hyderabad
200	IBM	Chennai
300	HCL	Hyderabad
400	HP	Mumbai
500	CSC	Bangalore

300 HCL Hyderabad

### **Reading multiple records from the internal table based on condition**

#### **Syntax:**

Loop at <internal table> into <work area> where <condition>.

-----

-----

-----

Endloop.

#### **Ex:**

Loop at IT\_t001 into WA\_t001 where ORT01 = 'Hyderabad'.

Write: / WA\_t001-BUKRS, WA\_t001-BUTXT, WA\_t001-ORT01.

Endloop.

#### **OP:**

100	TCS	Hyderabad
300	HCL	Hyderabad

### **Modify the data in internal table by using modify keyword**

This is 2 step procedure.

1. Fill the latest information into the work area.

2. Modify the internal table based on work area.

#### **Syntax:**

Modify <internal table> from <work area> transporting <field1> <field2> where <condition>.

#### **Ex:**

WA\_t001-ORT01 = 'BAN'.

Modify IT\_t001 from WA\_t001 transporting ORT01 where BUKRS = '2000'.

If we are maintaining all the fields information in the work area (key field)

**Syntax:** - Modify <internal table> from <work area>.

### **Delete the data from internal table by using delete keyword**

#### **Based on index:**

#### **Syntax:**

Delete <internal table> index <index no>.

#### **Ex:**

Delete IT\_t001 index 3.

#### **Based on condition:**

#### **Syntax:**

Delete <internal table> where <condition>.

#### **Ex:**

Delete IT\_t001 where ORT01 = 'Hyderabad'.

### **Sort the data in an internal table**

#### **Syntax:**

Sort <internal table> by <field1> <field2>.

#### **Ex:**

Sort IT\_t001 by BUKRS.

BUKRS	BUTXT	ORT01
100	TCS	Hyderabad
200	IBM	Chennai
300	HCL	Hyderabad
400	HP	Mumbai
500	CSC	Bangalore

OP: -

By default sort is ascending order. If you want to descending order

Syntax: - Sort <internal table> by <field 1> descending.

Ex: - Sort IT\_t001 by BUKRS descending.

Syntax of delete adjacent duplicates: -

Delete adjacent duplicates from <internal table> comparing <field 1> <field 2>.

Note: - Before using the delete adjacent duplicates, to delete duplicates we must sort the table based on comparing fields.

Ex: - Delete adjacent duplicates from IT comparing BUKRS.

✓ 100	TCS	Hyderabad	200	IBM	Chennai
100	TCS	Hyderabad	100	TCS	Hyderabad
100	TCS	Hyderabad	300	HCL	Hyderabad
✓ 200	IBM	Chennai	400	HP	Mumbai
200	IBM	Chennai	100	TCS	Hyderabad
✓ 300	HCL	Hyderabad	400	HP	Mumbai
✓ 400	HP	Mumbai	100	TCS	Hyderabad
400	HP	Mumbai	200	IBM	Chennai

← After sorted

## Transferring the data from one internal table to another table which are similar structure

If the second internal table is empty

Syntax: -

<internal table 2> = <internal table 1>.

Ex: -

IT = IT\_t001.

If the second internal table has data

Syntax: -

Append lines of <internal table 1> to <internal table 2>.

Ex: -

Append lines of IT\_t001 to IT.

Syntax: -

Insert lines of <internal table 1> into table <internal table 2>.

Ex: -

Insert lines of IT\_t001 into table IT.

```
Data: begin of WA,
      Eid(10) type C,
      Esal type I,
      End of WA.
```

```
Data IT like hashed table of WA with unique key Eid.
```

```
WA-Eid = '1'.
```

```
WA-Esal = '10000'.
```

एम एन सतीष कुमार रेडि

IT\_t001

bukrs	butxt	ort01
100	TCS	Hyd
200	IBM	Che
300	HCL	Ban

WA\_t001

bukrs	butxt	ort01

IT

bukrs	butxt	ort01
400	HP	Ban

X Loop at IT\_t001 into WA\_t001.  
Append WA\_t001 to IT.  
Endloop.

```
Collect WA into IT.
```

```
WA-Eid = '2'.
WA-Esal = '30000'.
Collect WA into IT.
WA-Eid = '4'.
WA-Esal = '60000'.
Collect WA into IT.
```

```
Loop at IT into WA.
  Write: / WA-Eid, WA-Esal.
Endloop.
Uline.
```

```
WA-Eid = '3'.
WA-Esal = '30000'.
Collect WA into IT.
```

```
Loop at IT into Wa.
  Write: / WA-Eid, WA-Esal.
Endloop.
```

Working with collect	
1	10000
2	30000
4	60000
1	10000
2	30000
4	60000
3	30000

```
Types: begin of ty,
      Eid(10) type C,
      Ename(25) type C,
      End of ty.
Data WA type ty.
Data IT like sorted table of WA with unique key Eid.
```

```
WA-Eid = '1'.
```

```
WA-Ename = 'HARSHA'.
```

```
Insert WA into table IT.
```

```
WA-Eid = '3'.
```

```
WA-Ename = 'Latha'.
```

```
Insert WA into table IT.
```

```
Loop at IT into WA.
```

```
  Write : / WA-Eid, WA-Ename.
```

```
Endloop.
```

```
Uline.
```

```
WA-Eid = '2'.
```

```
WA-Ename = 'SATISH'.
```

```
Insert WA into table IT.
```

```
Loop at IT into WA.
```

```
  Write: / WA-Eid, WA-Ename.
```

```
Endloop.
```

→ Based on the employee, salary internal table data display the Eid, Ename and Esal.

```
Data: begin of WA_Emp,
      Eid(10) type C,
      Ename(25) type C,
      End of WA_Emp.
```

1	HARSHA
3	LATHA
<hr/>	
1	HARSHA
2	SATISH
3	LATHA

```

Data IT_Emp like table of WA_Emp.
Data: begin of WA_Sal,
      Eid(10) type C,
      Esal(10) type C,
      End of WA_Sal.
Data IT_Sal like table of WA_sal.

```

```

Data: begin of WA,
      Eid(10) type C,
      Ename(25) type C,
      Esal(10) type C,
      End of WA.
Data IT like table of WA.

```

\* filling the employee information

```

WA_Emp-Eid = '1'.
WA_Emp-Ename = 'Vidya'.
Append WA_Emp to IT_Emp.
WA_Emp-Eid = '2'.
WA_Emp-Ename = 'Satish'.
Append WA_Emp to IT_Emp.
WA_Emp-Eid = '3'.
WA_Emp-Ename = 'Latha'.
Append WA_Emp to IT_Emp.
WA_Emp-Eid = '4'.
WA_Emp-Ename = 'Sanjay'.
Append WA_Emp to IT_Emp.

```

Loop at IT\_Emp into WA\_Emp.

```

Write: / WA_Emp-Eid, WA_Emp-Ename.
Endloop.
Uline.

```

\* Filling the data salary internal table

```

WA_Sal-Eid = '2'.
WA_Sal-Esal = '20000'.
Append WA_Sal to IT_Sal.
WA_Sal-Eid = '3'.
WA_Sal-Esal = '10000'.
Append WA_Sal to IT_Sal.
WA_Sal-Eid = '4'.
WA_Sal-Esal = '15000'.
Append WA_Sal to IT_Sal.
WA_Sal-Eid = '1'.
WA_Sal-Esal = '16000'.
Append WA_Sal to IT_Sal.

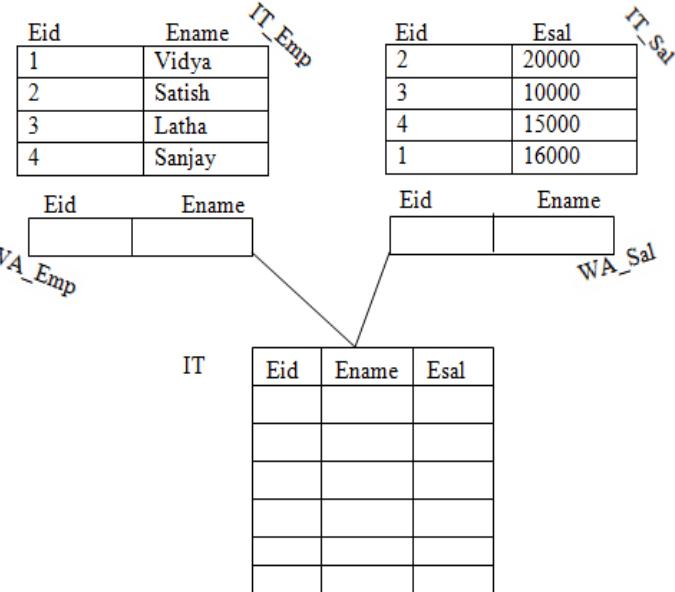
```

Loop at IT\_Sal into WA\_Sal.

```

Write: / WA_Sal-Eid, WA_Sal-Esal.
Endloop.

```



```

Uline.
* Based on emp and salary table fill the final table
Loop at IT_Emp into WA_Emp.
  WA_Eid = WA_Emp-Eid.
  WA_Ename = WA_Emp-Ename.
  Read table IT_Sal into WA_Sal with key Eid = WA_Emp-Eid.
  WA_Esal = WA_Sal-Esal.
  Append WA to IT.
  Clear WA.
Endloop.
Loop at IT into WA.
  Write: / WA_Eid, WA_Ename, WA_Esal.
Endloop.

```

→ Update the rank field in the student internal table based on rank internal table data.

```

Data: begin of WA_Stu,
      Sid(5) type C,
      Sname(25) type C,
      Srank(5) type C,
      End of WA_Stu.

Data IT_Stu like table of WA_Stu.

Data: begin of WA_rank,
      Sid(5) type C,
      Srank(5) type C,
      End of WA_rank.

Data IT_rank like table of WA_rank.

* Filling the student internal table
WA_Stu-Sid = '1'.
WA_Stu-Sname = 'Vidya'.
Append WA_Stu to IT_Stu.

WA_Stu-Sid = '2'.
WA_Stu-Sname = 'Satish'.
Append WA_Stu to IT_Stu.

WA_Stu-Sid = '3'.
WA_Stu-Sname = 'Latha'.
Append WA_Stu to IT_Stu.

WA_Stu-Sid = '4'.
WA_Stu-Sname = 'Sharadha'.
Append WA_Stu to IT_Stu.

Loop at IT_Stu into WA_Stu.
  Write: / WA_Stu-Sid, WA_Stu-Sname, WA_Stu-Srank.
Endloop.

Uline.
* Filing the rank internal table
WA_rank-Sid = '4'.
WA_rank-Srank = '3'.
Append WA_rank to IT_rank.

WA_rank-Sid = '3'.
WA_rank-Srank = '4'.

```

```

Append WA_rank to IT_rank.
WA_rank-Sid = '1'.
WA_rank-Srank = '1'.
Append WA_rank to IT_rank.
WA_rank-Sid = '2'.
WA_rank-Srank = '2'.
Append WA_rank to IT_rank.

Loop at IT_rank into WA_rank.
  Write: / WA_rank-Sid, WA_rank-Srank.
Endloop.
Uline.
* Modify student rank with rank internal table data.

Loop at IT_Stu into WA_Stu.
  Read table IT_rank into WA_rank with key
  Sid = WA_Stu-Sid.
  WA_Stu-Srank = WA_rank-Srank.
  Modify IT_Stu from WA_Stu.
Endloop.
* Display
Loop at IT_Stu into WA_Stu.
  Write: / WA_Stu-Sid, WA_Stu-Sname, WA_Stu-Srank.
Endloop.

```

### **Search Help: -**

Search help is used to provide the list of possible values to the input variable. There are two types of search helps. 1. **Elementary search help** 2. **Collective search help**.

This is two step procedure.

1. Create the search help
2. Attach the search help to data element.

→ **Create the elementary search help to identify the company code based on the company name.**

### **Steps to create elementary search help: -**

Execute SE11. Select the radio button search help. Provide the search help name (zsashi1). Click on create. Click on enter. Provide short description. Provide the selection method is table name (T001). Provide the hot key as any name. Enter. Provide search help parameter.

<u>Search help parameter</u>	<u>IMP</u>	<u>EXP</u>	<u>LPOS</u>	<u>SPOS</u>
BUKRS	✓	✓	1	1
BUTXT		✓	2	2

Save, check, activate the search help.

### **Steps to attach the search help to data element: -**

Execute SE11. Select the radio button data type. Provide the data element name (ZSPT\_9\_BUKRS). Click on create. Enter. Provide short description. Provide domain name [(zbukrs) (4, char)]. Click on further characteristics tab. Provide the search help name (zsashi1), parameter (BUKRS). Save, check, activate.

Execute SE38.

Parameter P\_BUKRS type ZSPT\_9\_BUKRS.

Save, check, activate, execute.

### **LFB1 (Vendors under company): -**

BUKRS → Company code

LIFNR → Vendor number

AKONT → Recon Account.

**Collective search help:** - Collective search help is the collection of elementary search help.

→ **Create the collective search help to identify the company code based on company name & identify the company code based on vendor number.**

Execute SE11. Select the radio button search help. Provide the collective search help name (zsashi3). Click on create. Select the radio button collective search help. Enter. Provide short description. Provide the search help parameter (BUKRS) & data element (BUKRS). Import, export, click on include search help tab. Provide the all the elementary search helps. Select the each elementary search help. Click on parameter assignment. Click on yes. Enter. Save, check, activate the collective search help.

### **Steps to create data element for collective search help: -**

Execute SE11. Select the radio button data type. Provide the data element name. Click on create. Enter. Provide short description. Provide the domain name. Click on further characteristic tab. Provide the collective search help name, parameter. Save, check, activate.

Execute /OSE38.

Parameter P\_BUKRS type ZSPT\_9\_BUKRS.

### **Hot key:** -

The Hot key permits the user to select an elementary search help from the collective search help directly in the input field with the short notation.

After executing the program we provide the input as equal to hot key (=A) & click on **F4**. Then we get the elementary search help, which contains the specified hot key as default.  
Letters & digits are allowed as a hot key.

### **Search help can call without assign to the data element**

parameter `kunnr type char10 MATCHCODE OBJECT ZCUST_NUM3.`

### **Lock Objects:** -

Lock object is used to avoid the concurrent access of multiple users on the same data base. When ever we create & activate the lock object it generates two function modules. 1. Enqueue (locking) 2. Dequeue (unlock). The lock object name must be start with EZ or EY.

### **Steps to create lock object:** -

Execute SE11. Select the radio button lock object. Provide the lock object name. Click on create. Provide short description. Click on tables tab. Provide the table name (T001). Select the lock mode as read lock. Save, check, activate. In the menu bar click on go to → lock modules. Identify the functional modules. (ENQUEUE – lock object name, DEQUEUE – Lock object name).

### **Types of lock modules:** -

#### **Write lock (or) exclusive lock:** -

The locked data can be read or processed by one user only.

#### **Read lock (or) shared lock:** -

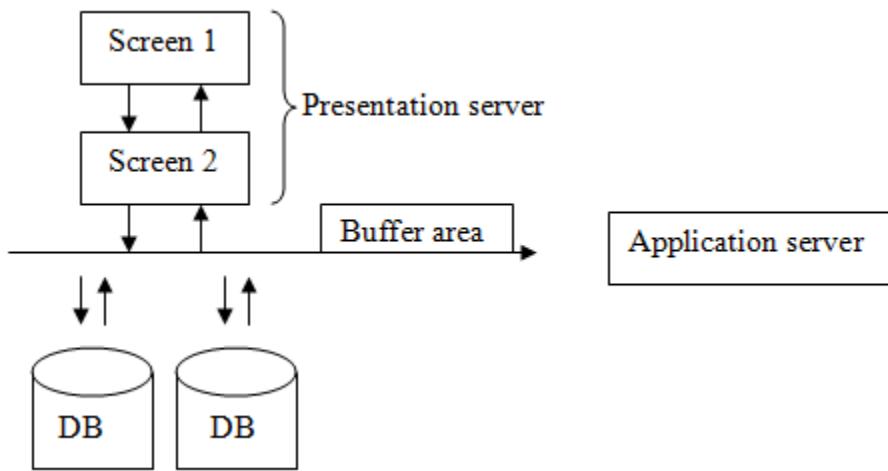
Several users can read the same data at the same time but only one user can edit the data.

#### **Enchanted write lock (Exclusive lock without cumulating):** -

Several users can access the same data as well as update the same data.

## **Buffering: -**

Buffering is the temporary place in the application server. When ever we execute any object then the system goes to application server and check the required data is available or not in buffer area. If the data is available then it gets from buffer area & displays it. If the data isn't available in the buffer area then it goes to data base & picks the data from data base server & placed into buffer area & displays it.



**Note:** - Buffering is always available in the technical setting of a table.

### **Types of buffering: -**

#### **Single record buffering:** -

In this kind of buffering the selected data will be stored into buffered area.

#### **Generic area buffering:** -

In this kind of buffering the key information stored in the buffered area.

#### **Fully buffered:** -

In this type of buffering the entire data of database is load into the buffering.

**Note:** - If you want to display the data in a single line then you must provide the line-size.

**Ex:** - Report <Report name> Line-size 1023.

### **MAKT (Material Description table)**

MATNR → Material number

SPRAS → Language

MAKTX → Material Description

### **EKKO (Purchasing document table): -**

EBELN → Purchasing document number

BEDAT → Document date

LIFNR → Vendor number

BUKRS → Company code

BSART → Document type.

### **T001W (Plant description table): -**

WERKS → Plant number

NAME1 → Plant name

### **EKPO (Purchasing document item table): -**

EBELN → Purchasing document number

EBELP → Item number

MATNR → Material number

MENGE → Quantity

MEINS → Unit of measurement

NETPR → Net price

### **MARC (Material & Plant table): -**

MATNR → Material number

WERKS → Plant number

### **MARD (Material, Plant storage location): -**

MATNR → Material number

WERKS → Plant number

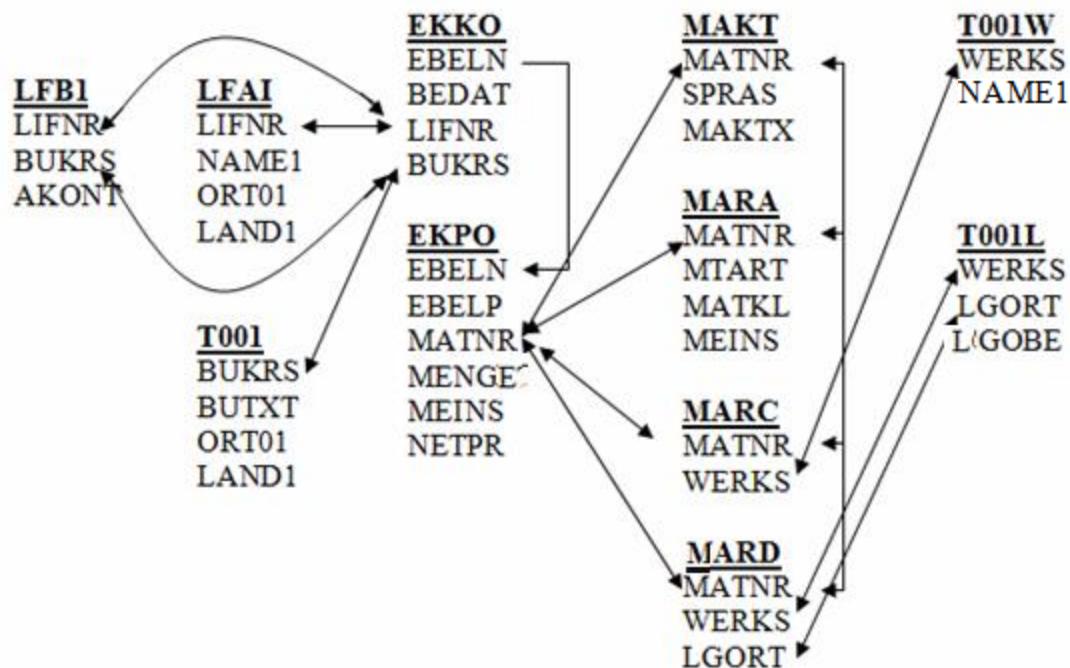
LGORT → Storage location

### **T001L (Storage location description table): -**

WERKS → Plant number

LGORT → Storage location

LGOBE → Storage location description



## **TMG (Table Maintenance Generator)**

Table Maintenance Generator is used to insert, update and delete the data of data base table with out any code (with out DML Commands). Table Maintenance Generator is only possible for custom tables. The transaction code for TMG is SM30.

### **Steps to create Table Maintenance Generator: -**

Execute SE11. Select the radio button data base table. Provide the data base table name for which table we want to create the TMG. Click on change. In the menu bar click on utilities → Table Maintenance Generator. Select the authorization group which is provided by BASIS people. Provide the function group as table name. Select the Maintenance type is one step or two step.

Provide the screen number (any number). Click on create. Save in our own package. Click on save, back, active.

### **Steps to maintain the data by using TMG: -**

Execute SM30. Provide table or view name as our table name (ZHAI11). Click on maintain. Enter. And perform the operations of the data (insert, update, delete).

### **Steps to create transaction code for the table or TMG: -**

Execute SE93. Provide the transaction code as same table name. Click on create. Provide short description. Select the radio button transaction with parameter. Provide transaction (SM30). Select the check box skip initial screen. Select the GUI check boxes. Provide the default values.

#### **Name of the screen field**

#### **value**

VIEWNAME

HAI11

UPDATE

X

Click on save.

Now we execute this table name as a transaction code then we get the screen and perform the operations. There are two types of Maintenance. One step and two step.

### **One Step Maintenance: -**

It means both maintaining the data and display the data in a single screen.

### **Two step Maintenance: -**

It means maintain the data in one screen and display the data in some other screen.

### **Some of the events in TMG**

1. Before saving the data in the data base
2. After saving the data in the data base.
3. Before deleting the data display
4. After deleting the data display.
5. Creating a new entry

### **Steps to implement the events in TMG: -**

Execute SE11. Open the table in change mode. In the menu bar click on utilities → Table Maintenance Generator. In the menu bar click on environment → modification events. Enter. Click on new entries in the application tool bar. Select the event. Provide the form name. click on save. Click on editor. Enter. Implement the code in between form end form. Save, check, activate. Back, save, back.

**Table type:** - Table type is the collection of structure records or structure fields. By using table type we declare the internal table in the ABAP editor. Creation of table type is two step procedures. 1. Create the structure, 2. Create the table type based on the structure.

### → **Create the table type with BUKRS BUTXT ORT01**

#### Steps to create the structure: -

Execute SE11. Select the radio button data type. Provide the structure name. click on create. Select the radio button structure. Enter. Provide short description. Provide the component, component type.

<b>Component</b>	<b>Component type</b>
BUKRS	BUKRS
BUTXT	BUTXT
ORT01	ORT01

Save, check, activate.

#### Steps to create table type based on the structure: -

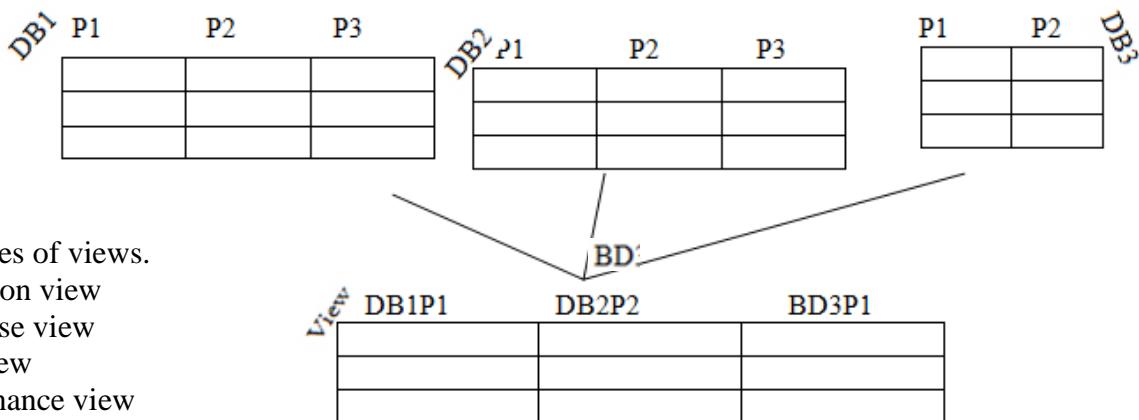
Execute SE11. Select the radio button table type. Provide the table type name. click on create. Select the radio button table type. Enter. Provide short description. Provide line type as structure name. Enter. Save, check, activate the table type.

```
DATA it_t001 TYPE zhai11.  
DATA wa_t001 LIKE LINE OF it_t001.  
SELECT bukrs butxt ort01 FROM t001 INTO TABLE it_t001.  
LOOP AT it_t001 INTO wa_t001.  
  WRITE: / wa_t001-bukrs, wa_t001-butxt, wa_t001-ort01.  
ENDLOOP.
```

### **Views: -**

Each application has its own data base tables. If you want to display the part of data from each table then we pick the data from each table & merge the data & display the data. If it's regular activity then it's better to create a view.

Views are logical databases. It doesn't contain the data permanently. At run time only view contains the data.



There are 4 types of views.

1. Projection view
2. Data base view
3. Help view
4. Maintenance view

### **Projection view: -**

If you want to display the part of data from single database table, if it's a regular activity then it's better to create projection view.

Projection view is always involving single database table.

### **Steps to create projection view: -**

Execute SE11. Select the radio button view. Provide the projection view name. Click on create. Select the radio button projection view. Enter. Provide short description. Provide the basis table. Click on table fields button. Select the required fields check box. Enter. Save, check, activate.

#### **Ex: -**

Select BUKRS BUTXT ORT01 from ZSPT\_9AM\_PV into table IT\_T001.

**Note:** - Fetching the data from view is faster than fetching the data from database table.

### **Data base view: -**

If you want to display the data from more than one table then we pick the data from each table & merge it & display it. If it's a regular activity, then it's better to create database view. Database view is always involved with more than one table.

### **→ Create the database view with BUKRS BUTXT LIFNR**

### **Steps to create database view: -**

Execute SE11. Select the radio button view. Provide the data base view name. Click on create. Enter. Provide short description. Provide the related tables in left table. Select all tables. Click on relationships. Select the check box. Enter. Click on view fields tab. Click on table fields button. Select the each table. Click on choose. Select the required fields. Enter. Save, check, activate.

#### **Ex: -**

Select BUKRS BUTXT LIFNR from ZSPT\_9AM\_DV into table IT\_Final.

Database view picks the data from both the tables. If and only if there is one or more entries is available in the right hand side table with corresponding left hand side table.

The diagram illustrates the creation of a database view. It shows two separate tables at the top: 'BUKRS BUTXT' (with rows 1000-TCS, 2000-IBM, 3000-HCL, 4000-HP, 5000-CSC) and 'BUKRS LIFNR' (with rows 1000-241, 1000-251, 3000-116, 4000-745, 4000-795). A curved arrow points from the 'BUTXT' column of the first table to the 'LIFNR' column of the second table, indicating a relationship. The word 'TOOL' is written vertically above the first table, and 'LFBI' is written vertically above the second table.

BUKRS	BUTXT
1000	TCS
2000	IBM
3000	HCL
4000	HP
5000	CSC

BUKRS	LIFNR
1000	241
1000	251
3000	116
4000	745
4000	795

The resulting database view table is shown below, combining the data from both tables where there is a matching BUKRS value:

BUKRS	BUTXT	LIFNR
1000	TCS	241
1000	TCS	251
3000	HCL	116
4000	HP	745
4000	HP	795

#### Help view: -

Help view pick the data from left hand side table. Even though there is no match in the right hand side table.

Help view always involve in two data base tables.

**Note:** - Help view is used as a selection method in an elementary search help to provide the possible entries from multiple tables.

#### Steps to create the help view: -

Execute SE11. Select the radio button view. Provide the help view name. Click on create. Select the radio button help view. Enter.

Provide short description. Provide the initial table. Click on relationships button. Select the required table check box. Enter. Click on view fields tab. Click on table fields. Select the each table. Click on choose. Select the required fields. Enter. Save, check, activate.

**Maintenance view:** Maintenance view is defined to maintain multiple tables data using the transaction code SM30.

## **Control break statement / events in internal table: -**

Control break statements are worked with in the loop of internal table. Before using the control break statements, we must sort the internal table based on At new field. Control break statements are 1> At First  
2> At New <field name> 3> At End of <field name> 4> At last  
Each control break statement ends with Endat.

### **AT FIRST: -**

This is an event which is triggered at the first record of internal table.

**Advantage:** - This is used to display the header information for internal table.

### **AT NEW <field name>: -**

It's an event which is triggered at the first record of each block.

**Advantage:** - It's used to display the individual fields.

### **AT END OF <field name>: -**

This event triggered at the last record of each block.

**Advantage:** -This is used to display the sub total.

### **AT LAST: -**

This is an event which is triggered at the last record of internal table.

**Advantage:** - It's used to display the grand total.

**→ Based on the given purchasing document numbers display the purchasing item details as shown in the figure.**

EBELN	EBELP	MENGE	MEINS	NETPR
3004	01	10	Kg	250.00
3004	02	2	Pcs	150.00
3005	01	3	Nos	450.00
3006	01	9	Pcs	120.00
3006	02	2	Box	180.00
3006	03	2	Nos	200.00

EBELN	EBELP	MENGE	MEINS	NETPR

At first =====► These are PO details

At new =====► 3004

01	10	Kg	250.00
02	2	Pcs	150.00

At end of =====► Sub total 400.00

3005	01	3	Nos	450.00
			Sub total	450.00

3006	01	9	Pcs	120.00
	02	2	Box	180.00
	03	2	Nos	200.00
			Sub total	500.00

At last =====► Grand total 1350.00

**Note:** - When ever we are working with at new & at end of the right side fields of mention field values display as **stars** if it's a character data type & displayed as **zeros** if it's a numeric data type.

```
DATA V1 TYPE EKKO-EBELN.  
SELECT-OPTIONS S_EBELN FOR V1.  
DATA: V2 TYPE EKPO-NETPR, V3 TYPE EKPO-NETPR.  
TYPES: BEGIN OF TY_EKPO,  
        EBELN TYPE EKPO-EBELN,  
        EBELP TYPE EKPO-EBELP,  
        MENGE TYPE EKPO-MENGE,  
        MEINS TYPE EKPO-MEINS,  
        NETPR TYPE EKPO-NETPR,  
        END OF TY_EKPO.  
DATA WA_EKPO TYPE TY_EKPO.  
DATA IT_EKPO TYPE TABLE OF TY_EKPO.  
DATA WA LIKE WA_EKPO.  
SELECT EBELN EBELP MENGE MEINS NETPR FROM EKPO INTO TABLE IT_EKPO WHERE  
EBELN IN S_EBELN.  
SORT IT_EKPO BY EBELN.  
LOOP AT IT_EKPO INTO WA_EKPO.  
WA = WA_EKPO.  
AT FIRST.  
WRITE / 'THESE ARE PO DETAILS'.  
ENDAT.  
AT NEW EBELN.  
WRITE: / WA_EKPO-EBELN, WA-EBELP, WA-MENGE.  
ENDAT.  
WRITE: / WA_EKPO-EBELP, WA_EKPO-MENGE, WA_EKPO-MEINS,  
WA_EKPO-NETPR.  
V2 = V2 + WA_EKPO-NETPR.  
V3 = V3 + WA_EKPO-NETPR.  
AT END OF EBELN.  
WRITE: / 'SUB TOTAL', V2.  
CLEAR V2.  
ENDAT.  
AT LAST.  
WRITE:/ 'GRAND TOTAL', V3.  
ENDAT.  
ENDLOOP.
```

### Working with ON CHANGE OF: -

Ex:-

Loop at IT\_EKPO into WA\_EKPO.

ON CHANGE OF WA\_EKPO-EBELN.

Write:/ WA\_EKPO-EBELN, WA\_EKPO-EBELP, WA\_EKPO-MENGE.

Endon.

Endloop.

→ Write the program by using control break statements to know difference between at new and on change of.

```
Data: BEGIN OF wa,
  code type i,
  name(4) type c,
  city(4) type c,
  end of wa.
  data it like table of wa.

  wa-code = '100'.
  wa-name = 'TCS'.
  wa-city = 'HYD'.
  APPEND WA TO IT.
  CLEAR WA.
  wa-code = '100'.
  wa-name = 'IBM'.
  wa-city = 'CHE'.
  APPEND WA TO IT.
  CLEAR WA.
  wa-code = '200'.
  wa-name = 'IBM'.
  wa-city = 'BAN'.
  APPEND WA TO IT.
  CLEAR WA.
  wa-code = '200'.
  wa-name = 'HCL'.
  wa-city = 'PUNE'.
  APPEND WA TO IT.
  CLEAR WA.
LOOP AT IT INTO WA.
AT NEW NAME.
  WRITE:/ WA-CODE, WA-NAME,WA-CITY.
ENDAT.
ENDLOOP.
```

At new NAME

100	TCS	****
100	IBM	****
200	IBM	****
200	HCL	****

At new CODE

100	*****	*****
200	*****	*****

### Explanation :-

First we have to sort the internal table based on at new field. If I take AT NEW CODE , then first record, third record only display. Because second and forth records are duplicate. If I use AT NEW NAME, then all records will display. Because first it will display 1<sup>st</sup> record. After, name is different. That's why 2<sup>nd</sup> record also display. Next, 3<sup>rd</sup> record also display. Because name is same but code is different. It'll check left side fields also. If there is a new data then it'll display. If the data is same as 2<sup>nd</sup> record, then it'll not display. Next it'll display the 4<sup>th</sup> record. Because name is different. By using on change of it's not possible. Here 3<sup>rd</sup> record isn't display by using on change of. It'll check that column only. 2<sup>nd</sup> record name, 3<sup>rd</sup> record name same. So on change of don't allow to display 3<sup>rd</sup> record. On change of will allow to display 4<sup>th</sup> record. Because name is different. But code is same. In this situation on change of don't check left side data.

## Differences between AT NEW, ON CHANGE OF

### AT NEW

1. AT NEW work within the loop of internal table
2. The right side fields of the mentioned field values display as **stars** if it's a character data type & display '0's if it's numeric data type.
3. In the AT NEW we can't use the logical operations (and, or, not)
4. This is used in ABAP objects.

**Note:** - Now a days ON CHANGE OF is out dated (Don't use).

If you want to remove the title in the output, then you must provide '**NO STANDARD PAGE HEADING**' in the name of the report.

**SY-ULINE** is the system variable which is used to draw the horizontal line. **SY-VLINE** is the system variable which is used to draw the vertical line.

```

DATA V1 TYPE T001-BUKRS.
SELECT-OPTIONS S_BUKRS FOR V1.
TYPES: BEGIN OF TY_T001,
        BUKRS TYPE T001-BUKRS,
        BUTXT TYPE T001-BUTXT,
        ORT01 TYPE T001-ORT01,
        END OF TY_T001.
DATA WA_T001 TYPE TY_T001.
DATA IT_T001 TYPE TABLE OF TY_T001.
SELECT BUKRS BUTXT ORT01 FROM T001 INTO TABLE IT_T001 WHERE BUKRS IN
S_BUKRS.
WRITE SY-ULINE(57).
WRITE: / SY-VLINE, 2 'COCD' COLOR 5, 6 SY-VLINE, 7 'COMPANY NAME'
      COLOR 5, 32 SY-VLINE, 33 'COMPANY CITY' COLOR 5, 57 SY-VLINE.
WRITE / SY-ULINE(57).
LOOP AT IT_T001 INTO WA_T001.
  WRITE: / SY-VLINE, 2 WA_T001-BUKRS, 6 SY-VLINE, 7 WA_T001-BUTXT,
         32 SY-VLINE, 33 WA_T001-ORT01, 57 SY-VLINE.
  WRITE / SY-ULINE(57).
ENDLOOP.

```

### VBAK (Sales document header table)

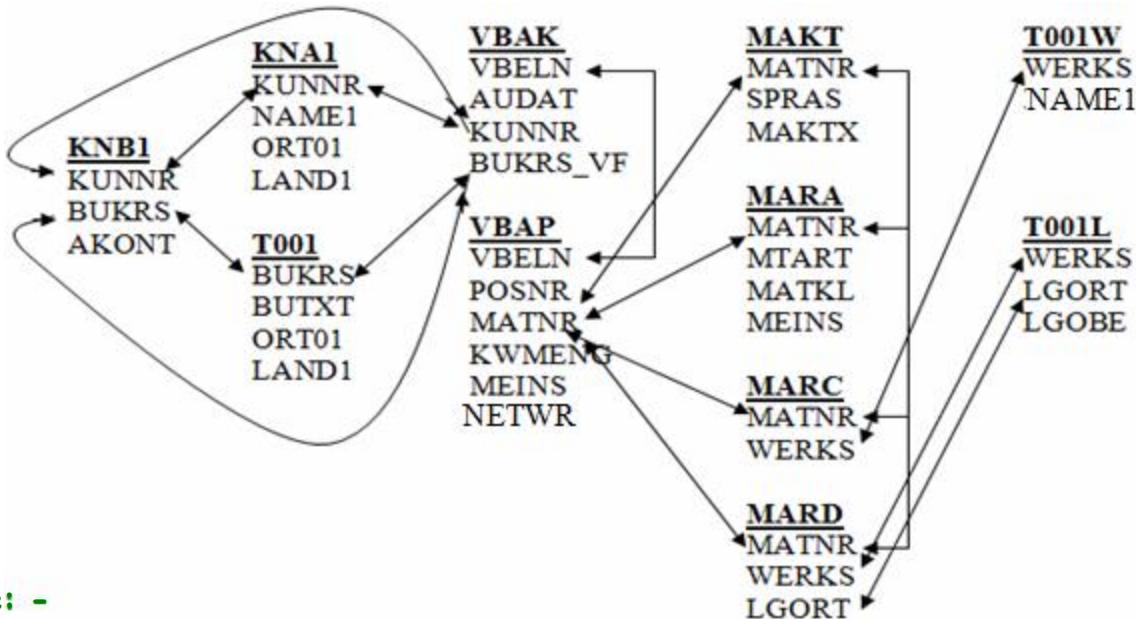
**VBELN** → Sales document number  
**AUART** → Sales document type.  
**AUDAT** → Sales document date  
**KUNNR** → Customer number  
**BUKRS\_VF** → Company code

### ON CHANGE OF

1. ON CHANGE OF work with in the any loop.
2. The right side field values of mention fields never change.
3. We can use logical operations.
4. This isn't used in ABAP objects.

### VBAP (Sales document item table)

**VBELN** → Sales document number  
**POSNR** → Item number  
**MATNR** → Material number  
**KWMENG** → Material quantity  
**MEINS** → UOM  
**NETWR** → Net value



### Continue: -

The continue statement only used in loop if it's used the current loop pass is ended immediately & the program flow is continue. continue will go to next index of the loop.

→ Based on the given purchasing document numbers, display the item numbers, quantity, units of measurements & price if the amount is more than 20 by using continue statement.

### Ex: -

```

LOOP AT it_ekpo INTO wa_ekpo.
  IF wa_ekpo-netpr < 20.
    CONTINUE.
  ENDIF.
  WRITE:/ wa_ekpo-ebeln, wa_ekpo-ebelp, wa_ekpo-menge, wa_ekpo-meins, wa_ekpo-netpr.
ENDLOOP.
    
```

### Exit: -

Exit statement is used within a loop. It leaves the loop by ending the current loop process. EXIT will completely go out of the LOOP statement or DO statement.

→ Based on the given purchasing document numbers to display the first 5 item details by using exit command

```

SELECT ebeln ebelp menge meins netpr FROM ekpo INTO it_ekpo WHERE ebeln IN s_ebeln.
LOOP AT it_ekpo INTO wa_ekpo.
  WRITE:/ wa_ekpo-ebelp, wa_ekpo-ebeln, wa_ekpo-menge, wa_ekpo-meins, wa_ekpo-
netpr.
  v2 = v2 + 1.
  IF v2 = 5.
    EXIT.
  ENDIF.
ENDLOOP.
    
```

If we use Exit statement in any subroutine (form, endform), It will skip that subroutine program execution from that Exit statement onwards. Cursor will come outside of the subroutine.

```

FORM fetch_data.
  SELECT ebeln ebelp menge meins netpr FROM ekpo INTO it_ekpo WHERE ebeln IN s_ebeln.
  EXIT.
  LOOP AT it_ekpo INTO wa_ekpo.
    WRITE:/ wa_ekpo-ebelp, wa_ekpo-ebeln, wa_ekpo-menge, wa_ekpo-meins, wa_ekpo-
netpr.
  ENDLOOP.
ENDFORM.
    
```

### **Check: -**

If we use this Check statement in subroutine by giving condition, it will skip to execute that subroutine, cursor will go outside of the form.

```
FORM fetch_data.
  SELECT ebeln ebelp menge netpr FROM ekpo INTO TABLE it_ekpo WHERE ebeln IN s_ebeln.
  CHECK it_ekpo IS INITIAL.
  LOOP AT it_ekpo INTO wa_ekpo.
    WRITE:/ wa_ekpo-ebelp, wa_ekpo-ebeln, wa_ekpo-menge, wa_ekpo-netpr.
  ENDLOOP.
ENDFORM.
```

If we use Check statement inside the loop statement, then that looping line will not execute. It will loop next record.

```
FORM fetch_data.
  SELECT ebeln ebelp menge netpr FROM ekpo INTO TABLE it_ekpo WHERE ebeln IN s_ebeln.
  LOOP AT it_ekpo INTO wa_ekpo.
    CHECK wa_ekpo-netpr IS INITIAL.
    WRITE:/ wa_ekpo-ebelp, wa_ekpo-ebeln, wa_ekpo-menge, wa_ekpo-netpr.
  ENDLOOP.
ENDFORM.
```

### **Stop: -**

If we use Stop statement either in subroutine or looping statement, program execution will stop of that total event. Cursor will go to next event.

```
START-OF-SELECTION.
  STOP.
  SELECT ebeln ebelp menge netpr FROM ekpo INTO TABLE it_ekpo WHERE ebeln IN s_ebeln.

END-OF-SELECTION.
  LOOP AT it_ekpo INTO wa_ekpo.
    WRITE:/ wa_ekpo-ebelp, wa_ekpo-ebeln, wa_ekpo-menge, wa_ekpo-netpr.
  ENDLOOP.
```

### **Return: -**

If we use Return statement in looping statement or in subroutine, the cursor will come outside of that form.

```
FORM fetch_data.
  SELECT ebeln ebelp menge netpr FROM ekpo INTO TABLE it_ekpo WHERE ebeln IN s_ebeln.
  RETURN.
  LOOP AT it_ekpo INTO wa_ekpo.
    WRITE:/ wa_ekpo-ebelp, wa_ekpo-ebeln, wa_ekpo-menge, wa_ekpo-netpr.
  ENDLOOP.
ENDFORM.
```

### **Open SQL**

#### **DDIC**

Data dictionary  
(Which is used to create or  
Alter the table)

#### **DML**

Insert  
Update  
Modify  
Delete

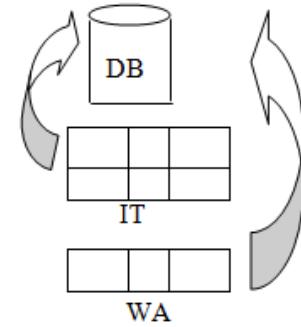
#### **DCL**

Commit work  
Rollback work

**Note:** - Insert, update, modify a single record into the data base table is always through work area & multiple records through internal table.

**Note:** - When ever we are working with data base tables then we must maintain the structure of the work area as well as structure of the internal table must be the similar structure of data base table.

<b>Data base tables</b>	<b>Key fields</b>
T001	→ BUKRS
KNA1	→ KUNNR
LFA1	→ LIFNR
KNB1	→ KUNNR, BUKRS
LFB1	→ LIFNR, BUKRS
MARA	→ MATNR
MAKT	→ MATNR, SPRAS
EKKO	→ EBELN
EKPO	→ EBELN, EHELP
MARC	→ MATNR, WERKS
MARD	→ MATNR, WERKS, LGORT
T001W	→ WERKS
T001L	→ WORKS, LGORT
VBAK	→ VBELN
VBAP	→ VBELN, POSNR



### **Insert (single record): -**

Insert inserts a record into the data base table based on the key field if there is no match found in the data base. Other wise it ignores the record.

#### **Syntax: -**

Insert <data base table> from <work area>.

**Note:** - SY-SUBRC is the system variable which contains zero with the above statement executed successfully otherwise it contains non zero. Most of the times it contains '4'.

```

Data wa_T001 like T001.
WA_T001-BUKRS = '0786'.
WA_T001-BUTXT = 'SATISH INFO'.
WA_T001-ORT01 = 'HYD'.
WA_T001-LAND1 = 'IN'.
Insert T001 from WA_T001.
If SY-SUBRC = 0.
Write 'Inserted'.
Else.
Write 'Not inserted'.
Endif.
  
```

### **Insert (Multiple records): -**

Insert inserts multiple records from internal table to data base table if there is no match found in the data base for all the records of internal table based on the key field. If at least one record is matched then it simply ignores the all records of internal table as well as terminates the entire transaction.

#### **Syntax: -**

Insert <data base table> from table <internal table>.

```

Data: WA_T001 LIKE T001,
      IT_T001 LIKE TABLE OF WA_T001.
WA_T001-BUKRS = '0888'.
WA_T001-BUTXT = 'SATISH TECH'.
WA_T001-ORT01 = 'HYD'.
Append WA_T001 to IT_T001.
Clear WA_T001.
WA_T001-BUKRS = '0999'.
WA_T001-BUTXT = 'DHAWAN TECH'.
WA_T001-ORT01 = 'CHE'.
Append WA_T001 to IT_T001.
Clear WA_T001.
WA_T001-BUKRS = '0777'.
WA_T001-BUTXT = 'DHAWAN INFO'.
WA_T001-ORT01 = 'BAN'.
Append WA_T001 to IT_T001.
Clear WA_T001.
Insert T001 from table IT_T001.

```

Here '0777' company already exists in the data base. So it ignores all other records in internal table as well as it terminates the entire transaction.

If you want avoid the termination of the program then you must place accepting duplicate keys in the syntax of inserting.

#### Syntax:-

Insert <data base table> from table <internal table> accepting duplicate keys.

The above syntax avoids the termination of the program as well as inserts the non duplicate records & ignores the duplicate records.

**Note:** - SY-DBCNT is the system variable which contains the number of records as successfully processed into the data base.

#### Ex: -

Insert T001 from table IT\_T001 accepting duplicate keys.

Write SY-DBCNT.

Here 0888 & 0999 companies are inserted & 0777 is ignored.

#### **Update (single record) / over write: -**

Update updates a record into the data base table if there is a match found into the data base base on the key field. Otherwise it ignores the record.

#### Syntax:-

Update <data base table> from <work area>.

**Note:** - when ever we are working with update then we must maintain change field information & also non change field information. Other wise non change field information may be lost.

#### Ex: -

```

Data WA_T001 like T001.
WA_T001-BUKRS = '0777'.
WA_T001-BUTXT = 'SATISH INFO'.
Update T001 from WA_T001.

```

```
If SY-SUBRC = 0.  
Write 'UPDATED'.  
Else.  
Write 'NOT UPDATED'.  
Endif.
```

### **Update (Multiple records): -**

This functionality is similar as update single record functionality.

#### **Syntax:-**

Update <data base table> from table <internal table>.

### **Update particular record: -**

#### **Syntax: -**

Update <data base table> set <field 1> = <value 1> <field 2> = <value 2> ..... where <condition>

#### **Ex: -**

```
Update T001 set ORT01 = 'MUM' LAND1 = 'IN' where BUKRS = '0777'.  
If SY-SUBRC = 0.  
Write 'updated'.  
Else.  
Write 'not updated'.  
Endif.
```

**Note:** - Modify acts like update if there is a match found in data base based on the key field otherwise it acts like insert. Modify never failed.

#### **Syntax: -**

Modify <data base table> from <work area>.

Modify <data base table> from table <internal table>.

```
Data: WA like T001,  
      IT like table of WA.  
WA-BUKRS = '0786'.  
WA-BUTXT = 'SJF'.  
WA-ORT01 = 'CHE'.  
Append WA to IT.  
Clear WA.  
WA-BUKRS = '0787'.  
WA-BUTXT = 'SJF'.  
WA-ORT01 = 'BAN'.  
Append WA to IT.  
Clear WA.  
Modify T001 from table IT.
```

In this example '0786' company already exist in the data base. So it acts like update or over write & '0787' company details aren't available in data base. So it acts as insert.

### **Delete: -**

Delete deletes the data from data base based on condition.

#### **Syntax: -**

Delete from <data base table> where <condition>.

**Ex:** -

```
Delete from T001 where BUKRS = '0787'.  
If SY-SUBRC = 0.  
Write 'deleted'.  
Else.  
Write 'not deleted'.  
Endif.
```

#### **Commit work:** -

This command is used to commit the data base changes.

**Ex:** -

```
Delete from T001 where BUKRS = '0786'.  
Commit work.
```

#### **Rollback work:** -

This command is used to reverse the data base changes.

**Ex:** -

```
Delete from T001 where BUKRS = '0787'.  
Rollback work.
```

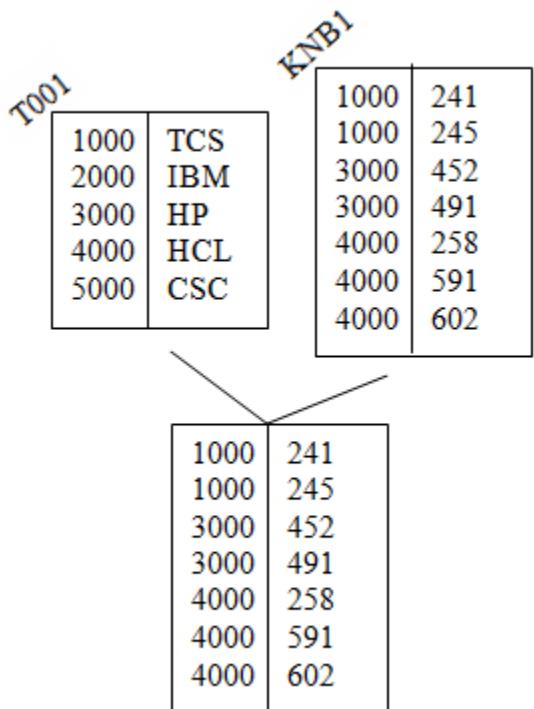
# Joins

Joins are used to fetch the data from more than one table. There are two types of joins.

1. Inner join
2. Left outer join

## **Inner join: -**

Inner join pick the data from both the tables if & only if there is one or more than one entry is available in the right hand side table with corresponding left hand side table.



## Syntax: -

Select <data base table 1> ~ <field 1> <data base table 1> ~ <field 2> -----  
           <data base table 2> ~ <field 1> <data base table 2> ~ <field 2> -----  
                       | |  
                       | |  
                       Where <condition>.

**Note:** - The link field must be primary fields in at least one data base table.

**→ Display the company codes, company names & vendor numbers of the company details by using inner join.**

```

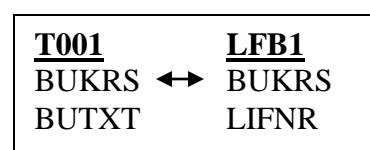
TYPES: Begin of TY_FINAL,
      BUKRS Type T001-BUKRS,
      BUTXT Type T001-BUTXT,
      LIFNR Type LFB1-LIFNR,
      End of TY_FINAL.

```

```

DATA: WA_FINAL TYPE TY_FINAL,
      IT_FINAL TYPE TABLE OF TY_FINAL.
Select T001~BUKRS T001~BUTXT LFB1~LIFNR
into table IT_FINAL
from T001 inner join LFB1
on T001~BUKRS = LFB1~BUKRS.

```



```

Sort IT_FINAL by BUKRS.
Loop at IT_FINAL into WA_FINAL.
  Write:/ WA_FINAL-BUKRS, WA_FINAL-BUTXT, WA_FINAL-LIFNR.
ENDLOOP.

```

→ Based on the given purchasing document numbers display the purchasing document numbers, document dates, vendor numbers, item numbers, quantity, unit of measurements & net price.

```

Data V1 type EKKO-EBELN.
Select-options S_ebeln for V1.
Types: Begin of ty_final,
  EBELN type EKKO-EBELN,
  BEDAT type EKKO-BEDAT,
  LIFNR type EKKO-LIFNR,
  EBELP type EKPO-EBELP,
  MENGE type EKPO-MENGE,
  MEINS type EKPO-MEINS,
  NETPR type EKPO-NETPR,
End of ty_final.

```

```

Data: wa_final type ty_final,
      it_final type table of ty_final.
Select EKKO~EBELN EKKO~BEDAT EKKO~LIFNR EKPO~EBELP EKPO~MENGE
EKPO~MEINS EKPO~NETPR into table it_final from EKKO inner join EKPO on
EKKO~EBELN = EKPO~EBELN where EKKO~EBELN in S_ebeln.

```

Loop at it\_final into wa\_final.

```

  Write:/ wa_final-ebeln, wa_final-bedat, wa_final-lifnr, wa_final-
ebelp, wa_final-menge, wa_final-meins, wa_final-netpr.
Endloop.

```

→ Based on the given material numbers display the material numbers, material types, plant numbers, plant names by using inner join.

```

Data V1 type MARA-MATNR.
Select-options s_matnr for V1.
Types: begin of ty_final,
  MATNR type MARA-MATNR,
  MTART type MARA-MTART,
  WERKS type MARC-WERKS,
  NAME1 type T001W-NAME1,
End of ty_final.

```

```

Data: wa_final type ty_final,
      it_final type table of ty_final.
Select MARA~MATNR MARA~MTART MARC~WERKS T001W~NAME1 into table it_final
from MARA inner join MARC on MARA~MATNR = MARC~MATNR inner join T001W
on MARC~WERKS = T001W~WERKS where MARA~MATNR in S_matnr.

```

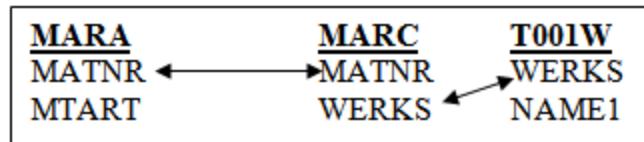
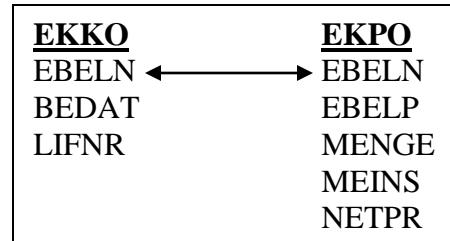
Loop at it\_final into wa\_final.

```

  Write:/ wa_final-MATNR, wa_final-MTART, wa_final-WERKS, wa_final-
NAME1.
Endloop.

```

→ Based on the given customer numbers display the customer numbers, customer names, sales document numbers, document date, item number, material number, material description, quantity, unit of measurement, net price by using inner join.

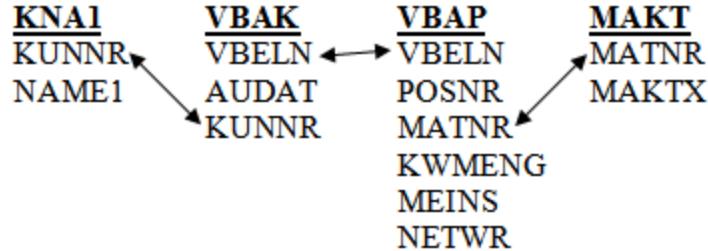


```

Data V1 type KNA1-kunnr.
Select-options s_kunnr for V1.
Types: begin of ty_final,
        KUNNR type KNA1-KUNNR,
        NAME1 type KNA1-NAME1,
        VBELN type VBAK-VBELN,
        AUDAT type VBAK-AUDAT,
        POSNR type VBAP-POSNR,
        MATNR type VBAP-MATNR,
        MAKTX type MAKT-MAKTX,
        KWMENG type VBAP-KWMENG,
        MEINS type VBAP-MEINS,
        NETWR TYPE VBAP-NETWR,
      End of ty_final.

```

KUNNR, NAME1, VBELN, AUDAT, POSNR, MATNR,  
MAKTX, KWMENG, MEINS, NETWR



```

Data: wa_final type ty_final,
      it_final type table of ty_final.

```

```

Select KNA1~KUNNR KNA1~NAME1 VBAK~VBELN VBAK~AUDAT VBAP~POSNR
VBAP~MATNR MAKT~MAKTX VBAP~KWMENG VBAP~MEINS VBAP~NETWR into table
it_final from KNA1 inner join VBAK on KNA1~KUNNR = VBAK~KUNNR inner
join VBAP on VBAK~VBELN = VBAP~VBELN inner join MAKT on VBAP~MATNR =
MAKT~MATNR where KNA1~KUNNR in s_kunnr.

```

Loop at it\_final into wa\_final.

Write:/ wa\_final-KUNNR, wa\_final-NAME1, wa\_final-VBELN, wa\_final-
AUDAT, wa\_final-POSNR, wa\_final-MATNR, wa\_final-MAKTX, wa\_final-KWMENG,
wa\_final-MEINS, wa\_final-NETWR.

Endloop.

### Left outer join: -

Left outer join pick the data from left hand side table even though there is no match found in right hand side table. It's possible for only two data base tables.

T001		KNB1	
1000	TCS	1000	241
2000	IBM	1000	245
3000	HP	3000	452
4000	HCL	3000	491
5000	CSC	4000	258

1000	241
1000	245
2000	-----
3000	452
3000	491
4000	258
4000	591
4000	602
5000	-----

In the inner join syntax instead of inner join we paste the left outer join.

→ Based on the given company codes display the company codes, company names & customer number based on left outer join.

```

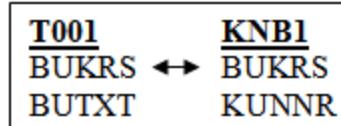
TYPES : Begin of TY_FINAL,
        BUKRS Type T001-BUKRS,
        BUTXT Type T001-BUTXT,
        KUNNR Type KNB1-KUNNR,
      End of TY_FINAL.

```

```

DATA: WA_FINAL TYPE TY_FINAL,
      IT_FINAL TYPE TABLE OF TY_FINAL.
Select T001~BUKRS T001~BUTXT KNB1~KUNNR
into table IT_FINAL
from T001 left outer join KNB1
on T001~BUKRS = KNB1~BUKRS.
Sort IT_FINAL by BUKRS.

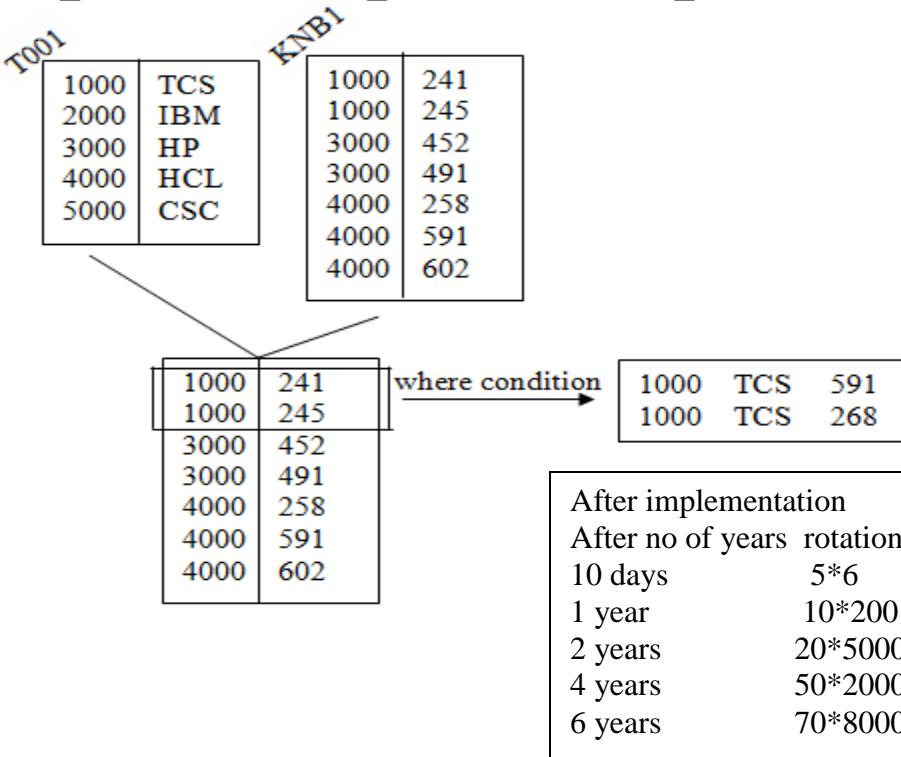
```



```

Loop at IT_FINAL into WA_FINAL.
Write:/ WA_FINAL-BUKRS, WA_FINAL-BUTXT, WA_FINAL-KUNNR.
Endloop.

```



In the real time the maximum program execution time in four grounds [when you press F8 it's called four ground] 600 seconds are max time. In the program execution time exceeds 600 seconds then it goes to dump [time up]. Some times more than two tables join leads to time out. So we go for 'for all entries'.

'For all entries' pick the data based on the where condition first, next it based on on-condition.

At the time of implementing the SAP performance of the inner join & 'For all entries' are same. Day by day, day by day the data base sizes are increased then the performance of the inner join is decreased the 'For all entries' is same. So we go for 'For all entries'.

**Note:** - Inner join isn't possible for pooled & clustered tables. Only possible for transparent tables.

### Steps to work for all entries:-

1. Declare one final data internal table which data we want to display & also declare one work area & internal table for each participated data base table.
2. Based on the given input we will fetch the data from data base & filled into data base internal table.
3. Based on the data base internal table data we fill the final internal table data.

### Procedure to fill the final internal table:-

First we identify the number of primary keys in each participated data base table. If the number of primary keys are '1' that is read. If the number of primary keys is more than '1' that is loop. Loop the many primary keys data base internal table & read the single primary key internal table & populate the final internal table.

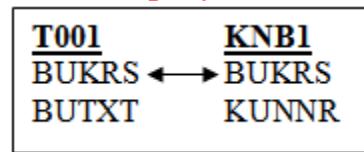
**Note:** - when ever we are working with for all entries then you must consider higher level internal table having the data. Suppose if you're not consider & higher level internal table having no data. Then it'll pick the entire data from next level.

\* if not IT\_T001 is initial.  
 Select BUKRS KUNNR from KNB1 into table IT\_KNB1 for all entries in IT\_T001 where BUKRS = IT\_T001-BUKRS.  
 \* endif.

Suppose IT\_T001 internal table having no data then it'll fetch entire data from KNB1 data base table & placed into IT\_KNB1.

**→ Based on the given company codes display the company codes, company names & customers under the company based on for all entries.**

```
Data V1 type T001-BUKRS.
Select-options s_bukrs for V1.
Types : begin of ty_final,
         BUKRS type T001-BUKRS,
         BUTXT type T001-BUTXT,
         KUNNR type KNB1-KUNNR,
         End of ty_final.
```



```
Data: wa_final type ty_final,
      it_final type table of ty_final.
Types: begin of ty_t001,
       BUKRS type T001-BUKRS,
       BUTXT type T001-BUTXT,
       End of ty_t001.
Data : wa_t001 type ty_t001,
      it_t001 type table of ty_t001.
Types: begin of ty_knb1,
       BUKRS type KNB1-BUKRS,
       KUNNR type KNB1-KUNNR,
       End of ty_knb1.
Data: wa_knb1 type ty_knb1,
      it_knb1 type table of ty_knb1.
```

```
Select BUKRS BUTXT from T001 into table it_t001 where BUKRS in s_bukrs.
If not it_t001 is initial.
  Select BUKRS KUNNR from KNB1 into table it_knb1 for all entries in
  it_t001 where BUKRS = it_t001-BUKRS.
Endif.
```

```
Loop at it_knb1 into wa_knb1.
  wa_final-BUKRS = wa_knb1-BUKRS.
  wa_final-KUNNR = wa_knb1-KUNNR.
  Read table it_t001 into wa_t001 with key BUKRS = wa_knb1-BUKRS.
  wa_final-BUTXT = wa_t001-BUTXT.
  Append wa_final to it_final.
  Clear: wa_final, wa_t001, wa_knb1.
Endloop.
Sort it_final by BUKRS.
Loop at it_final into wa_final.
  Write:/ wa_final-BUKRS, wa_final-BUTXT, wa_final-KUNNR.
Endloop.
```

→ Based on the given material numbers display the material numbers, material types, plant numbers, and plant names by using for all entries.

```
DATA v1 TYPE mara-matnr.
SELECT-OPTIONS s_matnr FOR v1.
TYPES: BEGIN OF ty_mara,
    matnr TYPE mara-matnr,
    mtart TYPE mara-mtart,
  END OF ty_mara.
DATA: wa_mara TYPE ty_mara,
      it_mara TYPE TABLE OF ty_mara.
```

```
TYPES: BEGIN OF ty_marc,
    matnr TYPE marc-matnr,
    werks TYPE marc-werks,
  END OF ty_marc.
DATA: wa_marc TYPE ty_marc,
      it_marc TYPE TABLE OF ty_marc.
```

```
TYPES: BEGIN OF ty_t001w,
    werks TYPE t001w-werks,
    name1 TYPE t001w-name1,
  END OF ty_t001w.
DATA: wa_t001w TYPE ty_t001w,
      it_t001w TYPE TABLE OF ty_t001w.
```

```
TYPES: BEGIN OF ty_final,
    matnr TYPE mara-matnr,
    mtart TYPE mara-mtart,
    werks TYPE marc-werks,
    name1 TYPE t001w-name1,
  END OF ty_final.
```

```
DATA: wa_final TYPE ty_final,
      it_final TYPE TABLE OF ty_final.
```

```
SELECT matnr mtart FROM mara INTO TABLE it_mara WHERE matnr IN s_matnr.
```

IF NOT it\_mara IS INITIAL.

```
  SELECT matnr werks FROM marc INTO TABLE it_marc FOR ALL ENTRIES IN it_mara WHERE matnr = it_mara-matnr.
```

ENDIF.

IF NOT it\_marc IS INITIAL.

```
  SELECT werks name1 FROM t001w INTO TABLE it_t001w FOR ALL ENTRIES IN it_marc WHERE werks = it_marc-werks.
```

ENDIF.

LOOP AT it\_marc INTO wa\_marc.

```
  wa_final-matnr = wa_marc-matnr.
```

```
  wa_final-werks = wa_marc-werks.
```

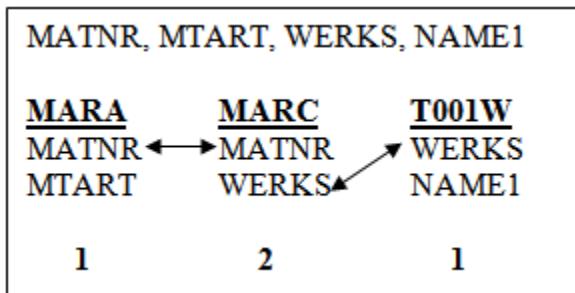
```
READ TABLE it_mara INTO wa_mara WITH KEY matnr = wa_marc-matnr.
```

```
  wa_final-mtart = wa_mara-mtart.
```

```
READ TABLE it_t001w INTO wa_t001w WITH KEY werks = wa_marc-werks.
```

```
  wa_final-name1 = wa_t001w-name1.
```

```
APPEND wa_final TO it_final.
```



**CLEAR:** wa\_final, wa\_mara, wa\_marc, wa\_t001w.

**ENDLOOP.**

**LOOP AT** it\_final **INTO** wa\_final.

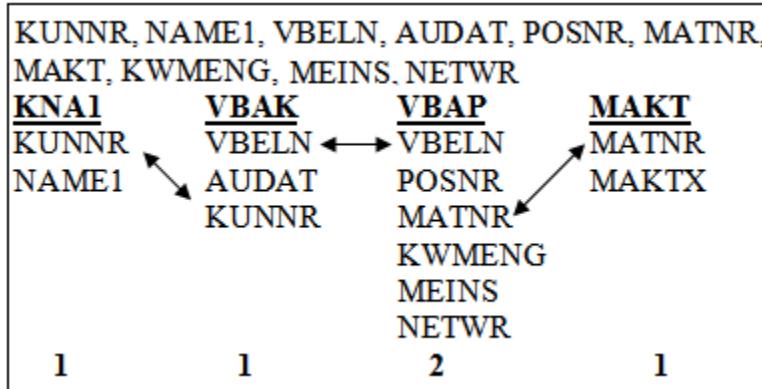
**WRITE:** / wa\_final-matnr, wa\_final-mtart, wa\_final-werks, wa\_final-name1.

**ENDLOOP.**

→ Based on the given vendor numbers  
display the vendor numbers vendor  
names, purchasing document numbers,  
document date, item number, quantity,  
unit of measurement & net price by  
using for all entries.

If not VBAP is initial.

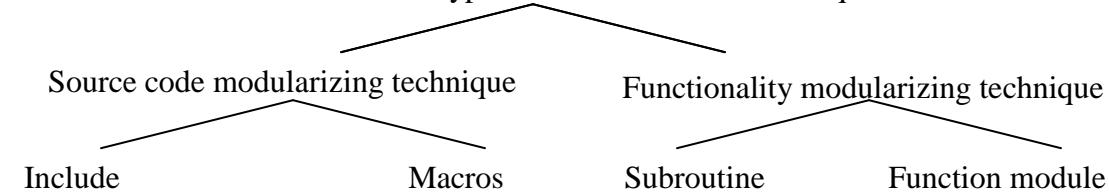
Select MATNR MAKTX from MAKT  
into table IT\_MAKT for all entries in  
IT\_VBAP where MATNR = IT\_VBAP-  
MATNR and SPRAS = SY-LANGU.



## Modularization techniques:

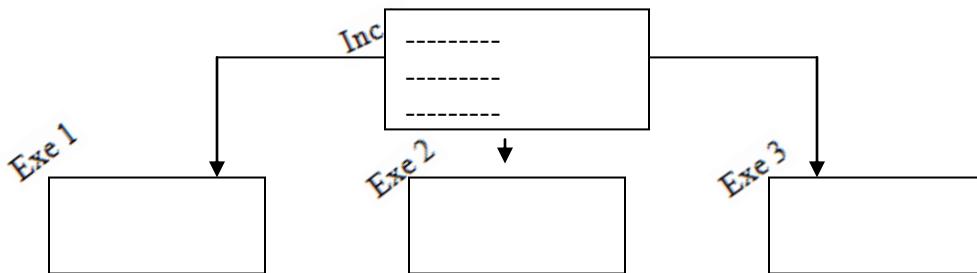
Modularization techniques are used to divide the business processing logic into reusable block of statements. This is two step procedure. 1. Define the reusable block, 2. Calling the reusable block.

2 types of modularization techniques



### Include: -

We can't execute an include independently where as the same include program can be included to any number of executable programs. Include programs are used to improve the readability of the program. In the real time include programs are used to maintain the all declarations of the program.



### Steps to create include: -

Execute SE38. Provide the include program name. Click on create. Provide title. Select the type is "Include program". Click on save, local object.

### Ex: -

```
Types: begin of ty_t001,
        Bukrs type t001-bukrs,
        Butxt type t001-butxt,
        Ort01 type t001-ort01,
        End of ty_t001.
Data: wa_t001 type ty_t001,
      it_t001 type table of ty_t001.
```

Save, check, Activate it.

### Syntax of calling the include program: -

Include <include program name>.

### Ex: -

```
Include ZWASTE101.
Select bukrs butxt ort01 from t001 into table it_t001.
Loop at it_t001 into wa_t001.
  Write: wa_t001-bukrs, wa_t001-ort01, wa_t001-butxt.
Endloop.
```

### Macros: -

Macros are used to perform the arithmetical operations. Macros can take up to 9 place holders. (&1, &2, ---&9). If you want to maintain the same set of statements more than one location of the same program instead of this we maintain those statements in macro definition later we call the same macro definition from different locations of the same program.

### Syntax of calling the macro: -

<macro name> <place holder1 value> <place holder 2 value>.

**Note:** - In macros definition should be the first and calling should be the next.

### **→ Perform the addition of two numbers by using macros.**

Data R type I.  
 Define add.  
 R = &1 + &2.  
 End-of-definition.  
 Add 20 10.  
 Write R.

Data result type I.  
 Define cal.  
 Result = &1 &2 &3.  
 End-of-definition.  
 Cal 10 \* 2.  
 Write result.  
 Cal 25 – 10.  
 Write result.

### **→ Manually filling the internal table by using macros.**

```
Types: begin of ty_t001,
        Bukrs type t001-bukrs,
        Butxt type t001-butxt,
        Ort01 type t001-ort01,
        End of ty_t001.

Data: wa_t001 type ty_t001,
      it_t001 type table of ty_t001.

Define fill_tab.
  wa_t001-bukrs = &1.
  wa_t001-butxt = &2.
  wa_t001-ort01 = &3.
  Append wa_t001 to it_t001.
  Clear wa_t001.

End-of-definition.

Fill_tab '1000' 'TCS' 'HYD'.
Fill_tab '2000' 'IBM' 'CHE'.
Loop at it_t001 into wa_t001.
  Write: / wa_t001-bukrs, wa_t001-butxt, wa_t001-ort01.
Endloop.
```

Define xyz.  
 Define abc.  
 -----  
 -----  
 End-of-definition.  
 -----  
 -----  
 End-of-definition.

**Note:** - We can't next the definition of macro. ←

### **Subroutines: -**

Subroutines are procedures. That we can define in any ABAP program & calling from the same or some other ABAP program. Procedure is the collection of statements.

### Syntax of defining the subroutine: -

Form <form name / subroutine name> using <IV 1> type <DT> <IV 2> type <DT> --- changing <OV 1> type <DT> <OV 2> type <DT> -----  
 ----- } logic  
 -----

Endform.

### Syntax of calling the subroutine: -

Perform <form name> using <IP 1> <IP 2> ---- changing <OP 1> <OP 2>-----

**Note:** - In subroutine calling is the first & definition is the next. In subroutines all the using parameters are called input parameters. All the changing parameters are called as output parameters.

We can't place the any executable statement after the definition of the subroutine.

### → Perform the addition of two numbers by using subroutines.

```
Data R type I.  
Parameter: P_input1 type I, P_input2 type I.  
Perform add1 using P_input1 P_input2 changing R.  
Write R.  
Form add1 using A type I B type I changing C type I.  
C = A + B.  
Endform.
```

There are two types of subroutines.

1. Internal subroutines.
2. External subroutines.

### Internal subroutines: -

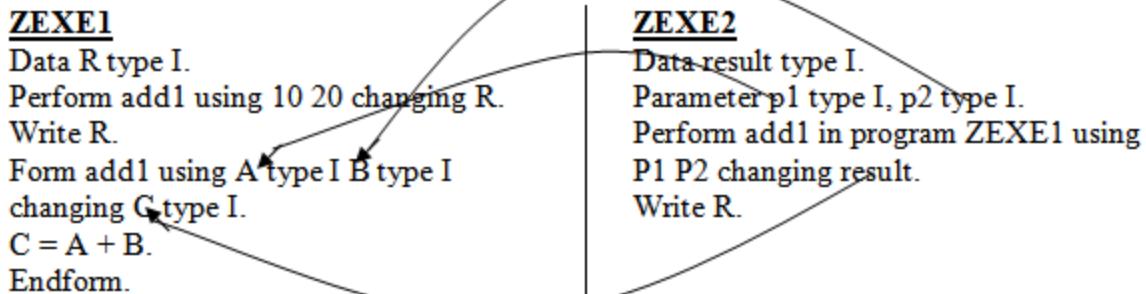
It's nothing but the definition of the subroutine as well as calling of the subroutine must be in the same program.

### External subroutine: -

It's nothing but the definition of the subroutines taken from one program & the calling of the subroutine is taken from some other ABAP program.

### Syntax of calling the external subroutine: -

Perform <form name> in program <program name> using <IP 1> <IP 2> ---- changing <OP 1> <OP2> ---



### Differences between Macros & subroutines: -

#### Macros

1. In macros definition & calling in the same program
2. In macros definition should be the first & calling should be next.
3. Macros contain up to 9 inputs.
4. After the definition of macro we can place any executable statement.
5. Macros are used in HR-ABAP.

#### Subroutines

1. In subroutines definition & calling may / mayn't in the same program.
2. In subroutines calling should be first & definition should be next.
3. Subroutine contain any number of inputs.
4. After the definition of subroutine we can't place any executable statement.
5. Subroutines are used in both HR-ABAP & ABAP.

### **Termination of subroutine: -**

Subroutines are normally ends with endform. If you want to terminate the subroutine so earlier then we use exit or check command. Exit command is used to terminate the subroutine unconditionally. Check command is used to terminate the subroutine conditionally.

```
Perform spexit.
```

```
Form spexit.
```

```
Write 'SJ Technologies' .
```

```
Write / 'SR Nagar' .
```

```
Exit.
```

```
Write / 'HYD' .
```

```
Endform.
```

OP: - SJ Technologies  
SR Nagar

Parameter P\_BSART type EKKO-BSART.

Perform spcheck using P\_BSART.

Form spcheck using A type EKKO-BSART.

Check A = 'DOM'.

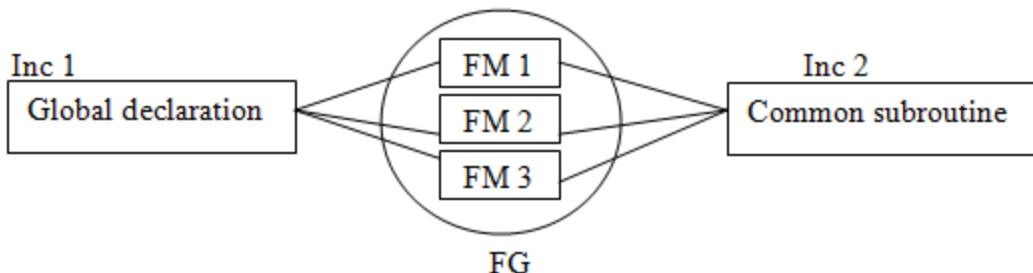
-----

-----

Endform.

### **Function module: -**

Function modules are reusable components that are defined in functional library.



Each function module must be attached to one function group. Each function group contains two include programs by default. One is for global declaration another one is for common subroutines. All the function modules can access the both the include programs.

When ever we are calling any one of the function module then all other function modules will be loaded into the memory of calling program. So it's better to group related function module into one function group.

Function modules allow you to encapsulate and reuse global functions in the R/3 System. They are stored in a central library, Unlike subroutines, you do not define function modules in the source code of your program.

**Note:** - 'SE37' is the transaction code for function builder.

### **Steps to create function group: -**

Execute SE37. In the menu bar click on GO TO → Function groups → Create group. Provide the function group name, short description. Save. Click on local object.

### **Steps to activate function group: -**

In the menu bar click on environment → inactive objects. Expand the function group under local objects. Select the function group. Right click on it. Click active. Enter.

## **Components of function module: -**

1. Attributes
2. Import
3. Export
4. Changing
5. Tables
6. Exceptions
7. Source code

### **Attributes: -**

Attributes specify the type of function module whether it's a **normal** or **remote enable** or **Updated Function Module**. If the function module type is normal then we can access the function module within the server only. If the function module type is remote enable then we can access the function module within the server as well as outside the server also. Update Function modules are used to update the database tables. Using the function module in updating the database table helps in updating the database without any irregularities.

### **Import: -**

Import acts like input parameters [using in the server].

### **Export: -**

Export acts like output parameters [changing in the server].

### **Changing: -**

Changing acts like both import & export. Tables parameters are currently replaced by **CHANGING**.

### **Tables: -**

Tables acts like both import & export only for internal tables.

### **Exceptions: -**

These are used to handle the errors in function module.

### **Source code: -**

It contains the logic related function module.

**→ Develop a function module to display the all the purchasing order header details based on the given purchasing document number.**

### **Steps to create function module: -**

Execute SE37. Provide the function module name. Click on create. Provide the function group name. Provide the short note. Click on save. Enter. Click on import. Provide parameter name, type, associated type. Click on export. Provide parameter name, type, associated type. Click on source code. Write source code.

### **Ex: -**

Select single \* from EKKO into E\_WA where EBELN = I\_EBELN.

Endfunction.

Save, check, activate the function.

**Note:** – Function module writes single value & multiple values. So we no need to maintain any display statement (write) in the function module definition.

**Note:** - We can test the function module independently without calling the function module.

### **Function module definition**

Import -----  
Export -----  
Changing -----  
Tables -----

### **Function module calling**

Export  
Import  
Changing  
Tables

### **Steps to call the function module:**

Place the cursor where you want to call the function module. Click on pattern in the application tool bar. Provide the function module name.

Parameter p\_ebeln type ekko-ebeln.  
Data wa\_ekko like ekko.  
Call function 'ZSPT\_930\_fun'  
Exporting  
i\_ebeln = p\_ebeln.  
Importing ←  
e\_wa = wa\_ekko.  
Write: / wa\_ekko-ebeln, wa\_ekko-bedat,  
wa\_ekko-lifnr.

Function module definition  
ZSP\_930\_fun  
→ Import  
i\_ebeln type ekko-ebeln.  
Export  
e\_wa like ekko.  
Source code  
Select single \* from ekko into e\_wa  
where ebeln = i\_ebeln.

### **Difference between subroutines & function module:**

#### **Subroutine**

1. Subroutines are local that means we can access the subroutine within the server only.
2. We can't test the subroutine independently without calling the subroutine.
3. We can't handle the errors in subroutine.
4. Subroutines are defined in ABAP-Editor.

#### **Function module**

1. Function modules are global. That means we can access the function module with in the server or as well as outside the server also.
2. We can test the function module independently without calling.
3. We can handle the errors in function module through exceptions.
4. Function modules are defined in function builder SE37.

### **→ Develop a function module to display the company codes company name & city based on the given company.**

If you want to declare the work area with some of the fields in function module then we must create one structure with those fields in the data dictionary & later we refer the structure in the function module [export tab].

In this object we must create the structure with bukrs, butxt, ort01.

### **Steps to create structure:**

Execute SE11. Select the radio button data type. Provide the structure name. Click on create. Select the radio button structure. Enter. Provide the short description. Provide component, component type.

Bukrs        bukrs  
Butxt        butxt  
Ort01        ort01

Save, check, activate.

Function module ZSPT\_930\_FM1.

### Import

<u>Parameter name</u>	<u>type</u>	<u>associated type</u>
i_bukrs	type	t001-bukrs

### Export

<u>parameter name</u>	<u>type</u>	<u>associated type</u>
e_wa	like	ZSPT_930_FS1.

### Source code

Select single bukrs butxt ort01 from t001 into e\_wa where bukrs = i\_bukrs.

Endfunction.

```
Parameter p_bukrs type t001-bukrs.  
Data wa_t001 like ZSPT_930_FS1.  
Call function 'ZSPT_930_FS2'.  
Exporting  
i_bukrs = p_bukrs.  
Write: / wa_t001-bukrs, wa_t001-butxt,  
wa_t001-ort01.
```

### Function module definition

ZSPT\_930\_FM2.

#### Import

i\_bukrs type t001-bukrs.

#### Export

E\_wa like ZSPT\_930\_FS1.

#### Source code

Select single bukrs butxt ort01 from t001  
into e\_wa where bukrs = i\_bukrs.

→ Based on the given vendor number display the vendor number, purchasing document numbers, document types by using function module.

If you want to declare the internal table with some of the fields in the function module then we must create one structure with those fields & later we refer the structure in the tables tab of function module.

In this object we must create one structure with lifnr, ebeln, bsart after we refer the structure in the tables tab of function module.

Structure – ZSPT\_930\_FS2

<u>Component</u>	<u>Component type</u>
Lifnr	elifnr
Ebeln	ebeln
Bsart	esart

Save, check, activate.

Go to SE37 function module

### Import

<u>Parameter name</u>	<u>type</u>	<u>associated type</u>
i_lifnr	type	ekko-lifnr

### Tables

<u>Parameter name</u>	<u>type</u>	<u>associated type</u>
IT	like	ZSPT_930_FS2

### Exception

<u>Exception</u>	<u>Short note</u>
NODATA	invalid input

### Source code

Select lifnr ebeln bsart from ekko into table IT where lifnr = i\_lifnr.

```

If sy-subrc NE 0.
Raise NODATA.
Endif.

Execute SE38.
Parameter p_lifnr type ekko-lifnr.
Data IT_ekko type ZSPT_930_FS2.
Data wa_ekko like line of it_ekko.
Call function 'ZSPT_930_FM3'.
Exporting.
i_lifnr = p_lifnr.
Tables
IT = IT_ekko.
Exporting
NODATA = 1
OTHERS = 2.
If SY-SUBRC <> 0.
Write: 'INVALID INPUT'.
Else.
Loop at IT_ekko into wa_ekko.
Write: / wa_ekko-lifnr, wa_ekko-ebeln, wa_ekko-bsart.
Endloop.
Endif.

```

**Note:** - In the function module all parameters are either pass by value or pass by reference. By default all parameters are pass by reference. Pass by reference means from calling to definition & definition to calling all parameter values are passing along with the memory. Pass by value means from calling to definition & definition to calling only parameters values are passed.

**Note:** - In the remote enable function module all parameters are pass by value only.

### **Select-options:** -

The name of the select options acts like an internal table with header line that means the name of the work area as well as the name of the internal table is the similar name of the select options.

### **Syntax:** -

Select-options <select-options name> for <variable>.

### **Ex:** -

Data v1 type t001-bukrs.

Select-options s\_bukrs for v1.

**Note:** - If you want to declare the work area with the some of the fields in the function module then we must create one structure with those fields in the data dictionary & later we refer the structure in the import, export, changing tab.

**Note:** - If you want to declare the internal table with some of the fields in the function module then we must create one structure with those fields & later we refer the structure in the tables tab (or) Create one table type with those fields in the data dictionary later we refer the table type in the import, export or changing tab.

**Note:** - If you want to declare the select-options in the function module then we must create one structure with the following fields later we refer the structure in the tables tab of function module.

#### **Components of select-options:** -

1. Sign → Include (I) / Exclude (E).
2. Option → between (BE) / Not Between (NB) / Equals (EQ) ....
3. Low → The low value of the select-options.
4. High → the high value of the select-options.

<i>s_bukrs</i>	Sign	Opcion	Low	High

Sign = I  
Option = BT  
**X** 1000 **✓** 4000 **X**

<i>s_bukrs</i>	Sign	Opcion	Low	High

Sign = I  
Options = NB.      Opcion  
**✓** 1000 **X** 4000 **✓**

Sign = E  
Option = BE

**X** 1000 **X** 4000 **X**

Sign = E  
Option = NB

**✓** **X** 1000 **X** 4000 **✓**

**→ Based on the given company code, display the company codes, names, cities by using function module.**

If you want to declare the select-option the function module then we must create one structure with the following fields in the data dictionary & later we refer the structure in the tables tab.

Sign → (C,1)

Option → (C,2)

Low }  
High } Depends of fields

In this object we must create two structures one is for input parameter (sign option low, high). One is for output (bukrs, butxt, ort01) both structures are referred in tables tab only.

#### **Steps to create structure select-options:** -

Execute SE11. Select the radio button data type. Provide the structure name, click on create. Select the radio button structure. Enter. Provide short description. Click on pre defined type.

<u>Component</u>	<u>data type</u>	<u>length</u>
SIGN	CHAR	1
OPTION	CHAR	2
LOW	CHAR	4
HIGH	CHAR	4

Save, check, activate the structure.

Function module: ZSPT\_930\_fm4.

#### Tables

<u>Parameter name</u>	<u>type</u>	<u>associated type</u>
i_bukrs	like	zspt_930_fs_so
e_it	like	zspt_930_fs1

#### source code

```
select bukrs butxt ort01 from t001 into table e_it where bukrs in i_bukrs.
Endfunction.
```

Data v1 type t001-bukrs.

Select-options s\_bukrs for v1.

Data it\_t001 like table of zspt\_930\_fs1.

Data wa\_t001 like line of it\_t001.

Call function 'zspt\_930\_fm4'.

Tables

i\_bukrs = s\_bukrs

Loop at it\_t001 into wa\_t001.

Write: / wa\_t001-bukrs, wa\_t001-butxt, wa\_t001-ort01.

Endloop.

→ Based on the given vendor numbers display the vendor numbers, vendor names, purchasing document numbers by using function module.

#### Structure ystr5

Lifnr	lifnr	char	10
Name1	name1	char	30
Ebeln	ebeln	char	10

#### Structure ystr6

Sign	char	1
Option	char	2
Low	char	10
High	char	10

Function module yfm5

#### Tables

<u>Parameter name</u>	<u>kind</u>	<u>associated type</u>
i_lifnr	like	ystr6
e_it	like	ystr5

#### Source code

Select LFA1~LIFNR LFA1~NAME1 LFA1~EBELN into table e\_it from LFA1 inner join EKKO on LFA1~LIFNR = EKKO~LIFNR where LFA1~LIFNR in i\_lifnr.  
Endfunction.

## Report yprogram2

Data v1 type LFA1-LIFNR.

Select-options s\_lifnr for v1.

Data it\_final like table of ystr5.

Data wa\_final like line of it\_final.

Call function 'yfm5'

Tables

i\_lifnr = s\_lifnr

e\_it = it\_final.

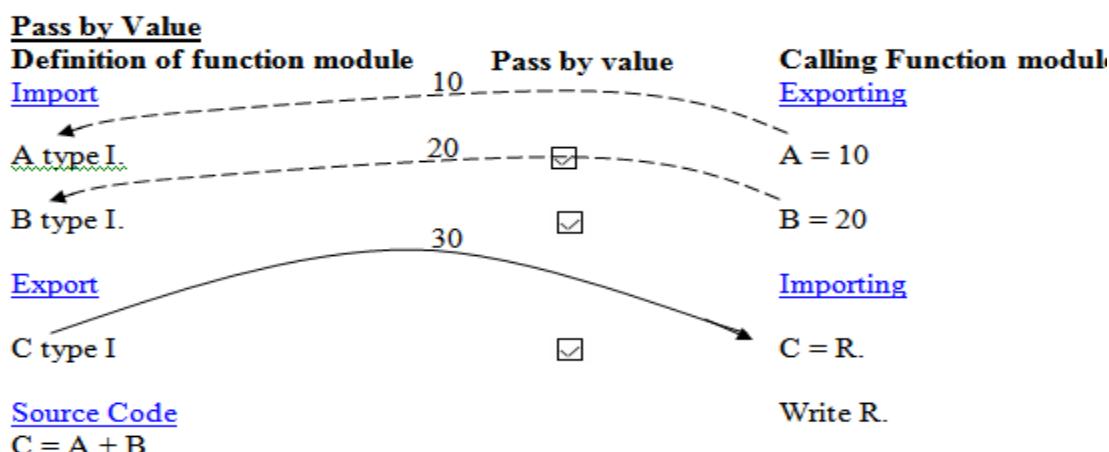
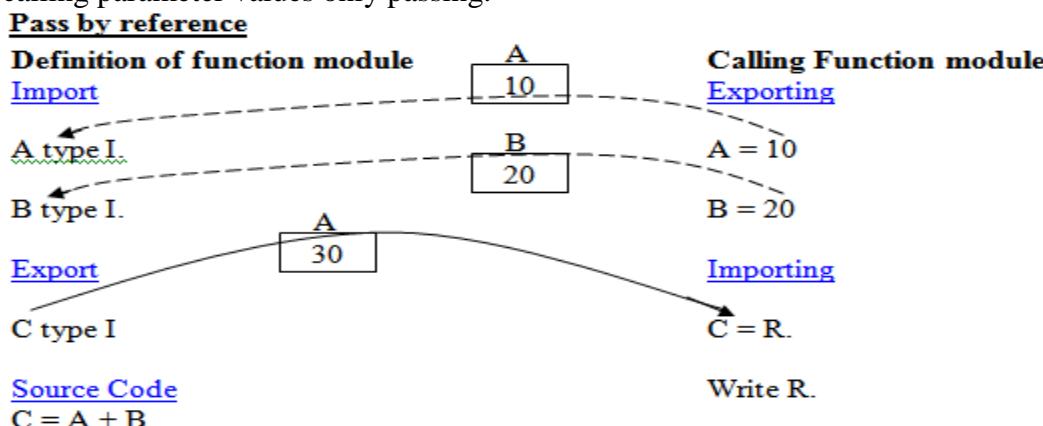
Loop at it\_final into wa\_final.

Write: / wa\_final-lifnr, wa\_final-name1, wa\_final-ebeln.

Endloop.

**Note:** - When ever we are working with remote enable function module then we must select the all parameters are pass by value.

**Note:** - In the function module all parameters are either pass by reference or pass by value. By default all parameters are pass by reference. Pass by reference means from calling to definition & definition to calling all parameter values along with the memory is transferring. Pass by value means from calling to definition & definition to calling parameter values only passing.



**Note:** - In Pass by value only values are passed from calling to definition part. Both actual & formal parameters will share different memory locations. Any changes made in formal parameters, will not be effected to actual ones.

```
DATA X(2) TYPE C VALUE '10'.
PERFORM HAI USING X.
WRITE:/ X.
FORM HAI USING P.
P = '20'.
WRITE:/ P.
ENDFORM.
```

Output will be like this

20  
10

**Note:** - In Pass by reference a reference of actual parameters is passed to formal parameters. Any changes made in formal parameters are automatically reflected to actual ones.

```
DATA X(2) TYPE C VALUE '10'.
PERFORM HAI CHANGING X.
WRITE:/ X.
FORM HAI CHANGING P.
P = '20'.
WRITE:/ P.
ENDFORM.
```

Output will be like this

20  
20

Here Actual parameters are nothing but which parameters are in Definition part. Formal parameters are nothing but the parameters which are in calling part.

Actually function modules are 4 types. Those are **Normal FM**, **Remote Enable FM**, **ALV FM**, **Update FM**.

Normal, Remote Enable function modules are already explained at the time of beginning of Function modules.

**ALV Function Module:** - ALV's are normal function modules. As they improves performance of a program (by avoiding loop ---- endloop). So they are separated from normal ones.

**Update Function Module:** - It's for LUW (logical Unit of Work).

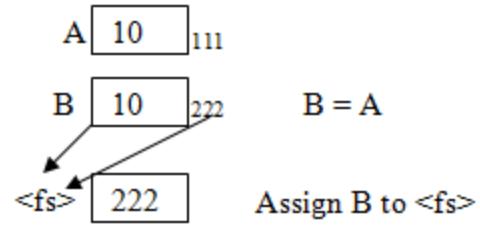
**Note:** - Exporting & Importing parameters work based on 'Pass By Value'. Tables & Changing parameters work based on 'Pass By Reference'.

**Note:** - Field symbols hold the reference of other variables and writens value stored in the reference.

```

FIELD-SYMBOLS <FS>.
DATA A TYPE I VALUE 10.
DATA B TYPE I.
B = A.
ASSIGN B TO <FS>.
IF <FS> IS ASSIGNED.
  WRITE:/ <FS>.
ENDIF.

```



**Note:** – The difference between ordinary variable and field symbol is ordinary variables memory allocated during at runtime. Field symbols memory is allocated when ever assign keyword is executed.

Field symbols improve the performance of the programming.

### → Find the time taking between work area performance and field symbol performance.

```

TYPES: BEGIN OF TY_LFA1,
  LIFNR TYPE LFA1-LIFNR,
  NAME1 TYPE LFA1-NAME1,
  ORT01 TYPE LFA1-ORT01,
END OF TY_LFA1.

DATA: WA_LFA1 TYPE TY_LFA1,
  IT_LFA1 TYPE TABLE OF TY_LFA1.

FIELD-SYMBOLS <FS> TYPE TY_LFA1.
DATA : TIME1 TYPE I,
  TIME2 TYPE I,
  TIME3 TYPE I.

SELECT LIFNR NAME1 ORT01 FROM LFA1 INTO TABLE IT_LFA1.

GET RUN TIME FIELD TIME1. "CURRENT EXECUTION TIME
LOOP AT IT_LFA1 INTO WA_LFA1.
  WA_LFA1-ORT01 = 'HYDERABAD'.
  MODIFY IT_LFA1 FROM WA_LFA1 TRANSPORTING ORT01.
ENDLOOP.
GET RUN TIME FIELD TIME2. "CURRENT EXECUTION TIME
TIME3 = TIME2 - TIME1. "TOTAL TIME TAKEN FOR EXECUTION
WRITE:/ 'TIME TAKEN FOR EXECUTIN NORMAL LOOP', TIME3.
CLEAR: TIME1, TIME2, TIME3.

GET RUN TIME FIELD TIME1.
LOOP AT IT_LFA1 ASSIGNING <FS>.
  <FS>-ORT01 = 'HYDERABAD'.
ENDLOOP.
GET RUN TIME FIELD TIME2.
TIME3 = TIME2 - TIME1.
WRITE:/ 'TIME TAKEN FOR EXECUTIN FIELD SYMBOL LOOP', TIME3.

```

## REPORTS

Report is the combination of given inputs to the selection-screen retrieving the data from data base based on the given input & display the output in a predefined format.

### Syntax of selection-screen: -

Selection-screen begin of block <block name> with frame title text-.

Optional      Optional

Selection-screen end of block  
    <block name>.

**Syntax of check box: -**

Parameter <name of the check box> as check box.

Ex: -

Parameter P\_DIS as checkbox.



`NO INTERVALS` is the keyword to display multiple single values in select-options. It'll display like parameter but work like select-options.

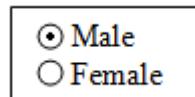
NO-EXTENSION can allow only one range. Multiple selections are not possible & push button will be disappeared.

**Syntax of radio button: -**

Parameter <name of the radio button> radiobutton group <group name>.

**Ex:** -

Parameter: Male radiobutton group g,  
Female radiobutton group g.



→ Design the following selection-screen.

Selection criteria	
S_KUNNR	<input type="text"/>
to <input type="text"/> ▶	
<input type="checkbox"/> Display <input type="checkbox"/> Non display	

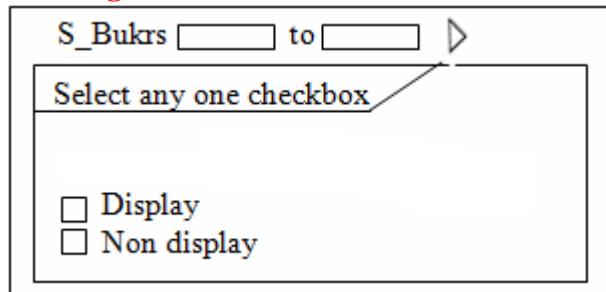
```
data v1 type kna1-kunnr.  
selection-screen begin of block a with frame title text-001.  
select-options s_kunnr for v1.  
parameter p_dis as checkbox.  
parameter p_nondis as checkbox.  
selection-screen end of block a.
```

If you want to provide the meaningful descriptions to the input variables then in the menu bar click on go to → text elements → selection texts. It displays the all available inputs. If the field is coming from dictionary then select the dictionary check box otherwise we manually provide the text.

P_DIS	Display	
P_Non display	Non display	
P_KUNNR		✓

Save, activate, back.

→ Design the selection-screen as shown in the figure.



```
data v1 type t001-bukrs.  
selection-screen begin of block a with frame.  
select-options s_BUKRS for v1.  
selection-screen begin of block b with frame title text-001.  
parameter p_dis as checkbox.  
parameter p_nondis as checkbox.  
selection-screen end of block b.  
selection-screen end of block a.
```

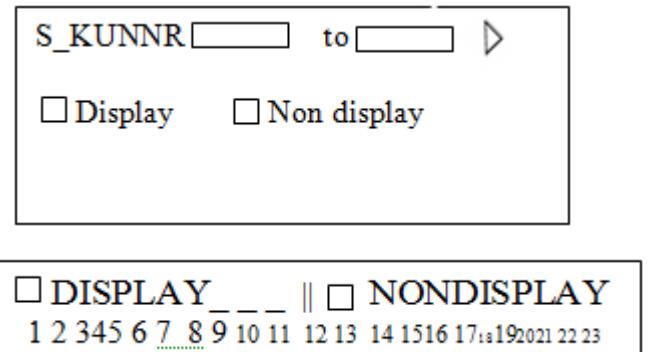
When ever we are working with begin of line & end of line then the name of the parameter is disable. So we must provide comment before, after block of a checkbox or radiobutton.

Syntax: - Total length of the parameter

Selection-screen comment x(y) text-<no> → 3 digit number (For name of the parameter)  
Starting position

→ Design the selection-screen as shown in the figure.

```
data v1 type knal-kunnnr.  
selection-screen begin of block a  
with frame.  
select-options s_kunnnr for v1.  
selection-screen begin of line.  
parameter p_dis as checkbox.  
selection-screen comment 2(10)  
text-002.  
parameter p_nondis as checkbox.  
selection-screen comment 14(11)  
text-003.  
selection-screen: end of line,  
end of block a.
```

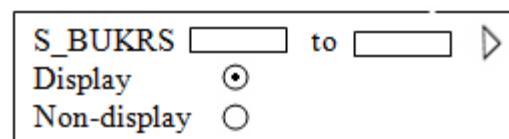


→ Design the selection-screen as shown in the figure.

Skip is the keyword to provide the space in between any two input variables in the selections name. By default skip is one line. Maximum we can skip up to 9 lines at a line.

```
DATA V1 TYPE T001-BUKRS.  
SELECTION-SCREEN BEGIN OF BLOCK A  
WITH FRAME.  
SELECT-OPTIONS S_BUKRS FOR V1.
```

एम एन सतीष कुमार रेडि



```

SELECTION-SCREEN BEGIN OF LINE.
SELECTION-SCREEN COMMENT 1(12) TEXT-001.
PARAMETER P_DIS RADIobutton GROUP B.
SELECTION-SCREEN END OF LINE.
SELECTION-SCREEN BEGIN OF LINE.
SELECTION-SCREEN COMMENT 1(12) TEXT-002.
PARAMETER P_NONDIS RADIobutton GROUP B.
SELECTION-SCREEN: END OF LINE,
END OF BLOCK A.

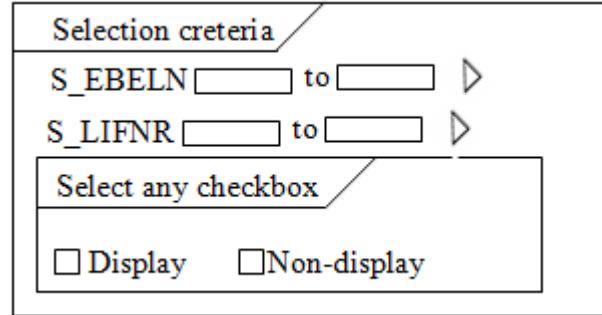
```

→ Design the selection-screen as shown in the figure

```

DATA V1 TYPE EKKO-EBELN.
DATA V2 TYPE EKKO-LIFNR.
SELECTION-SCREEN BEGIN OF BLOCK
A WITH FRAME TITLE TEXT-003.
SELECT-OPTIONS S_EBELN FOR V1.
SELECT-OPTIONS S_LIFNR FOR V2.
SELECTION-SCREEN BEGIN OF BLOCK
B WITH FRAME TITLE TEXT-004.
PARAMETER P_DIS AS CHECKBOX.
PARAMETER P_NONDIS AS CHECKBOX.
SELECTION-SCREEN: END OF BLOCK B,
END OF BLOCK A.

```



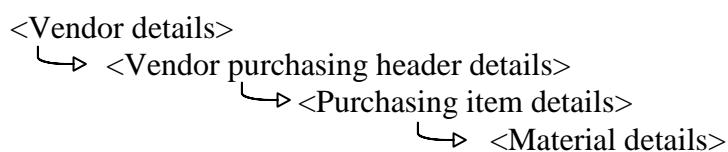
Types of reports as per ABAPer requirement: -

**Types of Reports:-**

1. Classical Reports
2. Interactive Reports
3. ALV Reports

**Classical Reports:** - It's nothing but to display the entire information in a single list

Ex: -



**Events in classical reports :** -

1. Initialization
2. At selection-screen
3. At selection-screen on
4. Start-of-selection
5. End-of-selection
6. Top-of-page
7. End-of-page

**Initialization:** -

It's an event which is triggered before displaying the selection-screen.

**ADV:** - it's used to provide the default values to the selection-screen.

### **At selection-screen: -**

It's an event which is triggered after providing the input to the screen & before leaving the selection-screen.

**ADV:** - This is used to validate the given input.

### **At selection-screen on: -**

It's an event which is triggered at the selection-screen based on particular input field.

**ADV:** - This is used to validate the particular input field.

### **Start-of-selection: -**

It's an event which is triggered after leaving the selection-screen & before displaying the output.

**ADV:** - This is used to fetch the data from database & place into internal table.

### **End-of-selection: -**

It's an event which is triggered after completion of the logic.

**ADV:** - This is used to display the output.

### **Top-of-page: -**

It's an event which is triggered at the top of each page.

**ADV:** - It's used to display the header information.

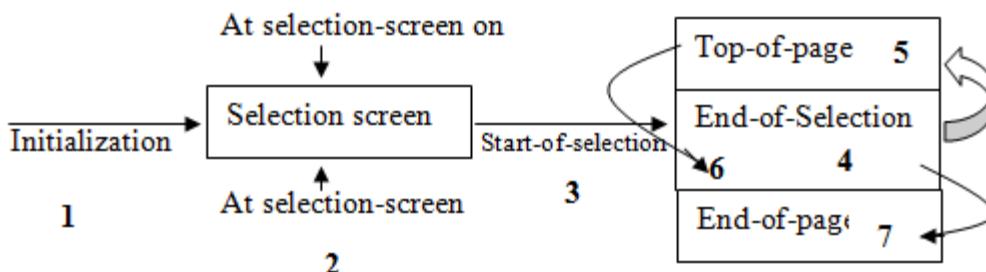
### **End-of-page: -**

It's an event which is triggered at the end of each page.

**ADV:** - It's used to display the footer information.

**Note:** - Start-of-selection is the default event in the classical report

### **Process flow of events: -**



### **Some more events in classical report: -**

1. At selection-screen output
2. At selection-screen on value-request
3. At selection-screen on help-request

### **At selection-screen output: -**

It's an event which is triggered at the selection-screen based on the user action.

**ADV:** - This is used to modify the selection-screen.

### **At selection-screen on value-request: -**

It's an event which is triggered at the time of click on F4 button.

**ADV:** - This is used to provide the list of possible values to the input variables.

## At selection-screen on help-request: -

It's an event which is triggered at the time of click on F1 button.

**ADV:** - This is used to provide the help document to the input variable

At selection-screen output is the first triggering event in the selection-screen.

**Note:** - When ever the program is loaded into the ABAP editor then automatically load-of-program event will be triggered. We never write any code on load-of-program event.

→ Design as well as provide the default values in the selection-screen as shown in the figure.

Data v1 type vbak-vbeln.

Selection-screen begin of block A with frame.

Select-options s\_vbeln for v1.

Parameter p\_dis as checkbox.

Parameter p\_nondis as checkbox.

Selection-screen end of block A.

Initialization.

s\_vbeln-sign = 'I'.

s\_vbeln-option = 'BT'.

s\_vbeln-low = '4969'.

s\_vbeln-high = '4975'.

Append s\_vbeln.

Clear s\_vbeln.

s\_vbeln-sign = 'I'.

s\_vbeln-option = 'BT'.

s\_vbeln-low = '4980'.

s\_vbeln-high = '4988'.

Append s\_vbeln.

Clear s\_vbeln.

s\_vbeln-sign = 'I'.

s\_vbeln-option = 'EQ'.

s\_vbeln-low = '4990'.

Append s\_vbeln.

Clear s\_vbeln.

p\_dis = 'X'.

→ Design as well as assigned the default values to the selection screen as shown in the figure.

Data v1 type ekko-bukrs.

Data v2 type ekko-ebeln.

Selection-screen begin of block A with Frame title text-001.

Select-options s\_bukrs for v1.

Select-options s\_ebeln for v2.

Selection-screen begin of line.

Parameter p\_dis as checkbox.

Selection-screen comment 2(11) text-002.

Parameter p\_nondis as checkbox.

Selection-screen comment 15(11) text-003.

Selection-screen end of line.

Selection-screen end of block A.

S_VBELN	4969	to	4975	▷
	4980		4985	
	4990			
<input checked="" type="checkbox"/> Display				
<input type="checkbox"/> Non-display				

Selection criteria				
S_BUKRS	1000	to	3000	▷
	5000		7000	
S_EBELN	3000000004	to	3000000010	▷
	3000000010			
<input checked="" type="checkbox"/> Display <input type="checkbox"/> Non-display				

```

Initialization.
s_bukrs-sign = 'I'.
s_bukrs-option = 'BT'.
s_bukrs-low = '1000'.
s_bukrs-high = '3000'.
Append s_bukrs.
Clear s_bukrs.
s_bukrs-sign = 'I'.
s_bukrs-option = 'BT'.
s_bukrs-low = '5000'.
s_bukrs-high = '7000'.
Append s_bukrs.
Clear s_bukrs.
s_ebeln-sign = 'I'.
s_ebeln-option = 'BT'.
s_ebeln-low = '3000000004'.
s_ebeln-high = '3000000008'.
Append s_ebeln.
Clear s_ebeln.
s_ebeln-sign = 'I'.
s_ebeln-option = 'EQ'.
s_ebeln-low = '3000000010'.
Append s_ebeln.
Clear s_ebeln.
p_dis = 'X'.

```

### Differences between select single, select up to 1 rows

#### Select single

1. It fetch the only one record.
2. Here we must pass entire primary key combination in where condition.
3. It hits the data base only once.
4. This is used to fetch the exact record.

#### Select up to 1 rows

1. It also fetch only one record.
2. Enough to pass part of key combination in the where condition it always pick the first record among the method once.
3. It hit the data base twice.
4. This is used for validation.

#### Syntax: -

Select single <field1> <filed2> ... from <data base> table into <work area> where condition.

#### Syntax: -

Select <field1> <filed2> ... from <data base table> into <work area> up to 1 rows where <condition>. Endselect.

```

Types: begin of ty_knb1,
       bukrs type knb1-bukrs,
       kunnr type knb1-kunnr,
       akont type knb1-akont,
       end of ty_knb1.

```

```
Data wa_knb1 type ty_knb1.
```

```
Select single bukrs kunnr akont from knb1 into wa_knb1 where kunnr
= '000000224' and
```

```

bukrs = '5000'.
Write: / wa_knb1-kunnr, wa_knb1-bukrs, wa_knb1-akont.
* select bukrs kunnr akont from knb1 into wa_knb1 up to 1 rows
where kunnr = '0000000224'.
* endselect.
* write: / wa_knb1-kunnr, wa_knb1-bukrs, wa_knb1-akont.

```

**Note:** - We always choose the select single if you know the entire primary key combination otherwise we choose the select up to 1 rows.

### **Message:** -

We've 4 different types messages.

1. Abend message (A) → <Message> **Enter** Terminates the entire transaction
  
2. Warning message (W) → Status bar **Enter**
  - i. If you are in the basic list then it goes to program.
  - ii. If you are in the secondary list then it goes to previous list.
  
3. Information message (I) → <Message> **Enter** Goes to selection-screen.
  
4. Success message (S) → Status bar **Enter** Nothing happened

### **Syntax:** -

Message <message type><message number> (<message class>).

↳ 3 digit number

**Note:** - The transaction code for message class creation is 'SE91'

### **Steps to create the message class:** -

Execute SE91. Provide the message class name. Click on create. Provide short description. Click on save. Click on messages tab. Provide messages against the messages.

000 Less than 10

001 Grater than 10

002 Equal to 10

Click on save.

### **Ex:** -

Parameter P type I.

If P < 10.

Message I000(zmessage1).

Elseif p > 10.

Message I001(zmessage1).

Else

Message I002(zmessage1).

Endif.

**Note:** - In the real time we always use only one message class only one message number against that number we maintain the place holders.

### Syntax:-

Message <message type><message number> (<message class>) with ‘<message>’.

Zmessage2

000		&	&	&	
-----	--	---	---	---	--

Parameter p type I.

If p < 10.

Message I000(zmessage2) with ‘less than 10’.

Elseif p > 10.

Message I000(zmessage2) with ‘grater than 10’.

Else

Message I000(zmessage2) with ‘equal to 10’.

Endif.

**Note:** – When ever the system execute any message then the further statements are not executed.  
(Execution is stopped.)

### Field validation table

BUKRS	→ T001
KUNNR	→ KNA1
LIFNR	→ LFA1
EBELN	→ EKKO
VBELN	→ VBAK
MATNR	→ MARA
WERKS	→ T001W
LGORT	→ T001L

```

tables kna1.
select-options s_kunnr for kna1-kunnr.
data v type kna1-kunnr.
At selection-screen.
  Select kunnr from kna1 into v up to 1 rows where kunnr in s_kunnr.
Endselect.
If sy-subrc <> 0.
  Message E000(zmessage2) with 'Invalid customer'.
ENDIF.

```

**Note:** – All inputs are wrong then only SY-SUBRC value is not equal to zero.

**Note:** – Now-a-days in the real time we always use select single to validate the input. Because select single is faster than up to 1 rows (select single hit the database once. Up to 1 rows hit the database twice.)

```

tables: ekko, lfa1.
data: v type ekko-ebeln, v1 type lfa1-lifnr.
select-options: s_ebeln for ekko-ebeln, s_lifnr for lfa1-lifnr.
At selection-screen.
  Select single ebeln from ekko into v where ebeln in s_ebeln.
  If sy-subrc <> 0.
    Message E000(zmessage3) with 'Invalid number'.

```

Endif.

At selection-screen on s\_lifnr.

Select single lifnr from lfa1 into V1 where lifnr in s\_lifnr.

If sy-subrc <> 0.

Message E000(zmessage3) with 'Invalid vendor'.

Endif.

**Note:** - If you want to validate the entire selection-screen then we use at selection-screen event. If you want to validate a particular field then we go for 'at selection-screen on'. In this case also at selection-screen work but performance wise poor compare to 'at selection-screen on'.

```
tables t001.  
selection-screen begin of block a with frame.  
select-options s_bukrs for t001-bukrs.  
parameter p_dis as checkbox.  
parameter p_nondis as checkbox.  
selection-screen end of block a.  
types: begin of ty_t001,  
        bukrs type t001-bukrs,  
        butxt type t001-butxt,  
        ort01 type t001-ort01,  
        end of ty_t001.  
data: wa_t001 type ty_t001,  
      it_t001 type table of ty_t001.  
Start-of-selection.  
  Select bukrs butxt ort01 from t001 into table it_t001 where  
bukrs in s_bukrs.
```

End-of-selection.

If p\_dis = 'X'.

Loop at it\_t001 into wa\_t001.

Write: / wa\_t001-bukrs, wa\_t001-butxt, wa\_t001-ort01.

Endloop.

Else.

message I000(zmessage1) with 'select the display checkbox'.

endif.

Top-of-page.

Write: 'SATISH TECHNOLOGIES'.

End-of-page.

Write '301, Tirumalagiri complex, SR Nagar'.

**Note:** - One event is always ends with another event.

**Note:** - When ever we are working with events then we no need to follow the order of the events.

**Note:** - When ever we are working with end-of-page then we must provide the LINE-COUNT in the name of the report. Other wise the footer information is not printed.

**Syntax:-**

Report <report name> line-count X(Y).  
                          Number of footer lines  
                          Number of lines per page

**Note:** - If you want to avoid the title in the output then we must provide NO STANDARD PAGE HEADING in the name of the report.

→ Based on the given purchasing document numbers display the purchasing document numbers, document dates & vendor numbers by using classical event reports & also display the top-of-page as 'DHAWAN TECHNOLOGIES' & end of page as 'SR Nagar'

REPORT ZWASTE102 line-count 10(1).

Data v1 type ekko-ebeln.

Selection-screen begin of block A with frame.

Select-options s\_ebeln for v1.

Selection-screen end of block A.

Types: Begin of ty\_ekko,

    Ebeln type ekko-ebeln,

    Bedat type ekko-bedat,

    Lifnr type ekko-lifnr,

    End of ty\_ekko.

Data: wa\_ekko type ty\_ekko,

    it\_ekko type table of ty\_ekko.

Data V type ekko-ebeln.

At selection-screen.

    Select single ebeln from ekko into V where ebeln in s\_ebeln.

    If sy-subrc <> 0.

        Message E000(zmessage3) with 'Invalid purchasing document'.

    Endif.

start-of-selection.

    Select ebeln bedat lifnr from ekko into table it\_ekko where ebeln in s\_ebeln.

End-of-selection.

    Loop at it\_ekko into wa\_ekko.

        Write: / wa\_ekko-ebeln, wa\_ekko-lifnr, wa\_ekko-bedat.

    Endloop.

Top-of-page.

    Write 'DHAWAN TECHNOLOGIES'.

End-of-page.

    Write 'SR NAGAR'.

→ Based on the given customer numbers display the customer numbers, customer names & cities by using classical report events & also provide top of page as THESE ARE CUSTOMER DETAILS' & end of page as 'SATISH TECHNOLOGIES'.

Data v1 type knal-kunnr.

Selection-screen begin of block A with frame.

Select-options s\_kunrn for v1.

Selection-screen end of block A.

Types: begin of ty\_knal,

    Kunrn type knal-kunrn,

    Name1 type knal-name1,

    Ort01 type knal-ort01,

    End of ty\_knal.

Data: wa type ty\_knal,

    it type table of ty\_knal.

```

At selection-screen.
Data v type knal-kunnr.
Select single kunnr from knal into v where kunnr in s_kunnr.
If sy-subrc <> 0.
  Message E000(zmessage3) with 'Invalid customer number'.
Endif.

Start-of-selection.
  Select kunnr name1 ort01 from knal into table it where kunnr in
  s_kunnr.

End-of-selection.
  Loop at it into wa.
    Write / wa-kunnr, wa-name1, wa-ort01.
  Endloop.

Top-of-page.
  Write / 'THESE ARE CUSTOMER DETAILS'.

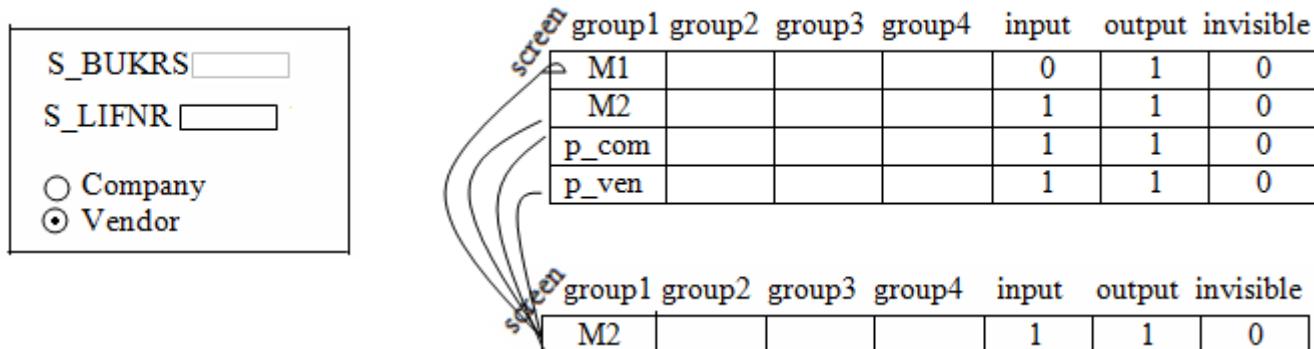
End-of-page.
  Write / 'SATISH TECHNOLOGIES'.

```

When ever we are working with at selection-screen output event then we must provide <MODIF ID> <ID NAME> for each modified field & also provide user-command based on which field you want to modify.

**→ Design the selection-screen as shown in the figure.**

If the user selects the company radio button then company field is enabled and vendor field is disable. If the user select vendor radio button then company field is disable.



Selection-screen begin of block A with frame.  
Parameter P\_BUFRS type lfb1-bukrs MODIF ID M1.  
Parameter P\_LIFNR type lfb1-lifnr MODIF ID M2.  
Parameter p\_com radiobutton group B USER-COMMAND UC.  
Parameter p\_ven radiobutton group B default 'X'.  
Selection-screen end of block A.

At selection-screen output.  
If p\_com = 'X'.  
 Loop at screen.  
 If screen-group1 = 'M1'.  
 Screen-input = 1.

```

        Modify screen.
Elseif screen-group1 = 'M2'.
    Screen-input = 0.
    Modify screen.
Endif.
Endloop.
Elseif p_ven = 'X'.
    Loop at screen.
        If screen-group1 = 'M1'.
            Screen-input = 0.
            Modify screen.
        Elseif screen-group1 = 'M2'.
            Screen-input = 1.
            Modify screen.
        Endif.
    Endloop.
Endif.

```

Screen-active = 1 → Display  
Screen active = 0 → Not display

→ Design the selection-screen as shown in the figure.

If the user select the sales radio button then sales field enabled & rest of the fields are disabled. If the user select S\_BUKRS then company field enabled & rest of the fields are disabled.

```

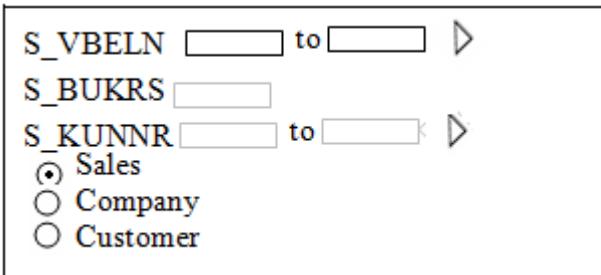
Data V1 type VBAK-VBELN.
Data V2 type KNA1-KUNNR.
Selection-screen begin of block A with frame.
Select-options S_VBELN for V1 MODIF ID M1.
Parameter S_BUKRS type T001-BUKRS MODIF ID M2.
Select-options S_KUNNR for V2 MODIF ID M3.
Parameter p_sal radiobutton group B USER-COMMAND UC.
Parameter p_com radiobutton group B.
Parameter p_cus radiobutton group B default 'X'.
Selection-screen end of block A.
At selection-screen output.

```

```

If p_sal = 'X'.
    Loop at screen.
        If screen-group1 = 'M1'.
            Screen-input = 1.
            Modify screen.
        Elseif screen-group1 = 'M2'.
            Screen-input = 0.
            Modify screen.
        Elseif screen-group1 = 'M3'.
            Screen-input = 0.
            Modify screen.
        Endif.
    Endloop.
Elseif p_com = 'X'.
    Loop at screen.
        If screen-group1 = 'M1'.
            Screen-input = 0.
            Modify screen.
        Elseif screen-group1 = 'M2'.

```



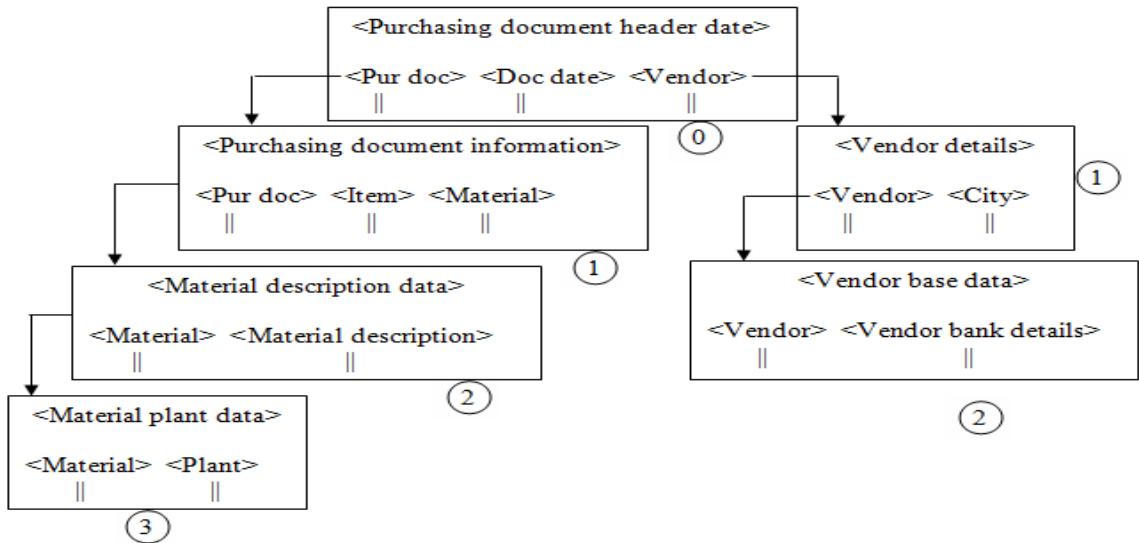
```

        Screen-input = 1.
        Modify screen.
    Elseif screen-group1 = 'M3'.
        Screen-input = 0.
        Modify screen.
    Endif.
Endloop.
Elseif p_cus = 'X'.
    Loop at screen.
        If screen-group1 = 'M1'.
            Screen-input = 0.
            Modify screen.
        Elseif screen-group1 = 'M2'.
            Screen-input = 0.
            Modify screen.
        Elseif screen-group1 = 'M3'.
            Screen-input = 1.
            Modify screen.
        Endif.
    Endloop.
Endif.

```

### **Interactive report: -**

It's nothing but to display the summarized information in the basic list & detailed information in the secondary list.



**Note:** - We can have only one basic list & up to 20 secondary lists (1-20).

### **Events in Interactive report: -**

1. At line selection
2. At user-command.
3. Top-of-page during line-selection.
4. At PF(N)
5. Set PF-status.

### **At line selection: -**

It's an event which is triggered at the time of user clicks on any record of any list.

### **At user-command: -**

It's an event which is triggered at the time of user clicks on any menu item.

### **Top-of-page during line-selection: -**

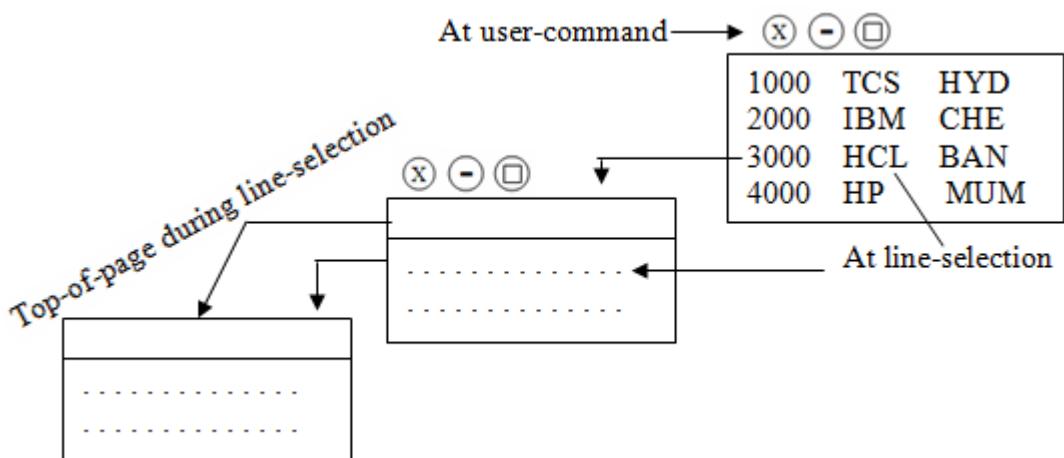
It's an event which is triggered at the top of each secondary list.

### At PF (N): -

It's an event which is triggered at the time of user clicks on F1 to F12 function keys.

### Set PF-status: -

It's an event which is triggered at the time of attaching our own GUI to the program.



**Note:**- Classical events are also triggered for basic list.

### Some of the system variables related to interactive report: -

1. SY-LSIND
2. SY-LISEL
3. SY-LILLI
4. SY-UCOMM
5. SY-LINNO

### SY-LSIND: -

It's the system variable which contains the current list index number.

### SY-LISEL: -

It's the system variable which contains the contents of the selected record.

### SY-LILLI: -

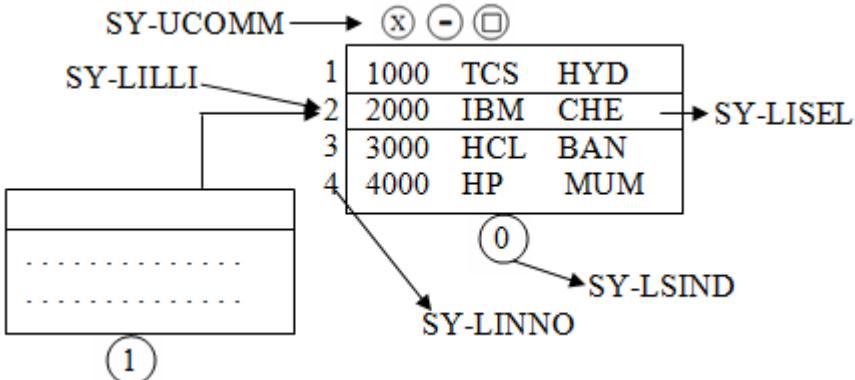
It's the system variable which contains the exact line number of the selected record.

### SY-UCOMM: -

It's the system variable which contains the function code of the selected menu item.

### SY-LINNO: -

It's the system variable which contains the line number of the last record display.



**Note:** - Interactive reports support the user interaction is always through double click. When ever the user double clicks on any record at any list then at line selection event will be triggered & list index is incremented by 1. If you want to retrieve the data for current list then we should know the record which is clicked by the user in the previous list.

The following techniques are used to identify the records which are clicked by the user in the previous list.

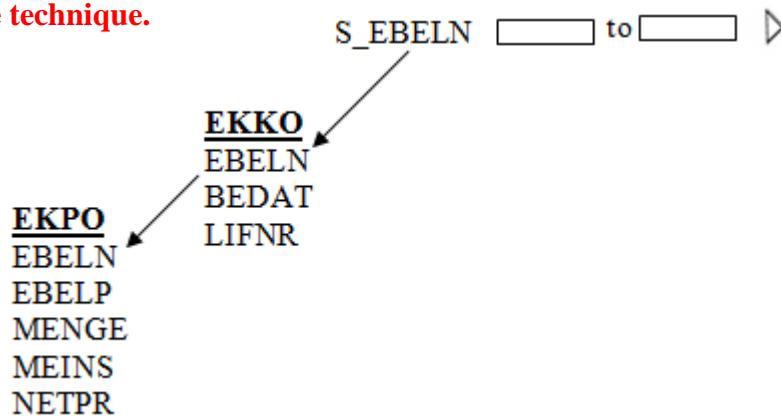
1. Hide technique.
2. SY-LISEL technique.
3. Get cursor technique.

#### **Hide technique: -**

Hide maintain the copy of the previous list with output line numbers & their contents. When ever the user clicks on any record of any list then at line-selection event will be triggered & list index is increased by '1' & that particular record will move from hide area to work area. Based on the work area we fetch the data for current list.

**Note:** - Hide is always maintain after write statement.

**→ Based on the given purchasing document numbers display the purchasing document number, document dates & vendor numbers. In the basic list. If the user clicks on any record then we display the purchasing document item details (EBELN, EBELP, MENGE, MEINS, NETPR) in the first secondary list by using hide technique.**



```
Data V1 type EKKO-EBELN.  
Select-options S_EBELN for V1.
```

```
Types: Begin of ty_ekko,  
       EBELN type EKKO-EBELN,  
       BEDAT type EKKO-BEDAT,  
       LIFNR type EKKO-LIFNR,  
       End of ty_ekko.
```

```
Data: wa_ekko type ty_ekko,  
      it_ekko type table of ty_ekko.
```

```
Types: Begin of ty_ekpo,  
       EBELN type EKPO-EBELN,  
       EBELP type EKPO-EBELP,  
       MENGE type EKPO-MENGE,  
       MEINS type EKPO-MEINS,  
       NETPR type EKPO-NETPR,  
       End of ty_ekpo.
```

```
Data: wa_ekpo type ty_ekpo,  
      it_ekpo type table of ty_ekpo.
```

```
Select EBELN BEDAT LIFNR from EKKO into table it_ekko where EBELN in  
S_EBELN.
```

```

Loop at it_ekko into wa_ekko.
  Write: / wa_ekko-EBELN, wa_ekko-BEDAT, wa_ekko-LIFNR.
  Hide : wa_ekko-EBELN, wa_ekko-BEDAT, wa_ekko-LIFNR.
Endloop.
At line-selection.
If SY-LSIND = '1'.
  Select EBELN EBELP MENGE MEINS NETPR from EKPO into table it_ekpo
  where EBELN = wa_ekko-EBELN.
  Loop at it_ekpo into wa_ekpo.
    Write: / wa_ekpo-EBELN, wa_ekpo-EBELP, wa_ekpo-MENGE, wa_ekpo-
      S_EBELN [ ] to [ ] ▷
    Endloop.
ENDIF.

```

Hide

1	30004	1.5.03	1910
2	30005	2.6.04	2606
3	30006	3.6.07	2787

wa\_ekpo

ebeln	ebelp	menge	meins	netpr
[ ]	[ ]	[ ]	[ ]	[ ]

it\_ekpo

ebeln	ebelp	menge	meins	netpr
30005	02	2	NO	120.00

wa\_ekko

Eblen	bedat	lifnr
30005	2.6.04	2606

it\_ekpo

Eblen	bedat	lifnr
3004	1.5.03	1910
30005	2.6.04	2606
30006	3.6.07	2787

OP

30004 1.5.03 1910  
 30005 2.6.04 2606  
 30006 3.6.07 2787

(0)

30005 01 10 KG 250.00  
 30005 02 2 NO 120.00

(1)

→ Based on the given company codes display the company codes, company name & cities in the basic list. If the user clicks on any record then we display the customers under company details (BUKRS, KUNNR, AKONT) in the first secondary list. If the user clicks on any record then we display the customers details (KUNNR, NAME1, ORT01) in the second secondary list by using hide technique.

KNA1  
 KUNNR  
 NAME1  
 ORT01

(2)

```

Data V1 type T001-BUKRS.
Select-options S_BUKRS for V1.
TYPES: BEGIN OF TY_T001,
  BUKRS TYPE T001-BUKRS,
  BUTXT TYPE T001-BUTXT,
  ORT01 TYPE T001-ORT01,
  END OF TY_T001.
DATA: WA_T001 TYPE TY_T001,
      IT_T001 TYPE TABLE OF TY_T001.
TYPES: BEGIN OF TY_KNB1,

```

एम एन सतीष कुमार रेडि

S\_BUKRS [ ] to [ ] ▷

T001  
 BUKRS  
 BUTXT  
 ORT01

(0)

KNB1  
 BUKRS  
 KUNNR  
 AKONT

(1)

```

BUKRS TYPE KNB1-BUKRS,
KUNNR TYPE KNB1-KUNNR,
AKONT TYPE KNB1-AKONT,
END OF TY_KNB1.

DATA: WA_KNB1 TYPE TY_KNB1,
      IT_KNB1 TYPE TABLE OF TY_KNB1.

TYPES: BEGIN OF TY_KNA1,
       KUNNR TYPE KNA1-KUNNR,
       NAME1 TYPE KNA1-NAME1,
       ORT01 TYPE KNA1-ORT01,
       END OF TY_KNA1.

DATA: WA_KNA1 TYPE TY_KNA1,
      IT_KNA1 TYPE TABLE OF TY_KNA1.

Select BUKRS BUTXT ORT01 from T001 into table it_t001 where BUKRS in
S_BUKRS.

Loop at it_t001 into wa_t001.
  Write:/ wa_t001-BUKRS, wa_t001-BUTXT, wa_t001-ORT01.
  Hide: wa_t001-BUKRS, wa_t001-BUTXT, wa_t001-ORT01.

Endloop.

At line-selection.
  If SY-LSIND = '1'.
    Select BUKRS KUNNR AKONT from KNB1 into table it_knb1 WHERE BUKRS
= wa_t001-BUKRS.
    Loop at it_knb1 into wa_knb1.
      Write: / wa_knb1-BUKRS, wa_knb1-KUNNR, wa_knb1-AKONT.
      Hide: wa_knb1-BUKRS, wa_knb1-KUNNR, wa_knb1-AKONT.

    Endloop.

  Elself SY-LSIND = '2'.
    Select KUNNR NAME1 ORT01 from KNA1 into table it_kna1 where KUNNR
= wa_kna1-KUNNR.
    Loop at it_kna1 into wa_kna1.
      Write: / wa_kna1-KUNNR, wa_kna1-NAME1, wa_kna1-ORT01.

    Endloop.

  Endif.

```

#### Syntax of accessing the part of data from any variable: -

<variable name>+X(Y). → Number of characters  
                           ↓ → Starting position

#### Ex: -

Data A(10) type C value 'SHREE JANANI TECHNOLOGIES'.

WRITE A.

Write A+0(5). → SHREE JANANI TECHNOLOGIES  
                           ↓  
                           0 1 2 3 4 5 6 7 8 9 10  
                           SHREE

**Note:** - when ever we use any one of the classical report event then we must use the start of selection event.

**Note:** - CONVERSION\_EXIT\_ALPHA\_INPUT is the function module which is used to add the leading zeros to the input variable based on the length of the input variable.

#### Ex: -

Data A(5) type C.

A = 415.

Call function CONVERSION-EXIT-ALPHA-INPUT.

Write C.

OP: - 00415.

→ Based on the given company codes display the company codes company names & cities in the basic list. If the user clicks on any record then display the vendors under company details (BUKRS, LIFNR, AKONT) in the first secondary list. If the user clicks on any record then display the vendor details (LIFNR, NAME1, ORT01) in the second secondary list by using SY-LISEL technique & also validate the given company code.

DATA V1 TYPE T001-BUKRS.

SELECT-OPTIONS S\_BUKRS FOR V1.

TYPES: BEGIN OF TY\_T001,

  BUKRS TYPE T001-BUKRS,

  BUTXT TYPE T001-BUTXT,

  ORT01 TYPE T001-ORT01,

  END OF TY\_T001.

DATA WA\_T001 TYPE TY\_T001.

DATA IT\_T001 TYPE TABLE OF TY\_T001.

TYPES: BEGIN OF TY\_LFB1,

  BUKRS TYPE LFB1-BUKRS,

  LIFNR TYPE LFB1-LIFNR,

  AKONT TYPE LFB1-AKONT,

  END OF TY\_LFB1.

DATA WA\_LFB1 TYPE TY\_LFB1.

DATA IT\_LFB1 TYPE TABLE OF TY\_LFB1.

TYPES: BEGIN OF TY\_LFA1,

  LIFNR TYPE LFA1-LIFNR,

  NAME1 TYPE LFA1-NAME1,

  ORT01 TYPE LFA1-ORT01,

  END OF TY\_LFA1.

DATA WA\_LFA1 TYPE TY\_LFA1.

DATA IT\_LFA1 TYPE TABLE OF TY\_LFA1.

DATA V TYPE T001-BUKRS.

AT SELECTION-SCREEN.

  SELECT SINGLE BUKRS FROM T001 INTO V WHERE BUKRS IN S\_BUKRS.

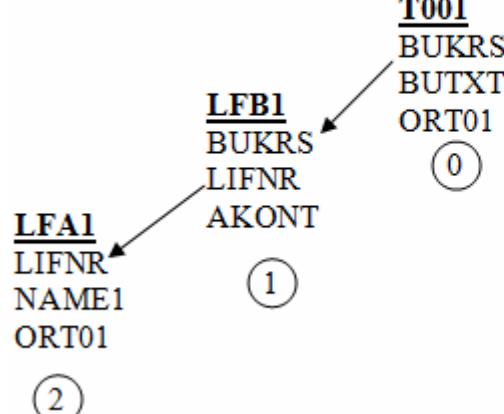
  IF SY-SUBRC <> 0.

    MESSAGE E000(ZMESSAGE1) WITH 'INVALID NUMBER'.

  ENDIF.

START-OF-SELECTION.

  SELECT BUKRS BUTXT ORT01 FROM T001 INTO TABLE IT\_T001 WHERE BUKRS  
  IN S\_BUKRS.



```

LOOP AT IT_T001 INTO WA_T001.
  WRITE:/ WA_T001-BUKRS, WA_T001-BUTXT, WA_T001-ORT01.
ENDLOOP.

AT LINE-SELECTION.
  IF SY-LSIND = '1'.
    SELECT BUKRS LIFNR AKONT FROM LFB1 INTO TABLE IT_LFB1 WHERE BUKRS
    = SY-LISEL+0(4).
    LOOP AT IT_LFB1 INTO WA_LFB1.
      WRITE:/ WA_LFB1-BUKRS, WA_LFB1-LIFNR, WA_LFB1-AKONT.
    ENDLOOP.
  ELSEIF SY-LSIND = '2'.
    CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
      EXPORTING
        INPUT = SY-LISEL+5(10)
      IMPORTING
        OUTPUT = SY-LISEL+5(10).
    SELECT LIFNR NAME1 ORT01 FROM LFA1 INTO TABLE IT_LFA1 WHERE LIFNR
    = SY-LISEL+5(10).
    LOOP AT IT_LFA1 INTO WA_LFA1.
      WRITE:/ WA_LFA1-LIFNR, WA_LFA1-NAME1, WA_LFA1-ORT01.
    ENDLOOP.
  ENDIF.

```

TOP-OF-PAGE DURING LINE-SELECTION.

```

  IF SY-LSIND = '1'.
    WRITE: 'THESE ARE VENDORS UNDER COMPANY:', SY-LISEL+0(4).
  ELSEIF SY-LSIND = '2'.
    WRITE: 'THESE ARE VENDOR DETAILS OF :', SY-LISEL+5(10).
  ENDIF.

```

→ Based on the given customer numbers display the customer numbers, customer names, customer cities in the basic list. If the user clicks on any record then display the sales document header details of customer (VBELN AUDAT KUNNR) in the first secondary list. If the user clicks on any record then we display the sales document item details (VBELN POSNR KWMENG MEINS NETWR) in the second secondary list by using HIDE technique & also validate the customer number.

```

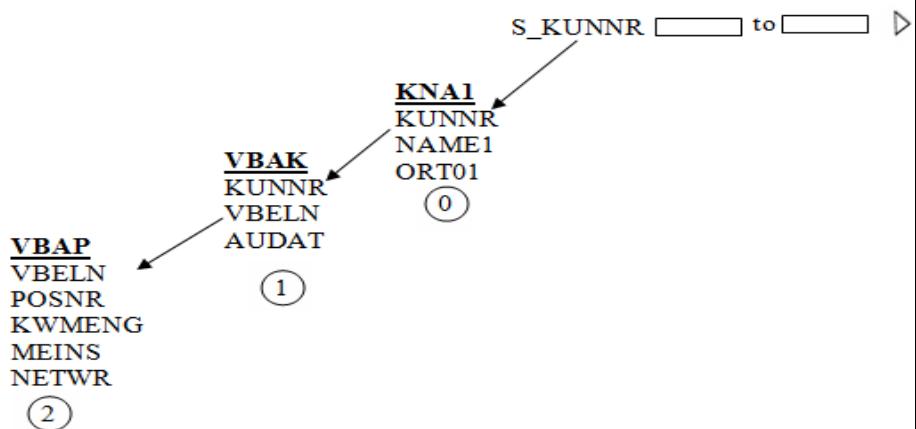
DATA V1 TYPE KNA1-KUNNR.
SELECTION-SCREEN BEGIN OF BLOCK A WITH FRAME.
SELECT-OPTIONS S_KUNNR FOR V1.
SELECTION-SCREEN END OF BLOCK A.

```

```

TYPES: BEGIN OF TY_KNA1,
  KUNNR TYPE KNA1-KUNNR,
  NAME1 TYPE KNA1-NAME1,

```



```

ORT01 TYPE KNA1-ORT01,
END OF TY_KNA1.

DATA WA_KNA1 TYPE TY_KNA1.
DATA IT_KNA1 TYPE TABLE OF TY_KNA1.

TYPES: BEGIN OF TY_VBAK,
        VBELN TYPE VBAK-VBELN,
        AUDAT TYPE VBAK-AUDAT,
        KUNNR TYPE VBAK-KUNNR,
        END OF TY_VBAK.
DATA WA_VBAK TYPE TY_VBAK.
DATA IT_VBAK TYPE TABLE OF TY_VBAK.

TYPES: BEGIN OF TY_VBAP,
        VBELN TYPE VBAP-VBELN,
        POSNR TYPE VBAP-POSNR,
        KWMENG TYPE VBAP-KWMENG,
        MEINS TYPE VBAP-MEINS,
        NETWR TYPE VBAP-NETWR,
        END OF TY_VBAP.

DATA WA_VBAP TYPE TY_VBAP.
DATA IT_VBAP TYPE TABLE OF TY_VBAP.

DATA V TYPE KNA1-KUNNR.

AT SELECTION-SCREEN.
  SELECT SINGLE KUNNR FROM KNA1 INTO V WHERE KUNNR IN S_KUNNR.
  IF SY-SUBRC <> 0.
    MESSAGE E000(ZMESSAGE1) WITH 'INVALID CUSTOMER NUMBER'.
  ENDIF.

START-OF-SELECTION.
  SELECT KUNNR NAME1 ORT01 FROM KNA1 INTO TABLE IT_KNA1 WHERE KUNNR
IN S_KUNNR.

END-OF-SELECTION.
  LOOP AT IT_KNA1 INTO WA_KNA1.
    WRITE:/ WA_KNA1-KUNNR, WA_KNA1-NAME1, WA_KNA1-ORT01.
    HIDE: WA_KNA1-KUNNR, WA_KNA1-NAME1, WA_KNA1-ORT01.
  ENDLOOP.

AT LINE-SELECTION.
  IF SY-LSIND = '1'.
    SELECT VBELN AUDAT KUNNR FROM VBAK INTO TABLE IT_VBAK WHERE KUNNR
= WA_KNA1-KUNNR.
    LOOP AT IT_VBAK INTO WA_VBAK.
      WRITE:/ WA_VBAK-VBELN, WA_VBAK-AUDAT, WA_VBAK-KUNNR.
      HIDE: WA_VBAK-VBELN, WA_VBAK-AUDAT, WA_VBAK-KUNNR.
    ENDLOOP.

```

```

ELSEIF SY-LSIND = '2'.
  SELECT VBELN POSNR KWMENG MEINS NETWR FROM VBAP INTO TABLE
IT_VBAP WHERE VBELN = WA_VBAK-VBELN.
  LOOP AT IT_VBAP INTO WA_VBAP.
    WRITE: / WA_VBAP-VBELN, WA_VBAP-POSNR, WA_VBAP-KWMENG, WA_VBAP-
MEINS, WA_VBAP-NETWR.
  ENDLOOP.
ENDIF.

```

**Note:** - Hide & SY-LISEL techniques generates the next list based on the line-selection. If you want to generate the next list based on the field selection then we go for get-cursor technique.

**Get-cursor technique:** - Get-cursor technique writes the field name as well as field value which is clicked by the user.

**Syntax:** -    
Get-cursor field1 <variable1> value <variable2>

→ Based on the given sales document numbers display the sales document numbers, document dates & customer numbers in the basic list. If the user clicks on the any sales document number only then we display the sales document item details [VBELN POSNR KWMENG MEINS NETWR] in the first secondary list if the user clicks on any customer number only then we display the customer details [KUNNR NAME1 ORT01] in the first secondary list by using get-cursor technique.

```

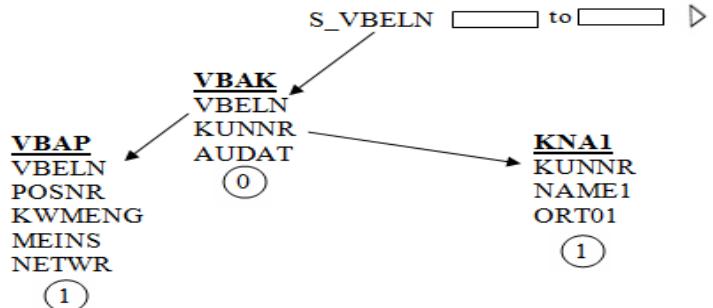
data v3(10) type c.
data v1 type vbak-vbeln.
select-options s_vbeln for v1.
DATA V2(15) TYPE C.
types: begin of ty_vbak,
       vbeln type vbak-vbeln,
       audat type vbak-audat,
       kunnr type vbak-kunnr,
       end of ty_vbak.
data: wa_vbak type ty_vbak,
      it_vbak type table of ty_vbak.

types: begin of ty_vbap,
       vbeln type vbap-vbeln,
       posnr type vbap-posnr,
       kwmeng type vbap-kwmeng,
       meins type vbap-meins,
       netwr type vbap-netwr,
       end of ty_vbap.

data: wa_vbap type ty_vbap,
      it_vbap type table of ty_vbap.

types: begin of ty_knal,
       kunnr type knal-kunnr,

```



```

name1 type kna1-name1,
ort01 type kna1-ort01,
end of ty_kna1.
data: wa_kna1 type ty_kna1,
      it_kna1 type table of ty_kna1.

select vbeln audat kunnr from vbak into table it_vbak where vbeln in
s_vbeln.
loop at it_vbak into wa_vbak.
  write:/ wa_vbak-vbeln, wa_vbak-audat, wa_vbak-kunnr.
endloop.

at line-selection.
if sy-lsind = '1'.
  get cursor field v2 value v3.
  IF V2 = 'WA_VBAK-VBELN'.
    CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
      EXPORTING
        INPUT = V3
      IMPORTING
        OUTPUT = V3.

    SELECT VBELN POSNR KWMENG MEINS NETWR FROM VBAP INTO TABLE
IT_VBAP WHERE VBELN = V3.
    LOOP AT IT_VBAP INTO WA_VBAP.
      WRITE:/ WA_VBAP-VBELN, WA_VBAP-POSNR, WA_VBAP-MEINS, WA_VBAP-
KWMENG, WA_VBAP-NETWR.
    ENDLOOP.
  ELSEIF V2 = 'WA_VBAK-KUNNR'.
    CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
      EXPORTING
        INPUT = V3
      IMPORTING
        OUTPUT = V3.
    SELECT KUNNR NAME1 ORT01 FROM KNA1 INTO TABLE IT_KNA1 WHERE
KUNNR = V3.
    LOOP AT IT_KNA1 INTO WA_KNA1.
      WRITE:/ WA_KNA1-KUNNR, WA_KNA1-NAME1, WA_KNA1-ORT01.
    ENDLOOP.
  ENDIF.
ENDIF.

```

**Note:** - If you click on most of the records, if the output isn't coming then we open the table in SE11 & pass the values. If the data is available then you must check the any conversion routine is available or not.

#### **Steps to identify the conversion routine:-**

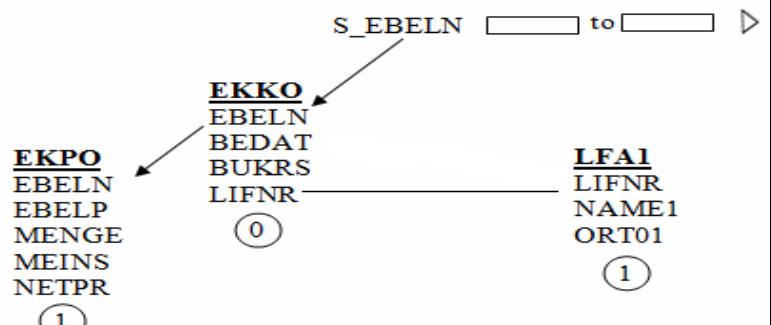
Double click on data element of the field. Double click on domain if any conversion routines are available, double click on it & identify the input routine & apply the input routine before select query.

→ Based on the given purchasing document numbers display the purchasing document numbers, document dates, company codes & vendor numbers if the user clicks on any purchasing document number only then we display the purchasing document item details. [EBELN, EHELP, MENGE, MEINS, NETPR] in the first secondary list if the user click on any vendor number then display the vendor details [LIFNR, NAME1, ORT01] in the first secondary list & also validate the given purchasing document number.

```

DATA V1 TYPE EKKO-EBELN.
DATA V2(15) TYPE C.
DATA V3(10) TYPE C.
SELECT-OPTIONS S_EBELN FOR V1.
TYPES: BEGIN OF TY_EKKO,
       EBELN TYPE EKKO-EBELN,
       BEDAT TYPE EKKO-BEDAT,
       BUKRS TYPE EKKO-BUKRS,
       LIFNR TYPE EKKO-LIFNR,
       END OF TY_EKKO.
DATA: WA_EKKO TYPE TY_EKKO,
      IT_EKKO TYPE TABLE OF TY_EKKO.
TYPES: BEGIN OF TY_EKPO,
       EBELN TYPE EKPO-EBELN,
       EHELP TYPE EKPO-EHELP,
       MENGE TYPE EKPO-MENGE,
       MEINS TYPE EKPO-MEINS,
       NETPR TYPE EKPO-NETPR,
       END OF TY_EKPO.
DATA: WA_EKPO TYPE TY_EKPO,
      IT_EKPO TYPE TABLE OF TY_EKPO.
TYPES: BEGIN OF TY_LFA1,
       LIFNR TYPE LFA1-LIFNR,
       NAME1 TYPE LFA1-NAME1,
       ORT01 TYPE LFA1-ORT01,
       END OF TY_LFA1.
DATA: WA_LFA1 TYPE TY_LFA1,
      IT_LFA1 TYPE TABLE OF TY_LFA1.
DATA X TYPE EKKO-EBELN.
AT SELECTION-SCREEN.
  SELECT SINGLE EBELN FROM EKKO INTO X WHERE EBELN IN S_EBELN.
  IF SY-SUBRC <> 0.
    MESSAGE E000(ZMESSAGE1) WITH 'INVALID NUMBER'.
  ENDIF.
START-OF-SELECTION.
  select EBELN BEDAT BUKRS LIFNR from EKKO into table IT_EKKO
  where EBELN IN S_EBELN.
  LOOP AT IT_EKKO INTO WA_EKKO.
    WRITE: WA_EKKO-EBELN, WA_EKKO-BEDAT, WA_EKKO-BUKRS, WA_EKKO-
          LIFNR.

```



```

ENDLOOP.

AT LINE-SELECTION.
IF SY-LSIND = '1'.
  GET CURSOR FIELD V2 VALUE V3.
  IF V2 = 'WA_EKKO-EBELN'.
    CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
      EXPORTING
        INPUT = V3
      IMPORTING
        OUTPUT = V3.

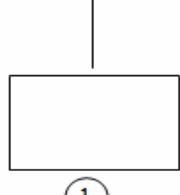
    SELECT EBELN EHELP MENGE MEINS NETPR FROM EKPO INTO TABLE
    IT_EKPO WHERE EBELN = V3.
    LOOP AT IT_EKPO INTO WA_EKPO.
      WRITE:/ WA_EKPO-EBELN, WA_EKPO-EHELP, WA_EKPO-MENGE,
      WA_EKPO-MEINS, WA_EKPO-NETPR.
    ENDLOOP.

    ELSEIF V2 = 'WA_EKKO-LIFNR'.
      CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
        EXPORTING
          INPUT = V3
        IMPORTING
          OUTPUT = V3.

      SELECT LIFNR NAME1 ORT01 FROM LFA1 INTO TABLE IT_LFA1 WHERE
      LIFNR = V3.
      LOOP AT IT_LFA1 INTO WA_LFA1.
        WRITE:/ WA_LFA1-LIFNR, WA_LFA1-ORT01, WA_LFA1-NAME1.
      ENDLOOP.
    ENDIF.
  ENDIF.

```

	WA_VBAK-VBELN	WA_VBAK-AUDAT	WA_VBAK-KUNNR
4969	01.05.2000	1175	
4970	05.09.2004	1095	
4971	15.02.2006	7150	
4972	29.03.2009	10710	



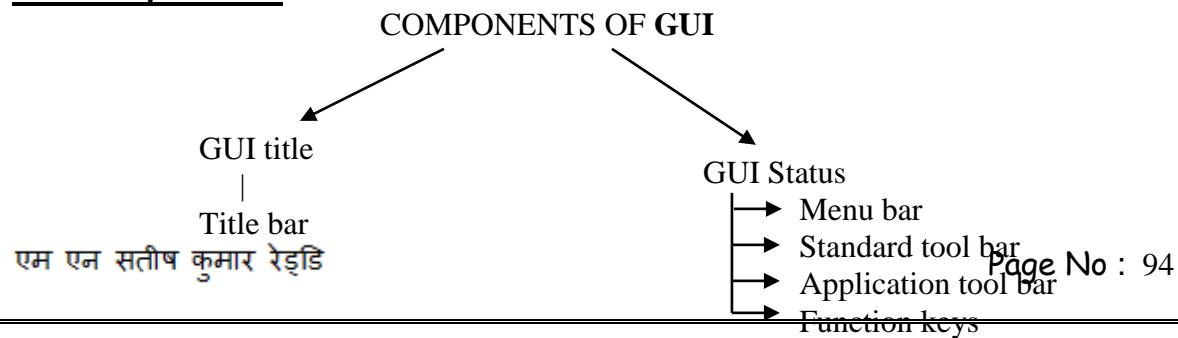
V2 = WA\_VBAK-VBELN

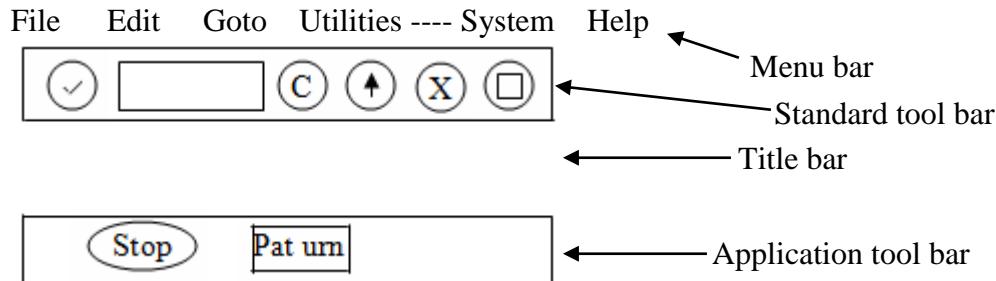
V3 = 4971

#### Working with menu painter: -

Menu painter is a tool to design the user interface to the program. The transaction code for menu painter is 'SE41'.

#### GUI Components: -





**Note:** - In the menu bar system & help are default menu items & we can create up to 6 menu items [total 8 items].

**Note:** - We can design up to 35 buttons in the application tool bar.

**→ Based on the given purchasing document numbers display the purchasing document numbers document dates & vendor numbers as shown in the below & also design one download button in the application tool bar. If the user clicks on download button then we download the display records into presentation server.**

xyz	◀
Download	
30004 01.05.2002	3910
30005 05.09.2005	5550
30006 19.02.2010	1910

### **Steps to create the GUI to the program: -**

Execute 'SE41'. Provide the program name. Provide the status (any name). Click on create. Provide the short description. Enter.

Expand the function key and enabling the back button by providing 'BACK'.

Expand the application tool bar. Provide the function code name. Double click on it. Enter. Provide function text [Download]. Enter. Select the shortcut key. Enter. Repeat the same steps for all other buttons. Save, check, activate.

### **Syntax of attaching our own GUI to the program: -**

SET PF-STATUS '<status name>'.

**Ex:** - set pf-status 'STAT'.

**Note:** - Download is the function module which is used to browse the file as well as download the data from internal table to file. The input for the above function module is

1. File type → 'DAT'.
2. Data internal table.

**Data V1 type EKKO-EBELN.**

**Select-options S\_EBELN for V1.**

**Types: Begin of TY\_EKKO,  
EBELN type EKKO-EBELN,**

```

BEDAT type EKKO-BEDAT,
LIFNR type EKKO-LIFNR,
End of TY_EKKO.

Data WA_EKKO type TY_EKKO.
Data IT_EKKO type table of TY_EKKO.
Select EBELN BEDAT LIFNR from EKKO INTO TABLE IT_EKKO where EBELN in
S_EBELN.
Loop at IT_EKKO into WA_EKKO.
  Write: / WA_EKKO-EBELN, WA_EKKO-BEDAT, WA_EKKO-LIFNR.
Endloop.
Set PF-STATUS 'STAT'.
At user-command.
If SY-UCOMM = 'DOWN'.
  CALL FUNCTION 'DOWNLOAD'
    EXPORTING
      FILETYPE = 'DAT'
    TABLES
      DATA_TAB = IT_EKKO.
Endif.

```

#### Syntax of reading the displayed records:-

Read line <line number> field value <Displayed field name1> into <variable1>  
                                   <displayed field name2> into <variable2> ----.

#### Ex: -

Read line 3 field value WA\_EKKO-EBELN into V1  
                                   WA\_EKKO-BEDAT into V2  
                                   WA\_EKKO-LIFNR into V3.

→ Based on the given sales document numbers display the sales document number, document dates & customer numbers as shown in below. If the user click on download button then we download the selected records into presentation server. If the user click on display button then we display the sales order details through VA03 transaction. If the user click on 'Selt all' then select the all check boxes.

#### Some of the standard transaction codes: -

1. XK03 → Display vendor
2. XD03 → Display customer
3. MM03 → Display material
4. ME53N → Display purchase order
5. VA03 → Display sales order
6. FB03 → Display accounting document.

*Note:* - 1 → Create 2 → Change 3 → Display

#### Syntax of call transaction: -

Call transaction '<Tcode>'.

#### Ex: -

Call transaction 'VA03'.

Before call the transaction we must set the value set parameter ID '<ID name>' field '<value>'.

#### Steps to identify the parameter ID:-

Execute the transaction code place the cursor on input field. Click on F1. Click on technical information identify the parameter id.

Set parameter ID '<ID Name>' field '<value>'.

/ xyz	DIS	SA
Download	Display	Selt all
<input type="checkbox"/>	4969	05.02.2009
<input checked="" type="checkbox"/>	4970	15.02.2010
<input type="checkbox"/>	4971	31.03.2013
<input checked="" type="checkbox"/>	4972	02.09.2013
		1175
		1015
		1510
		1170

Ex: -

Parameter P\_VBELN type VBAK-VBELN.  
Set parameter ID ‘AUN’ field P\_VBELN.  
CALL TRANSACTION ‘VA03’.

*Note:* - If you want to get the current document value which is opened.

Syntax: -

Get parameter ID ‘<ID Name>’ field <Variable>.

Ex: -

Data v1 type vbak-vbeln.  
Get parameter ID ‘AUN’ field v1.  
Write v1.

*Note:* - Set & get are called SAP memory & import, export, changing --- are called ABAP memory.

*Note:* - SY-INDEX is the system variable which contains the current loop pass.

*Note:* - SY-TABIX is the system variable which contains the exact line number of the record which is moving from internal table to work area.

data: a, b.

DATA: V1 TYPE VBAK-VBELN, V2 TYPE VBAK-AUDAT.

DATA V TYPE SYLINNO.

tables vbak.

select-options s\_vbeln for vbak-vbeln.

types: begin of ty\_vbak,

    vbeln type vbak-vbeln,

    audat type vbak-audat,

    kunnr type vbak-kunnr,

    a type c,

    end of ty\_vbak.

data: wa\_vbak type ty\_vbak,

    it\_vbak type table of ty\_vbak.

DATA: WA LIKE WA\_VBAK,

    IT LIKE TABLE OF WA.

select vbeln audat kunnr from vbak into table it\_vbak where vbeln in s\_vbeln.

loop at it\_vbak into wa\_vbak.

    write:/ a as checkbox, wa\_vbak-vbeln, wa\_vbak-audat input, wa\_vbak-kunnr.

endloop.

V = SY-LINNO.

set pf-status 'STAT'.

AT USER-COMMAND.

IF SY-UCOMM = 'DOWN' .

    DO V TIMES .

        READ LINE SY-INDEX FIELD VALUE A INTO B

            WA\_VBAK-VBELN INTO WA-VBELN

            WA\_VBAK-AUDAT INTO WA-AUDAT

            WA\_VBAK-KUNNR INTO WA-KUNNR.

```

IF B = 'X'.
  APPEND WA TO IT.
  CLEAR WA.
ENDIF.
ENDDO.
CALL FUNCTION 'DOWNLOAD'
  EXPORTING
    FILETYPE = 'DAT'
  TABLES
    DATA_TAB = IT.
  REFRESH IT.
ELSEIF SY-UCOMM = 'SA'.
  A = 'X'.
  LOOP AT IT_VBAK INTO WA_VBAK.
    WRITE:/ A AS CHECKBOX, WA_VBAK-VBELN, WA_VBAK-AUDAT, WA_VBAK-
KUNNR.
  ENDLOOP.
ELSEIF SY-UCOMM = 'UPD'.
  DO V TIMES.
    READ LINE SY-INDEX FIELD VALUE A INTO B
      WA_VBAK-VBELN INTO V1
      WA_VBAK-AUDAT INTO V2.
    IF B = 'X'.
      UPDATE VBAK SET AUDAT = V2 WHERE VBELN = V1.
    ENDIF.
  ENDDO.
  IF SY-SUBRC = 0.
    MESSAGE S000(ZMESSAGE1) WITH 'UPDATED SUCCESSFULLY'.
  ELSE.
    MESSAGE E000(ZMESSAGE1) WITH 'NOT UPDATED'.
  ENDIF.
ELSEIF SY-UCOMM = 'DIS'.
  DO V TIMES.
    READ LINE SY-INDEX FIELD VALUE A INTO B
      WA_VBAK-VBELN INTO V1.
    IF B = 'X'.
      SET PARAMETER ID 'AUN' FIELD V1.
      CALL TRANSACTION 'VA03' AND SKIP FIRST SCREEN.
    ENDIF.
  ENDDO.
ENDIF.

```

## **ALV (ABAP LIST VIEWERS): -**

ALV is used to display the output with predefined functionalities. Like

1. Sort the list in ascending order
2. Sort the list in descending order
3. Tables
4. Filtering
5. Down the list
6. Send an attachment
7. Word processing
8. Excel sheet
9. Change the layout
10. Graphics
11. Print previews

ALV is introduced from 4.6C version onwards. ALV is used to display the data from internal table only.

### **Steps to work with ALV: -**

1. Declare the final data internal table (which data we want to display) and implement the retrieving logic.
2. Prepare the field catalog (about the display field) i.e.
  1. Field name
  2. Column position
  3. Column Heading
  4. Colour
  5. Hotspot
3. Call the 'REUSE\_ALV\_GRID\_DISPLAY' function module.  
(OR)  
CALL THE 'REUSE\_ALV\_LIST\_DISPLAY' function module.

**Note:** - REUSE\_ALV\_GRID\_DISPLAY is the function module which is used to display the output in a grid format.

REUSE\_ALV\_LIST\_DISPLAY is the function module which is used to display the output in a list format.

The input for the above two functions modules are two internal tables.

1. Data internal table
2. Field Catalog internal table

### **→ Display the all sales documents details by using ALV.**

```
Data IT_VBAP like table of VBAP.  
Select * from VBAP into table IT_VBAP.  
CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'  
  EXPORTING  
    I_STRUCTURE_NAME = 'VBAP'  
  TABLES  
    T_OUTTAB      = IT_VBAP.
```

When ever we are working with all the fields from any one of the database table or structure, then we no need to prepare field catalog internal table. We simply pass i\_structure\_name as data base table name structure.

**Note:** - Here the function module picks the column headings from the data element of each field and also display the fields in the similar order of the fields in the table.

## FILLING THE FIELD CATALOG

If we are working with all the fields from any data base table / structure then we no need to prepare the field catalog. We simply pass the database name/ structure name as `i_structure_name`.

Manually filling the field catalog.

`REUSE_ALV_FIELDCATALOG_MERGE` function.

### Some of the fields in field catalog internal table: -

1. Field name → Name of the field
2. Col\_Pos → Column position
3. Seltext\_S }  
Seltext\_M } Column heading  
Seltext\_L }
4. Emphasize → Colour
5. Output length → Length of the output field
6. Hotspot → Hand / Symbol
7. Edit → Change mode
8. No-zero → Remove the leading zeros
9. No-sign → Remove the leading sign
10. No-out → Hide the display field
11. Do-sum → Calculate the total
12. Checkbox → Checkbox

Activate = 'X'  
Inactivate = ''

**Note:** - In `slis` we have one type i.e. `SLIS_T_FIELDCAT_ALV` which contains all the fields related to field catalog internal table. So we simply declare our internal table by referring this.

`Slis` is the type group which contains all the types related to ALV.

**Note:** - When ever we are referring any type under any type group then we must include the type group name by using “type-pools” keyword.

### Steps to create type-pools: -

Execute **SE11** & select the radio button type group. Provide the type group name (ZTG) & click on create. Provide short description (Type group) & press enter. Select local object.

**Note:** - All the names under the group must be starts with type group name\_ (underscore).

Type-pools ZTG.

```
Types: begin of ZTG_T001,
      Bukrs type t001-bukrs,
      Butxt type t001-butxt,
      Ort01 type t001-ort01,
      End of ZTG_T001.
```

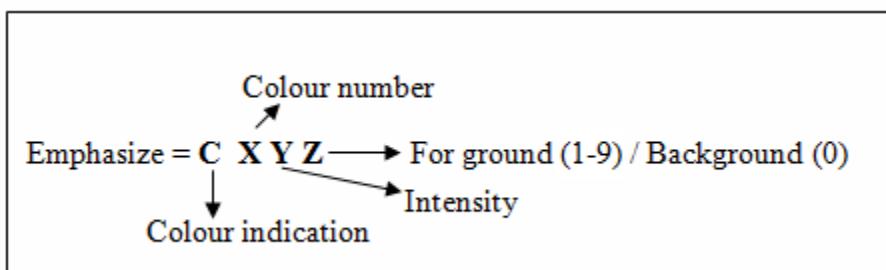
```
Types ZTG_T_T001 type table of ZTG_T001. ◀ (IT) (WA)
```

→ Display the company codes, company names and cities.

Type-pools ZTG.

```
Data: it_t001 type ZTG_T_T001,  
      Wa_t001 like line of it_t001.  
Select bukrs butxt ort01 from t001 into table it_t001.  
Loop at it_t001 into wa_t001.  
  Write:/ wa_t001-bukrs, wa_t001-butxt, wa_t001-ort01.  
Endloop.
```

→ Based on the given purchasing document number display the purchasing document numbers, document dates and vendor numbers by using ALV and also display the purchasing document number with yellow co lour, document date with edit and the vendor is hotspot.



Type-pools slis.

TABLES EKKO.

```
SELECT-OPTIONS S_EBELN FOR EKKO-EBELN.
```

\* Declare the data internal table.

```
Types: Begin of ty_ekko,  
       Ebeln type ekko-ebeln,  
       Bedat type ekko-bedat,  
       Lifnr type ekko-lifnr,  
       End of ty_ekko.
```

Data it\_ekko type table of ty\_ekko.

\* Filling the data internal table.

```
Select ebeln Bedat lifnr from ekko into table it_ekko where ebeln  
in s_ebeln.
```

\* Declaring the field catalog.

```
Data: it_fcat type SLIS_T_FIELDCAT_ALV,  
      Wa_fcat like line of it_fcat.
```

\* Filling the field catalog

```
WA_FCAT-FIELDNAME = 'EBELN'.
```

```
WA_FCAT-COL_POS = '1'.
```

```
WA_FCAT-SELTEXT_M = 'PUR.DOC'.
```

```
WA_FCAT-EMPHASIZE = 'C310'.
```

```
APPEND WA_FCAT TO IT_FCAT.
```

```
CLEAR WA_FCAT.
```

```
WA_FCAT-FIELDNAME = 'BEDAT'.
```

```
WA_FCAT-COL_POS = '2'.
```

```
WA_FCAT-SELTEXT_M = 'DOC.DT'.
```

```
WA_FCAT-EDIT = 'X'.
```

```
APPEND WA_FCAT TO IT_FCAT.
```

```
CLEAR WA_FCAT.
```

```
WA_FCAT-FIELDNAME = 'LIFNR'.
```

```

WA_FCAT-COL_POS = '3'.
WA_FCAT-SELTEXT_M = 'VENDOR'.
WA_FCAT-HOTSPOT = 'X'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.
* Display the output
CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
  EXPORTING
    It_fieldcat = it_fcat
  TABLES
    T_outtab      = it_ekko.

```

Field name	Col_pos	Seltext_m	Hotspot	Emphasize	Edit	Key	-----
EBELN	1	PUR.DOC		C310			
BEDAT	2	DOC DATE			X		
LIFNR	3	VENDOR	X				

Field name	Col_pos	Seltext_m	Hotspot	Emphasize	Edit	Key	-----

Pur.doc Ebeln Bedat Lifnr	Doc.date 01.05.2002 15.02.2007 19.05.2010	Vendor 3910 1510 5550
------------------------------------	--	--------------------------------

'REUSE\_ALV\_FIELDCATALOG MERGE' is the function module which is used to fill the field catalog internal table.

The input for the above function module is

- Data work area.
- Program name in where data work area is declared.
- Field catalog internal table.
- Program name, in where field catalog internal table is declared.

**Note:** - 'SY-CPROG' is the system variable which contains current program name.

**Note:** - When ever we are working with merge function module then we must consider the following things.

- We never declare the data internal table by using 'TYPES' keyword.
- We never refer the data internal table fields by using type. Instead of type we use like.
- We must maintain the code up to 72 columns.

**→ Based on the given purchasing document number display the purchasing document numbers, document dates and vendor numbers by using ALV and also display the purchasing document number with yellow color, document date with edit and the vendor is hotspot by using hotspot module.**

After the merge function module the field catalog as shown in the below.

it\_fcat

Field name	col_doc	seltext_m	Emphasize	Hotspot	Edit	key
EBELN	1	PUR.DOC			X	
BEDAT	2	DOC DATE				
LIFNR	3	VENDOR				-

)2

After merge function module we modify the field catalog based on the client requirement.

**Note:** - We can't modify the key fields directly first we remove the key then we modify it.

```
TYPE-POOLS SLIS.  
TABLES EKKO.  
SELECT-OPTIONS S_EBELN FOR EKKO-EBELN.  
DATA: BEGIN OF WA_EKKO,  
      EBELN LIKE EKKO-EBELN,  
      BEDAT LIKE EKKO-BEDAT,  
      LIFNR LIKE EKKO-LIFNR,  
      END OF WA_EKKO.  
DATA IT_EKKO LIKE TABLE OF WA_EKKO.  
SELECT EBELN BEDAT LIFNR FROM EKKO INTO TABLE IT_EKKO WHERE EBELN  
IN S_EBELN.  
DATA: IT_FCAT TYPE SLIS_T_FIELDCAT_ALV,  
      WA_FCAT LIKE LINE OF IT_FCAT.  
  
CALL FUNCTION 'REUSE_ALV_FIELDCATALOG_MERGE'  
EXPORTING  
  I_PROGRAM_NAME          = SY-CPROG  
  I_INTERNAL_TABNAME      = 'WA_EKKO'  
  I_INCLNAME              = SY-CPROG  
CHANGING  
  CT_FIELDCAT             = IT_FCAT.  
* Modify the field catalog based on requirement  
WA_FCAT-KEY = 'X'.  
WA_FCAT-EMPHASIZE = 'C510'.  
MODIFY IT_FCAT FROM WA_FCAT TRANSPORTING KEY EMPHASIZE WHERE  
FIELDNAME = 'EBELN'.  
WA_FCAT-HOTSPOT = 'X'.  
MODIFY IT_FCAT FROM WA_FCAT TRANSPORTING HOTSPOT WHERE  
FIELDNAME = 'LIFNR'.  
CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'  
EXPORTING  
  IT_FIELDCAT = IT_FCAT  
  TABLES  
    T_OUTTAB   = IT_EKKO.
```

→ Based on the given purchasing document numbers and dates to display the purchasing document numbers, document dates, vendor numbers, item numbers and price by using ALV.

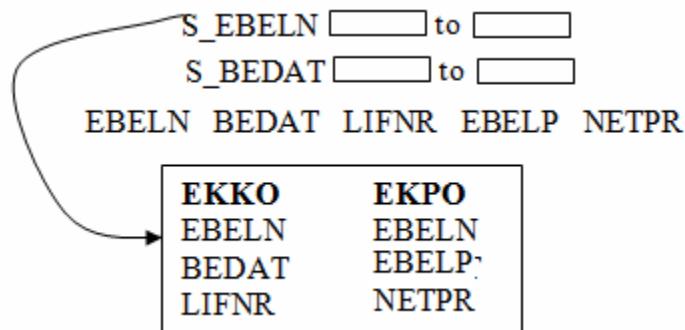
TYPE-POOLS SLIS.

Tables EKKO.

Select-options s\_ebeln for ekko-ebeln.

\* Declare the data IT.

Types: Begin of ty\_final,  
 Ebeln type ekko-ebeln,



```

Bedat type ekko-bedat,
Lifnr type ekko-lifnr,
Ebelp type ekpo-ebelp,
Netpr type ekpo-netpr,
End of ty_final.

Data it_final type table of ty_final.
Select ekko~ebeln ekko~bedat ekko~lifnr ekpo~ebelp ekpo~netpr into
table it_final from ekko inner join ekpo on ekko~ebeln = ekpo~ebeln
where ekko~ebeln in s_ebeln.

* Declare the field catalog.
Data: it_fcat type slis_t_fieldcat_alv,
      Wa_fcat like line of it_fcat.

* Filling the field catalog.
Wa_fcat-fieldname = 'EBELN'.
Wa_fcat-col_pos = '1'.
Wa_fcat-selttext_m = 'pur.doc'.
Append wa_fcat to it_fcat.
Clear wa_fcat.

Wa_fcat-fieldname = 'BEDAT'.
Wa_fcat-col_pos = '2'.
Wa_fcat-selttext_m = 'DOC DATE'.
Append wa_fcat to it_fcat.
Clear wa_fcat.

Wa_fcat-fieldname = 'LIFNR'.
Wa_fcat-col_pos = '3'.
Wa_fcat-selttext_m = 'VENDOR'.
Append wa_fcat to it_fcat.
Clear wa_fcat.

Wa_fcat-fieldname = 'EBELP'.
Wa_fcat-col_pos = '4'.
Wa_fcat-selttext_m = 'ITEM'.
Append wa_fcat to it_fcat.
Clear wa_fcat.

Wa_fcat-fieldname = 'NETPR'.
Wa_fcat-col_pos = '5'.
Wa_fcat-selttext_m = 'NETVALUE'.
Append wa_fcat to it_fcat.
Clear wa_fcat.

* Display output
CALL FUNCTION 'REUSE_ALV_LIST_DISPLAY'
  EXPORTING
    IT_FIELDCAT = IT_FCAT
  TABLES
    T_OUTTAB      = IT_FINAL.

```

## EVENTS IN ALV

In ALV, events are handle through ‘sub-routines’ (form, endform)

1. Top\_of\_page
2. Top\_of\_list
3. End\_of\_page
4. End\_of\_list
5. User\_command
6. Pf\_status\_set

### **Top\_of\_page:** -

It's an event which is triggered at the top of each page.

### **Top\_of\_list:** -

It's an event which is triggered at the top of displayed output list.

### **End\_of\_page:** -

It's an event which is triggered at the end of each page.

### **End\_of\_list:** -

It's an event which is triggered at the end of displayed output list.

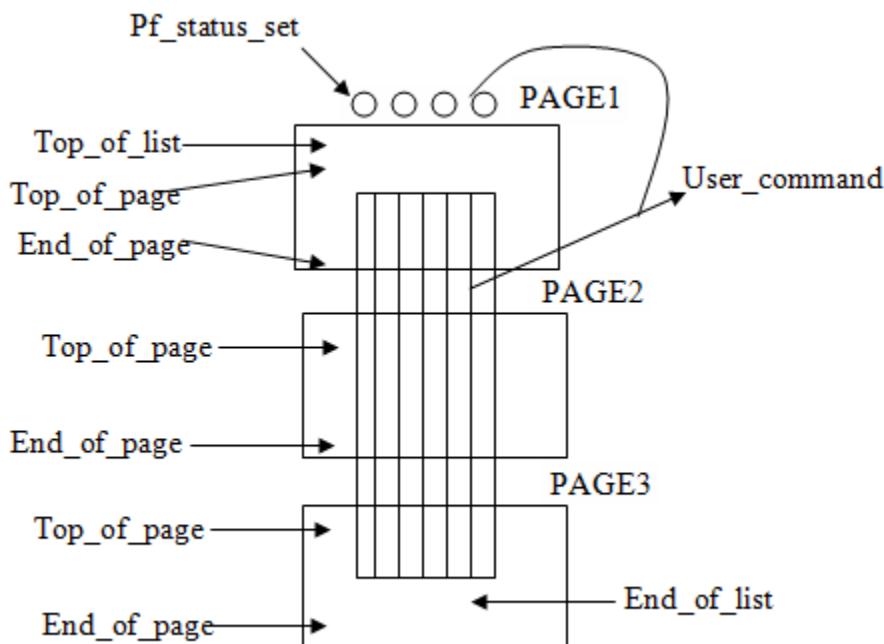
### **User\_command:** -

It's an event which is triggered at the time of user clicks on any record of any list as well as any menu item.

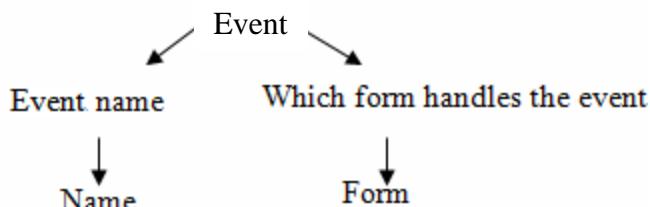
This event acts like both at line-selection & at user-command.

### **Pf\_status\_set:** -

It's an event which is triggered at the time of attaching our GUI to the program.

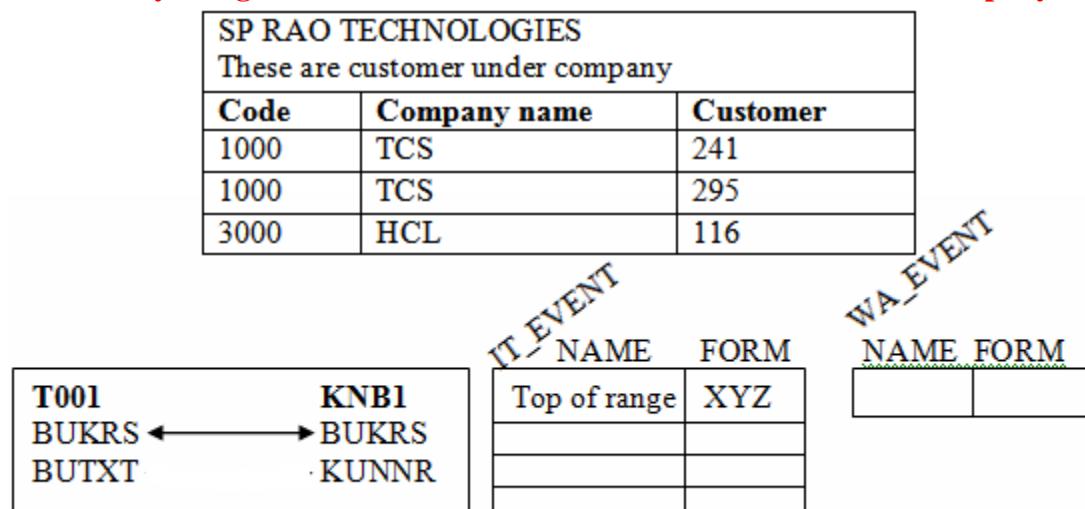


When ever we are working with events then we must declare an event internal table which contains 2 fields.



**Note:** - In slis we have one type i.e. `slis_t_event` which contains the above fields. So we simply declare the internal table by referring `slis_t_event`.

→ Based on the given company codes display the company codes, company names & customer numbers by using ALV as shown in the below and also validate the company code.



**Note:** - We can also print the data in start-of-selection event when ever we are fetching the data from data base. Then we must use end-of-selection to print the data.

**Note:** - When ever we are working with events then we must provide “I\_callback\_program” as current program name in the grid or list display.

TYPE-POOLS SLIS.

TABLES T001.

DATA V TYPE T001-BUKRS.

SELECT-OPTIONS S\_BUKRS FOR T001-BUKRS.

\* Declare the final data IT.

TYPES: BEGIN OF TY\_FINAL,  
        BUKRS TYPE T001-BUKRS,  
        BUTXT TYPE T001-BUTXT,  
        KUNNR TYPE KNB1-KUNNR,  
        END OF TY\_FINAL.

DATA: WA\_FINAL TYPE TY\_FINAL,  
        IT\_FINAL TYPE TABLE OF TY\_FINAL.

\* Declare the field catalog.

DATA: IT\_FCAT TYPE SLIS\_T\_FIELDCAT\_ALV,  
        WA\_FCAT LIKE LINE OF IT\_FCAT.

\* Declare the event internal table.

DATA: IT\_EVENT TYPE SLIS\_T\_EVENT,  
        WA\_EVENT LIKE LINE OF IT\_EVENT.

AT SELECTION-SCREEN.

SELECT SINGLE BUKRS FROM T001 INTO V WHERE BUKRS IN S\_BUKRS.

IF SY-SUBRC <> 0.

MESSAGE E000(ZMESSAGE1) WITH 'PLEASE SELECT VALID COMPANY CODE'.

ENDIF.

\* Filling the data internal table

```

SELECT T001~BUKRS T001~BUTXT KNB1~KUNNR INTO TABLE IT_FINAL FROM T001
INNER JOIN KNB1 ON T001~BUKRS = KNB1~BUKRS WHERE T001~BUKRS IN S_BUKRS.
* Filling field catalog
WA_FCAT-FIELDNAME = 'BUKRS'.
WA_FCAT-COL_POS = '1'.
WA_FCAT-SELTEXT_M = 'COCD'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.
WA_FCAT-FIELDNAME = 'BUTXT'.
WA_FCAT-COL_POS = '2'.
WA_FCAT-SELTEXT_M = 'COMPANY NAME'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.
WA_FCAT-FIELDNAME = 'KUNNR'.
WA_FCAT-COL_POS = '3'.
WA_FCAT-SELTEXT_M = 'CUSTOMER'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.
* Filling event IT.
WA_EVENT-NAME = 'TOP_OF_PAGE'.
WA_EVENT-FORM = 'TOP'.
* Perform TOP
APPEND WA_EVENT TO IT_EVENT.
* Display output
CALL FUNCTION 'REUSE_ALV_LIST_DISPLAY'
  EXPORTING
    I_CALLBACK_PROGRAM = SY-CPROG
    IT_FIELDCAT        = IT_FCAT
    IT_EVENTS          = IT_EVENT
  TABLES
    T_OUTTAB          = IT_FINAL.

FORM TOP.
  WRITE:/ 'SHREE JANANI TECHNOLOGIES'.
  WRITE:/ 'THESE ARE CUSTOMERS UNDER COMPANY'.
ENDFORM.

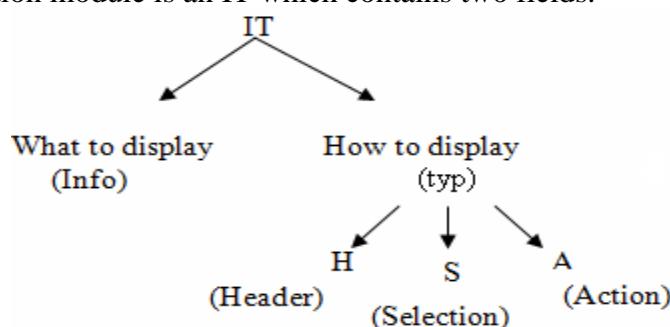
```

**Note:** - If you want to print the logo in the top of page then list display doesn't support. Only grid display supports the logo, but we can't print the text in the top or bottom events by using write statements in grid display.

#### REUSE\_ALV\_COMMENTARY\_WRITE: -

It's the function module which is used to display the text top or bottom events.

The input for the above function module is an IT which contains two fields.



**Note:** - In slis we have one type i.e. **slis\_t\_listheader** which contains the above two fields. So we simply declare our internal table by reffering **slis\_t\_listheader**.

→ Based on the given purchasing document numbers display the purchasing document number, document dates and vendor numbers by using ALV. As shown in the below.

Top-of-page	These are PO details S JF TECHNOLOGIES		
Pur.doc no	Doc date	Vendor	
300004	01.05.2000	5230	
300005	05.05.2004	3910	
300006	15.07.2010	1510	
End-of-list	SR NAGAR, HYDERABAD		

### Steps to upload the logo in ALV: -

Execute OAER

Provide the class name is 'pictures'  
Class type as 'OT'  
Object by as 'Any name'

Execute (F8).

→ Expand standard document types in the bottom window. Double click on screen. Browse the logo. Enter.

**Note:** - If you want to print the logo then you must pass the object key name into commentary write function module.

IT_List2	Info	Name
	SR Nagar	A

IT_Event	Name	Form
Top-of-page	XYZ	
End-of list	ABC	

WA_Event	Name	Form
SR Nagar	A	

IT_List	Info	Name
These are PO details	H	
SP RAO Technologies	S	

### TYPE-POOLS SLIS.

TABLES EKKO.

SELECT-OPTIONS S\_EBELN FOR EKKO-EBELN.

TYPES: BEGIN OF TY\_EKKO,  
EBELN TYPE EKKO-EBELN,  
BEDAT TYPE EKKO-BEDAT,  
KUNNR TYPE EKKO-KUNNR,  
END OF TY\_EKKO.

DATA: WA\_EKKO TYPE TY\_EKKO,  
IT\_EKKO TYPE TABLE OF TY\_EKKO.

\* Declaring the field catalog.

DATA: IT\_FCAT TYPE SLIS\_T\_FIELDCAT\_ALV,  
WA\_FCAT LIKE LINE OF IT\_FCAT.

\* Filling the data internal table.

एम एन सतीष कुमार रेडि

```

SELECT EBELN BEDAT KUNNR FROM EKKO INTO TABLE IT_EKKO WHERE EBELN IN
S_EBELN.
* Filling the field catalog
WA_FCAT-FIELDNAME = 'EBELN'.
WA_FCAT-COL_POS = '1'.
WA_FCAT-SELTEXT_M = 'PUR DOC'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME = 'BEDAT'.
WA_FCAT-COL_POS = '2'.
WA_FCAT-SELTEXT_M = 'DOC DATE'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME = 'KUNNR'.
WA_FCAT-COL_POS = '3'.
WA_FCAT-SELTEXT_M = 'CUSTOMER'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

* Declare two event internal tables.
DATA: IT_EVENT TYPE SLIS_T_EVENT,
      WA_EVENT LIKE LINE OF IT_EVENT.
* Filling the event IT.
WA_EVENT-NAME = 'TOP_OF_PAGE'.
WA_EVENT-FORM = 'TOP'.
* Perform TOP.
APPEND WA_EVENT TO IT_EVENT.
WA_EVENT-NAME = 'END_OF_LIST'.
WA_EVENT-FORM = 'END'.
APPEND WA_EVENT TO IT_EVENT.

* Display output
CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
  EXPORTING
    I_CALLBACK_PROGRAM = SY-CPROG
    IT_FIELDCAT        = IT_FCAT
    IT_EVENTS          = IT_EVENT
  TABLES
    T_OUTTAB           = IT_EKKO.

FORM TOP.
  DATA: IT_LIST TYPE SLIS_T_LISTHEADER,
        WA_LIST LIKE LINE OF IT_LIST.
  WA_LIST-INFO = 'THESE ARE PO DETAILS'.
  WA_LIST-TYP = 'H'.
  APPEND WA_LIST TO IT_LIST.
  WA_LIST-INFO = 'SJF TECHNOLOGIES'.
  WA_LIST-TYP = 'S'.
  APPEND WA_LIST TO IT_LIST.
  CALL FUNCTION 'REUSE_ALV_COMMENTARY_WRITE'

```

```

EXPORTING
  IT_LIST_COMMENTARY = IT_LIST
  I_LOGO             = 'SATISH'.
ENDFORM.

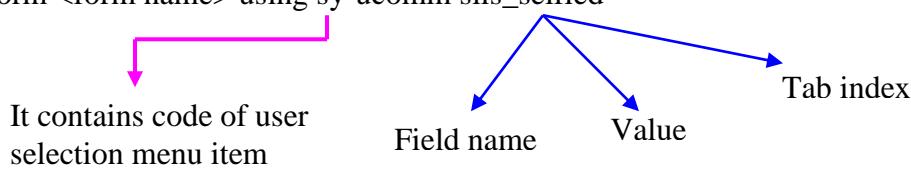
FORM END.

DATA: IT_LIST1 TYPE SLIS_T_LISTHEADER,
      WA_LIST1 LIKE LINE OF IT_LIST1.
WA_LIST1-INFO  = 'SR NAGAR'.
WA_LIST1-TYP   = 'A'.
APPEND WA_LIST1 TO IT_LIST1.
CALL FUNCTION 'REUSE_ALV_COMMENTARY_WRITE'
  EXPORTING
    IT_LIST_COMMENTARY = IT_LIST1.
ENDEFORM.

```

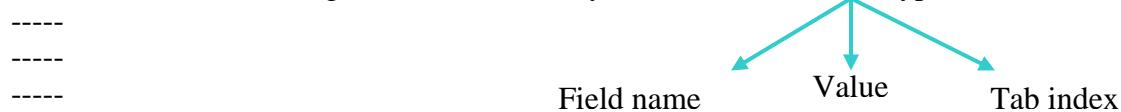
Syntax of calling subroutine for the event USER\_COMMAND: -

Perform <form name> using sv-ucomm slis selfied



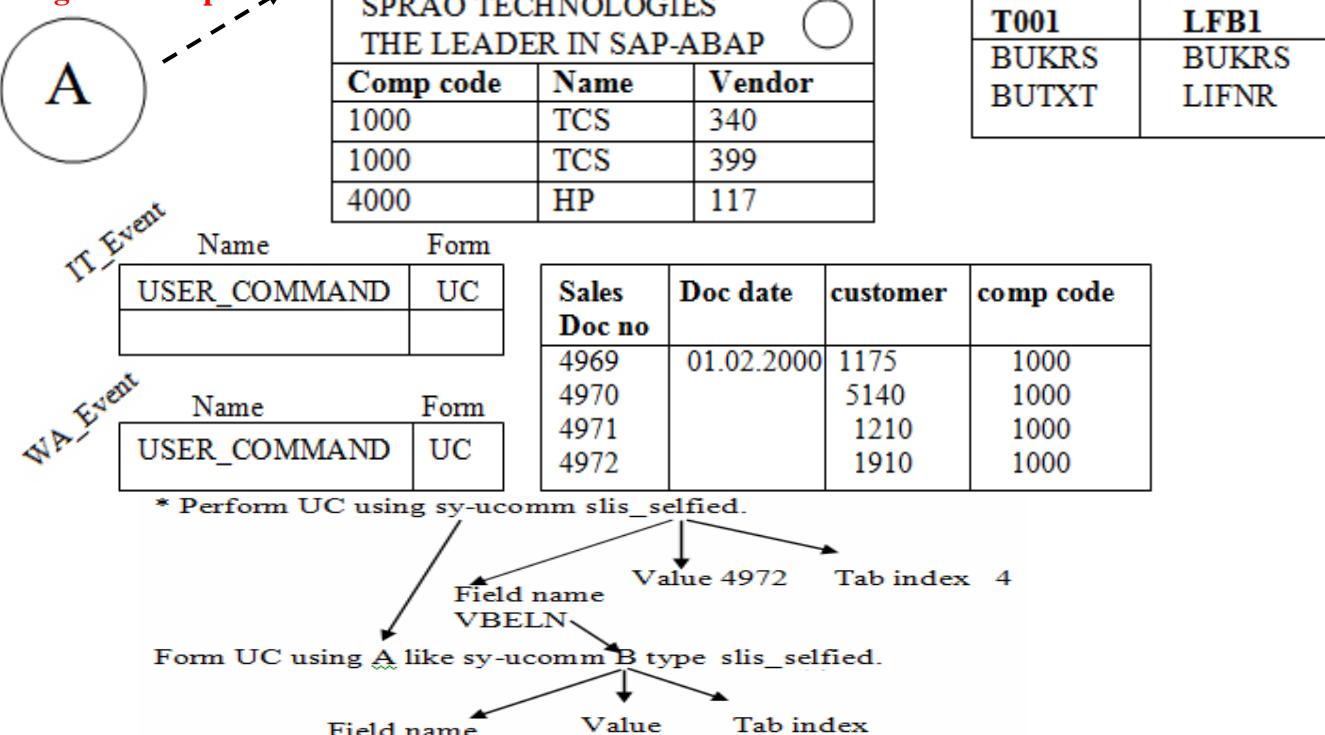
## **Syntax of definition: -**

Form <form name> using <variable 1> like sy-ucomm <variable 2> type slis\_selfied.



Endform

→ Based on the given company codes display the company codes, company names and vendors under company as shown in below by using ALV and also print the company code with green color along with hotspot.



```

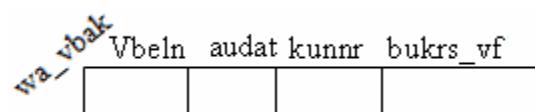
Type-pools slis.
Tables vbak.
Select-options s_vbeln for vbak-vbeln.
Data v type vbap-vbeln.
TYPES: BEGIN OF TY_VBAK,
        VBELN TYPE VBAK-VBELN,
        AUDAT TYPE VBAK-AUDAT,
        KUNNR TYPE VBAK-KUNNR,
        BUKRS_VF TYPE VBAK-BUKRS_VF,
        END OF TY_VBAK.
DATA: WA_VBAK TYPE TY_VBAK,
      IT_VBAK TYPE TABLE OF TY_VBAK.
DATA: IT_VBAP TYPE TABLE OF VBAP,
      WA_VBAP LIKE LINE OF IT_VBAP.
* Filling the data internal table.
Select vbeln audat kunnr bukrs_vf from vbak into table it_vbak where
vbeln in s_vbeln.
* Declare the field catalog
Data: it_fcat type slis_t_fieldcat_alv,
      Wa_fcat like line of it_fcat.
* Filling the field catalog.
WA_FCAT-FIELDNAME = 'VBELN'.
WA_FCAT-COL_POS = '1'.
WA_FCAT-SELTEXT_M = 'SALES DOC'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.
WA_FCAT-FIELDNAME = 'AUDAT'.
WA_FCAT-COL_POS = '2'.
WA_FCAT-SELTEXT_M = 'DOC DATE'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.
WA_FCAT-FIELDNAME = 'KUNNR'.
WA_FCAT-COL_POS = '3'.
WA_FCAT-SELTEXT_M = 'CUST NO'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.
WA_FCAT-FIELDNAME = 'BUKRS_VF'.
WA_FCAT-COL_POS = '4'.
WA_FCAT-SELTEXT_M = 'COCD'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.
* Declare the event IT.
Data: it_event type slis_t_event,
      wa_event like line of it_event.
* Filling the event IT.
wa_event-name = 'USER_COMMAND'.
wa_event-form = 'UC'.
* Perform UC using sy-ucomm slis_selfield
Append wa_event to it_event.
* Display output

```

```

CALL FUNCTION 'REUSE_ALV_LIST_DISPLAY'
EXPORTING
  I_CALLBACK_PROGRAM = SY-CPROG
  IT_FIELDCAT        = IT_FCAT
  IT_EVENTS          = IT_EVENT
TABLES
  T_OUTTAB           = IT_VBAK.
Form UC using A like sy-ucomm B type slis_selfield.
If B-fieldname = 'VBELN'.
  V = B-VALUE.
  CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
    EXPORTING
      INPUT = V
    IMPORTING
      OUTPUT = V.
  SELECT * FROM VBAP INTO TABLE IT_VBAP WHERE VBELN = V.
  CALL FUNCTION 'REUSE_ALV_LIST_DISPLAY'
    EXPORTING
      I_STRUCTURE_NAME = 'VBAP'
    TABLES
      T_OUTTAB           = IT_VBAP.
*read table it_vbak into wa_vbak index b-tabindex.
*select * from vbap into table it_vbap where vbeln = wa_vbak-vbeln.
*  CALL FUNCTION 'REUSE_ALV_LIST_DISPLAY'
*  EXPORTING
*    I_STRUCTURE_NAME = 'VBAP'
*  TABLES
*    T_OUTTAB           = IT_VBAP.
Endif.
Endform.

```



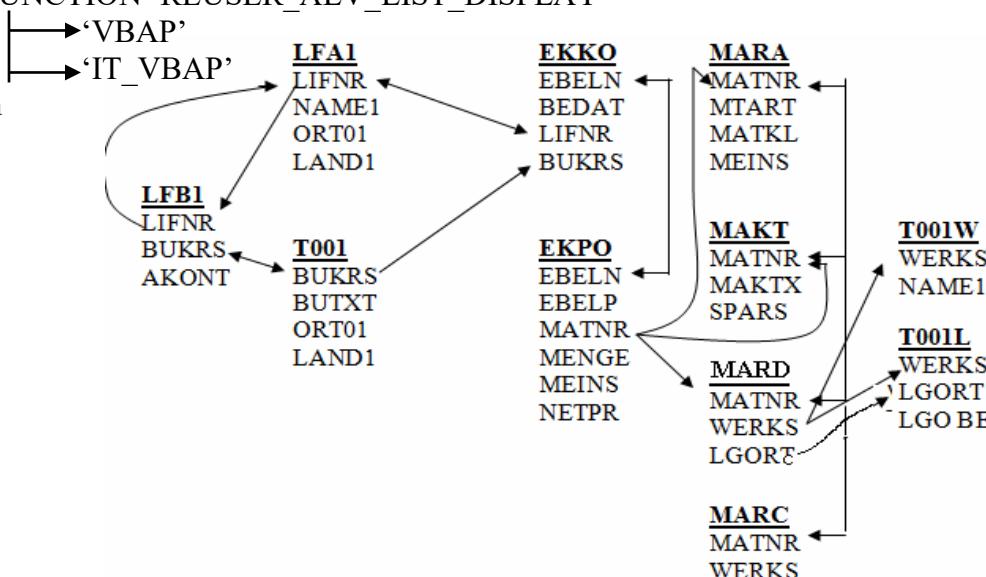
⇒ If the user clicks on any record then we display the sales document item details.

→ Form UC using A like sy-ucomm B type slis\_selfield.

Read table it\_vbak into wa\_vbak index B-tabindex.

Select \* from vbap into table it\_vbap where vbeln = wa\_vbak-vbeln.

CALL FUNCTION 'REUSER\_ALV\_LIST\_DISPLAY'



→ Based on the given vendor numbers, display the purchase document numbers, document dates & vendor numbers by using ALV & also display the purchase document numbers with yellow color. If the user clicks on any purchasing document number only then we display the all the purchasing document item details by using ALV.

```
Type-pools SLIS.  
Tables EKKO.  
Select-options s_EBELN for ekko-EBELN.  
* Declare the data internal table  
Types: Begin of ty_ekko,  
        EBELN type EKKO-EBELN,  
        BEDAT type EKKO-BEDAT,  
        LIFNR type EKKO-LIFNR,  
        End of ty_ekko.  
Data it_ekko type table of ty_ekko.  
Data it_ekpo type table of ekpo.  
* filling the data internal table  
Select ebeln bedat lifnr from ekko into table it_ekko where ebeln in  
s_ebeln.  
* Declare the field catalog  
DATA: IT_FCAT TYPE SLIS_T_FIELDCAT_ALV,  
      WA_FCAT LIKE LINE OF IT_FCAT.  
* Filling the field catalog  
Wa_fcat-fieldname      =      'EBELN'.  
Wa_fcat-col_pos         =      '1'.  
Wa_fcat-seltext_m       =      'PUR.DOC'.  
Wa_fcat-emphasize      =      'C310'.  
Append wa_fcat to it_fcat.  
Clear wa_fcat.  
Wa_fcat-fieldname      =      'BEDAT'.  
Wa_fcat-col_pos         =      '2'.  
Wa_fcat-seltext_m       =      'DOC DATE'.  
Append wa_fcat to it_fcat.  
Clear wa_fcat.  
Wa_fcat-fieldname      =      'LIFNR'.  
Wa_fcat-col_pos         =      '3'.  
Wa_fcat-seltext_m       =      'VENDOR'.  
Append wa_fcat to it_fcat.  
Clear wa_fcat.  
* Declare the event internal table  
Data: it_event type slis_t_event,  
      wa_event like line of it_event.  
* Filling the event internal table  
Wa_event-name          =      'USER_COMMAND'.  
WA_Event-form           =      'UC'.  
* Perform UC using sy-ucomm slis_selfield  
Append wa_event to it_event.  
* Display the output  
CALL FUNCTION 'REUSE_ALV_LIST_DISPLAY'  
  EXPORTING  
    I_CALLBACK_PROGRAM = SY-CPROG
```

```

IT_FIELDCAT      = IT_FCAT
IT_EVENTS        = IT_EVENT
TABLES
T_OUTTAB        = IT_EKKO.
FORM UC USING A LIKE SY-UCOMM B TYPE SLIS_SELFIELD.
IF B-fieldname = 'EBELN'.
  Select * from ekpo into table it_ekpo where ebeln = B-value.
  CALL FUNCTION 'REUSE_ALV_LIST_DISPLAY'
    EXPORTING
      I_STRUCTURE_NAME      = 'EKPO'
    TABLES
      T_OUTTAB      = IT_EKPO.
ENDIF.
ENDFORM.

```

### LOGIC: -

WA\_EVENT-NAME = 'USER\_COMMAND'.  
 WA\_EVENT-FORM = 'UC'.

\* PERFORM UC USING SY-UCOMM SLIS\_SELFIELD

Append wa\_event to it\_event.

Call function 'REUSE\_ALV\_LIST\_DISPLAY'.

- IT\_EKKO
- IT\_FCAT
- IT\_EVENT

Field name (EBELN)

Value (300012)

Tabindex (4)

FORM UC USING A LIKE SY-UCOMM B TYPE SLIS\_SELFIELD.

IF B-FIELDNAME = 'EBELN'.

----- } LOGIC

Tabindex (4)

Field name (EBELN)

Value (300012)

ENDIF.

ENDFORM.

### Some of the standard transaction codes: -

1. XK03 → Display the vendor
2. XD03 → Display the customer
3. MM03 → Display the material
4. ME23N → Display purchasing order
5. VA03 → Display sales order
6. FB03 → Display the accounting document

*Note: - 1 → create 2 → change 3 → display*

### Syntax of call the transaction: -

Call transaction '<transaction code>'.

If you want to skip the first screen: -

Call transaction '<transaction code>' and skip first screen.

Syntax of set the value before call the transaction: -

Set parameter id '<IDNAME>' field '<VALUE>'.

### Steps to identify the parameter id at transaction: -

Execute the transaction code XK03. Place the cursor on input field. Click on technical information. Identify the parameter id.

**EX: -**

Parameter p\_lifnr type lfal-lifnr.

Set parameter id 'LIF' field p\_lifnr.

Call transaction 'XK03' and skip first screen.

→ Based on the given sales document numbers to display the sales doc numbers, document dates & customer numbers by using ALV & also display the sales document number with green color. If the user clicks on any sales document only then we display the sales order details through 'VA03' transaction. If the user clicks on any customer number only then we display the customer details through 'XD03' transaction.

\* Declare select-options

\* Declare data internal table, field catalog, event internal table

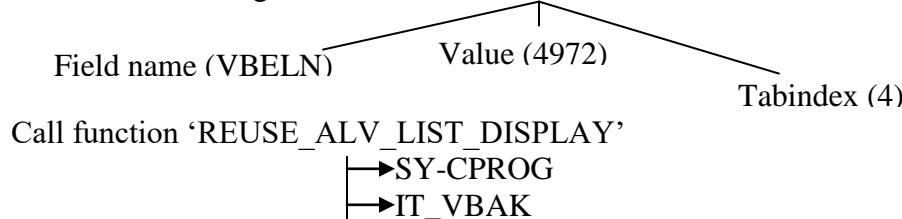
\* Fill the data internal table

\* Fill the field catalog

Wa\_event-name = 'USER\_COMMAND'.

Wa\_event-form = 'UC'.

\* Perform UC using SY-UCOMM SLIS\_SELFIELD.



FORM UC USING A LIKE SY-UCOMM B TYPE SLIS\_SELFIELD.

IF B-FIELDNAME = 'VBELN'.

set parameter id 'AUN' field B-value.

call transaction 'VA03' and skip first screen.

Endif.

Sales doc	Doc date	Customer
4969	01.05.2000	1175
4970	02.05.2012	1010
4971	05.09.2014	3050

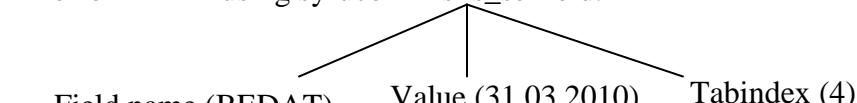
NAME	FORM
USER_COMMAND	UC

→ Based on the given purchasing document numbers, display the purchasing document numbers, document dates & vendor numbers by using ALV. If the user clicks on any record then we display the purchasing document details through 'ME23N' transaction.

Wa\_event-name = 'USER\_COMMAND'.

Wa\_event-form = 'XYZ'.

\* Perform XYZ using sy-ucomm slis\_selfield.



Append wa\_event to it\_event.

Call function 'REUSE\_ALV\_LIST\_DISPLAY'

IT\_EKKO

## IT\_FCAT IT\_EVENT

Form XYZ using A like sy-ucomm B type slis\_selfield.

Field name (BEDAT)      Value (31.03.2010)      Tabindex (4)

S\_EBELN  to

Read table it\_ekko into wa\_ekko index b-tabindex.

Set parameter id 'BES' field wa\_ekko-ebeln.

Call transaction 'ME23N'.

Endform.

### Syntax of calling subroutine for the pf status set event: -

Perform <form name> using slis\_t\_extab.

### Syntax of definition: -

Form <form name> using <variable 1> type slis\_t\_extab.

----- } logic

Endform.

**Note:** - When ever we are working with ALV then we never create our own GUI to the program we always copy the existing GUI from SAPLKKBL standard program.

If you create our own GUI to the ALV program then default functionality will loss. (Sorted list in ascending, in descending ---).

→ Based on the given purchasing document number display the purchasing document number, document dates and vendor numbers as shown in the below by using ALV. Here also attach display button in the application tool bar.



	Pur.doc	Doc date	Vendor
<input type="checkbox"/>	300004	01.05.2000	3910
<input type="checkbox"/>	300005	15.02.2002	5370
<input checked="" type="checkbox"/>	300006	31.03.2010	1910
<input type="checkbox"/>	300007	02.09/2012	5330

If the user click on display button then we display the selected check box purchasing doc details through ME23N transaction.

### Steps to copy the existing GUI: -

Execute SE41. Click on copy status in the application tool bar (copy status). Provide from program SAPLKKBL. Select the status 'STANDARD\_FULLSCREEN'. Provide to program (ZREPO12), Status (STAT). Click on copy and enter. Open our status in change mode. Expand the application tool bar. Provide transaction code for additional button (DIS) where the place is available. Double click on it and

press enter. Provide the function text (Display). Click on enter. Select the shortcut key. Enter. Save, check and activate GUI.

Type-pools slis.

Tables ekko.

Select-options s\_ebeln for ekko-ebeln.

\* Declare the data IT.

Data: begin of wa\_ekko,

    Ebeln type ekko-ebeln,

    Bedat type ekko-bedat,

    Lifnr type ekko-lifnr,

    Chk,

End of wa\_ekko.

Data it\_ekko like table of wa\_ekko.

\* Filling the data IT.

SELECT EBELN BEDAT LIFNR FROM EKKO INTO TABLE IT\_EKKO WHERE EBELN  
IN S\_EBELN.

\* Declare the field catalog.

DATA: IT\_FCAT TYPE SLIS\_T\_FIELDCAT\_ALV,  
      WA\_FCAT LIKE LINE OF IT\_FCAT.

\* Filling field catalog.

Wa\_fcat-fieldname = 'CHK'.

Wa\_fcat-COL\_POS = '1'.

Wa\_fcat-selttext\_m = ' '.

Wa\_fcat-checkbox = 'X'.

Wa\_fcat-edit = 'X'.

Wa\_fcat-outputlen = '3'.

Append wa\_fcat to it\_fcat.

Clear wa\_fcat.

Wa\_fcat-fieldname = 'EBELN'.

Wa\_fcat-col\_pos = '2'.

Wa\_fcat-selttext\_m = 'PUR DOC'.

Append wa\_fcat to it\_fcat.

Clear wa\_fcat.

Wa\_fcat-fieldname = 'BEDAT'.

Wa\_fcat-col\_pos = '3'.

Wa\_fcat-selttext\_m = 'DOC DATE'.

Append wa\_fcat to it\_fcat.

Clear wa\_fcat.

Wa\_fcat-fieldname = 'LIFNR'.

Wa\_fcat-col\_pos = '4'.

Wa\_fcat-selttext\_m = 'VENDOR'.

Append wa\_fcat to it\_fcat.

Clear wa\_fcat.

\* Declare the event IT.

Data: it\_event type slis\_t\_event,

      Wa\_event like line of it\_event.

wa\_event-name = 'PF\_STATUS\_SET'.

wa\_event-form = 'GUI'.

\* Perform GUI using slis\_t\_extab

Append wa\_event to it\_event.

clear wa\_event.

```

Wa_event-name = 'USER_COMMAND'.
Wa_event-form = 'UC'.
* Perform UC using sy-ucomm slis_selfield
Append wa_event to it_event.
* Display the output
CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
  EXPORTING
    I_callback_program = sy-cprog
    It_fieldcat        = it_fcat
    It_events          = it_event
  TABLES
    T_outtab           = it_ekko.

Form GUI using M type slis_t_extab.
  Set pf-status 'STAT'.
Endform.

Form UC using A like sy-ucomm B type slis_selfield.
  If A = 'DIS'.
    Read table it_ekko into wa_ekko index b-tabindex.
    Set parameter id 'RES' field wa_ekko-ebeln.
    CALL TRANSACTION 'ME23N' and skip first screen.
  ENDIF.
ENDFORM.

```

### Table Maintenance Generator: -

This is used to create the user interface to the data base table to perform the ‘DML’ operations (Insert, Update, Modify) without any code.

### Some of the fields in layout WA: -

Colwidth\_optimize → Compress the displayed field

Zebra → Stripped pattern

Info\_fieldname → colour field.

**Note:** - In slis we have one type I.e. slis\_layout\_alv which contains all the fields related to layout work area. So we simply declare our layout work area by referring above type.

I → Structure

IS → Work area

CT  
IT } Internal table  
T }

## Blocked ALV

### Blocked ALV: -

Blocked ALV is used to display the output in a block wise.

### Steps to work with blocked ALV: -

1. Initialize the blocked ALV by using ‘REUSE\_ALV\_BLOCK\_LIST\_INIT’ function module. The input for the above function module is ‘current program name’.
2. Append the data IT to the blocked ALV by using ‘REUSE\_ALV\_BLOCK\_LIST\_APPEND’ function module. The input for the above function module is
  1. <Data IT>
  2. <Field catalog IT>
  3. <event IT> } Dummy also ok
  4. <Layout WA>

3. Display the data in blocked ALV by using ‘REUSE\_ALV\_BLOCK\_LIST\_DISPLAY’ function module.

→ Based on the given purchasing document numbers display the purchasing document header (EBELN, BEDAT, LIFNR) and item (EBELN EBELP MENGE MEINS NETPR) details by using blocked ALV as shown in the below.

PUR_DOC	DOC_DATE	VENDOR
4500000500	24.01.2015	300005
4500000501	27.01.2015	300000
4500000502	27.01.2015	300001
4500000503	27.01.2015	300011
4500000504	27.01.2015	300224
4500000505	27.01.2015	300000

PUR_DOC	ITME	QTY	UOM	NET PRICE
4500000500	00010	100,000	NOS	950,00
4500000501	00010	1,000	NOS	10,00
4500000502	00010	100,000	NOS	1.000,00
4500000503	00010	10,000	NOS	103,00
4500000504	00010	4,000	EA	0,00
4500000505	00010	100,000	NOS	0,00

```

REPORT ZREPORT2007.

TYPE-POOLS SLIS.

TABLES EKKO.

SELECT-OPTIONS S_EBELN FOR EKKO-EBELN.

* Declare data IT (EKKO, EKPO)

TYPES: BEGIN OF TY_EKKO,
        EBELN TYPE EKKO-EBELN,
        BEDAT TYPE EKKO-BEDAT,
        LIFNR TYPE EKKO-LIFNR,
        END OF TY_EKKO.

DATA: WA_EKKO TYPE TY_EKKO,
      IT_EKKO TYPE TABLE OF TY_EKKO.

TYPES: BEGIN OF TY_EKPO,
        EBELN TYPE EKPO-EBELN,
        EBELP TYPE EKPO-EBELP,
        MENGE TYPE EKPO-MENGE,
        MEINS TYPE EKPO-MEINS,
        NETPR TYPE EKPO-NETPR,
        END OF TY_EKPO.

DATA: WA_EKPO TYPE TY_EKPO,
      IT_EKPO TYPE TABLE OF TY_EKPO.


```

```

* Filling the data IT.

SELECT EBELN BEDAT LIFNR FROM EKKO INTO TABLE IT_EKKO WHERE EBELN
IN S_EBELN.

IF IT_EKKO IS NOT INITIAL.

  SELECT EBELN EBELP MENGE MEINS NETPR FROM EKPO INTO TABLE IT_EKPO
  FOR ALL ENTRIES IN IT_EKKO WHERE EBELN = IT_EKKO-EBELN.

ENDIF.


```

```

* Declare the field catalogs

DATA: IT_FCAT TYPE SLIS_T_FIELDCAT_ALV,
      WA_FCAT LIKE LINE OF IT_FCAT.

DATA: IT_FCAT1 TYPE SLIS_T_FIELDCAT_ALV,
      WA_FCAT1 LIKE LINE OF IT_FCAT1.


```

\* Filling the field catalog

```

WA_FCAT-FIELDNAME = 'EBELN'.
WA_FCAT-COL_POS = '1'.
WA_FCAT-SELTEXT_M = 'PUR DOC'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.
WA_FCAT-FIELDNAME = 'BEDAT'.
WA_FCAT-COL_POS = '2'.
WA_FCAT-SELTEXT_M = 'DOC DATE'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.
WA_FCAT-FIELDNAME = 'LIFNR'.
WA_FCAT-COL_POS = '3'.
WA_FCAT-SELTEXT_M = 'VENDOR'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT1-FIELDNAME = 'EBELN'.
WA_FCAT1-COL_POS = '1'.
WA_FCAT1-SELTEXT_M = 'PUR DOC'.
APPEND WA_FCAT1 TO IT_FCAT1.
CLEAR WA_FCAT1.
WA_FCAT1-FIELDNAME = 'EBELP'.
WA_FCAT1-COL_POS = '2'.
WA_FCAT1-SELTEXT_M = 'ITME'.
APPEND WA_FCAT1 TO IT_FCAT1.
CLEAR WA_FCAT1.
WA_FCAT1-FIELDNAME = 'MENGE'.
WA_FCAT1-COL_POS = '3'.
WA_FCAT1-SELTEXT_M = 'QTY'.
APPEND WA_FCAT1 TO IT_FCAT1.
CLEAR WA_FCAT1.
WA_FCAT1-FIELDNAME = 'MEINS'.
WA_FCAT1-COL_POS = '4'.
WA_FCAT1-SELTEXT_M = 'UOM'.
APPEND WA_FCAT1 TO IT_FCAT1.
CLEAR WA_FCAT1.
WA_FCAT1-FIELDNAME = 'NETPR'.
WA_FCAT1-COL_POS = '5'.
WA_FCAT1-SELTEXT_M = 'NET PRICE'.
APPEND WA_FCAT1 TO IT_FCAT1.
CLEAR WA_FCAT1.

* Declare the event IT
DATA: IT_EVENT TYPE SLIS_T_EVENT,
      WA_EVENT LIKE LINE OF IT_EVENT.
DATA: IT_EVENT1 TYPE SLIS_T_EVENT,
      WA_EVENT1 LIKE LINE OF IT_EVENT1.

* Declare the layout WA (2).
DATA WA_LAYOUT TYPE SLIS_LAYOUT_ALV.
DATA WA_LAYOUT1 TYPE SLIS_LAYOUT_ALV.

```

```

* Initialize the blocked ALV.
CALL FUNCTION 'REUSE_ALV_BLOCK_LIST_INIT'
  EXPORTING
    I_CALLBACK_PROGRAM = SY-CPROG.
CALL FUNCTION 'REUSE_ALV_BLOCK_LIST_APPEND'
  EXPORTING
    IS_LAYOUT      = WA_LAYOUT
    IT_FIELDCAT   = IT_FCAT
    I_TABNAME     = 'WA_EKKO'
    IT_EVENTS     = IT_EVENT
  TABLES
    T_OUTTAB      = IT_EKKO.

CALL FUNCTION 'REUSE_ALV_BLOCK_LIST_APPEND'
  EXPORTING
    IS_LAYOUT      = WA_LAYOUT1
    IT_FIELDCAT   = IT_FCAT1
    I_TABNAME     = 'WA_EKPO'
    IT_EVENTS     = IT_EVENT1
  TABLES
    T_OUTTAB      = IT_EKPO.
* Display output.
CALL FUNCTION 'REUSE_ALV_BLOCK_LIST_DISPLAY'.

```

#### Steps to work with row colour: -

1. Declare the one additional colour field in the data IT, which is char and length 4.
2. Fill the data IT based on given input.
3. Fill the field catalog (not colour field)
4. Pass the colour field name into layout WA (info\_field name).
5. Modify the colour field based on the requirement.
6. Display output

→ Based on the given purchasing document no, display the purchase document numbers, item numbers, quantity, unit of measurement, net prize by using ALV & also display the line item details in red colour if the amt is more than 1000.

EBELN	EBELP	MENGE	MEINS	NETPR	XYZ
300004	01	10	KG	290.00	
300004	02	02	PCS	1050.00	C610
300005	01	02	PCS	200.00	
300005	02	03	BOX	2000.00	C610

EBELN	EBELP	MENGE	MEINS	NETPR	XYZ
300004	01	10	KG	290.00	

```

type-pools slis.
tables ekpo.
select-options s_ebeln for ekpo-ebeln.
types: begin of ty_ekpo,
         ebeln type ekpo-ebeln,
         ebelp type ekpo-ebelp,
         menge type ekpo-menge,
         meins type ekpo-meins,
         netpr type ekpo-netpr,
         xyz(4) type c,

```

```

        end of ty_ekpo.
data: wa_ekpo type ty_ekpo,
      it_ekpo type table OF ty_ekpo.
data: it_fcat type slis_t_fieldcat_alv,
      wa_fcat like line of it_fcat.
* Delcare the layout
data wa_layout type slis_layout_alv.
* Fill the layout wa.
wa_layout-info_fieldname = 'XYZ'.

select ebeln ebelp menge meins netpr from ekpo into table it_ekpo where
ebeln in s_ebeln.
wa_fcat-fieldname = 'EBELN'.
wa_fcat-col_pos = '1'.
wa_fcat-seltext_m = 'PUR DOC'.
append wa_fcat to it_fcat.
clear wa_fcat.
wa_fcat-fieldname = 'EBELP'.
wa_fcat-col_pos = '2'.
wa_fcat-seltext_m = 'ITEM'.
append wa_fcat to it_fcat.
clear wa_fcat.
wa_fcat-fieldname = 'MENGE'.
wa_fcat-col_pos = '3'.
wa_fcat-seltext_m = 'QTY'.
append wa_fcat to it_fcat.
clear wa_fcat.
wa_fcat-fieldname = 'MEINS'.
wa_fcat-col_pos = '4'.
wa_fcat-seltext_m = 'UOM'.
append wa_fcat to it_fcat.
clear wa_fcat.
wa_fcat-fieldname = 'NETPR'.
wa_fcat-col_pos = '5'.
wa_fcat-seltext_m = 'NET PRICE'.
append wa_fcat to it_fcat.
clear wa_fcat.

* Modify the color field
wa_ekpo-xyz = 'C610'.
modify it_ekpo from wa_ekpo transporting XYZ where NETPR > 1000.

CALL FUNCTION 'REUSE_ALV_LIST_DISPLAY'
EXPORTING
  IS_LAYOUT    = WA_LAYOUT
  IT_FIELDCAT = IT_FCAT
TABLES
  T_OUTTAB    = IT_EKPO.

```



→ Based on the Purchase document number, display the pur doc num, item, material, quantity, units, net prise. Here one pur doc number can have so many items. I want sub total for each document number. Finally I want grand total also. It'll display as in below picture.

```

TABLES EKPO.
SELECT-OPTIONS S_EBELN FOR EKPO-EBELN.

TYPES: BEGIN OF TY_EKPO,
       EBELN TYPE EKPO-EBELN,
       EBELP TYPE EKPO-EBELP,
       MATNR TYPE EKPO-MATNR,
       MENGE TYPE EKPO-MENGE,
       MEINS TYPE EKPO-MEINS,
       NETPR TYPE EKPO-NETPR,
     END OF TY_EKPO.

DATA: WA_EKPO TYPE TY_EKPO,
      IT_EKPO TYPE TABLE OF TY_EKPO.

SELECT EBELN EBELP MATNR MENGE MEINS NETPR FROM EKPO INTO TABLE IT_EKPO WHERE EBELN IN
N S_EBELN.

DATA: IT_FCAT TYPE SLIS_T_FIELDCAT_ALV,
      WA_FCAT LIKE LINE OF IT_FCAT.

DATA: IT_SORTINFO TYPE SLIS_T_SORTINFO_ALV,
      WA_SORTINFO LIKE LINE OF IT_SORTINFO.

DATA WA_LAYOUT TYPE SLIS_LAYOUT_ALV.
WA_LAYOUT-COLWIDTH_OPTIMIZE = 'X'.

WA_FCAT-FIELDNAME = 'EBELN'.
WA_FCAT-COL_POS = '1'.
WA_FCAT-REF_TABNAME = 'EKPO'.
WA_FCAT-REF_FIELDNAME = 'EBELN'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME = 'EBELP'.
WA_FCAT-COL_POS = '2'.
WA_FCAT-REF_TABNAME = 'EKPO'.
WA_FCAT-REF_FIELDNAME = 'EBELP'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME = 'MATNR'.
WA_FCAT-COL_POS = '3'.
WA_FCAT-REF_TABNAME = 'EKPO'.
WA_FCAT-REF_FIELDNAME = 'MATNR'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME = 'MENGE'.
WA_FCAT-COL_POS = '4'.
WA_FCAT-REF_TABNAME = 'EKPO'.
WA_FCAT-REF_FIELDNAME = 'MENGE'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME = 'MEINS'.
WA_FCAT-COL_POS = '5'.
WA_FCAT-REF_TABNAME = 'EKPO'.

```

Purch.Doc...	Item	Material	Quantity	OU	Σ	Net Price
1236990	1	312	56.000	KG		1,050.00
1236990					▪	1,050.00
1236991	1	312	60.000	KG		1,050.00
1236991					▪	1,050.00
1236992	1	312	43.000	KG		1,050.00
1236992					▪	1,050.00
1236993	1	312	50.000	KG		1,050.00
1236993	2	319	20.000	KG		900.00
1236993	3	312	30.000	KG		1,050.00
1236993					▪	3,000.00
1236994	1	312	63.000	KG		1,050.00
1236994	2	312	25.000	KG		1,050.00
1236994	3	319	26.000	KG		900.00
1236994					▪	3,000.00
1236995	1	312	10.000	KG		1,050.00
1236995					▪	1,050.00
1236996	1	312	10.000	KG		1,050.00
1236996					▪	1,050.00
1236997	1	312	5.000	KG		1,050.00
1236997					▪	1,050.00
1236998	1	312	10.000	KG		1,050.00
1236998					▪	1,050.00
1236999	1	312	12.000	KG		1,050.00
1236999					▪	1,050.00
						▪ ▪ 14,400.00

```

WA_FCAT-REF_FIELDNAME = 'MEINS'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME = 'NETPR'.
WA_FCAT-COL_POS = '6'.
WA_FCAT-REF_TABNAME = 'EKPO'.
WA_FCAT-REF_FIELDNAME = 'NETPR'.
WA_FCAT-DO_SUM = 'X'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_SORTINFO-FIELDNAME = 'EBELN'.
WA_SORTINFO-SUBTOT = 'X'.
APPEND WA_SORTINFO TO IT_SORTINFO.
CLEAR WA_SORTINFO.

SORT IT_EKPO BY EBELN.

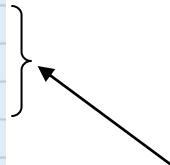
CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
  EXPORTING
    I_CALLBACK_PROGRAM = SY-CPROG
    IS_LAYOUT          = WA_LAYOUT
    IT_FIELDCAT        = IT_FCAT
    IT_SORT            = IT_SORTINFO
  TABLES
    T_OUTTAB           = IT_EKPO.

-----
***** -----

```

In the above program if you take WA\_SORTINFO-UP = 'X' instead of WA\_SOFTINFO-SUBTOT = 'X', the output will be like this.

Purch.Doc.	Item	Material	PO Quantity	OUr	Σ Net Price
1236987	10	312	1.000	KG	1,050.00
1236988	1	312	125.000	KG	1,050.00
1236989	1	312	85.000	KG	1,050.00
1236990	1	312	56.000	KG	1,050.00
1236991	1	312	60.000	KG	1,050.00
1236992	1	312	43.000	KG	1,050.00
1236993	1	312	50.000	KG	1,050.00
	2	319	20.000	KG	900.00
	3	312	30.000	KG	1,050.00
1236994	1	312	63.000	KG	1,050.00
	2	312	25.000	KG	1,050.00
	3	319	26.000	KG	900.00
1236995	1	312	10.000	KG	1,050.00
1236996	1	312	10.000	KG	1,050.00
1236997	1	312	5.000	KG	1,050.00
1236998	1	312	10.000	KG	1,050.00
1236999	1	312	12.000	KG	1,050.00



This will work in GRID display only.  
If you take LIST display, Grand total & Sub total only work. Hiding of same record is not possible

## → Display the traffic lights in the ALV report

```
TABLES VBAP.  
SELECT-OPTIONS S_VBELN FOR VBAP-VBELN.  
TYPES: BEGIN OF TY_VBAP,  
        VBELN TYPE VBAP-VBELN,  
        KWMENG TYPE VBAP-KWMENG,  
        MEINS TYPE VBAP-MEINS,  
        NETWR TYPE VBAP-NETWR,  
        ICON TYPE C,  
      END OF TY_VBAP.  
  
DATA: WA_VBAP TYPE TY_VBAP,  
      IT_VBAP TYPE TABLE OF TY_VBAP.  
DATA: IT_FCAT TYPE SLIS_T_FIELDCAT_ALV,  
      WA_FCAT LIKE LINE OF IT_FCAT.  
  
DATA WA_LAYOUT TYPE SLIS_LAYOUT_ALV.  
SELECT VBELN KWMENG MEINS NETWR FROM VBAP INTO TABLE IT_VBAP WHERE VBELN IN S_VBELN.  
  
WA_FCAT-FIELDNAME = 'VBELN'.  
WA_FCAT-COL_POS = '1'.  
WA_FCAT-SELTEXT_M = 'SALES DOC NUM'.  
APPEND WA_FCAT TO IT_FCAT.  
CLEAR WA_FCAT.  
WA_FCAT-FIELDNAME = 'KWMENG'.  
WA_FCAT-COL_POS = '2'.  
WA_FCAT-SELTEXT_M = 'QUANTITY'.  
APPEND WA_FCAT TO IT_FCAT.  
CLEAR WA_FCAT.  
WA_FCAT-FIELDNAME = 'MEINS'.  
WA_FCAT-COL_POS = '3'.  
WA_FCAT-SELTEXT_M = 'UNITS'.  
APPEND WA_FCAT TO IT_FCAT.  
CLEAR WA_FCAT.  
WA_FCAT-FIELDNAME = 'NETWR'.  
WA_FCAT-COL_POS = '4'.  
WA_FCAT-SELTEXT_M = 'NET VALUE'.  
WA_FCAT-DO_SUM = 'X'.  
APPEND WA_FCAT TO IT_FCAT.  
CLEAR WA_FCAT.  
  
LOOP AT IT_VBAP INTO WA_VBAP.  
  IF WA_VBAP-NETWR <= 500.  
    WA_VBAP-ICON = 1.  
  ELSEIF WA_VBAP-NETWR >= 501 AND WA_VBAP-NETWR <= 1000.  
    WA_VBAP-ICON = 2.  
  ELSE.  
    WA_VBAP-ICON = 3.  
 ENDIF.  
  MODIFY IT_VBAP FROM WA_VBAP TRANSPORTING ICON.  
  CLEAR WA_VBAP.  
ENDLOOP.  
WA_LAYOUT-LIGHTS_FIELDNAME = 'ICON'.  
  
CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'  
  EXPORTING  
    I_CALLBACK_PROGRAM          = SY-CPROG  
    IS_LAYOUT                   = WA_LAYOUT  
    IT_FIELDCAT                = IT_FCAT  
    TABLES  
    T_OUTTAB                   = IT_VBAP.
```

## HIERARCHICAL ALV

Hierarchical ALV is used to display the header and item details in a hierarchical manner.

**Note:** - ‘REUSE\_ALV\_HIERSEQ\_LIST\_DISPLAY’ is the function module which is used to display the header and item details in a hierarchical manner.

The input for the above function module is

- Two data IT (Header of item)
- Only one field catalog IT
- Key info WA

→ Link field between header and item table

Key info WA contains link fields between the header and item table.

EKKO  
EKPO → EBELN key info wa

### Fields in key info WA: -

Header 01

Item 01

Header 02

Item 02

Header 05

Item 05

**Note:** - In slis we have one type I.e. ‘SLIS\_KEYINFO\_ALV’ which contains above fields. So we simply declare our IT by referring SLIS\_KEYINFO\_ALV.

### Ex: -

EKKO > EBELN

MARD > WERKS  
T001L > LGORT

HEADER 01 = ‘EBELN’

HEADER 01 = ‘WERKS’

ITEM01 = ‘EBELN’

ITEM 01 = ‘WERKS’

HEADER 02 = ‘LGORT’

ITEM 02 = ‘LGORT’.

**Note:** - When ever we are working with hierarchical ALV then we must maintain link fields in between header and item internal table.

**Note:** - When ever we are working with hierarchical ALV then we must fill the field catalog IT.

1. REUSE\_ALV\_LIST\_DISPLAY
2. REUSE\_ALV\_GRID\_DISPLAY
3. REUSE\_ALV\_HIERSEQ\_LIST\_DISPLAY
4. REUSE\_ALV\_COMMENTARY\_WRITE
5. REUSE\_ALV\_BLOCK\_LIST\_DISPLAY
6. REUSE\_ALV\_FIELD CATALOG MERGE

1. SLIS\_T\_FIELDCAT\_ALV
2. SLIS\_T\_EVENT
3. SLIS\_T\_LIST HEADER
4. SLIS\_SELFIELD
5. SLIS\_T\_EXTAB
6. SLIS\_LAYOUT\_ALV
7. SLIS\_KEYINFO\_ALV

→ Based on the given purchasing document number display the purchasing document (EBELN, BEDAT, LIFNR) & purchasing document item details (EBELP, MENGE, MEINS, NETPR) in hierarchical manner as shown in the below by using ALV.

Type-pools slis.

Tables ekko.

Select-options s\_ebeln for ekko-ebeln.

\* Declare the data IT (IT\_EKKO, IT\_EKPO)

```
TYPES: BEGIN OF TY_EKKO,
       EBELN TYPE EKKO-EBELN,
       BEDAT TYPE EKKO-BEDAT,
       LIFNR TYPE EKKO-LIFNR,
       END OF TY_EKKO.
```

```
DATA: WA_EKKO TYPE TY_EKKO,
      IT_EKKO TYPE TABLE OF TY_EKKO.
```

```
TYPES: BEGIN OF TY_EKPO,
       EBELN TYPE EKPO-EBELN,
       EBELP TYPE EKPO-EBELP,
       MENGE TYPE EKPO-MENGE,
       MEINS TYPE EKPO-MEINS,
       NETPR TYPE EKPO-NETPR,
       END OF TY_EKPO.
```

```
DATA: WA_EKPO TYPE TY_EKPO,
      IT_EKPO TYPE TABLE OF TY_EKPO.
```

```
SELECT EBELN BEDAT LIFNR FROM EKKO INTO TABLE IT_EKKO WHERE EBELN
IN S_EBELN.
```

```
SELECT EBELN EBELP MENGE MEINS NETPR FROM EKPO INTO TABLE IT_EKPO
WHERE EBELN IN S_EBELN.
```

\* Declare the field catalogs

```
data: it_fcat type slis_t_fieldcat_alv,
      wa_fcat like line of it_fcat.
```

```
WA_FCAT-FIELDNAME = 'EBELN'.
WA_FCAT-COL_POS = '1'.
WA_FCAT-SELTEXT_M = 'PUR DOC'.
WA_FCAT-TABNAME = 'IT_EKKO'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.
WA_FCAT-FIELDNAME = 'BEDAT'.
WA_FCAT-COL_POS = '2'.
```

PUR DOC	DOC DATE VENDOR			
PUR DOC	ITME	QTY	UOM	NETPRICE
3000000004 02112000 5550				
3000000004	1	1.000	PC	27.95
3000000004	2	1.000	PC	10.95
3000000005 02112000 5550				
3000000005	1	1.000	PC	27.95
3000000005	2	1.000	PC	10.95

```

WA_FCAT-SELTEXT_M = 'DOC DATE'.
WA_FCAT-TABNAME = 'IT_EKKO'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.
WA_FCAT-FIELDNAME = 'LIFNR'.
WA_FCAT-COL_POS = '3'.
WA_FCAT-SELTEXT_M = 'VENDOR'.
WA_FCAT-TABNAME = 'IT_EKKO'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME = 'EBELN'.
WA_FCAT-COL_POS = '1'.
WA_FCAT-SELTEXT_M = 'PUR DOC'.
WA_FCAT-TABNAME = 'IT_EKPO'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.
WA_FCAT-FIELDNAME = 'EBELP'.
WA_FCAT-COL_POS = '2'.
WA_FCAT-SELTEXT_M = 'ITME'.
WA_FCAT-TABNAME = 'IT_EKPO'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME = 'MENGE'.
WA_FCAT-COL_POS = '3'.
WA_FCAT-SELTEXT_M = 'QTY'.
WA_FCAT-TABNAME = 'IT_EKPO'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME = 'MEINS'.
WA_FCAT-COL_POS = '4'.
WA_FCAT-SELTEXT_M = 'UOM'.
WA_FCAT-TABNAME = 'IT_EKPO'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME = 'NETPR'.
WA_FCAT-COL_POS = '5'.
WA_FCAT-SELTEXT_M = 'NETPRICE'.
WA_FCAT-TABNAME = 'IT_EKPO'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

* Declare the key info WA
Data wa_key type slis_keyinfo_alv.
* Filling the key info WA.
Wa_key-header01 = 'EBELN'.
Wa_key-item01 = 'EBELN'.
* Display output
CALL FUNCTION 'REUSE_ALV_HIERSEQ_LIST_DISPLAY'
  EXPORTING
    IT_FIELDCAT      = IT_FCAT

```

```

I_TABNAME_HEADER = 'IT_EKKO'
I_TABNAME_ITEM   = 'IT_EKPO'
IS_KEYINFO        = WA_KEY
TABLES
T_OUTTAB_HEADER  = IT_EKKO
T_OUTTAB_ITEM    = IT_EKPO.

→ Based on the given input fetch data, send those data in mail.

DATA :
g_sent_to_all      TYPE sonv-flag,
g_tab_lines        TYPE i,
obj_descr(50)      type c,
date_c(12)         TYPE c.

"Types
TYPES:
t_document_data   TYPE sodocchgil,
t_packing_list    TYPE sopcklstil,
t_attachment       TYPE solistil,
t_body_msg         TYPE solistil,
t_receivers        TYPE somlrecil.

"Workareas
DATA :
w_document_data   TYPE t_document_data,
w_packing_list    TYPE t_packing_list,
w_attachment       TYPE t_attachment,
w_body_msg         TYPE t_body_msg,
w_receivers        TYPE t_receivers.

"Internal Tables
i_document_data   TYPE STANDARD TABLE OF t_document_data,
i_packing_list    TYPE STANDARD TABLE OF t_packing_list,
i_attachment       TYPE STANDARD TABLE OF t_attachment,
i_body_msg         TYPE STANDARD TABLE OF t_body_msg,
i_receivers        TYPE STANDARD TABLE OF t_receivers.

CLASS cl_abap_char_utilities DEFINITION LOAD.
CONSTANTS:
con_tab  TYPE C VALUE cl_abap_char_utilities=>horizontal_tab,
con_cret TYPE C VALUE cl_abap_char_utilities=>cr_lf.

CONCATENATE 'SKU Code' 'SKU Description' 'Depot/Factory/RDC' 'Batch
no' 'EXP DATE' 'L3M Average Sale' 'All India Avg Sale' 'CM FC'
      INTO w_attachment
SEPARATED BY con_tab.
CONCATENATE con_cret
w_attachment
INTO w_attachment.
APPEND w_attachment TO i_attachment.
CLEAR w_attachment.

```

```

LOOP AT it_mail INTO wa_mail.
  CONCATENATE wa_mail-matnr
    wa_mail-maktx
    wa_mail-werks
    wa_mail-charg
    wa_mail-vfdat
    wa_mail-l3m_avg_sales
    wa_mail-all_ind_sal
    wa_mail-cm_fc  INTO w_attachment
  SEPARATED BY con_tab.
  CONCATENATE con_cret w_attachment
  INTO w_attachment.
  APPEND w_attachment TO i_attachment.
  CLEAR w_attachment.
  CLEAR :wa_mail-matnr,wa_mail-maktx,wa_mail-werks,wa_mail-
charg,wa_mail-vfdat,
        wa_mail-l3m_avg_sales,wa_mail-all_ind_sal,wa_mail-cm_fc.
ENDLOOP.

"Subject of the mail.
w_document_data-obj_name = 'MAIL_TO_HEAD'.
CONCATENATE 'Transfer & Partical transfer stock ' '' sy-
datum+6(2) '.' sy-datum+4(2) '.' sy-datum+0(4) into obj_descr.
w_document_data-obj_descr = obj_descr.

"Body of mail
CLEAR w_body_msg.
w_body_msg = 'Dear Sir/Madam,'.
APPEND w_body_msg TO i_body_msg.
CLEAR w_body_msg.

w_body_msg = ' '.
APPEND w_body_msg TO i_body_msg.
CLEAR w_body_msg.

CONCATENATE sy-datum+6(2) sy-datum+4(2) sy-
datum+0(4) INTO date_c SEPARATED BY '.'.
CONCATENATE 'You can check the Transfer stock and Partial transfe
r stock in the attached excel sheet' date_c '.' INTO w_body_msg SEP
ARATED BY ''.
APPEND w_body_msg TO i_body_msg.
CLEAR w_body_msg.

"Write Packing List for Body
DESCRIBE TABLE i_body_msg LINES g_tab_lines.
w_packing_list-head_start = 1.
w_packing_list-head_num = 0.
w_packing_list-body_start = 1.
w_packing_list-body_num = g_tab_lines.
w_packing_list-doc_type = 'RAW'.
APPEND w_packing_list TO i_packing_list.

```

```

CLEAR w_packing_list.

"Write Packing List for Attachment
w_packing_list-transf_bin = space.
w_packing_list-head_start = 1.
w_packing_list-head_num = 1.
w_packing_list-body_start = g_tab_lines + 1.
DESCRIBE TABLE i_attachment LINES w_packing_list-body_num.
w_packing_list-doc_type = 'XLS'.'XXL'.
w_packing_list-obj_descr = 'Transfer Partical transfer stock'.
w_packing_list-obj_name = 'XLS_ATTACHMENT'.
w_packing_list-doc_size = w_packing_list-body_num * 255.
APPEND w_packing_list TO i_packing_list.
CLEAR w_packing_list.

APPEND LINES OF i_attachment TO i_body_msg.
"Fill the document data and get size of attachment
w_document_data-obj_langu = sy-langu.
READ TABLE i_body_msg INTO w_body_msg INDEX g_tab_lines.
w_document_data-doc_size = ( g_tab_lines -
1 ) * 255 + strlen( w_body_msg ).

" Receivers List.
w_receivers-receiver = 'itproject21@nerolac.com'.
PERFORM get_mails.      "Get list of Mails

w_receivers-receiver = 'satishkumar.r@unisoftinfotech.com'.
PERFORM get_mails.      "Get list of Mails

w_receivers-receiver = 'satishkumarreddy.mn@gmail.com'.
PERFORM get_mails.      "Get list of Mails

"Function module to send mail to Recipients
CALL FUNCTION 'SO_NEW_DOCUMENT_ATT_SEND_API1'
EXPORTING
  document_data          = w_document_data
  put_in_outbox          = 'X'
  commit_work             = 'X'
IMPORTING
  sent_to_all              = g_sent_to_all
TABLES
  packing_list            = i_packing_list
  contents_txt             = i_body_msg
  receivers                = i_receivers
EXCEPTIONS
  too_many_receivers       = 1
  document_not_sent        = 2
  document_type_not_exist   = 3
  operation_no_authorization = 4
  parameter_error           = 5
  x_error                   = 6

```

```

enqueue_error          = 7
OTHERS                 = 8.

IF sy-subrc = 0 .
  MESSAGE i303(me) WITH 'Mail has been Successfully Sent.'.
ENDIF.
FORM get_mails .
  w_receivers-rec_type   = 'U'.  "Internet address
  w_receivers-com_type   = 'INT'.
  w_receivers-notif_del  = 'X'.
  w_receivers-notif_ndel = 'X'.
  APPEND w_receivers TO i_receivers .
ENDFORM.

```

" Here IT\_MAIL is the internal table which contains some data.

→ Based on given input fetch the data. In the output I want one checkbox. If select check box, then one field display in editable mode. If I changed the data, all data with the changes goes to next screen.

```

type-pools slis.
types: begin of ty_vbak,
  vbeln type vbak-vbeln,
  audat type vbak-audat,
  netwr type vbak-netwr,
  a(1) type c,
  KUNNR TYPE KNA1-KUNNR,
  field_style  TYPE lvc_t_styl, "FOR DISABLE
end of ty_vbak.

data: it_vbak type TABLE OF ty_vbak,
  wa_vbak type ty_vbak.
data: it_fcat type lvc_t_fcat,
  wa_fcat like line of it_fcat.
DATA: IT_EVENT TYPE SLIS_T_EVENT,
  WA_EVENT LIKE LINE OF IT_EVENT.
DATA: WA_LAYOUT TYPE lvc_s_layo.

tables vbak.
select-OPTIONS s_vbeln for vbak-vbeln.
* select vbeln audat netwr from vbak into table it_vbak where vbeln in s_vbeln.

```

"Here I'm using appending. I'm not using select query. Because in the internal table I've one field that is field\_style. The field\_style type is lvc\_t\_styl. We can't use select query for fetch data and keep in one internal table, which internal table have this data type. If you want select query, then declare one more internal table like this except that lvc\_t\_styl data type field. After that pass the data into this internal table which has lvc\_t\_styl data type.

```

WA_VBAK-VBELN = '101'.
WA_VBAK-AUDAT = 20170402.
WA_VBAK-NETWR = 2000.
APPEND WA_VBAK TO IT_VBAK.

WA_VBAK-VBELN = '102'.
WA_VBAK-AUDAT = 20170403.
WA_VBAK-NETWR = 5000.
APPEND WA_VBAK TO IT_VBAK.
DATA ls_stylerow TYPE lvc_s_styl .
DATA lt_styletab TYPE lvc_t_styl .
LOOP AT IT_VBAK INTO WA_VBAK.

ls_stylerow-fieldname = 'KUNNR' .
ls_stylerow-
style = cl_gui_alv_grid->mc_style_disabled.      " Disable input for t
he field initially.
"set field to disabled
APPEND ls_stylerow TO wa_VBAK-field_style.
MODIFY IT_VBAK FROM wa_VBAK.
ENDLOOP.

WA_LAYOUT-stylefname = 'FIELD_STYLE'.

wa_fcat-fieldname = 'VBELN'.
WA_FCAT-COLTEXT = 'Sales Doc Num'.
wa_fcat-col_pos = '1'.
APPEND wa_fcat to it_fcat.
clear wa_fcat.

wa_fcat-fieldname = 'AUDAT'.
WA_FCAT-COLTEXT = 'Doc Date'.
wa_fcat-col_pos = '2'.
APPEND wa_fcat to it_fcat.
clear wa_fcat.

wa_fcat-fieldname = 'NETWR'.
WA_FCAT-COLTEXT = 'Net price'.
wa_fcat-col_pos = '3'.
APPEND wa_fcat to it_fcat.
clear wa_fcat.

wa_fcat-fieldname = 'A'.
WA_FCAT-COLTEXT = 'checkbox'.
wa_fcat-col_pos = '4'.
wa_fcat-checkbox = 'X'.
WA_FCAT-HOTSPOT = 'X'.
WA_FCAT-EDIT = 'X'.
APPEND wa_fcat to it_fcat.
clear wa_fcat.

```

```

wa_fcat-fieldname = 'KUNNR'.
WA_FCAT-COLTEXT = 'Customer'.
wa_fcat-col_pos = '5'.
*wa_fcat-checkbox = 'X'.
*WA_FCAT-EDIT = 'X'.
APPEND wa_fcat to it_fcat.
clear wa_fcat.

CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY_LVC'
EXPORTING
*   I_INTERFACE_CHECK           =
*   I_BYPASSING_BUFFER          =
*   I_BUFFER_ACTIVE              =
I_CALLBACK_PROGRAM           = sy-repid
*   I_CALLBACK_PF_STATUS_SET    =
I_CALLBACK_USER_COMMAND        = 'USER_COMMAND'
*   I_CALLBACK_TOP_OF_PAGE      =
*   I_CALLBACK_HTML_TOP_OF_PAGE =
*   I_CALLBACK_HTML_END_OF_LIST =
*   I_STRUCTURE_NAME             =
*   I_BACKGROUND_ID              =
*   I_GRID_TITLE                 =
*   I_GRID_SETTINGS              =
IS_LAYOUT_LVC                  = wa_layout
IT_FIELDCAT_LVC                = IT_FCAT
*   IT_EXCLUDING                 =
*   IT_SPECIAL_GROUPS_LVC        =
*   IT_SORT_LVC                  =
*   IT_FILTER_LVC                =
*   IT_HYPERLINK                 =
*   IS_SEL_HIDE                  =
*   I_DEFAULT                     = 'X'
I_SAVE                         = 'X'
*   IS_VARIANT                   =
*   IT_EVENTS                     =
*   IT_EVENT_EXIT                 =
*   IS_PRINT_LVC                  =
*   IS_REPREP_ID_LVC              =
*   I_SCREEN_START_COLUMN         = 0
*   I_SCREEN_START_LINE           = 0
*   I_SCREEN_END_COLUMN           = 0
*   I_SCREEN_END_LINE             = 0
*   I_HTML_HEIGHT_TOP             =
*   I_HTML_HEIGHT_END             =
*   IT_ALV_GRAPHICS               =
*   IT_EXCEPT_QINFO_LVC          =
*   IR_SALV_FULLSCREEN_ADAPTER   =
*   IMPORTING
*     E_EXIT CAUSED BY CALLER    =
*     ES_EXIT CAUSED BY USER     =
TABLES

```

```

        t_outtab          = it_vbak
EXCEPTIONS
  PROGRAM_ERROR      = 1
  OTHERS             = 2

  .
IF sy-subrc <> 0.
* Implement suitable error handling here
ENDIF.

FORM user_command USING r_ucomm      LIKE sy-ucomm
               rs_selfield TYPE slis_selfield.
DATA ls_stylerow TYPE lvc_s_styl .
DATA lt_styletab TYPE lvc_t_styl .

if r_ucomm = '&IC1'." This is the fcode for hotspot link.

  read table IT_VBAK into wa_VBAK index rs_selfield-tabindex.
  if wa_VBAK-
A = ' '." The value for checkbox will not be stored with hotspot
link, thats why the check on the previous value.
  wa_VBAK-A = 'X'.
  ls_stylerow-fieldname = 'KUNNR'.
  ls_stylerow-style = cl_gui_alv_grid=>mc_style_enabled.
  "set field to disabled
  delete wa_VBAK-field_style where fieldname = 'KUNNR'.

" I want to enable KUNNR field only. That is why I used APPEND in
the below line. If you want to enable more than one field, then you
have to use INSERT statement in place of APPEND statement. Because
that structure is sorted internal table.

  APPEND ls_stylerow TO wa_VBAK-field_style.
  MODIFY IT_VBAK from wa_VBAK INDEX rs_selfield-tabindex.
else.
  clear wa_VBAK-A.
  ls_stylerow-fieldname = 'KUNNR'.
  ls_stylerow-style = cl_gui_alv_grid=>mc_style_disabled.
  "set field to disabled
  delete wa_VBAK-field_style where fieldname = 'KUNNR'.
  APPEND ls_stylerow TO wa_VBAK-field_style.
  MODIFY IT_VBAK from wa_VBAK INDEX rs_selfield-tabindex.
  MODIFY IT_VBAK from wa_VBAK INDEX rs_selfield-tabindex.
endif.

ENDIF.
rs_selfield-refresh = 'X'.
ENDFORM.

```

## Differences between GRID & LIST display: -

### Grid display

1. Grid display supports OOP's ALV.
2. Grid display slower
3. Edit and logo is possible in grid display
4. Blocked and hierarchical ALV is not possible in grid display
5. In the output  symbol indicates F1 help.
6. By using 'commentary write' function module only we can print the text on top or bottom event.
7. This is used to display the output in grid format.

### List display

1. List display does not supports OOP's ALV.
2. List display faster
3. These are not possible in list display.
4. These are possible in list display.
5. In list display  indicates number of records is displayed.
6. Either by using write or commentary writes function module we can print the text on top or bottom event.
7. This is used display the output in list format.

**Note:** - When ever we are working with ALV Reports in real time then we must pass layout WA.

Data wa\_layout type slis\_layout\_alv.

Wa\_layout-colwidth\_optimize = 'X'.

Wa\_layout-zebra = 'X'.

## Steps to identify the fields in field catalog: -

Execute SE37. Open any one of the ALV function module. Example is REUSE\_ALV\_LIST\_DISPLAY. Click on import tab against the field catalog, double click on associated type. Double click on their reference types. Double click on includes. Absorb the fields in field catalog.

**Note:** - When ever we are filling the field catalog manually, if you want to refer the database table, field description to print as a heading then we use following fields in field catalog.

Ref\_fieldname

Ref\_tablename.

EX: -

Wa\_fcat-fieldname = 'VBELN'.

Wa\_fcat-col\_pos = '1'.

Wa\_fcat-ref\_tablename = 'VBAK'.

Wa\_fcat-ref\_fieldname = 'VBELN'.

Append wa\_fcat to it\_fcat.

Clear wa\_fcat.

→ Create a program for AT SELECTION-SCREEN ON VALUE-REQUEST FOR event. In this program, take first parameter as Customer Number & second parameter as Sales Doc Num. First I'll give Customer number. If we click on F4 button for Sales Doc Num then the Sales Doc Num have to display which contains the above Customer Numbers only.

```
PARAMETERS: p_kunnr TYPE kunnr,
            p_vbeln TYPE vbeln.

AT SELECTION-SCREEN ON VALUE-REQUEST FOR p_vbeln.
DATA: it_sh TYPE TABLE OF dynpread,
      wa_sh TYPE dynpread,
      lt_ret TYPE TABLE OF ddshretval,
      ls_ret TYPE ddshretval.

wa_sh-fieldname = 'P_KUNNR'.
APPEND wa_sh TO it_sh.
CALL FUNCTION 'DYNP_VALUES_READ'
  EXPORTING
    dyname      = sy-repid
    dynumb      = sy-dynnr
  TABLES
    dynpfields = it_sh.

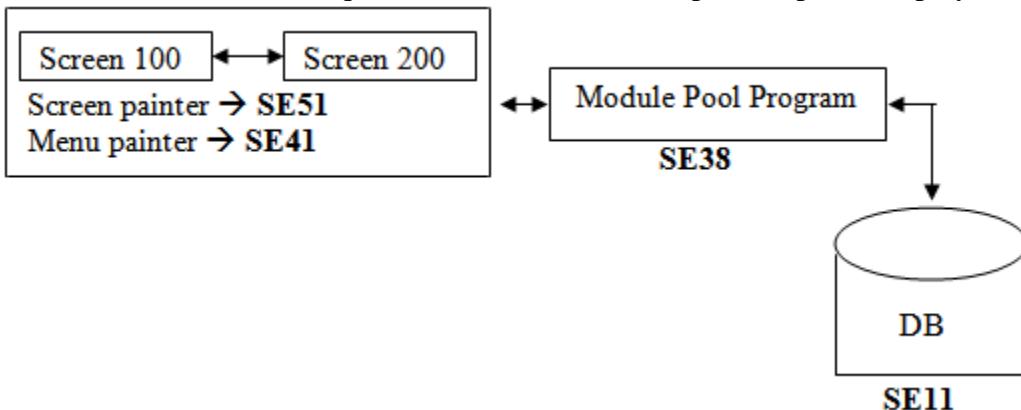
READ TABLE it_sh INTO wa_sh INDEX 1.
IF sy-subrc = 0.
  DATA(lv_kunnr) = wa_sh-fieldvalue.
ENDIF.

SELECT k~kunnr, k~name1, v~vbeln INTO TABLE @DATA(lt_vbak) FROM knal AS k INNER JOIN vbak AS v ON k~kunnr = v~kunnr WHERE k~kunnr = @lv_kunnr.
CALL FUNCTION 'F4IF_INT_TABLE_VALUE_REQUEST'
  EXPORTING
    retfield   = 'VBELN'
    value_org  = 'S'
  TABLES
    value_tab  = lt_vbak
    return_tab = lt_ret.

READ TABLE lt_ret INTO ls_ret INDEX 1.
IF sy-subrc = 0.
  p_vbeln = ls_ret-fieldval.
ENDIF.
```

## **Transaction / Dialog pool programming / Module Pool Programming: -**

A transaction is the collection of sequential screens which accept the input & display the output.



It's also called as Dialog Pool Programming since we have interaction between screens. It's also called as Module Pool Programming because the flow logic of each screen acts as a module. So it's a pool of module.

**Note:** - A transaction code contains either executable program or module pool program.

### **Differences between executable program, module pool program: -**

#### **Executable program**

1. We can execute the executable program either through transaction code or directly.
2. Type of the executable program is '1'.
3. All the standard reports are executable programs.

**Ex:** - ME2L, ME2K, ME2M

#### **Module Pool Program**

1. We execute the module pool program only through transaction.
2. Type of the module pool program is 'M'.
3. All the standard transactions are module pool programs.

**Ex:** - XK01, XD01

### **Steps to create transaction code for executable program: -**

Execute 'SE93'. Provide the transaction code. Click on create. Provide short text. Select the radio button program and selection screen. Enter. Provide the program name. Select the GUI check boxes. Save. Check.

**Note:** - In the real time when ever we develop a new object other than enhancement then we must create transaction code.

### **Steps to work with module pool program: -**

1. Create a module pool program (in SE38) & implement the logic.
2. Design the required screens (in SE51) & attached to the program.
3. Design the required menus (in SE41) & attached to screen.
4. Design the database tables (in SE11) & as per client requirement.
5. Create the transaction code (in SE93) & run the module pool program.

These 5 steps, we can work with single transaction. I.e. **SE80** (Object Navigator).

### **Steps to create the module pool program: -**

#### **Step 1: -**

Execute **SE80**. Click on 'Edit object' in the application tool bar. Click on program tab. Provide program name. Click on create. Remove the check box TOP INCL. Click on enter. Select the type is module pool. Enter. Click on save, local object. Select the program name. Provide program name in left panel. Enter, save.

## **Step 2: -**

Execute **SE38**. Provide the program name. Click on create. Provide title. Select the type is module pool. Click on save, local object. Click on display object list in the application tool bar.

### **Working with screen painter: -**

Screen painter is a tool which contains both the graphical as well as alpha numeric mode. The transaction code for screen painter is **SE51**.

### **Components of screen painter: -**

1. Attributes
2. Layout
3. Element list
4. Flow logic editor

#### **Attributes: -**

Attributes specify the type of the screen whether it's a normal sub screen or model dialog box.

#### **Layout: -**

Layout is the collection of screen elements. I.e. check box, radio button, input output field, push button, table control, tab strip, etc.

#### **Element list: -**

Element list contains the screen elements which are designed on the screen & their data types & lengths.

#### **Flow logic editor: -**

Flow logic editor contains the logic related to the screen.

#### **Events in flow logic editor: -**

1. PBO (Process Before Output)
2. PAI (Process After Input)
3. POV (Process On Value-request)
4. POH (Process On Help-request)

#### **PBO: -**

It's an event which is triggered before display the screen.

#### **ADV: -** This is used to provide the default values to the screen.

#### **PAI: -**

It's an event which is triggered after provide the input to the screen.

#### **ADV: -** This is used to implement the logic.

#### **POV: -**

It's an event which is triggered at the time of user clicks on F4 button.

#### **ADV: -** This is used to provide the list of possible values to the input variable.

#### **POH: -**

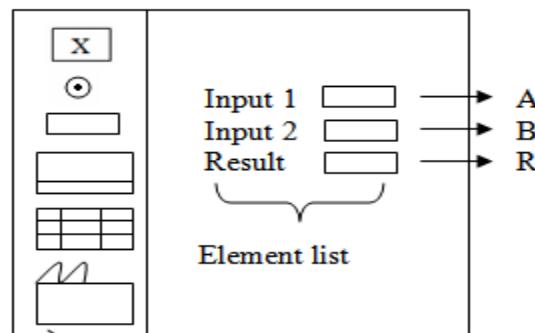
It's an event which is triggered at the time of user clicks on F1 button.

#### **ADV: -** This is used to provide the help document to the input variable.

**Note:** - The communication between flow logic editor to ABAP editor is always through screen elements. I.e. each element in the screen we must declare one equality declaration in ABAP editor.

### **Steps to work with module pool program as per ABAPer point of view: -**

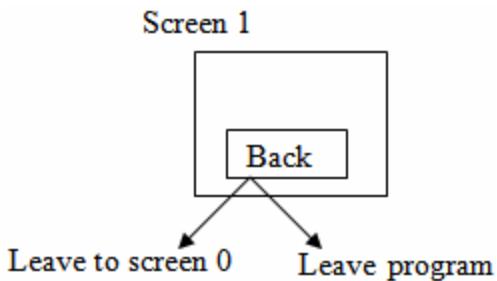
1. Create the module pool program
2. Design the required screens
3. Maintain the equality declaration in ABAP editor
4. Maintain or implement the PBO and PAI logics of each screen
5. Create the transaction code to run the program



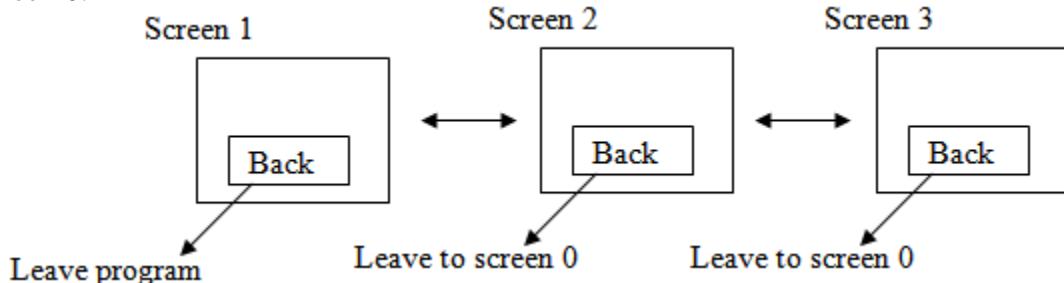
→ Design the screen as shown in the below.

After the user provide the input1 and input2 click on addition button then we display the output in the result field. If the user clicks on back button then we go to program.

**Note:** - If we are working with only one screen then the back button functionality of the screen is either leave to screen 0 or leave program.



If we are working with more than one screen then the back button functionality of the first screen is leave program, from second screen onwards leave to screen 0.



### Design the screen: -

Select the program in left panel. Right click on it. Create → screen. Provide screen number (4 digit). Enter. Provide short description. Click on save. Click on layout in the application tool bar. Design the screen as per client requirement.

### Screen designing: -

Select the text field screen element. Draw it. Double click on it. Provide the name, text. Select the input output field screen element. Draw it. Double click on it. Provide the name. Identify the data type & length. Repeat the same steps for all other fields. Select the push button screen element. Draw it. Double click on it. Provide the name, text, function code. Repeat the same steps for back button. Select the box screen element. Draw it.

### Steps to activate the program: -

Double click on program in the left panel. Right click → Activate. Enter.

### Steps to create the transaction code: -

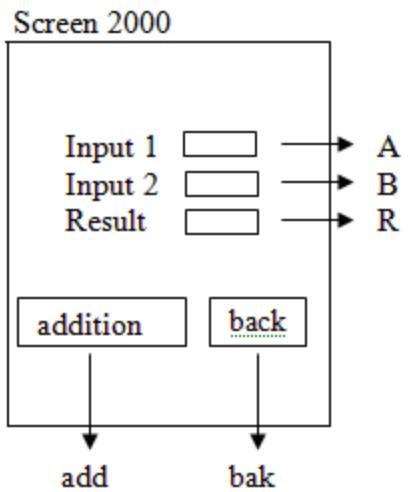
Select the program in the left panel. Right click, create → transaction. Provide the transaction code, short description. Enter. Provide the program name, screen number. Select the GUI checkbox. Save.

### Steps to execute the program: -

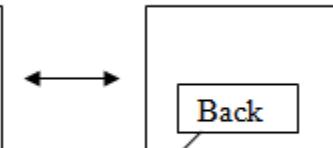
Select the transaction code in the left panel. Right click → execute → direct processing.

### Programming: -

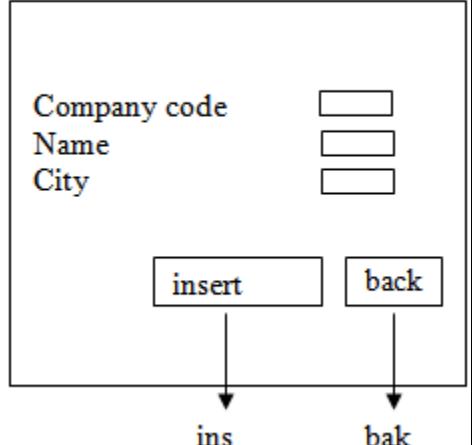
```
DATA A(10) TYPE C.  
DATA B(10) TYPE C.  
DATA C(10) TYPE C.  
MODULE USER_COMMAND_2000 INPUT.  
IF SY-UCOMM = 'ADD'.
```



Screen 3



Screen 1000



```

C = A + B.
WRITE C.
ELSEIF SY-UCOMM = 'BACK'.
LEAVE PROGRAM.
ENDIF.
ENDMODULE.

```

Flow logic of 2000 screen: -

```

PROCESS BEFORE OUTPUT.
* MODULE STATUS_2000.
PROCESS AFTER INPUT.
MODULE USER_COMMAND_2000.

```

→ Design the screen as shown in the below. After the user provide the company code, name & city, click on insert button then we insert company details into T001 data base table if the user clicks on back button then we come back to program.

Steps to design the screen: -

Click on layout in the application tool bar. If the fields are coming from any data base table or work area then click on dictionary / program field icon (F6) in the standard tool bar. Provide the table name. Enter. It displays the all fields. Select our required fields. Click on OK & place on the screen & design the rest of the buttons.

Programming: -

```

TABLES T001.
MODULE USER_COMMAND_2000 INPUT.
IF SY-UCOMM = 'INS'.
  INSERT T001 FROM T001.
  IF SY-SUBRC = 0.
    MESSAGE S000(ZMESSAGE1)
    WITH 'INSERTED SUCCESSFULLY'.
  ELSE.
    MESSAGE E000(ZMESSAGE1) WITH 'NOT INSERTED'.
  ENDIF.
ELSEIF SY-UCOMM = 'BACK'.
  LEAVE PROGRAM.
ENDIF.
ENDMODULE.

```

→ Design the screen as shown in the below. After the user provide the company code & click on display button then we display the customers under company details (BUKRS, KUNNR, AKONT) like ordinary report.

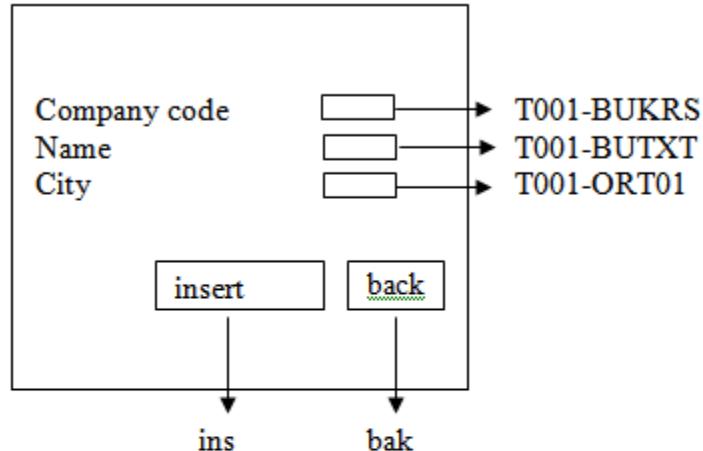
*Note: -* If you want to display the output like ordinary report then we must place leave to list processing before display the output.

```

DATA A TYPE KNB1-BUKRS.
TYPES: BEGIN OF TY_KNB1,
        BUKRS TYPE KNB1-BUKRS,
        KUNNR TYPE KNB1-KUNNR,
        AKONT TYPE KNB1-AKONT,

```

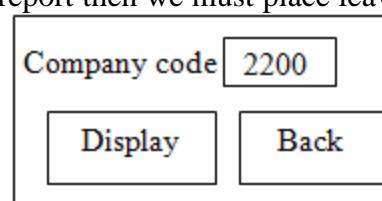
Screen 1000



Tables T001.

T001 BUKRS BUTXT ORT01 LAND1 -----  

--	--	--	--



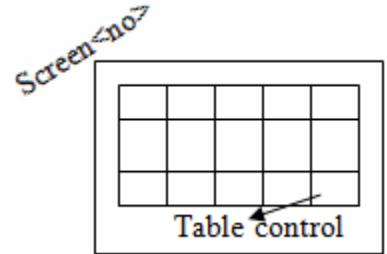
```

        END OF TY_KNB1.
DATA: WA_KNB1 TYPE TY_KNB1,
      IT_KNB1 TYPE TABLE OF TY_KNB1.
MODULE USER_COMMAND_2000 INPUT.
  IF SY-UCOMM = 'DIS'.
    SELECT BUKRS KUNNR AKONT FROM KNB1 INTO TABLE IT_KNB1 WHERE BUKRS = A.
    LEAVE TO LIST-PROCESSING.
  LOOP AT IT_KNB1 INTO WA_KNB1.
    WRITE:/ WA_KNB1-BUKRS, WA_KNB1-KUNNR, WA_KNB1-AKONT.
  ENDLOOP.
ELSEIF SY-UCOMM = 'BACK'.
  LEAVE PROGRAM.
ENDIF.
ENDMODULE.

```

### Working with table control: -

Table control is used to display the multiple records in a tabular format.



### Syntax of declaring the table control in ABAP editor: -

Controls <table control name> type table view using screen <screen no>.

#### → Design the screen as shown in the below.

After the user provide the purchasing doc number in the 1000's screen & click on display button then we display the purchasing document item details in a tabular format in 2000's screen.

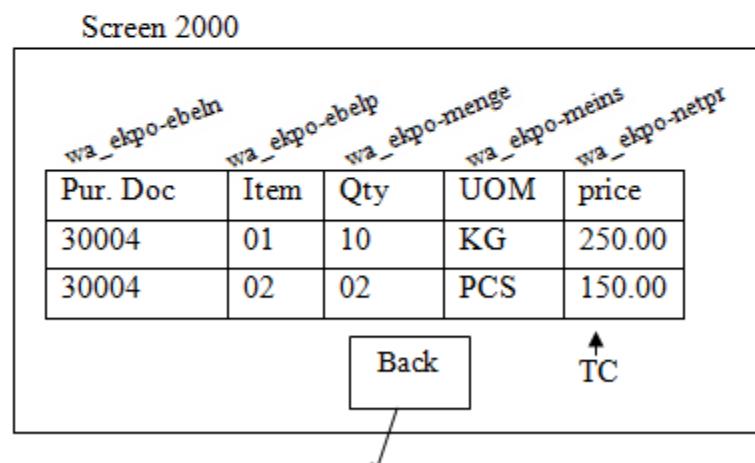
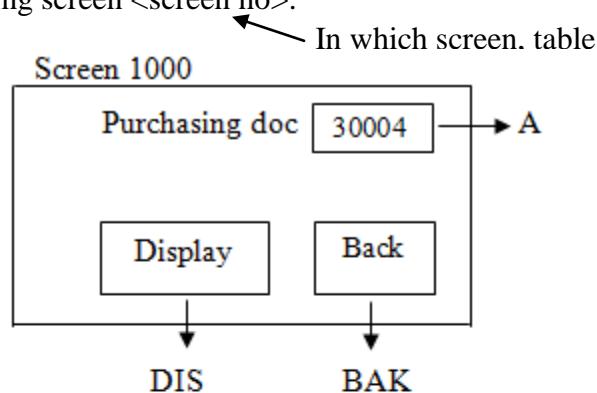
#### Steps to design table control: -

Select the table control screen element & draw it. Double click on it. Provide the name. Select vertical, horizontal check boxes, multiple radio buttons. Click on dictionary / program fields button (F6). Provide the table name. Select the required fields. Click on OK & place it top right. Double click on each input field. Provide the name as wa\_ekpo-ebeln, wa\_ekpo-ebelp, wa\_ekpo-meins, and wa\_ekpo-netpr like this. Design the back button. Save.

#### Syntax of transferring the data from internal table to table control in the flow logic editor: -

Loop at <IT> into <WA> with control <table control> cursor <table control> - current\_line.

Module <module name>. ←  
Endloop.



This is not required, if we are maintain table control field names & work area field names are same.

**Note:** - When ever we are working with loop & endloop in any one of the event then we must declare a dummy loop & endloop in another event.

#### CONTROLS TBC TYPE TABLEVIEW USING SCREEN 2000.

DATA A TYPE EKPO-EBELN.

```
DATA: BEGIN OF WA_EKPO,
      EBELN TYPE EKPO-EBELN,
      EBELP TYPE EKPO-EBELP,
      MENGE TYPE EKPO-MENGE,
      MEINS TYPE EKPO-MEINS,
      NETPR TYPE EKPO-NETPR,
      END OF WA_EKPO.
```

```
DATA IT_EKPO LIKE TABLE OF WA_EKPO.
MODULE USER_COMMAND_1000 INPUT.
```

```
IF SY-UCOMM = 'DIS'.
```

```
  SELECT EBELN EBELP MENGE MEINS NETPR FROM EKPO INTO TABLE IT_EKPO
  WHERE EBELN = A.
```

```
  CALL SCREEN 2000.
```

```
ELSEIF SY-UCOMM = 'BACK'.
```

```
  LEAVE PROGRAM.
```

```
ENDIF.
```

```
ENDMODULE.
```

```
MODULE USER_COMMAND_2000 INPUT.
```

```
IF SY-UCOMM = 'BAK'.
```

```
  LEAVE TO SCREEN 0.
```

```
ENDIF.
```

```
ENDMODULE.
```

#### Flow logic of 1000 screen:-

```
PROCESS BEFORE OUTPUT.
```

```
* MODULE STATUS_1000.
```

```
PROCESS AFTER INPUT.
```

```
  MODULE USER_COMMAND_1000.
```

#### Flow logic of 2000 screen: -

```
PROCESS BEFORE OUTPUT.
```

```
* MODULE STATUS_2000.
```

```
LOOP AT IT_EKPO INTO WA_EKPO WITH CONTROL TBC CURSOR TBC-CURRENT_LINE.
```

```
ENDLOOP.
```

```
PROCESS AFTER INPUT.
```

```
LOOP.
```

```
ENDLOOP.
```

```
  MODULE USER_COMMAND_2000.
```

In the work area field names, table control field names are different suppose V1, V2, V3, V4, V5.

#### PBO of 2000: -

Loop at it\_ekpo into wa\_ekpo with control TC cursor TC-current\_line.

Module xyz.

Endloop.

#### Program: -

Module xyz output.

V1 = wa\_ekpo-ebeln.

V2 = wa\_ekpo-ebelp.

V3 = wa\_ekpo-menge.

Pur. Doc	Item	Qty	UOM	price
30004	01	10	KG	250.00

Pur. Doc	Item	Qty	UOM	price
30004	01	10	KG	250.00
30004	02	02	PCS	150.00

V4 = wa\_ekpo-meins.

V5 = wa\_ekpo-netpr.

Endmodule.

→ Design the screen as shown in the below.

After the user provide the company code & city on display button then we display the customers under company details in a tabular format in 2000 screen. If the user clicks on download button in 2000 screen then we download the selected records into presentation server.

**Note:** - In the PAI dummy loop & endloop acts like loop at table control.

#### Steps to provide the check box to the table control:-

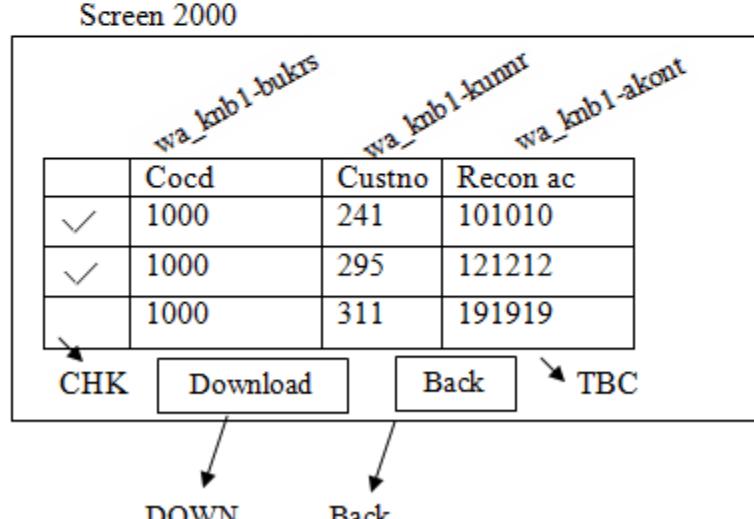
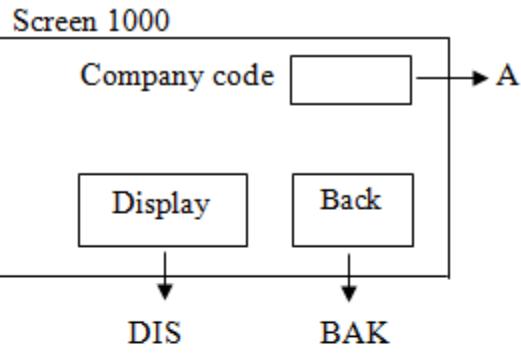
Double click on table control. Provide the name. provide the vertical, horizontal check boxes, multiple radio buttons. Select the w/SelColumn check box & provide the cheek box name.

```
CONTROLS TBC TYPE TABLEVIEW  
USING SCREEN 2000.  
DATA CHK.  
DATA A TYPE KNB1-BUKRS.  
DATA: BEGIN OF WA_KNB1,  
      BUKRS TYPE KNB1-BUKRS,  
      KUNNR TYPE KNB1-KUNNR,  
      AKONT TYPE KNB1-AKONT,  
      END OF WA_KNB1.  
DATA IT_KNB1 LIKE TABLE OF WA_KNB1.  
DATA IT LIKE IT_KNB1.  
MODULE USER_COMMAND_1000 INPUT.
```

```
  IF SY-UCOMM = 'DIS'.  
    SELECT BUKRS KUNNR AKONT FROM KNB1 INTO TABLE IT_KNB1 WHERE  
    BUKRS = A.  
    CALL SCREEN 2000.  
  ELSEIF SY-UCOMM = 'BACK'.  
    LEAVE PROGRAM.  
  ENDIF.
```

ENDMODULE.

```
MODULE USER_COMMAND_2000 INPUT.  
  IF SY-UCOMM = 'DOWN'.  
    CALL FUNCTION 'DOWNLOAD'  
    EXPORTING  
      FILETYPE = 'DAT'  
    TABLES  
      DATA_TAB = IT.
```



it

	Cocd	Custno	Recon ac
	1000	241	101010
	1000	311	191919

```

ELSEIF SY-UCOMM = 'BAK'.
    LEAVE TO SCREEN 0.
ENDIF.
ENDMODULE.

```

```

MODULE XYZ INPUT.
    IF CHK = 'X'.
        APPEND WA_KNB1 TO IT.
    ENDIF.
ENDMODULE.

```

#### Flow logic of 2000 screen: -

```

PROCESS BEFORE OUTPUT.
* MODULE STATUS_2000.
LOOP AT IT_KNB1 INTO WA_KNB1 WITH CONTROL TBC_CURSOR TBC-
CURRENT_LINE.
ENDLOOP.

```

PROCESS AFTER INPUT.

LOOP.

MODULE XYZ.

ENDLOOP.

MODULE USER\_COMMAND\_2000.

#### Steps to provide the mask to the particular field: -

Double click on the field. In the pop of window click on program tab in attributes block. Select the input field is 'NOT POSSIBLE'.

#### Steps to provide the input field is mandatory: -

Double click on the field. In the pop of window click on program tab in attributes block. Select the input field is 'REQUIRED'.

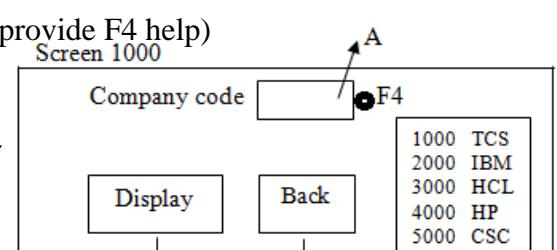
#### Working with POV: -

POV is an event which is triggered at the time of user clicks on F4 button. This is used to provide the list of possible values.

**Note:** – F4IF\_INT\_TABLE\_VALUE\_REQUEST is function module which is used to provide the F4 help to the input variable. The input for the above function module is

#### F4IF\_INT\_TABLE\_VALUE\_REQUEST

- <DATA INTERNAL TABLE> (which data we want to provide F4 help)
- <Return field in the data internal table>
- <Field name> (for which field, we want to provide F4)
- <Screen number>
- <Program name>
- <Value\_ORG> = 'S'



→ Design the screen as shown in the below.

If the user click on F4 button then we display the all company codes & company names as F4. After user select the company code & click on display button then we display the customers under company details (BUKRS, KUNNR, AKONT) like an ordinary report. If the user click on BACK button then we come back to program.

### Steps of module in POV event: -

Field <field name> module <module name>.

For which field we want to provide F4.

```

DATA A TYPE T001-BUKRS.
DATA: BEGIN OF WA_KNB1,
      BUKRS TYPE KNB1-BUKRS,
      KUNNR TYPE KNB1-KUNNR,
      AKONT TYPE KNB1-AKONT,
      END OF WA_KNB1.
DATA IT_KNB1 LIKE TABLE OF WA_KNB1.
DATA: BEGIN OF WA_T001,
      BUKRS TYPE T001-BUKRS,
      BUTXT TYPE T001-BUTXT,
      END OF WA_T001.
DATA IT_T001 LIKE TABLE OF WA_T001.
MODULE USER_COMMAND_1000 INPUT.
  IF SY-UCOMM = 'DIS'.
    SELECT BUKRS KUNNR AKONT FROM KNB1 INTO TABLE IT_KNB1 WHERE
BUKRS = A.
    LEAVE TO LIST-PROCESSING.
    LOOP AT IT_KNB1 INTO WA_KNB1.
      WRITE:/ WA_KNB1-BUKRS, WA_KNB1-KUNNR, WA_KNB1-AKONT.
    ENDLOOP.
  ELSEIF SY-UCOMM = 'BACK'.
    LEAVE PROGRAM.
  ENDIF.
ENDMODULE.
MODULE XYZ INPUT.
  SELECT BUKRS BUTXT FROM T001 INTO TABLE IT_T001.
  CALL FUNCTION 'F4IF_INT_TABLE_VALUE_REQUEST'
    EXPORTING
      RETFIELD      = 'BUKRS'
      DYNPPROG      = 'ZMPPR6'
      DYNPNR        = '1000'
      DYNPROFIELD   = 'A'
      VALUE_ORG     = 'S'
    TABLES
      VALUE_TAB    = IT_T001.
ENDMODULE.

```

### Flow logic of 1111 screen: -

PROCESS BEFORE OUTPUT.

\* MODULE STATUS\_1000.

PROCESS AFTER INPUT.

MODULE USER\_COMMAND\_1000.

PROCESS ON VALUE-REQUEST.

FIELD A MODULE XYZ.

### Working with menu painter: -

It's a tool to design the user interface to the program the transaction code for menu painter is 'SE41'.

## GUI Components

GUI Title                          2000    3000

### GUI Status

- Menu bar
- Standard tool bar
- Application tool bar
- Function keys

**Note:** - The PBO of each screen contains by default GUI title & GUI status.

### Working with sub screen areas:-

Sub screen area must be placed in normal screen only. Each sub screen area can call only one sub screen at a time.

→ Design the screen as shown in the below.

### Syntax of calling the sub screen from sub screen area:-

Call subscreen <subscreen area name> including <program name> ‘<screen number>’.

**Ex:-**

Call subscreen SA1 including SY-REPID ‘0100’.

**Note:** - SY-REPID is the system variable which contains the current program name.

### Steps to design the back button (working with GUI):-

Remove the comment of module in the PBO event. Double click on module name. By default it contains GUI status & GUI title. Uncomment the PF-Status. Provide the status name. Double click on it. Provide the short description. Enter. Expand the function keys. Enable the back button. Save, check, activate the GUI. Click on back.

**Note:** - If you want to navigate to subscreens from normal screen then we must call the subscreen area names in the PAI of normal screen.

**TABLES: T001, KNA1.**

**MODULE STATUS\_1000 OUTPUT.**

    SET PF-STATUS 'STAT'.

**ENDMODULE.**

**MODULE USER\_COMMAND\_1000 INPUT.**

    IF SY-UCOMM = 'BACK'.

        LEAVE TO SCREEN 0.

    ENDIF.

**ENDMODULE.**

**MODULE USER\_COMMAND\_2000 INPUT.**

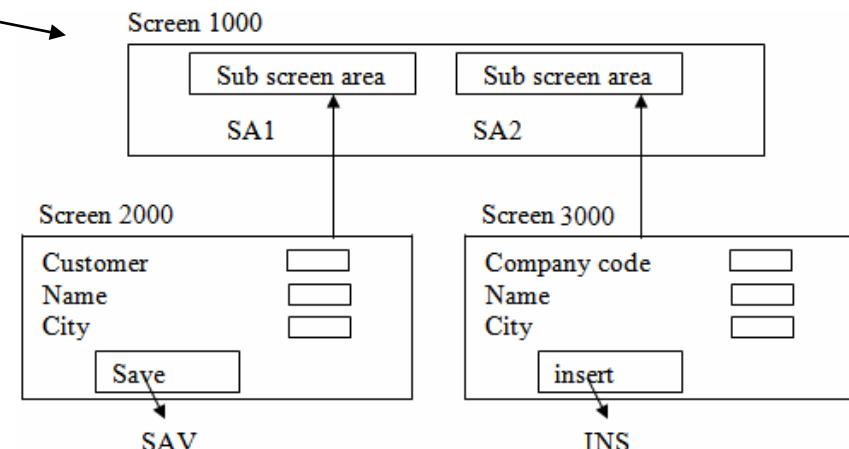
    IF SY-UCOMM = 'SAV'.

        INSERT KNA1 FROM KNA1.

        IF SY-SUBRC = 0.

            MESSAGE S000 (ZMESSAGE1) WITH 'SAVED SUCCESSFULLY'.

    ELSE.



```

MESSAGE E000(ZMESSAGE1) WITH 'NOT SAVED'.
ENDIF.
ENDIF.
ENDMODULE.

MODULE USER_COMMAND_3000 INPUT.
IF SY-UCOMM = 'INS'.
INSERT T001 FROM T001.
IF SY-SUBRC = 0.
MESSAGE S000(ZMESSAGE1) WITH 'INSERTED SUCCESSFULLY'.
ELSE.
MESSAGE E000(ZMESSAGE1) WITH 'NOT INSERTED'.
ENDIF.
ENDIF.
ENDIF.
ENDMODULE.

```

### Flow logic of 1000 screen.

```

PROCESS BEFORE OUTPUT.
MODULE STATUS_1000.
CALL SUBSCREEN: SA1 INCLUDING SY-REPID '2000',
SA2 INCLUDING SY-REPID '3000'.

```

PROCESS AFTER INPUT.

```

MODULE USER_COMMAND_1000.
CALL SUBSCREEN: SA1, SA2.

```

### Flow logic of 2000 screen: -

PROCESS BEFORE OUTPUT.

\* MODULE STATUS\_2000.

PROCESS AFTER INPUT.

MODULE USER\_COMMAND\_2000.

### Flow logic of 3000 screen: -

PROCESS BEFORE OUTPUT.

\* MODULE STATUS\_3000.

PROCESS AFTER INPUT.

MODULE USER\_COMMAND\_3000.

### Working with tab strip: -

- Strip of tabs.
- Each tab must contain at least one sub screen area.
- Each sub screen area can call only one sub screen at a time.
- By default tab strip contains 2 tabs.
- Only one tab is always activated.

→ Design the screen as shown in the below.

### Syntax of declaring the tab strip in ABAP editor:-

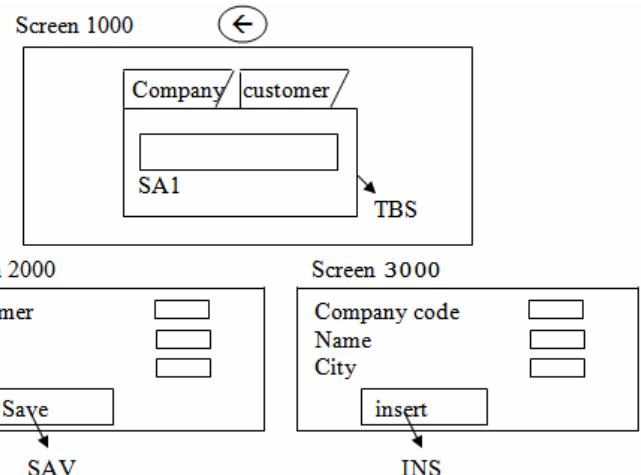
Controls <table strip name> type tabstrip.

**Note:** – When ever we are working with tabstrip then it's better to maintain tab name & function code of the tab name same.

### Syntax of activate the tab: -

<tabstrip name> - Active tab = '<tab name>'.

### Steps to design the strip control: -



Select the tabstrip control screen element. Draw it. Double click on it. Provide the name (TBS). Provide number of tab titles. Double click on tab1. Provide the name of tab1 (TAB1), provide the text function code as tab1(Company). Select the subscreen area screen element. Draw it. Double click on it. Provide the name(SA1). Repeat the steps for all the tabs.

```
CONTROLS TBS TYPE TABSTRIP.  
TABLES: T001, KNA1.  
MODULE USER_COMMAND_1000 INPUT.  
  IF SY-UCOMM = 'BACK'.  
    LEAVE TO SCREEN 0.  
  ELSEIF SY-UCOMM = 'TAB1'.  
    TBS-ACTIVETAB = 'TAB1'.  
  ELSEIF SY-UCOMM = 'TAB2'.  
    TBS-ACTIVETAB = 'TAB2'.  
  ENDIF.  
ENDMODULE.  
MODULE STATUS_1000 OUTPUT.  
  SET PF-STATUS 'STAT'.  
ENDMODULE.  
MODULE USER_COMMAND_0100 INPUT.  
  IF SY-UCOMM = 'INS'.  
    INSERT T001 FROM T001.  
  ENDIF.  
ENDMODULE.  
MODULE USER_COMMAND_0200 INPUT.  
  IF SY-UCOMM = 'SAV'.  
    INSERT KNA1 FROM KNA1.  
  ENDIF.  
ENDMODULE.  
Flow logic of 1000 screen: -  
PROCESS BEFORE OUTPUT.  
  MODULE STATUS_1000.  
    CALL SUBSCREEN: SA1 INCLUDING SY-REPID '0100',  
                  SA2 INCLUDING SY-REPID '0200'.  
PROCESS AFTER INPUT.  
  MODULE USER_COMMAND_1000.  
    CALL SUBSCREEN: SA1, SA2.
```

#### **Working with drop down list: -**

**Note:** - ‘VRM\_SET\_VALUES’ is the function module which is used to provide the drop down list to the input variable. The input for the above function module is ID (fieldname, for which field we want to provide dropdown), TEXT (Data Internal Table).

**Note:** - In VRM, we have one type that is VRM\_VALUES which contains the above two fields. So we simply declare our internal table by referring VRM\_VALUES.

→ Design the screen as shown in the below.

**Note:** - At the time of designing the screen double click on the drop down field & select the drop down as list box with key in the pop up window.

**Note:** - T005T is the standard data base table which contains all the country keys & country names.

TYPE-POOLS VRM.

TABLES T001.

```
TYPES: BEGIN OF TY_T005T,
      LAND1 TYPE T005T-LAND1,
      LANDX TYPE T005T-LANDX,
      END OF TY_T005T.
```

```
DATA: WA_T005T TYPE TY_T005T,
      IT_T005T TYPE TABLE OF TY_T005T.
```

MODULE STATUS\_2000 OUTPUT.

```
SET PF-STATUS 'STAT'.
DATA: IT TYPE VRM_VALUES,
      WA LIKE LINE OF IT.
```

```
*WA-KEY = 'IN'.
*WA-TEXT = 'INDIA'.
*APPEND WA TO IT.
*WA-KEY = 'CN'.
*WA-TEXT = 'CHINA'.
*APPEND WA TO IT.
*WA-KEY = 'KW'.
*WA-TEXT = 'KUWAIT'.
* APPEND WA TO IT.
```

```
SELECT LAND1 LANDX FROM T005T INTO TABLE IT_T005T WHERE SPRAS = SY-LANGU.
```

LOOP AT IT\_T005T INTO WA\_T005T.

```
WA-KEY = WA_T005T-LAND1.
WA-TEXT = WA_T005T-LANDX.
APPEND WA TO IT.
```

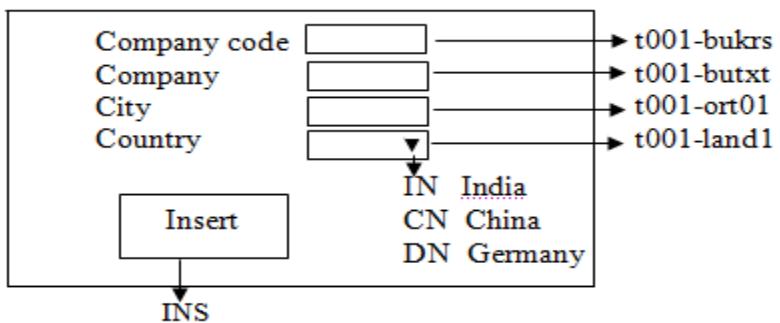
ENDLOOP.

```
CALL FUNCTION 'VRM_SET_VALUES'
  EXPORTING
    ID      = 'T001-LAND1'
    VALUES  = IT.
REFRESH IT.
```

ENDMODULE.

```
MODULE USER_COMMAND_2000 INPUT.
IF SY-UCOMM = 'BACK'.
  LEAVE PROGRAM.
ELSEIF SY-UCOMM = 'INS'.
```

Screen 1000



it KEY TEXT

IN	INDIA
CN	CHINA
DC	GERMANY

wa KEY TEXT

<input type="text"/>	<input type="text"/>
----------------------	----------------------

Wa\_t005t

LAND1 LANDX

A	Australia
IN	India

```

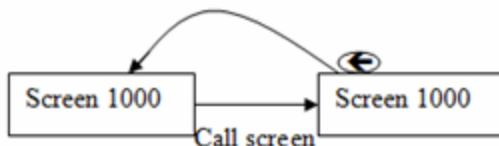
INSERT T001.
IF SY-SUBRC = 0.
  MESSAGE S000 (ZMESSAGE1) WITH 'INSERTED SUCCESSFULLY'.
ELSE.
  MESSAGE E000 (ZMESSAGE1) WITH 'NOT INSERTED'.
ENDIF.
ENDIF.
ENDMODULE.

```

### Differences between call screen & set screen

#### **Call screen**

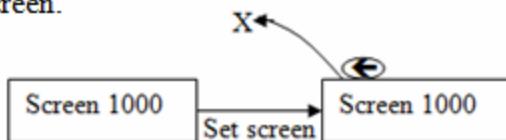
It calls the new screen in the new screen, if the user click on back button then it come back to previous screen.



Call screen

#### **Set screen**

It calls the new screen. The new screen acts like an industrial screen. If the user clicks on back button in the new screen, then it never come back to previous screen.



### **Ranges: -**

Ranges is the key word which accepts single value, multiple values, single range, multiple ranges.

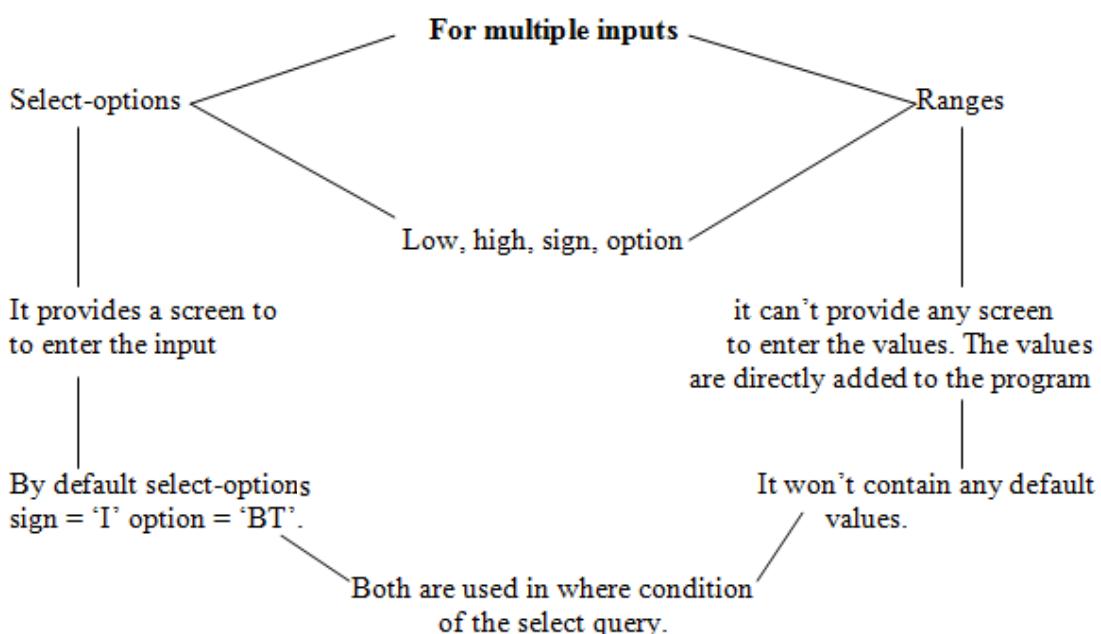
### Syntax: -

Ranges <name of the ranges> for <variable name>.

### Ex: -

Data V1 type t001-bukrs.

Ranges r\_bukrs for v1.



## → Design the screen as shown in the below.

After the user provide the company codes & click on display button then we display the vendors under company details by using ordinary report.

```
Data: A type t001-bukrs,  
      B type t001-bukrs.  
Data v1 like t001-bukrs.  
Ranges r_bukrs for v1.  
Data: begin of wa_lfb1,  
      Bukrs type lfb1-bukrs,  
      Lifnr type lfb1-lifnr,  
      Akont type lfb1-akont,  
      End of wa_lfb1.  
Data it_lfb1 like table of wa_lfb1.
```

```
MODULE USER_COMMAND_1000 INPUT.  
  if sy-ucomm = 'DIS'.  
    r_bukrs-low = A.  
    r_bukrs-high = B.  
    r_bukrs-sign = 'I'.  
    if b is not initial.  
      r_bukrs-option = 'BT'.  
    else.  
      r_bukrs-option = 'EQ'.  
    endif.  
    append r_bukrs.  
    select bukrs lifnr akont from lfb1 into table it_lfb1 where bukrs  
in r_bukrs.  
    leave to list-processing.  
    loop at it_lfb1 into wa_lfb1.  
      write:/ wa_lfb1-bukrs, wa_lfb1-lifnr, wa_lfb1-akont.  
    endloop.  
  elseif sy-ucomm = 'BACK'.  
    leave program.  
  endif.  
ENDMODULE.
```

### Working with validations: -

Validations always performed in the PAI event of any screen. There are three types of validations.

1. System validation
2. Validation of flow logic editor
3. validation of ABAP editor

### System validations: -

Whenever we are working with date & time format if you entered invalid date & time format then the system automatically validate & provide an error message.

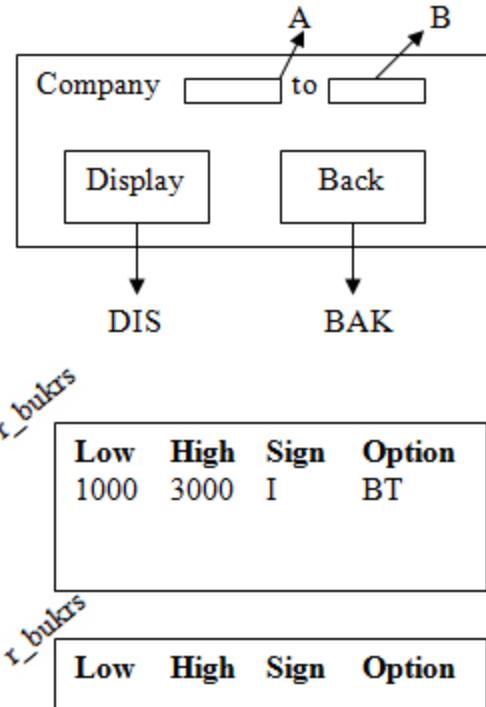
### Validation of flow logic editor: -

The validation logic is maintained in flow logic editor of the screen.

### Syntax: -

Field <field name> values ( '<value1>', '<value2>' ... ).

Which field we want to validate.



### Ex:-

Field A values ( '1000', '2000', '2200' ).

### **Validation of ABAP editor: -**

The validation logic is maintained in ABAP editor. This is similar as at selection-screen event logic.

### **Syntax in flow logic editor: -**

Field <field name> module <module name>

For which field we want to validate.

→ Design the screen as shown in the below.

### **CHAIN ---- ENDCHAIN: -**

This is used to validate the related input fields. If you aren't using chain & endchain, if you pass invalid input then it display the error field enabled mode & rest of the fields are disable mode. If we use chain & endchain if you get the error then all the fields are displayed in enable mode.

Whenever we're working with validations, if you get the error then the back button functionality isn't work. If you want to enable the back button functionality then we must provide function type as 'E' at the time of providing the function key & also provide at exit command for the module name of the back button.

In the PAI at exit command module is executed first & later rest of the modules are executed.

### Ex: -

PAI of 1000.

Module ABC.

Module XYZ.

First execute → Module XYZ1 at exit-command.

Module XYZ2

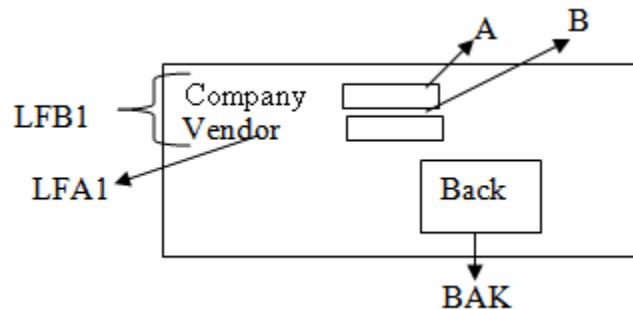
```

PROGRAM ZMPP99910.
DATA A TYPE LFB1-BUKRS.
DATA B TYPE LFB1-LIFNR.
DATA V1 TYPE LFA1-LIFNR.
DATA V2 TYPE LFB1-BUKRS.

MODULE USER_COMMAND_2000 INPUT.
  IF SY-UCOMM = 'BACK'.
    LEAVE TO SCREEN 0.
  ENDIF.
ENDMODULE.

MODULE ABC INPUT.
  SELECT SINGLE LIFNR FROM LFB1 INTO V1 WHERE LIFNR = B.
  IF SY-SUBRC <> 0.
    MESSAGE E000(ZMESSAGE1) WITH 'INVALID VENDOR'.
  ENDIF.
  SELECT SINGLE BUKRS LIFNR FROM LFB1 INTO (V2 , V1) WHERE LIFNR =
B AND BUKRS = A.

```



```

IF SY-SUBRC <> 0.
  MESSAGE E000(ZMESSAGE1) WITH 'THE VENDOR IS NOT UNDER THE
COMPANY'.
ENDIF.
ENDMODULE.
PROCESS BEFORE OUTPUT.
* MODULE STATUS_2000.
*
PROCESS AFTER INPUT.
MODULE USER_COMMAND_2000.

CHAIN.
FIELD A VALUES ('1000','2000','2200').
FIELD B MODULE ABC.
ENDCHAIN.
MODULE USER_COMMAND_2000 AT EXIT-COMMAND.

```

#### **Syntax of calling the model dialogue box: -**

Call screen <screen no> starting at <XPOS><YPOS> ending at <XPOS><YPOS>.

#### **Ex:-**

Call screen 2000 starting at 11 ending at 50 40.

#### **Syntax of calling the executable program: -**

Submit <report name> via selection-screen.

#### **Ex: -**

Submit ZSPT via selection-screen.

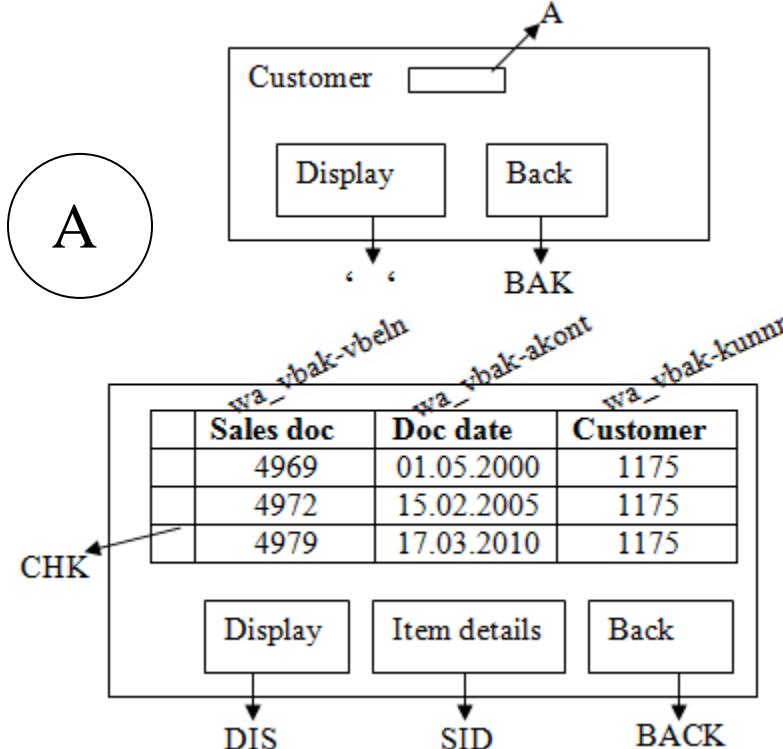
#### **Steps to provide input field as password: -**

Double click on input field. Provide name. Click on display tab on the attributes block. Select the check box ‘INVISIBLE’.

#### **→ Design the screen as shown in the below.**

After the user provide the customer & click on display or enter key then we display the sales document details of customer in a tabular format in 6000 screen.

If the user click on display button then we display the elected sales document details through VA03. If the user click on item details then we display the selected sales document details item details (VBELN, POSNR, KUNNR, MEINS,L NETPR) by using ALV & also validate the given customer.

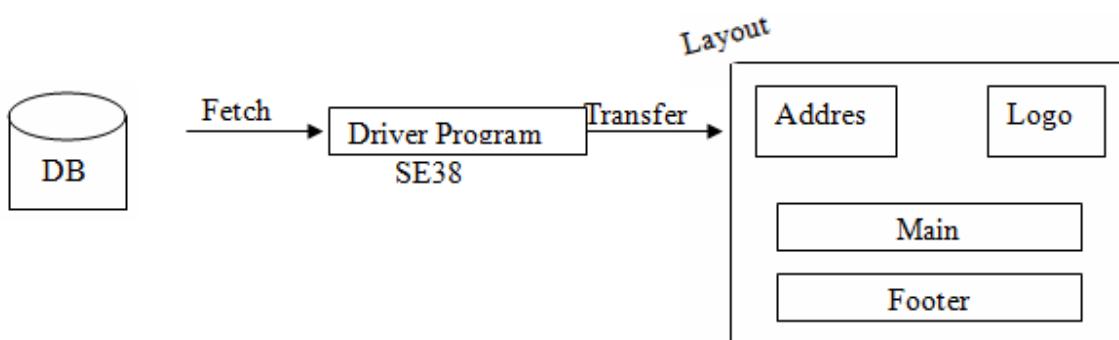
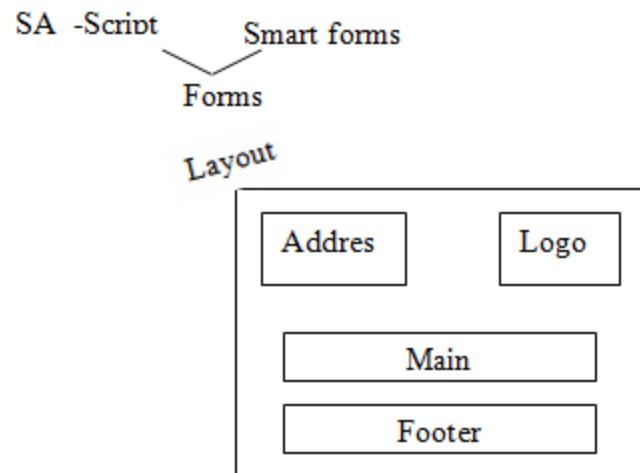


## SAP-SCRIPT

In the real time if you want to design the business documents, such as offer letters, experience letters, invoices, commercial invoices etc. we need layout sets. These are designed through forms. Forms are either SAP scripts or smart forms.

SAP-Script is a tool to display the business documents.

The standard SAP provided layout sets for almost all the applications. Most of the times the ABAPer job is either change the layout or adding some additional logic to the standard driver program.



Driver program is used to fetch the data from data base & transfer to the layout.

### Components of SAP script: -

1. Layout
2. Driver program

### Components of layout: -

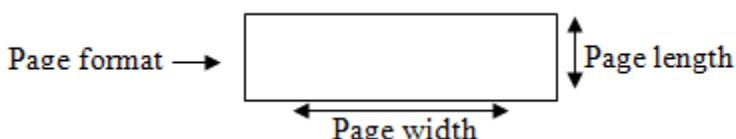
1. Header
2. Pages
3. Windows
4. Page window
5. Paragraph format
6. Character format
7. Documentation

### Header: -

Header is used to maintain the administrative information. i.e. form name, language & page format.

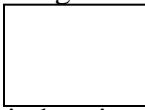
In the real time page formats are created by BASIS people through SPAD transaction.

Page format is the collection of page width & height of displayed document.



## **Pages: -**

Page is the physical area where we can place the windows. We can't print the data directly on the page.



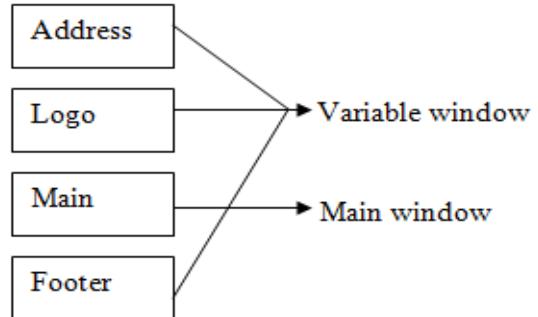
## **Windows: -**

We can paste the same window in any number of pages. We can't print the data directly on the window. There are two types of windows.

1. Main window
2. Variable window

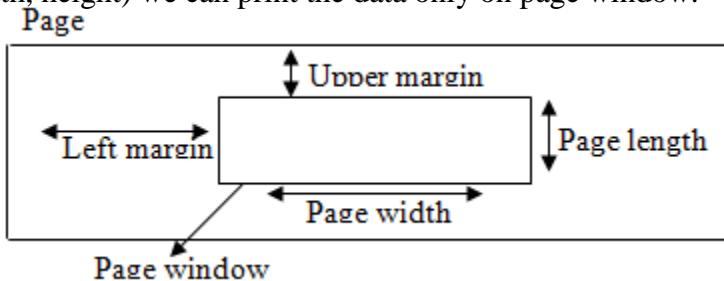
**Note:** - Main window is the default window in SAP script without a main window we can't design SAP script.

**Note:** - We can place the variable window only one time per page. Where as main window we can place up to 99 times per page.



## **Page window: -**

Page window is nothing but placing the window on the page with co-ordinates (left margin, upper margin, width, height) we can print the data only on page window.



## **Paragraph format: -**

This is used to print the entire paragraph to the required fonts & style.

## **Character format: -**

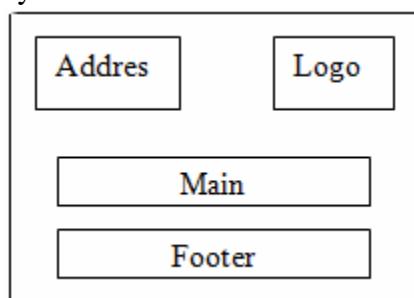
This is used to display the particular text with the required font & style.

## **Documentation: -**

This is used to maintain the document related to the form.

→ Design the screen as shown in the below. →

**Note:** - SE71 is the transaction code for form painter or form editor.



## **Steps to design the layout: -**

Execute SE71 . Provide the form name. Click on create. Provide short description. Click on pages in the application tool bar. In the menu bar click on edit → create element. Provide the page name, short description. Enter. Repeat the same steps for all pages. Click on windows in the application tool bar. Click on edit → create element. Provide the window name, short description. Enter. Repeat the same steps for all other windows. Click on page window. Click on edit → create element. It'll display the all available windows. Double click on the window & provide the co-ordinations. Click on paragraph formats in the application tool bar. Provide the default paragraph. Enter. Provide short description. Click on header in the application tool bar (F5). Click on basic settings. Provide the first page. Default paragraph. Save the layout. In the menu bar click on settings → form painter. Select the check box graphical form painter.

Enter. Click on layout in the application tool bar. Arrange the layouts by using drag & drop. Minimize the layout. Once again click on settings in the menu bar → form painter remove the graphical form painter. Enter. Save the layout. Check the layout. (Form → check → definition). Activate the layout (Form → active).

We can print the data on the page window is always through symbols. Each symbol start with ‘&’ ends with ‘&’. There are 4 types of symbols.

1. Program symbols.
2. System symbols
3. Standard symbols
4. Text symbols

#### **Program symbols: -**

Program symbols are variables in the program.

**Ex: -**

**&WA\_T001-BUKRS&  
&WA\_KNA1-KUNNR&**

#### **System symbols: -**

System symbols are system variables.

**EX: -**

**&date&  
&month&**

#### **Standard symbols: -**

Standard symbols are coming from TTDTG standard data base table.

**Ex: -**

**&Mr&  
&Dear&**

#### **Text symbols:-**

Text symbols are variables, which are defined in page window.

**Ex: -**

1: define &A&.

#### **Steps to develop the driver program:-**

1. Create an executable program and implement the retrieving logic.
2. Access the layout from the driver program by using ‘OPEN\_FORM’ function module. The input for the above function module is form name (layout).
3. Transfer the data from driver program to particular page window by using ‘WRITE\_FORM’ function module the input for above function module is window name.  
Repeat the same steps for each page window, which contains program symbols.
4. Close the form by using ‘CLOSE\_FORM’ function module.

**→ Based on the given vendor number display the vendor address in the address window by using SAP-Script.**

#### **Steps to provide the symbols on the page window:-**

Execute SE71. Provide the form name. Click on change. Click on page window on the application tool bar. Double click on our required window. Click on text elements in the application tool beside header button. Define &WA\_LFA1-LIFNR& &WA\_LFA1-NAME1& &WA\_LFA1-ORT01&.

Click on back, save, check, activate the program.

#### **Steps to execute the driver program: -**

Execute the program. Provide the input. Execute. Provide the output is ‘LP01’. Click on print preview.

**PARAMETER P\_LIFNR TYPE LFA1-LIFNR.**

**TYPES: BEGIN OF TY\_LFA1,  
LIFNR TYPE LFA1-LIFNR,**

```

NAME1 TYPE LFA1-NAME1,
ORT01 TYPE LFA1-ORT01,
END OF TY_LFA1.

DATA WA_LFA1 TYPE TY_LFA1.
SELECT SINGLE LIFNR NAME1 ORT01 FROM LFA1 INTO WA_LFA1 WHERE LIFNR =
P_LIFNR.

* Access the layout from driver program
CALL FUNCTION 'OPEN_FORM'
  EXPORTING
    FORM = 'ZSCRIPT1'.
* Transfer the data to address window. P_LIFNR. 
CALL FUNCTION 'WRITE_FORM'
  EXPORTING
    WINDOW = 'ADDRESS'.
*Close the form
CALL FUNCTION 'CLOSE_FORM'.
CALL FUNCTION 'WRITE_FORM'
  EXPORTING
    ADDRESS
*      &WA_LFA1-LIFNR&
*      &WA_LFA1-NAME1&
*      &WA_LFA1-ORT01&

```

#### Working with logo: -

We can print either '.BMP' or '.TIFF' images only.

**Note:** - When ever we are working with '.BMP' image then we must convert '.BMP' to graphics image by using 'SE78' transaction.

**Note:** - If we are working with '.TIFF' image then we must convert TIFF to text image by using RSTXLDLC standard program.

#### Steps to convert '.BMP' to graphics image: -

Execute SE78. Expand the graphics. Double click on bitmap image. Provide the graphics name. Select the radio button 'color bitmap image'. Click on import in the application tool bar. Browse the bitmap image. Enter.

#### Steps to insert the logo in the page window: -

Execute SE71. Open the form in change mode. Click on page widows in the application tool bar. Double click on logo window. Click on text elements in the application tool bar. In the menu bar click on insert → graphics. Click on stored document server tab. Provide the graphics name. Select the radio button 'color bitmap image'. Enter. Click on back. Save, check, activate.

**→ Based on the given vendor number display the vendor purchase orders (LIFNR, EBELN, BEDAT) in the main window and vendor details in header window by using SAP-Script.**

**Note:** - If you get the error WRITE\_FORM is invalid START\_FORM is missing then we must provide the next page as same page in the form (page).

#### Steps to provide next page: -

Execute SE71. Open form in change mode. Click on the pages on the application tool bar. Provide next page is page1. Save, check, activate.

**Note:** - When ever we are working with main window then we must provide text element name otherwise the first information will be printed twice. The text element name start with '/E'.

**PARAMETER P\_LIFNR TYPE LFA1-LIFNR.**

**TYPES: BEGIN OF TY\_LFA1,**

```

LIFNR TYPE LFA1-LIFNR,
NAME1 TYPE LFA1-NAME1,
ORT01 TYPE LFA1-ORT01,
END OF TY_LFA1.

DATA WA_LFA1 TYPE TY_LFA1.
TYPES: BEGIN OF TY_EKKO,
        LIFNR TYPE EKKO-LIFNR,
        EBELN TYPE EKKO-EBELN,
        BEDAT TYPE EKKO-BEDAT,
        END OF TY_EKKO.

DATA: WA_EKKO TYPE TY_EKKO,
      IT_EKKO TYPE TABLE OF TY_EKKO.

SELECT SINGLE LIFNR NAME1 ORT01 FROM LFA1 INTO WA_LFA1 WHERE LIFNR =
P_LIFNR.

SELECT LIFNR EBELN BEDAT FROM EKKO INTO TABLE IT_EKKO WHERE LIFNR =
P_LIFNR.

* Access the layout from driver program
CALL FUNCTION 'OPEN_FORM'
  EXPORTING
    FORM = 'ZSCRIPT1'.

* Transfer the data to address window.
CALL FUNCTION 'WRITE_FORM'
  EXPORTING
    WINDOW = 'ADDRESS'.

* Transfer the data to main window.
LOOP AT IT_EKKO INTO WA_EKKO.
  CALL FUNCTION 'WRITE_FORM'
    EXPORTING
      ELEMENT = 'SATISH'
      WINDOW = 'MAIN'.

ENDLOOP.

*Close the form
CALL FUNCTION 'CLOSE_FORM'.

```

Loop at it\_ekko wa\_ekko.

CALL FUNCTION 'WRITE\_FORM'

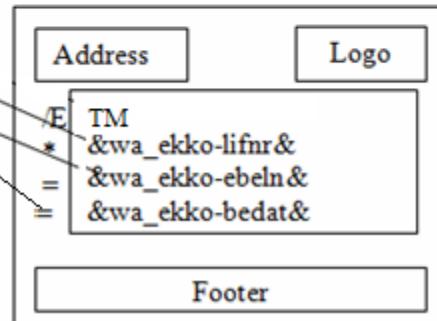
→ 'TM'  
→ 'MAIN'

Endloop.

P\_LIFNR:

LIFNR	EBELN	ORT01
5550	30004	01.05.2000
5550	30005	05.02.2005
5550	30009	31.03.2010

LIFNR	EBELN	ORT01
5550	30004	01.05.2000



In the real time most of the times footer window is used to print the page numbers & sign in last page.

#### Syntax of page numbers :-

\* PAGE & PAGE& at & SAPSCRIPT-FORMPAGES&  
    → Current page                          → number of pages

#### Syntax of sign in last page: -

/: if &nextpage& = 0.  
\* SHREE JANANI TECHNOLOGIES  
/: endif.

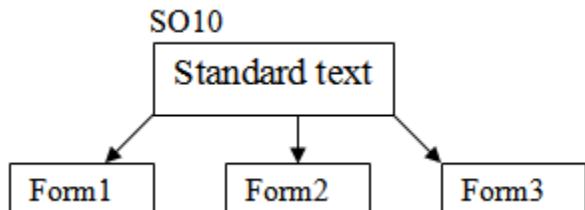
#### **Control commands: -**

Control commands are used to control the display output. Control commands start with ‘/’ control commands are

1. Include
2. Define
3. Address ----- endaddress
4. protect ----- endprotect
5. top ----- endtop
6. bottom ----- Endbottom
7. if ----- endif
8. case ----- endcase
9. set date / time mask
10. new-page
11. New window
12. box
13. perform ----- endperform

#### **Include: -**

This command is used to include the standard text which is defined in ‘SO10’ transaction into page window. If you want to maintain the same information in more than one form instead of maintaining these information in each form it’s better to maintain those statements in the standard text & later we include the standard text into each form.



#### **Steps to create standard text: -**

Execute SO10. Provide the standard text name. Click on create. Provide the screen text.

SHREE JANANI TECHNOLOGIES  
# 301, TIRUMALA MANASA COMPLEX  
ABOVE DCB BANK

#### **Steps to include the standard text in the page window: -**

Execute SE71. open the form in change mode. Click on the page window in the application tool bar. Double click on our required window. Click on text elements (F9). Place the cursor where we want to include the standard text. In the menu bar click on insert → text → standard. Provide the standard text name, enter. Click on back. Save, activate.

Whenever we create standard text it won’t ask any package or request number. So we must create a request number to standard text to transport to quality & live server.

#### **Steps to create the request number to standard text: -**

Execute SE38. Provide program name (RSTXTRAN). Execute. Provide the standard text name (ZSTN). Execute. Click on enter. Click on transfer text to correction in the application tool bar. Click on yes. Click on create request. Provide short description. Enter. Request no is ‘EC6K900496’.

This request number is given to basis people then the basis people move the standard text from development server to Quality / Live server.

**Define: -**

This command is used to declare the variables in the page window.

**Ex: -**

```
/: define &special& = 'DEC25'.
```

**Address - - - - endaddress: -**

This command is used to display the address in the format of target countries.

```
/: Address
* &wa_kna1-name1&
* &wa_kna1-ort01&
/: endaddress
```

**Protect - - - - endprotect: -**

This control command is used to print the continuous text without any page break. Here the system check each & every page which page is having the enough of page. If no page is having enough of page then it simply break the text & print in different places.

```
/: protect
* SHREE JANANI TECHNOLOGIES
* SR NAGAR
/: endprotect
```

**Top - - - - endtop: -**

This control command is used to display the header information in the main window.

**Ex: -**

```
/: top
* PUR-DOC, ITEM, QTY, UOM, PRICE
/: endtop.
```

**Bottom - - - - endbottom: -**

This control command is used to print the footer information in the main window.

```
/: bottom
* Total &v_total&
/: endbottom
```

**Note:** - Top & Bottom commands only work in main window.

**If - - - - - Endif: -**

This command functionality similar as ordinary of IF functionality.

```
/: If &wa_mara-matnr& = 'ROH'.
* Row material.
/: Elseif &wa_mara-matrn& = 'HALB'.
* Semi finished product.
/: Endif.
```

**Case - - - - - endcase: -**

This command functionally similar as ordinary case & end case functionality.

**Set date / time mask: -**

This command is used to print the date & time as per client required format.

```
/: Set date mask = 'DD/MM/YYYY'.
* &date&
12/07/2014
/: Set time mask = 'HH:MM:SS'.
* &time&
09:54:35
```

### New - page: -

This control command is used to break the page.

/: If &wa\_marc-matnr& = '100 – 200'.  
/: New-page.  
/: Endif.

### New-window: -

This control command is used to call the next window.

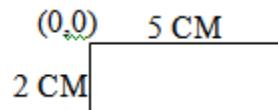
### Box: -

This command is used to draw the tables, horizontal lines & vertical lines.

#### Syntax of box: -

/: BOX xpos '<value>' <unit> ypos '<value>' <unit> width '<value>' <unit> height '<value>' <unit>  
intensity '<value>' <unit> frame '<value>' <unit>.

UNITS	DESCRIPTION
CM	CENTIMETER
MM	MILLIMETER
PT	POINT
IN	INCHES
CH	CHAR
TW	TWIP = $\frac{1}{20}$ PT



#### Ex: -

/: Box XPOS '0' cm YPOS '0' cm width '5' cm height '2' CM frame '20' TW.

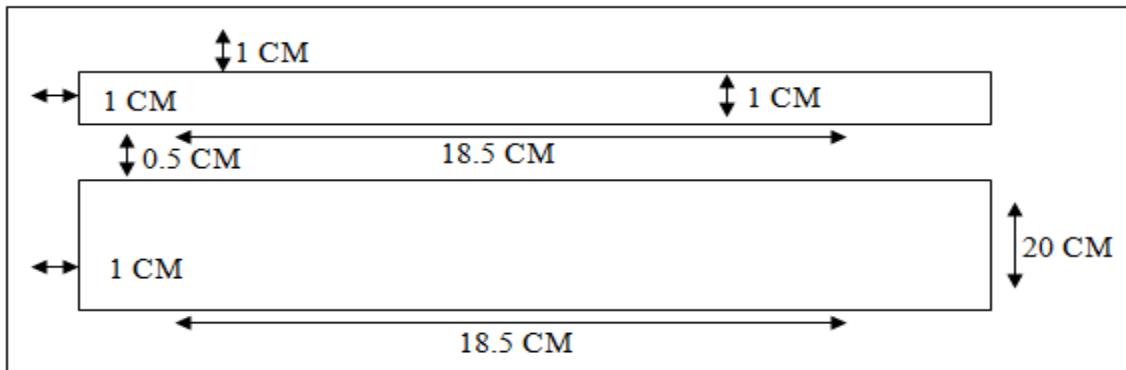
**Note:** - If we want to draw the horizontal line then height is 0, vertical line then width is 0.

/: Box xpos '0' cm ypos '0' cm width '7' cm height '0' cm frame '20' pw.

7 CM  
/: Box xpos '0' cm ypos '0' cm width '0' cm height '3' cm frame '20' pw. 3 CM

→ Based on the given purchasing document number display the purchasing document item details as shown below by using SAPSCRIPT.

PURCHASE ORDER				
Pur.doc	Item	Qty	UOM	Price
30004	01	10	Kg	250.00
30004	02	02	Pcs	150.00
30004	03	15	Nos	900.00
Total				1300.00

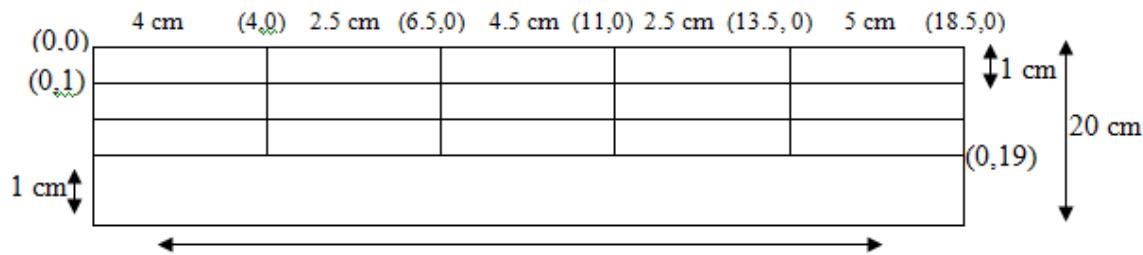


### **Title window: -**

Left margin 1 cm width 18.5 cm  
 Upper margin 1 cm height 1 cm

### **Main window: -**

Left margin 1 cm width 18.5 cm  
 Upper margin 2.5 cm height 20 cm



- /: Box frame '20' tw.
- /: Box xpos '0' cm ypos '1' cm width '18.5' cm height '0' cm frame '20' tw.
- /: Box xpos '0' cm ypos '19' cm width '18.5' cm height '0' cm frame '20' tw.
- /: Box xpos '4' cm ypos '0' cm width '0' cm height '19' cm frame '20' tw.
- /: Box xpos '6.5' cm ypos '0' cm width '0' cm height '19' cm frame '20' tw.
- /: Box xpos '11' cm ypos '0' cm width '0' cm height '19' cm frame '20' tw.
- /: Box xpos '13.5' cm ypos '0' cm width '0' cm height '19' cm frame '20' tw.

### **Steps to create paragraph formats: -**

Opens the form in change mode. Click on paragraph formats in the application tool bar. Provide the paragraph format. Enter. Provide short description, provide left margin 0.2 cm select the alignment 'center'. Click on font tab. Select font family 'HELVETICA'. Font size 14 pt. select the bold / italic / underline. Click on tabs button in the right side. Provide tab positions. Click on save. Repeat the same steps for all paragraph formats.

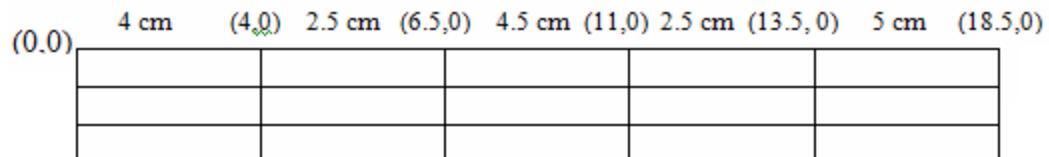
\* → default paragraph

/E → Text element

/: → Control command

/\* → Comment

T1  
 M1 } User created paragraphs  
 M2  
 = → Same line



Left align = left line + 0.2cm.

Right align = Right line - 0.2cm.

Center align = Left line + (column width/2).

**1 TAB = ,,**

### **Left alignment: -**

M1 PUR.DOC.,ITEM,,QTY,,UOM,,PRICE

M1

Left margin 0.2CM.

Number	Tab position	Alignment
1	4.2CM	Left
2	6.7CM	Left
3	11.2CM	Left
4	13.7CM	Left

Number	Tab position	Alignment
1	2CM	Center
2	5.2CM	Center
3	8.7CM	Center
4	12.2CM	Center
5	16CM	Center

Left line + width/2

$$0 + 4/2 = 2.$$

$$4 + 2.5/2 = 5.2$$

$$6.5 + 4.5/2 = 8.7$$

Center alignment

**Note:** - In the real time when ever we are working with quantity & amount fields then those information must be printed in right alignment.

M2 &WA\_EKPO-EBELN&&WA\_EKPO-EBELP&&WA\_EKPO-MENGE(ZC)&  
= &WA\_EKPO-MEINS&&WA\_EKPO-NETPR(ZC)&

M2

Left margin 0.2CM  
F1 „Total:,,&V\_Total&

F1

Left margin 0.2 CM

Number	Tab position	Alignment
1	4.2CM	Left
2	10.8CM	Right
3	11.2CM	Left
4	18.3CM	Right

Number	Tab position	Alignment
1	4.2 CM	Left
2	18.3 CM	Right

**PARAMETER P\_EBELN TYPE EKPO-EBELN .**

**TYPES: BEGIN OF TY\_EKPO,**

- EBELN TYPE EKPO-EBELN ,**
- EBELP TYPE EKPO-EBELP ,**
- MENGE TYPE EKPO-MENGE ,**
- MEINS TYPE EKPO-MEINS ,**
- NETPR TYPE EKPO-NETPR ,**
- END OF TY\_EKPO .**

**DATA: WA\_EKPO TYPE TY\_EKPO ,**  
**IT\_EKPO TYPE TABLE OF TY\_EKPO .**

**DATA V\_TOTAL TYPE EKPO-NETPR .**

**SELECT EBELN EBELP MENGE MEINS NETPR FROM EKPO INTO TABLE IT\_EKPO**  
**WHERE EBELN = P\_EBELN .**

**CALL FUNCTION 'OPEN\_FORM'**

**EXPORTING**

**FORM = 'ZSCRIPT2' .**

**LOOP AT IT\_EKPO INTO WA\_EKPO .**

**CALL FUNCTION 'WRITE\_FORM'**

**EXPORTING**

**ELEMENT = 'SATISH'**

**WINDOW = 'MAIN' .**

**V\_TOTAL = V\_TOTAL + WA\_EKPO-NETPR .**

**ENDLOOP .**

**CALL FUNCTION 'CLOSE\_FORM' .**

## FORM ZSCRIPT2

### MAIN WINDOW: -

```
/E SATISH
/: BOX FRAME '20' TW
/: BOX XPOS '0' CM YPOS '1' CM WIDTH '18.5' CM HEIGHT '0' CM FRAME '20' TW
/: BOX XPOS '0' CM YPOS '19' CM WIDTH '18.5' CM HEIGHT '0' CM FRAME '20' TW
/: BOX XPOS '4' CM YPOS '0' CM WIDTH '0' CM HEIGHT '20' CM FRAME '20' TW
/: BOX XPOS '6.5' CM YPOS '0' CM WIDTH '0' CM HEIGHT '20' CM FRAME '20' TW
/: BOX XPOS '11' CM YPOS '0' CM WIDTH '0' CM HEIGHT '20' CM FRAME '20' TW
/: BOX XPOS '13.5' CM YPOS '0' CM WIDTH '0' CM HEIGHT '20' CM FRAME '20' TW
/: TOP
M1 „PUR DOC,,ITEM,,QTY,,UOM,,PRICE
/: ENDTOP
```

```

M2  &WA_EKPO-EBELN&,,&WA_EKPO-EBELP&,,&WA_EKPO-MENGE(ZCT)&,
=   &WA_EKPO-MEINS(K)&,,&WA_EKPO-NETPR(ZCT)&
/:  BOTTOM
M3  ,,TOTAL:,,&V_TOTAL&
/:  END BOTTOM

```

### **Format options:-**

Format options are used to display the output or print the output as per client requirement.

<b>Format Options</b>	<b>Syntax</b>	<b>Example</b>	<b>Output</b>
Offset	&symbol+offset&	&a& = abcdefgh	&a+3& = defgh
Output length	&symbol(o/p length)&	&a& = abcdefgh	&a(3)& = abc
Omitting leading zeros	&symbol(z)&	&a& = 000100.00	&a(z)& = 100.00
Omitting sign	&symbol(s)&	&a& = -100.00	&a(s)& = 100.00
Leading sign at left	&symbol(<)&	&a& = 100.00-	&a(<)& = -100.00
Leading sign right	&symbol(>)&	&a& = -100.00	&a(>)& = 100.00-
Compress output	&symbol(c)&	&a& = 100.00	&a(c)& = 100.00
Ignore the separators	&symbol(t)&	&a& = 1,00,000.00	&a(t)& = 100000.00
Number of decimals	&symbol(.number)&	&a& = 10.3215	&a(.2)& = 10.32 &a(.0)& = 10
Ignore conversions	&symbol(k)&	&a& = &WA_EKPO-MEINS(K)&	

**Note:** – In the real time before modifying the SAP-SCRIPT we must maintain the backup of SAP-SCRIPT because SAP-SCRIPT doesn't have version management (ABAP editor have version management).

### **Steps to maintain the backup of SAP-SCRIPT or download the SAP-SCRIPT: -**

Execute SE38. Provide the program name RSTXSCRP. Execute. Provide object name as form name. Provide mode is EXPORT. Execute. Provide the file name with .txt. Click on save.

### **Steps to reload the backup SAP-SCRIPT: -**

Execute SE38. Provide the program name RSTXSCRP. Click on execute. Provide the object name as form name. Mode is IMPORT. Execute. Browse the file. Enter.

**Note:** – RSTXSCRP is the standard program which is used to download as well as upload the SAP-SCRIPT.

### **Steps to convert the SAP-SCRIPT output to PDF format: -**

This is 2 step procedure.

1. Generate / create the spool request number
2. Convert the spool request number to PDF format

### **Steps to generate the spool request number: -**

Execute the driver program. Provide the input. Execute. Provide the output device is LP01. Select the check box new spool request. Click on print.

### **Steps to identify the spool request number: -**

Execute SP01 or SP02. Provide the output devices is LP01. Click on execute. identify the spool.

### **Steps to convert the spool to PDF format: -**

Execute SE38. Provide the program name RSTXPDFT4. Execute. Provide the spool request number, execute. Provide the file name with .PDF.

**Note:** – We can convert any output to PDF format if the output is available in spool request. We can also convert the report output to PDF format. First we execute the report in background. Then we get the output in a spool, the spool request is convert into PDF by using RSTXPDFT4 standard program.

### **Steps to execute the report / program in background: -**

Execute SE38. Provide the program name. Execute. Provide the input. In the menu bar click on program → execute in background (F9). Click on continue. Click on immediate or click on date / time button. Provide the date & time.

If you want to print the output in both sides then open the form in SE71. Click on pages in the application tool bar. Select the print mode is ‘D’ in the print attributes. Save, check, activate the form.

If you want to print the terms & conditions in back side of each page, print mode D is not possible. We go for pre printed stationary (First we print the terms & conditions on the each paper those papers are loaded into the machine).

#### **Steps to call the multiple frames from driver program: -**

1. Create an executable program & implement the retrieving logic.
2. Access the layouts from driver program by using ‘OPEN\_FORM’ function module.
3.
  - i. Start the form by using ‘START\_FORM’ function module. The input for the above function module is form name.
  - ii. Transfer the data from driver program to particular page window by using ‘WRITE\_FORM’ function module. The input for the above function module is window name. Repeat the same step (ii) for each page window which contains program symbols.
  - iii. End the form by using ‘END\_FORM’ function module.

Repeat the step 3 for each form.

Close the form by using ‘CLOSE\_FORM’ function module.

**TABLES EKPO.**

**DATA V\_TOTAL TYPE EKPO-NETPR.**

**SELECT-OPTIONS S\_EBELN FOR EKPO-EBELN.**

**TYPES: BEGIN OF TY\_EKPO,**  
**EBELN TYPE EKPO-EBELN,**  
**EBELP TYPE EKPO-EBELP,**  
**MENGE TYPE EKPO-MENGE,**  
**MEINS TYPE EKPO-MEINS,**  
**NETPR TYPE EKPO-NETPR,**  
**END OF TY\_EKPO.**

**DATA: WA\_EKPO TYPE TY\_EKPO,**

**IT\_EKPO TYPE TABLE OF TY\_EKPO.**

**SELECT EBELN EBELP MENGE MEINS NETPR FROM EKPO INTO TABLE IT\_EKPO**  
**WHERE EBELN IN S\_EBELN.** ZTFORM3

**CALL FUNCTION 'OPEN\_FORM'.**

**CALL FUNCTION 'START\_FORM'**  
**EXPORTING**

**FORM = 'ZTFORM3'.**

**LOOP AT IT\_EKPO INTO WA\_EKPO.**

**CALL FUNCTION 'WRITE\_FORM'**  
**EXPORTING**

**ELEMENT = 'SATISH'**

**WINDOW = 'MAIN'.**

**V\_TOTAL = V\_TOTAL + WA\_EKPO-NETPR.** ZTFORM03

**ENDLOOP.**

**CALL FUNCTION 'END\_FORM'.**

**CALL FUNCTION 'START\_FORM'**  
**EXPORTING**

एम एन सतीष कुमार रेडि

PURCHASE ORDER				
Pur.doc	Item	Qty	UOM	Price
30004	01	10	Kg	250.00
30004	02	02	Pcs	150.00
30004	03	15	Nos	900.00
				Total 1300.00

MAIN FORM TERMS AND CONDITION
----------------------------------

```

FORM = 'ZTFORM03'.
CALL FUNCTION 'WRITE_FORM'
EXPORTING
WINDOW = 'MAIN'.
CALL FUNCTION 'END_FORM'.

CALL FUNCTION 'CLOSE_FORM'.

```

### Steps to debug the SAP-SCRIPT: -

#### METHOD 1: -

Execute SE71. Provide the form name. In the menu bar click on utilities → activate debugger. Now execute the driver program. Provide the input. Execute. Click on all. Click on print preview. Now the form is in debugging mode. Continuously click on 'F5' button. Identify the field names & their values.

#### METHOD 2: -

Execute SE38. Provide the program name RSTXDEBUG. Execute the driver program in a separate session. Provide the input. Execute. Click on OK. Click on print preview. Continuously click on 'F5' button. Identify the field names & field values.

**Note:** - From the driver program to main window data is transferred first. Next only other window data transfers.

#### MCHA (Material Batch Table) : -

MATNR → (Material number)

WERKS → (Plant number)

CHARG → (Batch number)

HSDAT → (Mfg. date)

VFDAT → (Exp. Date)

#### Working with labels: -

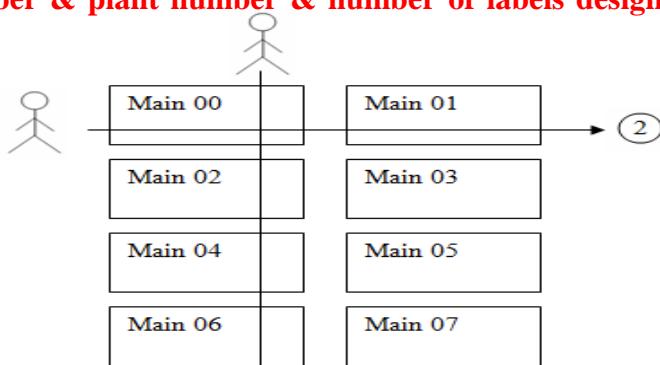
Labels are used to print the same information in each window.

In the real time in the where house module we design the so many labels such as accepted labels, rejected labels, on test labels, control sample label . . .

Depends the label size we split the main window into smaller window.

**→ Based on the given material number & plant number & number of labels design the accepted labels as shown in the below.**

Material: -----
Plant: -----
Batch: -----
Mfg.dt: -----
Exp.dt: -----



#### Steps to design the form: -

Execute SE71. Provide the form name. Click on create. Provide short description. Create the page. Click on windows in the application tool bar. In the menu bar click on edit → Main window. Provide the window area. Provide area → left margin 1.00 cm, upper margin 1.00 cm, area width 18.5 cm, area height 25.00 cm, horizontal → spacing 1.00 cm, number 2, vertical → spacing 1cm, number 4. Click on enter. Create the paragraph format. Provide header information. Save, check, activate.

**PARAMETER:** P\_MATNR TYPE MCHA-MATNR,  
P\_WERKS TYPE MCHA-WERKS,  
P\_NOL TYPE I.

```

DATA: BEGIN OF WA_MCHA,
      MATNR TYPE MCHA-MATNR,
      WERKS TYPE MCHA-WERKS,
      CHARG TYPE MCHA-CHARG,
      HSDAT TYPE MCHA-HSDAT,
      VFDAT TYPE MCHA-VFDAT,
   END OF WA_MCHA.

SELECT SINGLE MATNR WERKS CHARG HSDAT VFDAT FROM MCHA INTO WA_MCHA
WHERE MATNR = P_MATNR AND WERKS = P_WERKS.

* Access the layout from driver program
CALL FUNCTION 'OPEN_FORM'
  EXPORTING
    FORM = 'ZFORM195'.

* Transfer the data to main window
DO P_NOL TIMES.
  CALL FUNCTION 'WRITE_FORM'
    EXPORTING
      ELEMENT = 'HAI'
      WINDOW = 'MAIN'.
ENDDO.

* Close the form
CALL FUNCTION 'CLOSE_FORM'.

```

MAIN WINDOW: -

/E TM

\* Material,:&wa\_mcha-matnr&

\* Plant,: &wa\_mcha-werks&

\* Batch,: &wa\_mcha-charg&

\* Mfg.dt,: &wa\_mcha-hsdat&

\* Exp.dt,: &wa\_mcha-vfdat&

/: new-window

Working with Standard SAP-SCRIPT

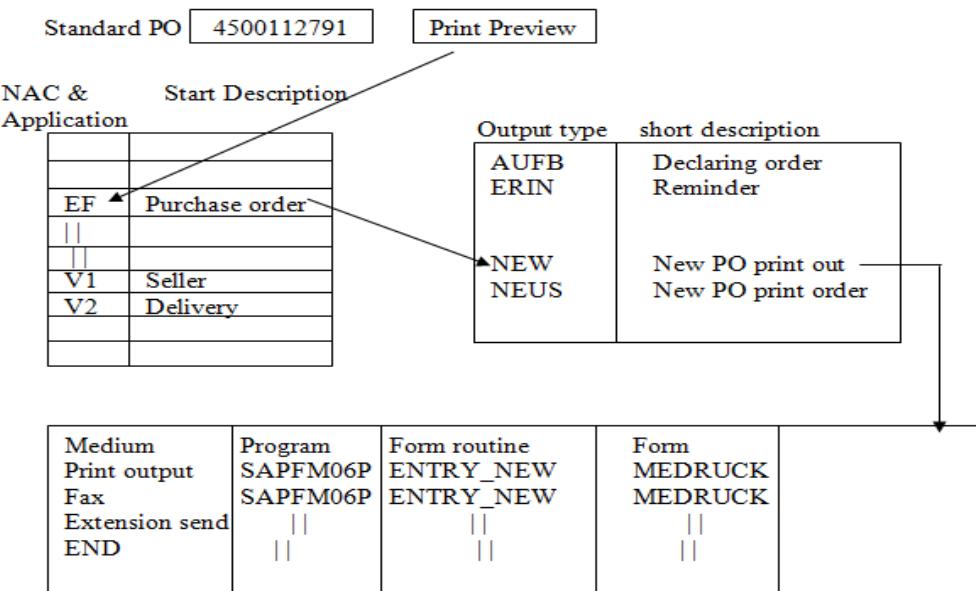
Change layout

Adding some additional logic to standard driver program

**Note:** - TNAPR is the standard data base table, which contains all the standard layouts & driver programs.

**Note:** - NACE is the transaction code, which contains all the application & their driver programs & layouts.

If we want to take the print preview of purchase order then execute ME23N transaction.



If you want to display the purchase order print preview then we execute the ME23N transaction & provide the purchase order number & click on print preview in the application tool bar. Then it goes to NACE transaction & identify the application at purchase order. Against the application it identify the output type (NEW) against the output type it identifies the medium print output. Against the medium it triggers the standard program & form. Based on the form we print the output.

In the real time output types are created by functional people based on the document category.

**Ex: -**

For domestic purchase order they create one output type. For amendment domestic purchase orders one output type. For import purchase order they create one output type. For stock transfer order they create one output type.

**Steps to change the layout: -**

1. Identify the standard layout / form
2. Copy the standard form into 'Z' form
3. Convert the language from original (DE) to our required language of the form based on requirement
4. Change 'Z' layout as per client requirement.
5. Place the new layout/form into NACE transaction.

→ **Modify the standard purchase order layout (MEDRUC) to incorporate the logo.**

**Steps to identify the standard layout or form: -**

Execute NACE transaction. Select the purchase order application (EF) & click on output types in the application tool bar. Select the output type which is provided by functional people. Double click on processing routines in the left panel. Against print output medium identify the form MEDRUCK.

**Steps to copy the standard form into 'Z' form: -**

Execute 'SE71'. In the menu bar click on utilities → copy from client. Provide the form name MEDRUCK. Source client is 000. Target form ZSPT\_930\_MEDRUCK. Click on execute, save, local object.

**Steps to convert the languages: -**

Execute 'SE71'. Provide the original language 'DE'. Click on change. In the menu bar click on utilities → convert original language. Provide to original language 'EN'. Enter.

**Note:** – We can create the windows page windows paragraph & character formats only in original language. That is automatically reflected to all other languages. The text in the windows isn't reflected to other languages.

**Steps to change the layout as per client requirement: -**

Execute 'SE71'. Provide the form name & original language. Click on change. Create the logo window. Place the logo window on the page. Align the layout & insert the graphics image in the logo window. Save, check, activate.

**Steps to place the new layout / form into the NACE transaction: -**

Execute NACE transaction. Select the application. Click on output types. Select the output type. Double click on processing routines in the left panel. Click on change mode. Remove the old form & place the new form. Save.

**Steps to identify the output type & language of purchase order: -**

Execute 'ME23N'. Provide the purchase order number. Click on messages in the application tool bar. Identify the output type & language.

**Steps to take the print preview of sales order: -**

Execute VA03 transaction. Provide the sales order number. In the menu bar click on sales document → issue output to select the message type. Click on print options. Provide destination LP01. Click on execute. Click on print preview in the bottom.

**Note:** – In the real time the output device is not always LP01, depends on the printer (laser, dot matrix) & depends on the language (EN, DE, CH) the output devices are created by basis people.

## **Perform - - - - endperform: -**

This control command is used to adding some additional logic to the standard driver program without disturbing the standard driver program.

### **Syntax of perform - - - endperform (calling) in page window: -**

```

/: Perform <form name> in program <subroutine pool program>
Where the definition is available ←
/: using &input1&
/: using &input2&
  ||
/: changing &output1&
/: changing &output2&
  ||
/: endperform

```

### **Syntax of definition in subroutine pool program: -**

Form <form name> tables <input> structure ITCSY <output> structure ITCSY.

----- }  
----- } business logic  
----- }

/: Endform.

Here input & output acts like an internal table with header line which contains two fields. That is name & value.

**Note:** - ITCSY is the structure which contains the two fields name & value.

Here all the using parameters & their values are stored into input internal tables. & all the changing parameters are stored into output internal tables. Here the business logic is read the input field & their value based on the values, we will fetch the data from data base & modify the output field value.

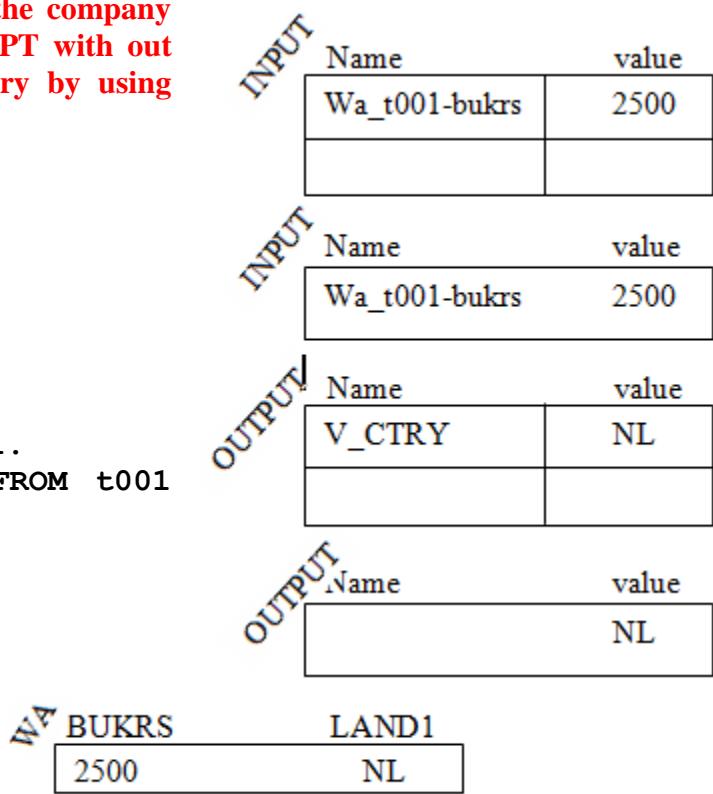
**→ Based on the given company code display the company code, company name, city by using SAP-SCRIPT with out disturbing the driver program add the country by using subroutine pool program.**

### **ZSPT 930 DRIVER PROGRAM3**

```

PARAMETER p_bukrs TYPE t001-bukrs.
TYPES: BEGIN OF ty_t001,
         bukrs TYPE t001-bukrs,
         butxt TYPE t001-butxt,
         ort01 TYPE t001-ort01,
         END OF ty_t001.
DATA: wa_t001 TYPE ty_t001,
      it_t001 TYPE TABLE OF ty_t001.
SELECT SINGLE bukrs butxt ort01 FROM t001
INTO wa_t001 WHERE bukrs =
p_bukrs.
CALL FUNCTION 'OPEN_FORM'
EXPORTING
  form = 'ZFEB26'.
CALL FUNCTION 'WRITE_FORM'
EXPORTING
  element = 'SATISH'
  window = 'MAIN'.
CALL FUNCTION 'CLOSE_FORM'.

```



### ZSPT 930 FORM3

```

/E   SATISH
*   &WA_T001-BUKRS&
*   &WA_T001-BUTXT&
*   &WA_T001-ORT01&
/:  PERFORM GET_COUNTRY IN PROGRAM Z11SCRIPT3
/: USING &WA_T001-BUKRS&
/:   CHANGING &V_LAND1&
/: ENDPERFORM
*: COMPANY COUNTRY: &V_LAND1&

```

### ZSPT 930 CTRY (SUBROUTINE POOL PROGRAM): -

```

DATA: BEGIN OF WA,
      BUKRS TYPE T001-BUKRS,
      LAND1 TYPE T001-LAND1,
      END OF WA.

FORM GET_COUNTRY TABLES ITAB STRUCTURE ITCSY OTAB STRUCTURE ITCSY.
  READ TABLE ITAB WITH KEY NAME = 'WA_T001-BUKRS'.
  SELECT SINGLE BUKRS LAND1 FROM T001 INTO WA WHERE BUKRS = ITAB-VALUE.
  OTAB-VALUE = WA-LAND1.
  MODIFY OTAB TRANSPORTING VALUE WHERE NAME = 'V_LAND1'.
ENDFORM.

```

→ Develop a driver program to display the material number & plant number with out disturbing the driver program we add the material description & plant description by using subroutine pool program.

PARAMETER P\_MATNR TYPE MARC-MATNR.

```

TYPES: BEGIN OF TY_MARC,
       MATNR TYPE MARC-MATNR,
       WERKS TYPE MARC-WERKS,
       END OF TY_MARC.

```

DATA WA\_MARC TYPE TY\_MARC.

```

SELECT SINGLE MATNR WERKS FROM MARC INTO WA_MARC WHERE MATNR =
P_MATNR.

```

CALL FUNCTION 'OPEN\_FORM'

```

  EXPORTING
    FORM                  = 'ZTSCRIPT2'.

```

CALL FUNCTION 'WRITE\_FORM'

```

  EXPORTING
    ELEMENT              = 'SATISH'
    WINDOW               = 'MAIN'.

```

CALL FUNCTION 'CLOSE\_FORM'.

```

/E   SATISH
*   &WA_MARC-MATNR&
*   &WA_MARC-WERKS&
/:  PERFORM GET_DES IN PROGRAM ZTSCRIPT02
/: USING &WA_MARC-MATNR&
/: USING &WA_MARC-WERKS&
/: CHANGING &V_MDES&
/: CHANGING &V_PDES&
/: ENDPERFORM
*: MAT DES: &V_MDES&
*: PLANT DES: &V_PDES&

```

एम एन सतीष कुमार रेडि

INPUT	
Name	value
Wa_marc-matnr	100-200
Wa_marc-werks	2010
OUTPUT	
Name	value
V_MDES	MARKERS
V_PDES	SR NAGAR
OUTPUT	
Name	value
	SR NAGAR

```

DATA: BEGIN OF WA_MAKT,
      MATNR TYPE MAKT-MATNR,
      MAKTX TYPE MAKT-MAKTX,
      END OF WA_MAKT.
DATA: BEGIN OF WA_T001W,
      WERKS TYPE T001W-WERKS,
      NAME1 TYPE T001W-NAME1,
      END OF WA_T001W.
FORM GET DES TABLES INPUT STRUCTURE ITCSY OUTPUT STRUCTURE ITCSY.
READ TABLE INPUT WITH KEY NAME = 'WA_MARC-MATNR'.
SELECT SINGLE MATNR MAKTX FROM MAKT INTO WA_MAKT WHERE MATNR =
INPUT-VALUE.
OUTPUT-VALUE = WA_MAKT-MAKTX.
MODIFY OUTPUT TRANSPORTING VALUE WHERE NAME = 'V_MDES'.
READ TABLE INPUT WITH KEY NAME = 'WA_MARC-WERKS'.
SELECT SINGLE WERKS NAME1 FROM T001W INTO WA_T001W WHERE WERKS =
INPUT-VALUE.
OUTPUT-VALUE = WA_T001W-NAME1.
MODIFY OUTPUT TRANSPORTING VALUE WHERE NAME = 'V_PDES'.
ENDFORM.

```

WA_MAKT		MATRN	MAKTX
		100-200	MARKERS
WA_T001W		WERKS	NAME1
		2010	SR NAGAR

### Text element: -

The text element is the name given to the block of statements in the page window. Text element name start with /E. Normally the write form functional module transfers the data from driver program to all the statements which are available in the page window.

Call function 'WRITE\_FORM'

Exporting

FORM = 'ADDRESS'.

ADDRESS	
*	&wa_t001-bukrs&
*	&wa_t001-ort01&

If you provide text element name to the write form function module then the 'WRITE\_FORM' function module transfers the data from driver program to all the statements which are available in the text element.

Call function 'WRITE\_FORM'

Exporting

Element = 'DHAWAN'.

Window = 'ADDRESS'

ADDRESS	
/E	SATISH
*	&WA_T001-BUKRS&
*	&WA_T001-ORT01&
/E	DHAWAN
*	&WA_LFA1-LIFNR&
*	&WA_LFA1-NAME1&

**Note:** - The page window contains at least one text element then we must provide the text element name to the write from function module.

### EKET (Scheduling agreement table) : -

EBELN → Purchasing document number

EBELP → Item number

EINDT → Delivery date

## → Modify the standard purchase order layout to incorporate the item wise delivery date.

In the object we add the additional logic of item wise delivery date to the standard driver program through subroutine pool program. Before adding the additional logic first we take a printout or print preview of existing layout & later we put the debugging mode & identify the right place with the help of printout or print preview. Implement the logic.

### Steps to identify the right place in the page window where we add the additional logic to the page window:

Execute 'ME23N'. provide PO number. Click on print preview or take a printout. Now execute the SE71. provide form name. in the menu bar click on utilities → Activate debugger. Now execute ME23N in a separate session. Click on print preview. Now the form is in debugging mode. Click on ok. Continuously click on F5 button. Identify the right window & text element. Right place is window name 'MAIN'. Element name is 'ITEM\_LINE\_1'

**FORM: ZSPT930\_MEDRUCK**

### WINDOW 'MAIN': -

IL

```
/:      PERFORM GET_DELDT IN PROGRAM ZTDELDT
/:      USING &EKPO-EBELN&
/:      USING &EKPO-EBELP&
/:      CHANGING &V_DD&
/:      ENDPERFORM
/      &EKPO-EBELP&,,&EKPO-EMATN&,,&EKPO-TXZ01&,,&V_DD+6(2)&.&V_DD+4(2)&.
=          &V_DD+0(4)&
```

### Sub routine pool program: -

PROGRAM ZTDELDT.

```
DATA V1 TYPE EKPO-EBELN.
DATA V2 TYPE EKPO-EBELP.
```

```
DATA: BEGIN OF WA_EKET,
      EBELN TYPE EKET-EBELN,
      EBELP TYPE EKET-EBELP,
      EINDT TYPE EKET-EINDT,
      END OF WA_EKET.
```

```
FORM GET_DELDT TABLES INPUT STRUCTURE
ITCSY OUTPUT STRUCTURE ITCSY.
  READ TABLE INPUT WITH KEY 'EKPO-EBELN'.
  V1 = INPUT-VALUE.
  READ TABLE INPUT WITH KEY 'EKPO-EBELP'.
  V2 = INPUT-VALUE.
  SELECT SINGLE EBELN EBELP EINDT FROM EKET INTO WA_EKET WHERE
EBELN = V1 AND EBELP = V2.
  OUTPUT-VALUE = WA_EKET-EINDT.
  MODIFY OUTPUT TRANSPORTING VALUE WHERE NAME = 'V_DD'.
ENDFORM.
```

INPUT	
Name	value
Ekpo-ebeln	4500011279
Ekpo-ebelp	10

INPUT	
Name	value
Ekpo-ebelp	10

OUTPUT	
Name	value
V_ADD	20010522

OUTPUT	
Name	value
	20010522

WA_EKET	EBELN	EBELP	EINDT
	4500011279	10	20010522

V\_ADD = 20010522

&v\_add+6(2)&.&v\_add+4(2)&.&v\_add+0(4)&

एम एन सतीष कुमार रेडि

### **Types of windows in the SAP-SCRIPT: -**

1. Main window
2. Variable window
3. Constant window

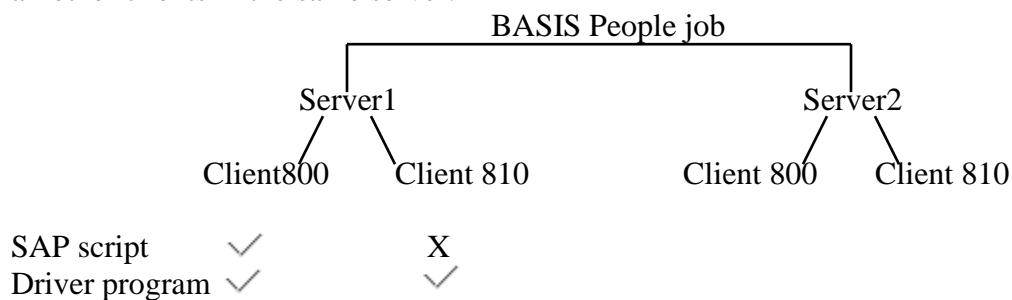
Control window is the fixed window in all the pages.

### **Differences between MAIN window and VARIABLE window: -**

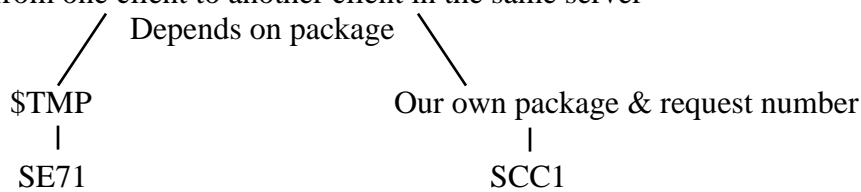
- | <b>MAIN WINDOW</b>  | <b>VARIABLE WINDOW</b>   |
|---|--|
| 1. It's used to print the continuous data.  | 1. Based on the window width & height only we can print the data.                          |
| 2. We can split the main window into smaller windows.   | 2. We can't split the variable window.   |
| 3. Without a main window we can't design SAP-SCRIPT.  | 3. Without a variable window we can design SAP-SCRIPT. We can create only variable window. |
| 4. Top & Bottom control commands only work in main window.  | 4. Top & Bottom aren't work in variable window.  |
| 5. Next page = 0 isn't possible in main window.   | 5. Next page = 0 is possible in variable window.   |
| 6. We must provide the text element name to the main window. Otherwise the first information will be printed twice. | 6. We no need to provide the text element name in the variable window.                     |



SAP script is client dependent that means if you design the SAP script in one client that isn't reflected to all other clients in the same server.



Copy the SAP script from one client to another client in the same server



### **Steps to copy the SAP script from one client to another client (800 to 810) into the script is available in \$TMP: -**

In 810 client execute SE71. In the menu bar click on utilities → copy from client. Provide the form name. Source client is 800. Provide the target form (ZSPT\_930\_FORM). Execute.

### **Steps to change the package at any object: -**

Execute SM30. Provide the table name ‘TADIR’. Click on maintain. Enter & check the our object type. If our object type isn’t available, then select the object type & provide the form name or object name. Click on execute. Double click on our object. Remove the old package. Place the new package. Click on save. Create the request (F8). Provide short description. Note down the request. Enter.

#### **Steps to copy the SAP script from 800 to 810 client if the form is available in our own package & request number: -**

In 810 client execute ‘SCC1’. Provide the source client number, request number. Select the include tasks in request checkbox. Click on start immediately in the application tool bar. Click on yes.

**Note:** – We can’t copy the SAP script one server to another server. This is done by BASIS people. We can copy the SAP script within the server only.

**Note:** – ‘RSTXFCPY’ is the standard program to copy the SAP script from one client to another client.

#### **READ\_TEXT: -**

It’s the function module which is used to read the standard text information. The input for the above function module is

1. Text ID
2. Name
3. Object
4. Language

The output for the above function module is an internal table which contains two fields. They are TDFORMAT & TDLINE.

If you want to identify the input of the above function module then open the standard text in the menu bar. Click on goto → header. Identify the name, language, text id, object.

**Note:** – In the real time we use this function module to read the purchasing document header text, purchase document item text, sales document header, sales document item text, invoice header text, invoice item text . . . .

When ever we are working with ‘READ\_TEXT’ module, we must uncomment the execution.

Data: IT like table of TDLINE,

WA like line of IT.

Call function ‘READ\_TEXT’

Exporting

ID = ‘ST’

Language = sy-langu

Name = ‘ZSPT\_930\_ST’

Object = ‘TEXT’

Lines = IT.

Exceptions

-----

-----.

Loop at IT into WA.

Write:/ WA-TDFORMAT, WA-TDLINE.

Endloop.

→ Based on the given material number, display the material numbers, material descriptions. If the functional people maintain the material description in the purchase order text then we print this information otherwise print the MAKTX.

**Steps to create the material: -**

Execute MM01. Select the provide industry sector, material type. Enter. Select Basic detail, purchase order text, enter. Provide short description. Provide the material group. Click on purchase order text tab. Provide the detailed description & save.

Data V like THEAD-TDNAME

Loop at it\_makt into wa\_makt.

S\_MATNR 100-200 to 100-200

\* Apply the conversion routine if it required

Call function 'READ\_TEXT'

Name = V

Object = 'MATERIAL'

Language = sy-langu

Id = 'BEST'

Lines = IT.

If sy-subrc = 0.

Read table it into wa index 1.

Write:/ wa\_makt-matnr, wa-pdformat.

Else.

Write:/ wa\_makt-mant, wa\_makt-maktx.

Endif.

Endloop.

MATNR	MAKTX
100-200	Markers
100-210	Chairs
100-300	Fans

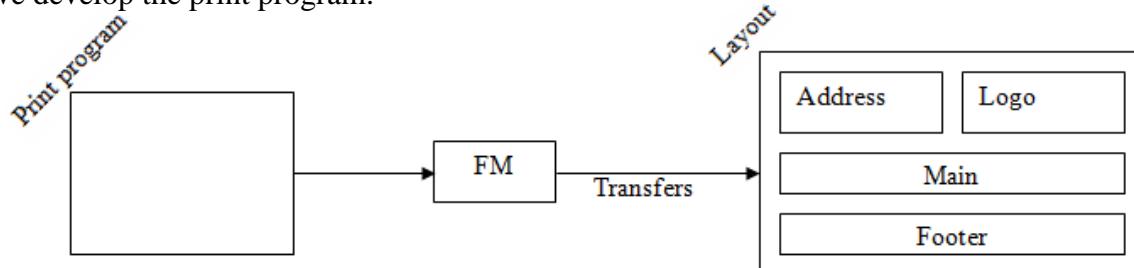
## **SMART FORMS: -**

Smart forms are used to design the business documents such as invoices, purchase orders, sales orders. . . smart form is introduced from 4.6C version onwards.

### **Procedure of the smart form: -**

Based on the client requirement we design the smart form layout by using ‘SMARTFORMS’ transaction code & provide the necessary symbols, save, check, activate.

When ever we activate the smart form it generates a function module. Based on the function module we develop the print program.



Function module is used to transfers the data from print program to layout.

### **Components of SMART FORMS: -**

1. Smart form layout
2. Function Module
3. Print program

### **Components of Smart form layout: -**

1. Global settings
2. Pages and windows

Global settings is the collection of form attribute, form interface, global definitions.

### **Form attributes: -**

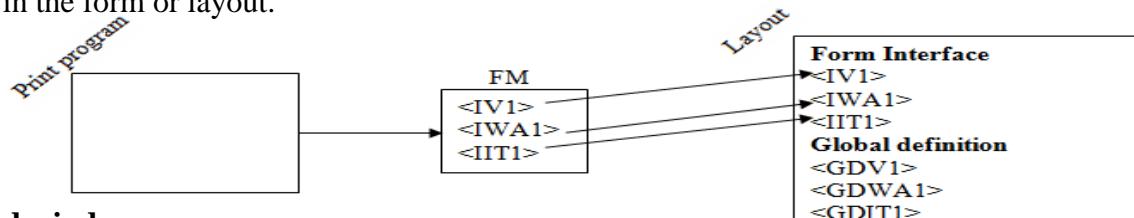
These are used to maintain the administrative information that is form name, language, page format, default style.

### **Form Interface: -**

This is used to declare the variables, work areas and internal tables which are needed to transfers the data from print program to layout

### **Global definition: -**

These are used to declare the variables, work area, internal tables which are needed to implement the logic in the form or layout.



### **Pages and windows: -**

These are used to design the smart form layout.

**Note:** – ‘SMARTFORMS’ is the transaction code to design the smart form layout.

**→ Based on the given company code, display the company code, comp name & city by using smart forms.**

### **Steps to design the smart form:-**

Execute ‘SMARTFORMS’. Select the radio button form. Provide the form name. Click on create. Provide short description. Double click on form interface in the left panel. The import tab (I\_BUKRS, type, t001-bukrs). Double click on global definitions in the left panel. Click on types tab. Declare the types.

Types: begin of ty\_t001,

Bukrs type t001-bukrs,

Butxt type t001-butxt,  
 Ort01 type t001-ort01,  
 End of ty\_t001.

Click on global data tab. Provide work area. (WA\_T001 TYPE TY\_T001).

Click on initialization tab & implement the logic. Provide input output parameters.

#### **INPUT PARAMETER      OUTPUT PARAMETER**

I\_BUKRS                    WA\_T001

```
select single bukrs butxt ort01 from t001 into wa_t001 where bukrs = i_bukrs.
```

Expand the page in the left panel. Select the main window. Right click → create → text. Double click on text. Click on editor under general attributes tab. Provide symbols.

&wa\_t001-bukrs&

&wa\_t001-butxt&

&wa\_t001-ort01&

Click on back. Save, check, activate. In the menu bar click on environment → function module name. based on this function module we develop the print program.

Report ZSPT\_930\_PRINT\_PROGRAM

Parameter p\_bukrs type t001-bukrs.

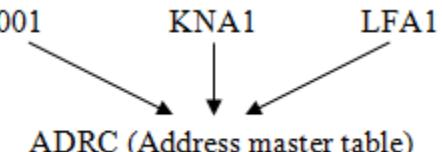
Call function '/1BCDWB/SF00000341'

Exporting

I\_BUKRS = P\_BUKRS.

**Note:** - In the SAP summarized data is available in t001, kna1,

lfa1 & their detailed information is available in 'ADRC' table. The link is 'ADRNR'.



#### **Working with address window: -**

In the real time if you want to print the address then you must use address window. This input for the address window is address number (ADRNR)

**→ Based on the given customer number, display the customer address by using address window in the smart form.**

#### **Steps to design the smart form: -**

SMART FORM NAME: ZSPT\_930\_SF2

#### **Form interface:**

##### **Import:**

I\_KUNNR      TYPE      KNA1\_KUNNR

##### **Global definition**

##### **Types:**

```
types: begin of ty_kna1,
      kunnr type kna1-kunnr,
      adrnr type kna1-adrnr,
      end of ty_kna1.
```

##### **Global data:**

Wa\_kna1      type      ty\_kna1

##### **Initialization:**

I\_KUNNR      WA\_KNA1

```
select single kunnr adrnr from kna1 into wa_kna1 where kunnr = i_kunnr.
```

Select the page in the left panel. Right click → create → address. Double click on address. Provide the address number in the general attributes tab. Save, check, activate. In the menu bar click on environment → function module name. Based on this function module we develop the print layout.

```

REPORT ZSM1.
parameter p_kunnr type kna1-kunnr.
CALL FUNCTION '/1BCDWB/SF00000260'
  EXPORTING
    I_KUNNR = p_kunnr.

```

**Note:** - If you want to declare the select-options in the smart form then we must declare one structure with the following fields in the data dictionary & later we refer the structure in the tables tab of form interface in the smart form.

→ Based on the given purchasing document numbers, display the purchasing document numbers, document dates & vendor numbers by using smart forms.

Sign → (C, 1)

Option → (C, 2)

Low }  
High }

Depends on the field

C  
VBELN

10

**Structure:** ZVBAK1

Sign char 1

Option char 2

Low char 10

High char 10

### Steps to design the smart form: -

#### Form interface:

#### Tables:

I\_SVBELN like ZVBAK1

#### Global definition

#### Types:

```

types: begin of ty_vbak,
      vbeln type vbak-vbeln,
      audat type vbak-audat,
      kunnr type vbak-kunnr,
      end of ty_vbak.

```

#### Global data:

Wa\_vbak type ty\_vbak

It\_vbak type table of ty\_vbak

#### Initialization:

```

I_VBELN          IT_VBAK
select vbeln audat kunnr from vbak into table it_vbak where vbeln in
i_svbeln.

```

Expand the page in left panel. Select the main window. Right click → create → flow logic → loop. Provide internal table name into work area name in the data tab. (IT\_VBAK INTO WA\_VBAK).

Select the loop into the left panel. Right click → create → text. Double click on text. Click on editor in the general attributes tab.

&wa\_vbak-vbeln& &wa\_vbak-audat& &wa\_vbak-kunnr&

Click on back, save, check, activate.

```

REPORT ZTEST199.
tables vbak.
select-options s_vbeln for vbak-vbeln.
CALL FUNCTION '/1BCDWB/SF00000254'
TABLES
I_SVBELN = s_vbeln.

```

**Note:** - We can test the smart form independently without using the print program.

#### **Working with graphic:** -

By using graphic window we can print the logo in the smart form. The input for the graphic window is graphics image.

We can print the .BMP images in the smart form. When ever we are working with .BMP image then we must convert graphic image by using SE78 transaction.

#### **Steps to place the logo in smart form:** -

Execute ‘SMARTFORMS’. Open the smart form in change mode. Select the page in the left panel. Right click → create → graphic. Double click on the graphics in the left panel. Provide the name (logo name), object (Graphics), ID (BMAP). Select the radio button color Bitmap image (BCOL). Click on form painter in the application tool bar. Align the windows. Once again click on form painter. Save, check, activate.

**Note:** - ‘RSTXPDFT4’ is the standard program. We can convert the smart form output to PDF format.

#### **Steps to print the watermark or background pictures in smart forms:** -

Execute ‘SMARTFORMS’. Open the form in change mode. Double click on page in the left panel. Click on background picture tab. Provide the name (SPG1), object (GRAPHICS), id (BMAP). Select the radio button colour. Select the output mode is print preview & print. Provide the position. Save, check, activate the smart form.

#### **Steps to convert SAP-SCRIPT to SMARTFORM:** -

Execute ‘SMARTFORMS’. Provide the smart form name. In the menu bar click on Utilities → Migration → import SAPSCRIPT FORM. Provide the script name (ZSJF\_7AM\_FORM). Enter. Click on save.

**Note:** - By using FB\_MIGRATE\_FORM function module we can convert the SAPSCRIPT to smart form.

#### **Working with SMARTSTYLES:** -

SMARTSTYLES are used to create the paragraphs as well as character formats in the smart form. The transaction code for smart styles is ‘SMARTSTYLES’.

#### **Steps to create paragraph & character formats:** -

Execute ‘SMARTSTYLES’. Provide the style name (ZSJF\_7AM\_SS1). Click on create. Provide short description. Select the paragraph formats in the left panel. Click on node. Provide the paragraph format name. Enter. Provide short description. Provide the left margin (0.2 CM). Click on font tab. Select the font family (Helve), font size (14), font style (bold). Select the checkbox color & select the colour. Click on tabs tab. Provide the tab positions. Click on save. Repeat the same steps for all the paragraph formats. Select the character format in the left panel. Right click → create node. Provide the character format name. Enter. Provide short description. Select the bar code name if it’s required. Click on font tab. Provide the font family, size, style. Click on color check box. Select color. Click on save. Repeat the same steps for all character formats. Double click on header data. Select the standard paragraph as default paragraph. Save, check, activate.

#### **Working with table:** -

Table is used to print the data in a tabular format. Table contains 3 sections.

1. Header
2. Main
3. Footer

Based on the client requirement which data is varying that part consider as a main area. Above main area consider as a header & below main area consider as footer.

**→ Based on the given purchasing document numbers, display the purchase order details as shown in the below.**

**Note:** - When ever we are working with table then we must create different lines with different cells based on the requirement.

#### For header

Line1	C1
-------	----

4	3	4.5	2.5	4.5
C1	C2	C3	C4	C5

#### For main area: -

We can use line2.

#### For footer

We can use line1.

**Note:** - SPELL\_AMOUNT is the function module which is used to convert the amount in words. The input for the above function module is amount & currency and output for the above function module is amount in words.

Smart form name: ZSJF\_7AM\_SF6

#### Form Attributes

##### Output options

Styles: ZSJF\_7AM\_SS2

#### Form interface: -

##### Tables:

I\_sebeln like ZSJF\_7AM\_SS

#### Global definitions

##### Types

```
types: begin of ty_ekpo,
      ebeln type ekpo-ebeln,
      ebelp type ekpo-ebelp,
      menge type ekpo-menge,
      meins type ekpo-meins,
      netpr type ekpo-netpr,
      end of ty_ekpo.
```

#### Global data

Wa_ekpo	type	ty_ekpo
It_ekpo	type	table of ty_ekpo
V_total	type	ekpo-netpr
AIW	type	SPELL

#### Initialization: -

I_SEBELN	IT_EKPO
IT_EKPO	V_TOTAL
WA_EKPO	AIW
V_TOTAL	

PURCHASE ORDER				
PURDOC	ITEM	QTY	UOM	PRICE
30004	01	10.00	PCS	200.00
30004	02	20.00	BOX	200.00
Amount				400.00
Amount in words: Four hundred rupees only.				

← 18.5 cm →

Header

Main

Footer

```

Select EBELN EHELP MENGE MEINS NETPR from EKPO into table it_ekpo
where Ebeln in I_SEBELN.
loop at it_ekpo into wa_ekpo.
v_total = v_total + wa_ekpo-netpr.
endloop.
CALL FUNCTION 'SPELL_AMOUNT'
EXPORTING
  AMOUNT          = v_total
  CURRENCY        = 'INR'
IMPORTING
  IN_WORDS        = AIW.

```

Double click on main window in the left panel. Click on output options tab. Provide the co ordinations of main window.

Left margin 1 CM      Width 18.5 CM

Upper margin 1CM      height 25 CM

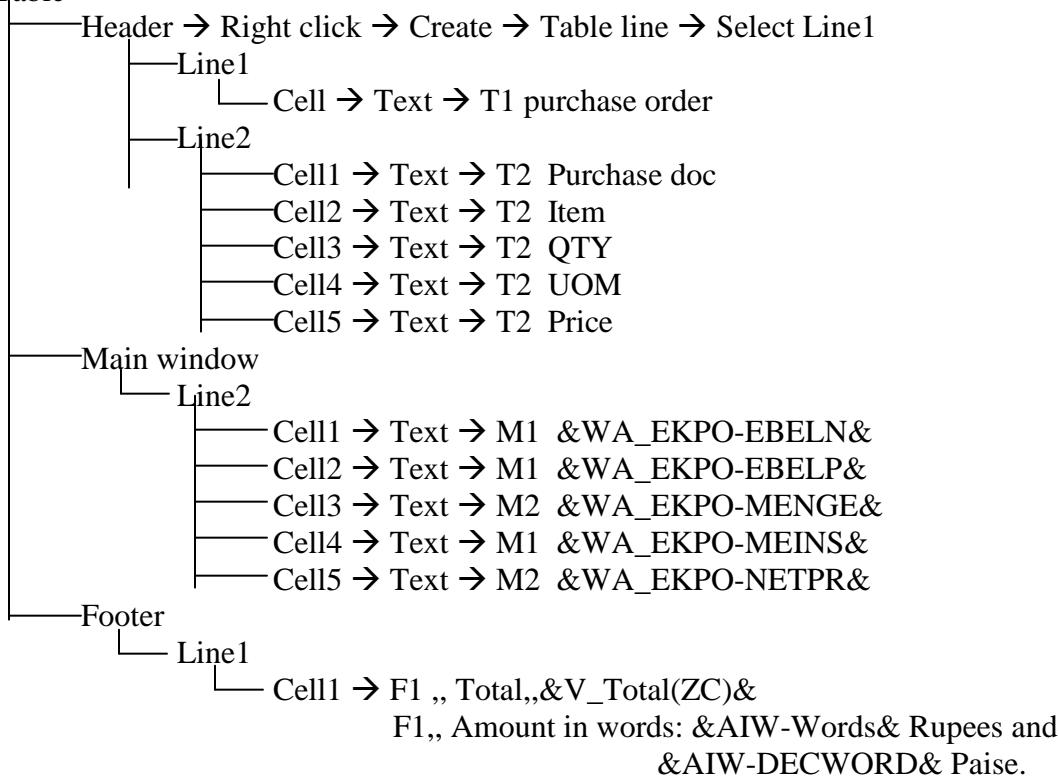
Select the main window in the left panel right click → create → table. Double click on table. In the data tab provide internal table name into work area name (IT\_EKPO into WA\_EKPO). Click on tables tab. Select the LTYPE1. Right click → rename line. Provide new name as line1. Select the ‘Line1’. Click on select pattern under table tab. Click on display framed pattern. Select the pattern. Select the ‘Line1’. Right click → insert → empty line underneath. Select the ‘Ltype1’. Right click → rename line. Provide the new name as ‘Line2’. Select the line2. Click on details in top right. Provide the each call width.

Line1 **18.5 cm**

Line2 **4cm 3cm 4.5cm 2.5cm 4.5cm**

Click on table painter in the top right. Select the ‘Line2’. Click on select pattern under table tab. Select the pattern.

#### Table



Save, check, activate the smart form.

## **T1**

Helve, 16, Bold, Center

## **T2**

Helve, 14, Bold, Center

## **M1**

Helve, 14, Bold, Left alignment, Left margin 0.2 cm

## **M2**

Helve, 14, Bold, Right alignment

## **F1**

### **Tab position**

1. 14 cm left alignment
2. 18.3 cm right alignment

### **Working with template: -**

Template is used to print the data in a fixed column & fixed rows.

→ Based on the given purchasing document number, display as show in below.

**Smart form name: ZSJF\_7AM\_SF7**

Form interface

#### **Import**

I\_EBELN TYPE EKKO-EBELN

Global definitions

#### **Types**

```

TYPES: BEGIN OF TY_EKKO,
       EBELN TYPE EKKO-EBELN,Column 1
       BEDAT TYPE EKKO-BEDAT,
       LIFNR TYPE EKKO-LIFNR,
       BUKRS TYPE EKKO-BUKRS,
       END OF TY_EKKO.
  
```

Global data

WA\_EKKO TYPE TY\_EKKO

Initialization

I\_EBELN WA\_EKKO

```

SELECT SINGLE EBELN BEDAT LIFNR BUKRS FROM EKKO INTO WA_EKKO WHERE
EBELN = I_EBELN.
  
```

Select the main window in the left panel. Right click → create → Template. In the template tab select the C1. Right click → rename line. Provide new name as line1. Select the line1. Click on details in the top right. Provide the height, each cell width. Click on table painter in the top right. Select the line1. Click on select pattern under template tab. Select the required pad. Repeat the same steps for all the lines. Select the template in the left panel. Right click → Create → Text. Double click on text. Click on editor under general attributes tab. (Pur.doc: &wa\_ekko-ebeln&). Click on back. Click on output options tab. Provide the line number and column number in the bottom. Repeat the same steps for all other text. Save, check activate the smart form.

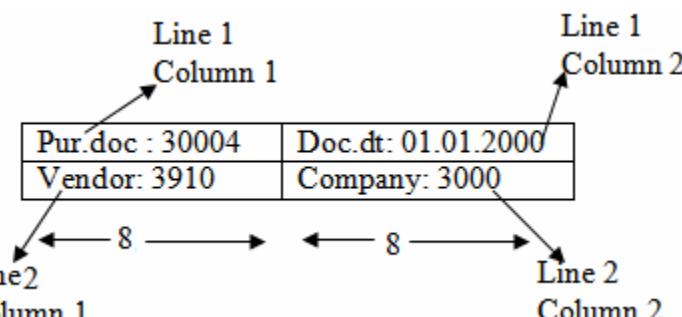
### **Events in SMARTFORMS: -**

1. Event on sort begin
2. Event on sort end

#### **Event on sort begin: -**

It's an event which is triggered at the first record of each block.

**ADV:** - It's used to display the individual headings



This is similar as at new field name in the control break statement.

#### **Event on sort end:** -

It's an event which is triggered at the last record of each block.

**ADV:** - It's used to display the subtotals.

This is similar as at end of field name in the control break statement.

**→ Based on the given purchasing document numbers, display the purchasing item details as shown in the below by using 'SMARTFORMS'.**

SMART FORM NAME: ZSJF\_7AM\_SF8

FORM INTERFACE

=====

Table

I\_SEBELN LIKE ZSJF\_7AM\_SS

Global definitions

Types:

```
TYPES: BEGIN OF TY_EKPO,
      EBELN TYPE EKPO-EBELN,
      EBELP TYPE EKPO-EBELP,
      MENGE TYPE EKPO-MENGE,
      MEINS TYPE EKPO-MEINS,
      NETPR TYPE EKPO-NETPR,
      END OF TY_EKPO.
```

Global data

Wa_ekko	type	ty_ekpo
It_ekpo	type table of	ty_ekpo
V1	type	ekpo-netpr
V2	type	ekpo-netpr

Initialization

I_sebeln	it_ekpo
Wa_ekpo	v2

It\_ekpo

```
SELECT EBELN EBELP MENGE MEINS NETPR FROM EKPO INTO TABLE IT_EKPO
WHERE EBELN IN I_SEBELN.
```

LOOP AT IT\_EKPO INTO WA\_EKPO.

V2 = V2 + WA\_EKPO-NETPR.

ENDLOOP.

Select the main window in the left panel. Right click → create → text. Double click on text. Click on editor.

\* grand total: &v2(zc)&

Click on back. Select the main window once again. Right click → create → flow logic → loop. Double click on loop. In the data tab provide

**It\_ekpo                   into                   wa\_ekpo**

In the below sort criteria block provide field name is Ebeln. Select the check box 'event on sort begin'. Select the 'event on sort begin' in the left panel. Right click → create → text. Double click on text.

\* &wa\_ekpo-ebeln&

Select the loop in left panel. Right click → create → text. Double click on the text. Click on editor.

\* &wa\_ekpo-ebelp& &wa\_ekpo-menge(zc)& &wa\_ekpo-meins& &wa\_ekpo-netpr(zc)&

Click on back. Select the text in left panel. Right click → create → flow logic → program lines. Double click on code in the left panel. Provide input, output parameters.

Wa\_ekpo                   v1

30004			
01	10	KG	250.00
02	20	PCS	150.00
Sub total			400.00
30005			
01	05	BOX	500.00
Sub total			500.00
Grand total			900.00

V1 = v1 + wa\_ekpo-netpr

Double click on loop in the left panel. In the sort criteria block provide field name as Ebeln and select the check box event on sort end. Select the event on sort end in left panel. Right click → create → text. Double click on text.

\* Sub total: &v1(zc)&

Click on back. Select the text in left panel. Right click → create → flow logic → program lines.

Double click on code. Implement the logic & input parameter is v1.

Clear v1.

Save, check, activate.

In the menu bar click on environment → function module name. based on this function module we develop the print program.

#### TABLES EKPO .

SELECT-OPTIONS S\_EBELN FOR EKPO-EBELN .

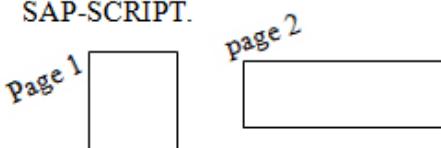
CALL FUNCTION '/1BCDWB/SF00000265'

#### TABLES

I\_SEBELN = S\_EBELN .

#### Differences between SAP Script & SMARTFORMS

##### SAP SCRIPT

1. Multiple page formats aren't possible in SAP-SCRIPT.  

2. Labels are possible in SAP-SCRIPT.
3. Without a Main window we can't create SAP-SCRIPT.
4. SAP-SCRIPT is a client dependent that means if you create a script in one client that's not reflected into all other clients in same server.
5. Control commands are worked in SAP-SCRIPT (Protect, top, bottom)
6. Colors aren't possible in this.
7. Water mark / background picture isn't possible in script.
8. Paragraph & character formats aren't reusable in script.
9. By using 'RSTXDEBUG' standard program we can debug the script.
10. We maintain the backup of script in .TXT file.
11. When ever we activate the script it won't generates function module.
12. We can't develop the code in script.
13. We can convert script to smart form.
14. Script is suitable for complex coding

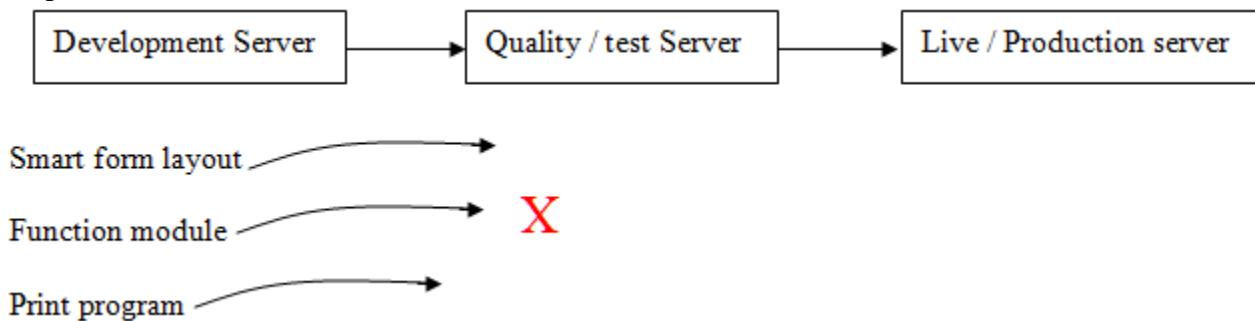
##### SMART FORM

1. Multiple page formats are possible in SMARTFORMS.
2. Labels aren't possible in smart form.
3. Without a Main window we can design smart forms.
4. Smart form is client independent. That means if we create smart form function module in any one of the client then automatically reflect into all other clients in same server.
5. Control commands are aren't possible in smart form.
6. Colors are possible in smart form.
7. Watermark / background picture is possible in smart form.
8. Paragraph & character formats are reusable in smart form.
9. By using static break points (break-point) we can debug the smart form.
10. We maintain the backup of smart form in .XML file.
11. When ever we activate the smart form it generates a function module.
12. We can develop the code in smart form.
13. We can't convert smart form to script.
14. Smart form is suitable for complex design.

**Note:** - In the smart forms function module number we can't transported to quality & live server.

Depends on server configuration function module number is generated. So we can't fix the function

module number in the print program. We always generate the function module by using 'SSF\_FUNCTION\_MODULE\_NAME'. The input for the above function module is smart form name. the output for the above function module is smart form number.



```

REPORT ZSF.
TABLES EKPO.
SELECT-OPTIONS S_EBELN FOR EKPO-EBELN.
DATA V_FM TYPE RS38L_FNAM.
CALL FUNCTION 'SSF_FUNCTION_MODULE_NAME'
  EXPORTING
    FORMNAME = 'ZSF2'
  IMPORTING
    FM_NAME = V_FM.
CALL FUNCTION V_FM
  TABLES
    I_SEBELN = S_EBELN.
  
```

**Note:** - Now-a-days most of the companies used zebra printers to print the labels. Because the cost of payable is cheap, quality is good. Gum thickness is also good. Zebra printers don't support sap script only. It supports smart forms. So we must design the labels through smart form.

→ Based on the given material number & number of labels design the accepted labels as shown in the below.

Material	:	_____
Plant	:	_____
Batch	:	_____
MFG.DT	:	_____
EXP.DT	:	_____

At the time of label design BASIS people create page format based on the table width & height through 'SPAD' transaction & given to us. Based on this page format we design the label.

Smart form name: ZSJF\_7AM\_SF9.

### FORM INTERFACE

#### Import

I_MATNR	TYPE	MCHA-MATNR
I_NOL	TYPE	I

#### Global definitions

##### Types

```

TYPES: BEGIN OF TY_MCHA,
        MATNR TYPE MCHA-MATNR,
        WERKS TYPE MCHA-WERKS,
        CHARG TYPE MCHA-CHARG,
      
```

```

HSDAT TYPE MCHA-HSDAT,
VFDAT TYPE MCHA-VFDAT,
END OF TY_MCHA.
TYPES: BEGIN OF TY,
       NO TYPE SYINDEX,
       END OF TY.

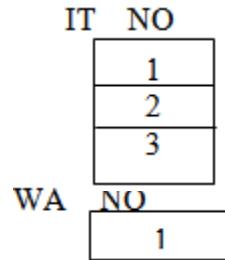
```

#### Global data

```

WA_MCHA TYPE          TY_MCHA
IT_MCHA  TYPE TABLE OF TY_MCHA
WA        TYPE          TY
IT        TYPE TABLE OF TY

```



#### Initialization

```

I_MATNR      WA_MCHA
I_NOL        WA
IT           IT

```

```

SELECT SINGLE MATNR WERKS CHARG HSDAT VFDAT FROM MCHA INTO WA_MCHA
WHERE MATNR = I_MATNR.

```

```
DO I_NOL TIMES.
```

```
WA-NO = SY-INDEX.
```

```
APPEND WA TO IT.
```

```
ENDDO.
```

Double click on main window in the left panel. Click on output options tab. Provide the co ordinates.

Left margin: 1CM width: 18.5 CM

Upper margin: 1 CM Window height 2.5 CM

Select the main window in the left panel. Right click → create → flow logic → loop. Double click on loop. Click on data tab. Provide (IT into WA). Select the loop. Right click → Create → Text. Double click on text.

```

MATERIAL : &WA_MCHA-MATNR&
PLANT    : &WA_MCHA-WERKS&
BATCH    : &WA_MCHA-CHARG&
MFG.DT   : &WA_MCHA-HSDAT&
EXP.DT   : &WA_MCHA-VFDAT&
PARAMETER: S_MATNR TYPE MCHA-MATNR,
            S_NOL TYPE I.

```

```
DATA V_FM TYPE RS38L_FNAM.
```

```
CALL FUNCTION 'SSF_FUNCTION_MODULE_NAME'
```

```
EXPORTING
```

```
  FORMNAME = 'ZSF3'
```

```
IMPORTING
```

```
  FM_NAME = V_FM.
```

```
CALL FUNCTION V_FM
```

```
EXPORTING
```

```
  I_MATNR = S_MATNR
```

```
  I_NOL = S_NOL.
```

Each & every window & text editor contains the following additional events in the conditions tab.

1. Only on first page
2. Not on first page
3. Only after end of Main window
4. Only before end of Main window
5. Only on page

**1. Only on first page:** - This is used to print the window or text editor information only on first page.

**2. Not on first page:-** This is used to print the window or text editor information from 2<sup>nd</sup> page on wards.

**3. Only after end of Main window:-** This is used to print the window or text editor information after main window data printing completion.

**4. Only Before end of Main window:-** - This is used to print the window or text information on each page until the main window is completed.

**5. Only on page:-** This is used to print the window or text editor information on a specified page.

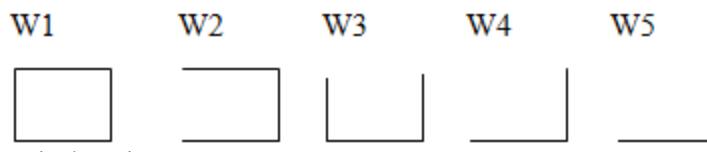
### **Working with BOX & Shading:-**

Each and every window & text editor contains BOX & shading in the output options tab.

#### **Steps to draw the box & shading: -**

Double click on required window or text editor. Click on output options tab. Select the check box lines with provide the spacing (0.2 cm) if it's required, color. If you want to add or remove any line then click on that line in the preview. If you want to shading, then select the shading color.

When ever we are working with Boxes in the smart form based on the requirement we consider **each** part as window & draw the boxes for the window & also move lines.



#### **Types of window in smart form**

1. Main window
2. Secondary window (It's similar as variable window in SAP Script)
3. Copies window
4. Final window

**Copies window: -** When ever we want to print the same document multiple copies with different headings then we use copies window.

In the real time the same invoice document is printed 3 copies with different headings. That is government form, company form, customer form. Here we use copies window to print the different headings on the same document.

#### **Steps to work with copies window: -**

Execute 'SMARTFORMS'. Open the smart form in change mode. Select the page in the left panel. Right click → create → window. Double click on window. Select window type as copies window in general attributes tab. Select the copies window in the left panel. Right click → create → flow logic → program lines. Double click on code.

```
IF SFSY-COPYCOUNT = 1.  
V = 'GOVERNMENT COPY'.  
ELSEIF SFSY-COPYCOUNT = 2.  
V = 'COMPANY COPY'.  
ELSEIF SFSY-COPYCOUNT = 3.  
V = 'CUSTOMER COPY'.  
ENDIF.
```

Output parameter V.

Select the code in left panel. Right click → create → text. Double click on text. Click on editor. &V&

Click on back. Save, check, activate.

At the time of print preview in the print program provide output device 'LP01'. Provide number of copies 3. click on print preview & absorb the different headings by click on next page.

**Note:** - SFSY is the structure which contains all the smart forms system variables.

If we want to print the page number

&SFSY-PAGE&

If we want to print total number of pages

&SFSY-FORM PAGES&

DATE → &SFSY-DATE&

Final window is used to print the total amount in the last page.

**Note:** - By using static break points (Break-point keyword) we can debug the smart form & also placing the dynamic break points in the function module we can debug the smart form.

**Folder:-** This is used to print the continuous text without any page break.

This functionality is similar as protect & end protect control command in SAP – SCRIPT.

**Alternative:** - This acts like if & endif control command in the SAP – SCRIPT.

**Command:** - This is used to break the page. This is similar as new page control command in SAP Script.

**Note:** - In the real time output device isn't always LP01. depending on the printer type (Laser printer, Dot Matrix printer) & language (English, Chinese, Japanese) the output device is created by BASIS people through 'SPAD' transaction.

### Steps to provide Microsoft word as a text editor in SAP SCRIPT and SMART FORM

#### Method 1:

- \* Execute I18N customizing
- \* Double click on MS word as editor
- \* Select the SAP SCRIPT and SMARTFORM check boxes and click on active.
- \* Yes

**Method 2:** Open any text editor. Then in menu bar click on goto → configure editor. Select graphical pc editor check box. Enter. You will get the MS Word editor.

**Note:** - In the realtime some times we develop the entire code in the print program only. Printed values are transferred to smartform layout. Here we must create one structure with the transferred fields in the data dictionary later we declare the work area and internal table in the form interface in the smart form based on structure.

**→ Based on the given company code, display the company code, company name, city by using smart form and develop the entire code in the print program only.**

Here we create one structure with BUKRS, BUTXT, ORT01 (which data we want to transfer). Based on this structure we declare the work area import tab of form interface to carry the data.

**Structure name: ZSJF\_7AM\_SS2.**

**SMART FORM Name: ZSJF\_7AM\_SF10.**

#### **Form interface**

##### Import

I\_WA like ZSJF\_7AM\_SS2.

Select the MAIN window → right click → text

```
&I_WA-BUKRS&
&I_WA-BUTXT&
&I_WA-ORT01&
```

#### Print Program

```
REPORT ZSF4.
```

```
PARAMETER P_BUKRS TYPE T001-BUKRS .
```

```
DATA: BEGIN OF WA_T001,
      BUKRS TYPE T001-BUKRS ,
      BUTXT TYPE T001-BUTXT ,
      ORT01 TYPE T001-ORT01 ,
      END OF WA_T001 .
```

```
SELECT SINGLE BUKRS BUTXT ORT01 FROM T001 INTO WA_T001 WHERE BUKRS =
P_BUKRS .
```

```
CALL FUNCTION '/1BCDWB/SF00000267'
  EXPORTING
    I_WA = WA_T001 .
```

→ Based on the given sales document number display the item details. I want to display the customer ship to address, billing address details and company name & it's wharehouse address. I don't want to print the smartform, directly I want to download that output in PDF format.

```

DATA V_E_DEVTYPE TYPE RSPOPTYPE. " It'll come under SSF_GET_DEVICE_TYPE FM
DATA: ST_JOB_OUTPUT_INFO      TYPE SSFCRESCL, "These all are come under our SF FM name
      ST_DOCUMENT_OUTPUT_INFO  TYPE SSFCRESPD,
      ST_JOB_OUTPUT_OPTIONS    TYPE SSFCRESOP.
DATA: OTF LIKE TABLE OF ITCOO, "These all are come under CONVERT_OTF_2_PDF FM
      TLINE LIKE TABLE OF TLINE,
      DOCS LIKE TABLE OF DOCS.
DATA bin_filesize type i. "It's for CONVERT_OTF_2_PDF FM & GUI_DOWNLOAD FM
DATA FILE TYPE IBIPPARMS-PATH. "FOR F4_FILENAME FM
DATA FILE1 TYPE STRING.
PARAMETER P_VBELN TYPE VBAK-VBELN.
DATA A TYPE KNB1-KUNNR.
DATA FM_NAM TYPE RS38L_FNAM.
DATA: WA_VBAK TYPE ZST_VBAK1,
      IT_VBAK LIKE TABLE OF WA_VBAK.
DATA: WA_VBAP TYPE ZST_VBAP1,
      IT_VBAP LIKE TABLE OF WA_VBAP.
DATA: WA_BILLADD TYPE ZST_BILLADD,
      IT_BILLADD LIKE TABLE OF WA_BILLADD.
DATA: WA_SHIPADD TYPE ZST_SHIPADD,
      IT_SHIPADD LIKE TABLE OF WA_SHIPADD.
DATA: WA_OUTPUT TYPE SSFCOMPOP, "These all are come under our SF FM name.
      WA_CONTROL TYPE SSFCTRLOP.

WA_OUTPUT-TDDEST = 'LP01'.
WA_CONTROL-NO_DIALOG = 'X'.
WA_CONTROL-GETOTF = 'X'.
WA_CONTROL-PREVIEW = 'X'.

DATA: BEGIN OF WA_VBPA,
      VBELN TYPE VBPA-VBELN,
      POSNR TYPE VBPA-POSNR,
      PARVW TYPE VBPA-PARVW,
      KUNNR TYPE VBPA-KUNNR,
      ADRNR TYPE VBPA-ADRNR,
      END OF WA_VBPA.
DATA IT_VBPA LIKE TABLE OF WA_VBPA.
DATA WA_VBPA1 LIKE WA_VBPA.
DATA IT_VBPA1 LIKE TABLE OF WA_VBPA1.
DATA: WA_FINAL LIKE ZST_VBAP1,
      IT_FINAL LIKE TABLE OF WA_FINAL.
DATA: BEGIN OF WA_T001W,
      KUNNR TYPE T001W-KUNNR,
      NAME1 TYPE T001W-NAME1,
      STRAS TYPE T001W-STRAS,
      ORT01 TYPE T001W-ORT01,
      LAND1 TYPE T001W-LAND1,
      PSTLZ TYPE T001W-PSTLZ,
      END OF WA_T001W.
DATA: BEGIN OF WA_KNB1,
      BUKRS TYPE KNB1-BUKRS,
      KUNNR TYPE KNB1-KUNNR,
      END OF WA_KNB1.

DATA: BEGIN OF WA_T001,
      BUKRS TYPE T001-BUKRS,
      BUTXT TYPE T001-BUTXT,
      END OF WA_T001.

```

```

DATA: WA_COMP LIKE ZST_COMP.

SELECT SINGLE VBELN AUDAT KUNNR FROM VBAK INTO WA_VBAK WHERE VBELN = P_VBELN.
IF SY-SUBRC = 0.
  SELECT SINGLE VBELN POSNR PARVW KUNNR ADRNR FROM VBPA INTO WA_VBPA WHERE VBELN = WA_VBAK-VBELN AND PARVW = 'AG'. "SOLD TO PARTY ADDRESS
  IF SY-SUBRC = 0.
    SELECT SINGLE ADDRNUMBER NAME1 CITY1 POST_CODE1 FROM ADRC INTO WA_BILLADD WHERE ADDRNUMBER = WA_VBPA-ADRNR.
  ENDIF.
  SELECT SINGLE VBELN POSNR PARVW KUNNR ADRNR FROM VBPA INTO WA_VBPA1 WHERE VBELN = WA_VBAK-VBELN AND PARVW = 'WE'. "SHIP TO PARTY
  IF SY-SUBRC = 0.
    SELECT SINGLE ADDRNUMBER NAME1 CITY1 POST_CODE1 FROM ADRC INTO WA_SHIPADD WHERE ADDRNUMBER = WA_VBPA1-ADRNR.
  ENDIF.
  SELECT SINGLE BUKRS KUNNR FROM KNB1 INTO WA_KNB1 WHERE KUNNR = WA_VBAK-KUNNR.
  IF SY-SUBRC = 0.
    SELECT SINGLE KUNNR NAME1 STRAS ORT01 LAND1 PSTLZ FROM T001W INTO WA_T001W WHERE KUNNR = WA_KNB1-KUNNR.
    SELECT SINGLE BUKRS BUTXT FROM T001 INTO WA_T001 WHERE BUKRS = WA_KNB1-BUKRS.
  ENDIF.
  SELECT VBELN POSNR KWMENG NETWR FROM VBAP INTO TABLE IT_VBAP WHERE VBELN = WA_VBAK-VBELN.
ENDIF.

WA_COMP-BUKRS = WA_KNB1-BUKRS.
WA_COMP-BUTXT = WA_T001-BUTXT.
WA_COMP-KUNNR = WA_T001W-KUNNR.
WA_COMP-NAME1 = WA_T001W-NAME1.
WA_COMP-STRAS = WA_T001W-STRAS.
WA_COMP-ORT01 = WA_T001W-ORT01.
WA_COMP-LAND1 = WA_T001W-LAND1.
WA_COMP-PSTLZ = WA_T001W-PSTLZ.

LOOP AT IT_VBAP INTO WA_VBAP.
  WA_FINAL-VBELN = WA_VBAP-VBELN.
  WA_FINAL-POSNR = WA_VBAP-POSNR.
  WA_FINAL-KWMENG = WA_VBAP-KWMENG.
  WA_FINAL-NETWR = WA_VBAP-NETWR.
  WA_FINAL-TOTAL = WA_VBAP-KWMENG * WA_VBAP-NETWR.
  APPEND WA_FINAL TO IT_FINAL.
  CLEAR WA_FINAL.
ENDLOOP.

CALL FUNCTION 'SSF_GET_DEVICE_TYPE'
  EXPORTING
    I_LANGUAGE      = SY-LANGU
    I_APPLICATION   = 'SAPDEFAULT'
  IMPORTING
    E_DEVTYPE       = V_E_DEVTYPE.

CALL FUNCTION 'SSF_FUNCTION_MODULE_NAME'
  EXPORTING
    FORMNAME = 'ZCDC1SF6'
  IMPORTING
    FM_NAME   = FM_NAM.

CALL FUNCTION FM_NAM
  EXPORTING
    CONTROL_PARAMETERS = WA_CONTROL

```

```

OUTPUT_OPTIONS      = WA_OUTPUT
USER_SETTINGS      =
P_VBELN           = P_VBELN
WA_VBAK            = WA_VBAK
WA_BILLADD         = WA_BILLADD
WA_SHIPADD         = WA_SHIPADD
WA_COMP             = WA_COMP
IMPORTING
DOCUMENT_OUTPUT_INFO = ST_DOCUMENT_OUTPUT_INFO
JOB_OUTPUT_INFO    = ST_JOB_OUTPUT_INFO
JOB_OUTPUT_OPTIONS = ST_JOB_OUTPUT_OPTIONS
TABLES
IT_FINAL           = IT_FINAL.

CALL FUNCTION 'CONVERT_OTF_2_PDF'
IMPORTING
BIN_FILESIZE       = BIN_FILESIZE
TABLES
OTF                = ST_JOB_OUTPUT_INFO-otfdata
DOCTAB_ARCHIVE     = DOCS
LINES              = TLINE.
CALL FUNCTION 'F4_FILENAME'
EXPORTING
PROGRAM_NAME        = SYST-CPROG
IMPORTING
FILE_NAME           = FILE.

FILE1 = FILE.
CALL FUNCTION 'GUI_DOWNLOAD'
EXPORTING
BIN_FILESIZE       = BIN_FILESIZE
FILENAME            = 'C:\Users\ULL@$\Desktop\HAI1.PDF'
FILETYPE            = 'BIN'
TABLES
DATA_TAB           = TLINE.

```

→ Based on the given purchasing document number display the document number, doc date, vendor (ALV). Select any check box of any record and click on PRINT button then display the purchasing item details in smart form.

```

REPORT ZALVR2.
TABLES EKKO.
SELECT-OPTIONS S_EBELN FOR EKKO-EBELN.
TYPE-POOLS SLIS.
TYPES: BEGIN OF TY_EKKO,
        EBELN TYPE EKKO-EBELN,
        BEDAT TYPE EKKO-BEDAT,
        LIFNR TYPE EKKO-LIFNR,
        A(2) TYPE C,
        END OF TY_EKKO.
DATA: WA_EKKO TYPE TY_EKKO,
      IT_EKKO TYPE TABLE OF TY_EKKO.
SELECT EBELN BEDAT LIFNR FROM EKKO INTO TABLE IT_EKKO WHERE EBELN IN
S_EBELN.
DATA: IT_FCAT TYPE SLIS_T_FIELDCAT_ALV,
      WA_FCAT LIKE LINE OF IT_FCAT.

WA_FCAT-FIELDNAME = 'A'.

```

```

WA_FCAT-COL_POS      = '1'.
WA_FCAT-EDIT      = 'X'.
WA_FCAT-CHECKBOX   = 'X'.
WA_FCAT-OUTPUTLEN  = '3'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME  = 'EBELN'.
WA_FCAT-COL_POS    = '2'.
WA_FCAT-SELTEXT_M  = 'PUR DOC'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME  = 'BEDAT'.
WA_FCAT-COL_POS    = '3'.
WA_FCAT-SELTEXT_M  = 'DOC DATE'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME  = 'LIFNR'.
WA_FCAT-COL_POS    = '4'.
WA_FCAT-SELTEXT_M  = 'VENDOR'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

DATA: IT_EVENT TYPE SLIS_T_EVENT,
      WA_EVENT LIKE LINE OF IT_EVENT.
WA_EVENT-NAME = 'PF_STATUS_SET'.
WA_EVENT-FORM = 'PF'.
APPEND WA_EVENT TO IT_EVENT.
WA_EVENT-NAME = 'USER_COMMAND'.
WA_EVENT-FORM = 'UC'.
APPEND WA_EVENT TO IT_EVENT.

CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
  EXPORTING
    I_CALLBACK_PROGRAM = SY-CPROG
    IT_FIELDCAT       = IT_FCAT
    IT_EVENTS          = IT_EVENT
  TABLES
    T_OUTTAB          = IT_EKKO.

FORM PF USING M TYPE SLIS_T_EXTAB.
  SET PF-STATUS 'STAT'.
ENDFORM.

FORM UC USING D LIKE SY-UCOMM E TYPE SLIS_SELFIELD.
  IF D = 'PRINT'.
    READ TABLE IT_EKKO INTO WA_EKKO INDEX E-TABINDEX.

    CALL FUNCTION '/1BCDWB/SF00000275'
      EXPORTING
        I_EBELN = WA_EKKO-EBELN.

```

ENDIF.

ENDFORM.

### Smart form

Global Settings

### Form interface

I\_EBELN TYPE EKKO-EBELN

### Global Definitions

#### Global data

```
WA_EKPO  TYPE          TY_EKPO
WA_EKKO  TYPE          TY_EKKO
IT_EKPO   TYPE TABLE OF  TY_EKPO
```

#### Types

```
TYPES: BEGIN OF TY_EKPO,
        EBELN TYPE EKPO-EBELN,
        EBELP TYPE EKPO-EBELP,
        MENGE TYPE EKPO-MENGE,
        MEINS TYPE EKPO-MEINS,
        NETPR TYPE EKPO-NETPR,
        END OF TY_EKPO.
```

```
TYPES: BEGIN OF TY_EKKO,
        EBELN TYPE EKKO-EBELN,
        BEDAT TYPE EKKO-BEDAT,
        END OF TY_EKKO.
```

Initialization SELECT EBELN EBELP MENGE MEINS NETPR FROM EKPO INTO TABLE IT\_EKPO where EBELN = I\_EBELN.

Input parameters (I\_EBELN), output parameters (IT\_EKPO, WA\_EKPO)

SELECT SINGLE EBELN BEDAT FROM EKPO INTO WA\_EKKO WHERE EBELN = I\_EBELN.

#### Table:-

##### Header

Line1 - create → text → purchasing document

Line 2: - row1 → create → text → pur doc: &wa\_ekko-ebeln&  
Row2 → create → text → doc date: &wa\_ekko-bedat&

Line 3: row1 → text → purchasing document

Row2 → text → item  
Row3 → text → qty  
Row4 → text → uom  
Row5 → text → price

#### Main: -

Line3: row1 → text → &wa\_ekpo-ebeln&  
Row2 → text → &wa\_ekpo-ebelp&  
Row3 → text → &wa\_ekpo-menge&  
Row4 → text → &wa\_ekpo-meins&  
Row5 → text → &wa\_ekpo-netpr&

→ Based on the given sales document date display the doc number, document date, item, quantity, price. I want to display one field that is Total amount. For example I have one material with 3 number of quantity. In this situation, I calculate quantity multiplication netprice. That will value will pass into Total variable. In smartform I want to display the 10 records. After 11<sup>th</sup> record to 20<sup>th</sup> record display in 2<sup>nd</sup> page. Per page I want to display 10 records only and I want to display the total of that page in that page footer. I want to display the total amount in that document in another window.

```

TABLES VBAK.
SELECT-OPTIONS S_AUDAT FOR VBAK-AUDAT.
DATA V2 TYPE I. "For containing total number of records

TYPES: BEGIN OF TY_VBAK,
        VBELN TYPE VBAK-VBELN,
        AUDAT TYPE VBAK-AUDAT,
        END OF TY_VBAK.
DATA: WA_VBAK TYPE TY_VBAK,
      IT_VBAK TYPE TABLE OF TY_VBAK.

TYPES: BEGIN OF TY_VBAP,
        VBELN TYPE VBAP-VBELN,
        POSNR TYPE VBAP-POSNR,
        NETWR TYPE VBAP-NETWR,
        KWMENG TYPE VBAP-KWMENG,
        END OF TY_VBAP.
DATA: WA_VBAP TYPE TY_VBAP,
      IT_VBAP TYPE TABLE OF TY_VBAP.

DATA: WA_FINAL LIKE ZST_VBAP2,
      IT_FINAL LIKE TABLE OF WA_FINAL.

DATA: WA_CONTROL TYPE SSFCTRLROP,
      WA_OUTPUT TYPE SSFCOMPOP.

WA_CONTROL-NO_DIALOG = 'X'.
WA_CONTROL-PREVIEW = 'X'.
WA_OUTPUT-TDDEST = 'LP01'.

DATA FORM_NAME TYPE RS38L_FNAM.

SELECT VBELN AUDAT FROM VBAK INTO TABLE IT_VBAK WHERE AUDAT IN S_AUDAT.
IF IT_VBAK IS NOT INITIAL.
  SELECT VBELN POSNR NETWR KWMENG FROM VBAP INTO TABLE IT_VBAP FOR ALL ENTRIES IN IT_VBAK WHERE VBELN = IT_VBAK-VBELN.
ENDIF.

LOOP AT IT_VBAP INTO WA_VBAP.
  WA_FINAL-VBELN = WA_VBAP-VBELN.
  WA_FINAL-POSNR = WA_VBAP-POSNR.
  WA_FINAL-NETWR = WA_VBAP-NETWR.
  WA_FINAL-KWMENG = WA_VBAP-KWMENG.
  WA_FINAL-TOTAL = WA_VBAP-KWMENG * WA_VBAP-NETWR.
  READ TABLE IT_VBAK INTO WA_VBAK WITH KEY VBELN = WA_VBAP-VBELN.
  WA_FINAL-AUDAT = WA_VBAK-AUDAT.
  APPEND WA_FINAL TO IT_FINAL.
  CLEAR WA_FINAL.
ENDLOOP.

DESCRIBE TABLE IT_FINAL LINES V2.

CALL FUNCTION 'SSF_FUNCTION_MODULE_NAME'

```

```

EXPORTING
  FORMNAME = 'ZCDC1SF7'
IMPORTING
  FM_NAME = FORM_NAME.

CALL FUNCTION '/1BCDWB/SF00000254'
EXPORTING
  CONTROL_PARAMETERS = WA_CONTROL
  OUTPUT_OPTIONS = WA_OUTPUT
  USER_SETTINGS = ''          " *****
  V2 = V2
  S_AUDAT = S_AUDAT
TABLES
  IT_FINAL = IT_FINAL.

```

**In smartform: -**

In form attributes I've taken the smartstyles name. In form interface I've taken like

```

V2      TYPE INTEGER
S_AUDAT  TYPE ZSO_AUDAT
In global definitions I've taken like this
WA_FINAL LIKE ZST_VBAP2
A      TYPE SY-INDEX
PRICE TYPE VBAP-NETWR
AIW    TYPE SPELL
AMT    TYPE VBAP-NETWR
AINW   TYPE SPELL
FLG    TYPE CHAR1

```

In global definitions → currency tab I've taken like this

```

WA_FINAL-KWMENG   WA_FINAL-KWMENG   QUAN
WA_FINAL-NETWRWA_FINAL-NETWRCURR
WA_FINAL-TOTAL WA_FINAL-TOTAL CURR

```

Expand page1, right click on Main window → create → table. Select table in left panel. In the data tab deactivate the check box internal table. In the table tab create the two lines. 1<sup>st</sup> line co-ordinates like this 0,80 CM 2,30 CM 2,75 CM 1,70 CM 2,45 CM 3,00 CM 3,00 CM. In 2<sup>nd</sup> line I'll take the co-ordinates like this. 2,68 CM 10,32 CM 3,00 CM Provide the pattern for 2 lines. Expand the table. Right click on header → create → table line. Select the 1<sup>st</sup> line which is create previously. Then automatically It'll display some cells. On Each cell Right click → create text. Provide the text what ever you want. You can apply smartstyles here as your wish. Select the main area, right click → create → flow logic → loop. Double click on loop. Select the check box internal table. Provide internal table name, work area name. select loop in left panel, right click → create → flow logic → command. Double click on command in left panel. Click on conditions tab. Provide like this. FLG = 'X'. Click on general attributes tab. Select the check box 'go to new page'. Provide the new page name as 'page2'. Right click on command → create → table line. Provide the line as 1<sup>st</sup> line. It'll automatically generates some cells. on first cell right click → create → flow logic → program line. Double click on program line. Provide the code as bellow.

```

A = A + 1.
DATA: NL TYPE I.
NL = A MOD 10.
IF NL = 0.
  FLG = 'X'.
ELSE.

```

```
FLG = ''.
ENDIF.
```

Provide output parameters as 'A', 'FLG'. Input parameters as 'A'.

Right click on 1<sup>st</sup> cell under main area → create → text. Provide the text as &SNO&.

Right click on 2<sup>nd</sup> cell under main area → create → text. Provide the text as &wa\_final-vbeln& Do same steps for all cell except last cell in the main area. Right click on last cell in main area → create → flow logic → program lines. In the program line provide like this.

```
AMT = AMT + WA_FINAL-TOTAL.
```

Provide the output parameter as 'AMT', input parameters like 'AMT', 'wa\_final-total'.

Right click on last cell in main area → create → text. Provide the text as &wa\_final-total(zc)&.

Right click on last cell in main area → create → flow logic → program line. Provide the code as bellow.

```
PRICE = PRICE + WA_FINAL-TOTAL.
```

```
CALL FUNCTION 'SPELL_AMOUNT'
EXPORTING
  AMOUNT      = PRICE
  CURRENCY    = 'INR'
  LANGUAGE    = SY-LANGU
IMPORTING
  IN_WORDS    = AIW.
```

Output parameter as 'PRICE', Input parameters as 'PRICE', 'WA\_FINAL-TOTAL', 'AIW'.

Right click on footer → create → table line. Select the 2<sup>nd</sup> line which is created previously. Right click on first cell → create → text. Provide text as 'Amount of this page'. Right click on send cell → create → text provide the text as '&AINW-WORD& RUPEES AND &AINW-DECWORD& PAISE.' . Right click on 2<sup>nd</sup> cell → create → flow logic → program line. In that program line provide text as

```
CALL FUNCTION 'SPELL_AMOUNT'
EXPORTING
  AMOUNT      = AMT
  CURRENCY    = 'INT'
IMPORTING
  IN_WORDS    = AINW.
```

Output parameter as 'AMT'. Right click on 3<sup>rd</sup> cell → create → flow logic → program line. In that one provide as '**CLEAR AMT**'. Input parameter as 'AMT'. Right click on 3<sup>rd</sup> cell → create → text. Provide the text as &AMT(ZC)&. Select the Main window, copy the main window. Select the page1, right click → create → page. Select the new page (page2), right click → paste. Save, check, activate the SF.

## How to work with dynamic logos in Smartform: -

If we want to display the different logos based on condition then we go for following steps.

First we upload the images through SE78 transaction. We have to give names for the logos. Open the smartform. Just click on Global definitaion. Provide one variable. I'm taking I\_LOGO. Click on Initialization tab under Global definition. Provide the logic. I'm giving the logic here.

```
IF WA_KNA1-LAND1 = 'IN'.
  I_LOGO = 'INDIA'.
ELSE.
  I_LOGO = 'GERMANI'.
ENDIF.
```

I/P parameter as WA\_KNA1-LAND1, O/P parameter as I\_LOGO.

Here 'INDIA', 'GERMANI' are Logo names as in SE78 transaction.

Now right click on Page1 → create → graphic. Provide the name as &I\_LOGO& and Object as GRAPHICS and ID as BMAP and select the Color Bitmap Image (BCOL) radio button. Save, check, activate.

## **How to work with Barcodes in Smartform: -**

First we have to create one Barcode name in SE73 transaction. Execute SE73. Select the SYSTEM BAR CODES radio button. If you want to create new one then change button. If you want to see already created one then click on Display button. Now I'm creating new one. So I'm clicking on Change button. Now click on create button which is in application tool bar. Click on New button. Provide the Bar code name and short text. Now I'm giving Bar code name as 'Harsha' and short text as 'Bar code created by Sathish Reddy'. Now select the Supported Bar Code Symbologies radio button. I'm selecting second one (Code 128). Click on enter. Select the Bar code alignment radio button. I'm selecting the Normal radio button. Click on enter. Click on enter. It'll ask a dialogue box to save the Bar code. Click on Yes. Click on Enter. Now your bar code is created. Now you have to pass this Bar code name to Smartstyles. Execute SMARTSTYLES transaction code. Provide the smartstyle name. Right click on paragraph format → create node. Provide paragraph format as 'DE'. Provide the short description as Default. Double click on header data. Select standard paragraph as DE. Right click on Character format → create node. Provide the Character format. I'm giving 'BC'. Click on enter. Provide the short description as 'Bar code purpose'. Provide the bar code name here which was created in SE73 transaction. At that time I was created as 'Harsha'. So I'm giving that name here. Save, check, activate the smartstyle. Open the smart form. In which place you want that bar code, in that place you provide the character format for that number. See, in the below I'm giving the bar code for Sales document number.

\* Sales Document Number : <BR>&WA\_VBAK-VBELN& </>

Here <BR> means, character format which was created in smart styles. It's ending with </>. VBELN number will display as Barcodes in Smartform output now.

## Debugging

Debugging is a tool to trace the program execution line by line. Debugging is used to change the field values at run time. Debugging is used to stop the program execution at any executable statement by using break points.

There are 2 types of break points.

1. Static break point
2. Dynamic break point

### Static break point

1. Static break points are placed by using BREAK-POINT keyword.
2. Static break points aren't a user specific. That means any user can execute the program. Then the cursor stops at break point keyword. (By using conditions we can set the static break points are user specific).
3. In any version at the program (Active / Inactive), we place static break point.

### Dynamic break point

1. Dynamic break points are placed by using the  button in the application tool bar.
2. Dynamic break points are user specific.
3. In an activate version of the program only we can place the dynamic break point.

**Note:** - SY-UNAME is the system variable which contains the current user name.

If SY-UNAME = 'SAPUSER'.

BREAK-POINT.

Endif.

### **Steps to place the dynamic break points: -**

Place the cursor where we want to stop the program execution. Click on stop button in the application tool bar. Then it automatically set the break point. If you want to remove the break point then place the cursor on the same line & click on stop button in the application tool bar.

**Note:** - We can place up to 30 break points in the program. At the time of debugging mode F5 → line by line execution, F6 → At a time one block is executed (At a time subroutine & function module is executed), F7 → come of the block, F8 → first it'll check is there any other break points available or not. If there is available then it goes to next break point. Otherwise come out of the program.

### **Watch Point: -**

Watch point is used to stop the program execution based on the condition. We can place up to 10 watch points in the program.

### **Steps to create watch point: -**

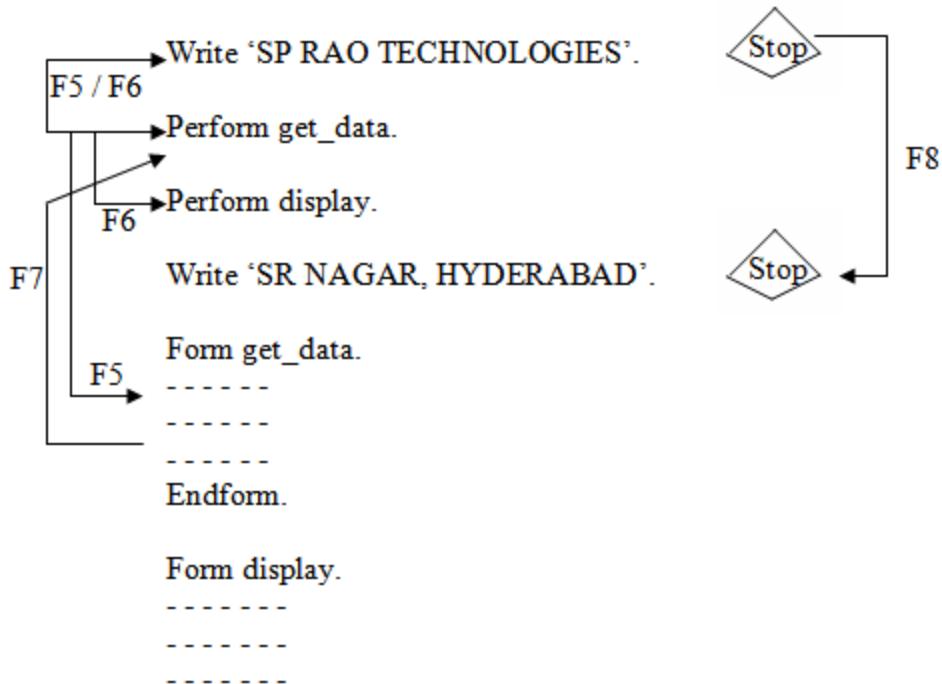
In the debugging mode click on watch point which is in the application tool bar. Provide the field name. Provide the relational operator & Comp.field/value. Enter. Click on F5 button. When ever the watch point is reached then the program is stopped.

### **Fields: -**

This is used to identify the fields or variable values and also we can change the values.

### **Steps to change the field values: -**

Provide the field name in left side. Click on enter. Then we get the value in the right side. Remove the value place the new value and click on change field (pencil symbol in right side).



#### **Table: -      Endform.**

This is used to display the internal table fields & their value & also perform the internal table operations (Append, Insert, and Delete).

#### **Break Points: -**

This is used to identify the all the break points which are placed in the program & their line number.

#### **Watch points: -**

This is used to identify the all available watch points & also we can change the watch point condition.

#### **Callstack: -**

This is used to identify the current execution event.

#### **Over view: -**

This is used to identify the all the events and all the blocks which are available in the program.

### **ABAP new debugger:** -

#### Desktop1: -

In this ABAP source code is displayed in the left side, global & local variables & their values are displayed in right side.

#### Desktop2: -

In this ABAP source code is displayed in the left side, ABAP stack is displayed in right side (currently which block is executed under which event).

#### Desktop3: -

In this source code is displayed in the top & global & local variables & their values are displayed in bottom.

#### Standard: -

In this source code is displayed in the left side. ABAP stack is displayed in right side top. Global & local variables are displayed in right side bottom.

Structures: -

This is used to identify the work area fields & their values & also change the values.

Tables: -

This is used to display the internal table fields & their values & also perform the internal table operations.

Objects: -

This is used to identify the all the methods of objects & also check their values.

Detail display: -

This is used to identify the detailed information of any particular field.

Break point / watch points: -

This is used to identify the all the break points as well watch points in the program.

Difference: -

This is used to compare the any two field values & also display their history.

**Differences between classic debugger & new debugger**

**Classic debugger**

1. In this we have no desktops.
2. By using this we can't debug object oriented program.
3. In this we can't compare any two field values.
4. In this it won't show global & local variables of the program.

**New debugger**

1. In this we have desktop1, desktop2, desktop3 & standard.
2. By using this we can debug the object oriented programs.
3. In this we can compare any two field values.
4. In this it provides the global & local variables of the program

There are two types of debugging

1. Place the break points in the program & run the program in debugging mode.
2. Execute the program & provide the input & set the program in debugging mode by using '/H'. '/H' is the runtime debugger.

The following ways are used to identify the errors in standard program

1. By using where used list we identify the error location.
2. By using watch point.
3. By using break point.
4. By using source code scanner.
5. By using ABAP runtime analysis [SE30].
6. By using SQL trace [ST05].

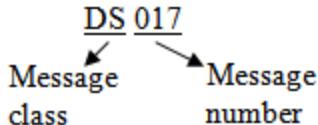
When ever we execute any transaction code if you get the error or message if you want to identify the location of the message then we use following techniques.

EX: -

When ever we try to open a program which isn't created, then it throws a message. This message is triggered from which location we identify now.

By using where used list: -

Execute SE38. provide the program which isn't created. Click on display. Then we get the message. Double click on that message. Identify the message number. In that last three digit is message number. Rest of the things is message class.



Execute SE91. Provide message class. Click on display. Select the message number. Click on where used list in the application tool bar. Enter. It displays the so many programs. Double click on each & every program & identify our message is available in which program.

#### **By using watch points:** -

Execute SE38. Provide the program name. Execute '/H' before display. Then debugging switched on. Click on display. In the menu bar click on classic debugger. Click on watch point in the application tool bar. Provide field name as SY-MSGID. Provide relational operator (=). Provide comparison value S017. Click on F8. & identify the right location.

#### **By using break points:** -

Execute SE38. Provide the program name. Execute '/H'. Click on display. In the menu bar click on break points. Break point at message / statement. If it is a statement then provide 'write'. Enter. Click on F8. Identify the right location of the error.

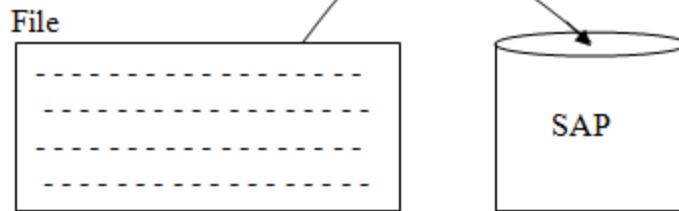
#### **By using source code:** -

Execute SE93. Provide the transaction code. Click on display. Identify the package. Execute SE38. Provide the program name RS\_ABAP\_SOURCE\_SCAN. Execute. Provide the package & provide string searched for (S017). Execute. it provide the all the locations where the message is available. Double click on each and every message or location. Place the break point. Execute SE38. Provide the program name. Click on display. Then the cursor is stops at right location.

*Note:* - By using CODE\_SCANNR transaction also we identify the right location.

## BDC **(Batch Data Conversion / Communication)**

BDC is used to upload the data from flat file to SAP system.



Developing a BDC program is nothing but to automate the existing transaction code. Each transaction can create only one record at a time. If you want to create thousands of records, one way is to execute the same transaction thousands of times. Another way is to develop a BDC program to automate the existing transaction.

### Steps of the standard transaction codes: -

1. XK01 / MK01 / FK01 → Create Vendor.
2. XD01 / VD01 / FD01 → Create Customer.
3. MM01 → Create Material
4. ME51N → Create Purchase Requisition
5. ME21N → Create Purchase Order
6. MB01 → Create Material Document
7. VA01 → Create Sales Order
8. VL01 → Create Delivery
9. VF01 → Create Billing
10. FI01 → Create Bank
11. KS01 → Create Cost Center
12. KE51N → Create Profit Center
13. FB01 → Create Accounting Document
14. CS01 → Create BOM (Bill of Material)
15. MSC1N → Create Batch
16. COR1 → Create Process Order
17. C201 → Create Recipe

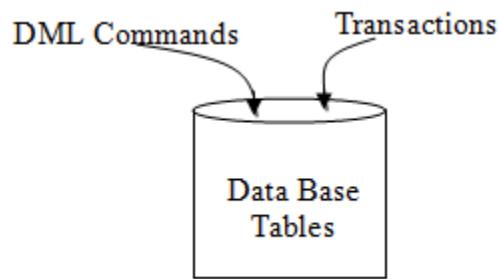
**Note:** - 1 → Create      2 → Change      3 → Display

### Steps to create a vendor : -

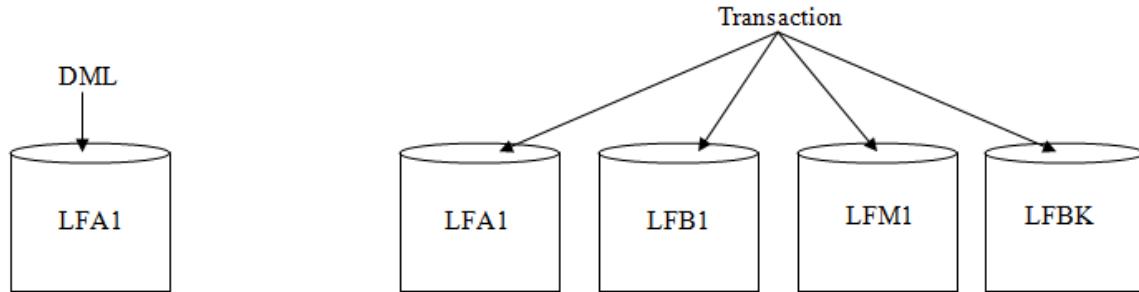
Execute XK01. Provide vendor number, account group. Click on enter. Provide the name, search term ½, country. Save.

**Note:** - We can perform the data base table operation either through DML commands or through Transaction codes.

DML commands are used to update only one data base table at a time, whereas transaction code is used to update their relevant data base tables at a time.



FM: TEXT\_CONVERT\_XLS\_TO\_SAP  
Type: TRUX T\_TEXT DATA  
Background: Submit RSBDCSUB via selection-screen



### Steps to develop the BDC program : -

1. Analyze the transaction code

    ↳ Analyzing the screen as well as field details

2. Prepare the flat file.

3. Upload the data from flat file to internal table / BDC program.

4. For each record in the internal table, we collect the screen and field details to automate the transaction.

5. For each record in the internal table, call the transaction.

### Steps in detail: -

#### **Step 1: -**

Analyzing the screen and field details is nothing but identifying the technical information of each screen and field. If you want to identify the technical information, execute the transaction. Place the cursor on input fields. Click on F1 button. Click on technical information. Identify the screen & field details. It's very difficult to identify the technical information of entire transaction. So we go for 'SHDB' transaction.

'SHDB' is the transaction code to collect the technical information of entire transaction (Do the recording).

#### Steps to Do The Recording : -

Execute 'SHDB'. Click on new recording in the application tool bar. Provide the recording name (any name). Provide the transaction code. Click on start recording or enter. Provide the vendor number, account group. Enter. Provide the name (any name), search term, country. Click on save.

**Note:** - When ever we click on save button recording will be stopped.

**Note:** - BDC\_OKCODE is the last entry of any screen.

**Note:** - In the real time recording is provided by functional people either in development server or in quality server depends on the data availability.

#### **Step 2: -**

In the real time functional people or end users provide a sample file in the development server to test the BDC program.

#### **Step 3: -**

'UPLOAD' is the function module which is used to browse the file as well as upload the data from file to internal table. The input for the above functional module is

1. File type → 'DAT'

2. Data internal table which is similar as file.

**Note:** - In the real time instead of upload functional module we always use GUI\_UPLOAD + F4\_filename function module.

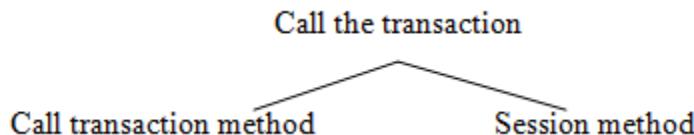
#### Step 4: -

Collect the screen & field details are nothing but fill an internal table which contains the following fields.

1. PROGRAM → Program name
2. DYNPRO → Screen number
3. DYNBEGIN → Starting position
4. FNAM → Field name
5. FVAL → Field value

**Note:** - In the DDIC we have one structure that is **BDCDATA** which contains above fields, so we simply declare our internal table by referring **BDCDATA** structure.

#### Step 5: -



#### Syntax of call transaction method: -

Call transaction ‘<Transaction code>’ using <BDCDATA internal table> mode ‘A/N/E’.

A → All screens

N → No screens

E → Error screens

→ Develop a conversion program to upload the vendor master data from flat file to SAP system by using BDC call transaction method through XK01 transaction. The flat file contains vendor numbers, names and search terms.

#### Step 1 (Do the record): -

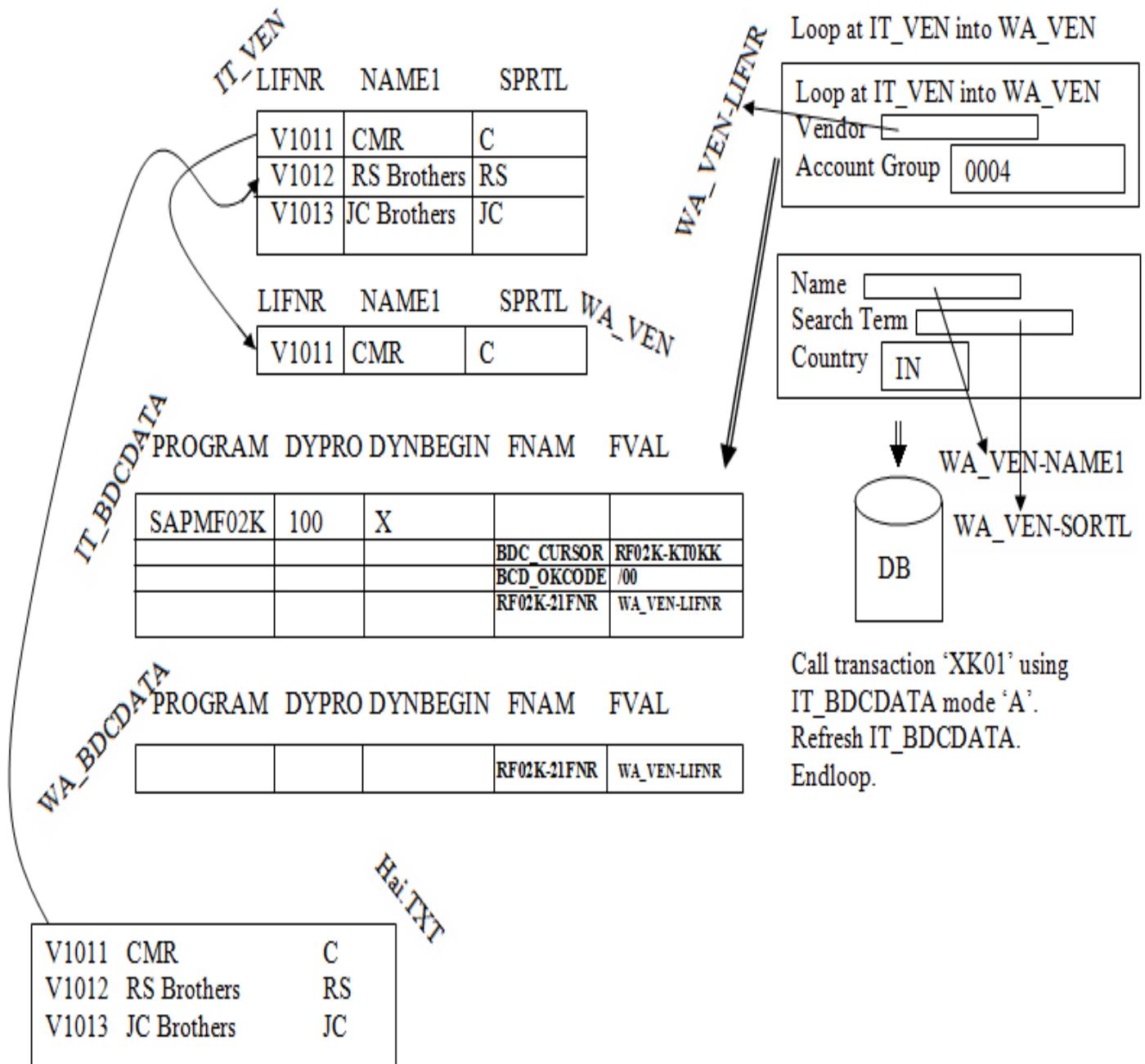
Execute ‘SHDB’. Click on new recording. Provide recording name (ZSXK01). Provide transaction code (XK01). Enter. Provide the vendor number (v1010), account group (0004). Enter. Provide the name (Big Bazar), search term (BB), country (IN). Save.

```
TYPES: BEGIN OF TY_VEN,
        LIFNR TYPE LFA1-LIFNR,
        NAME1 TYPE LFA1-NAME1,
        SORTL TYPE LFA1-SORTL,
        END OF TY_VEN.
DATA: WA_VEN TYPE TY_VEN,
      IT_VEN TYPE TABLE OF TY_VEN.
DATA: WA_BDCDATA LIKE BDCDATA,
      IT_BDCDATA LIKE TABLE OF WA_BDCDATA.
* Upload the data.
CALL FUNCTION 'UPLOAD'
  EXPORTING
    FILETYPE = 'DAT'
  TABLES
    DATA_TAB = IT_VEN.
```

```

LOOP AT IT_VEN INTO WA_VEN.
WA_BDCDATA-PROGRAM = 'SAPMF02K'.
WA_BDCDATA-DYNPRO = '0100'.
WA_BDCDATA-DYNBEGIN = 'X'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'BDC_CURSOR'.
WA_BDCDATA-FVAL = 'RF02K-KT0KK'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'BDC_OKCODE'.
WA_BDCDATA-FVAL = '/00'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'RF02K-LIFNR'.
WA_BDCDATA-FVAL = WA_VEN-LIFNR.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'RF02K-KTOKK'.
WA_BDCDATA-FVAL = '0004'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
* 2nd screen and field details
WA_BDCDATA-PROGRAM = 'SAPMF02K'.
WA_BDCDATA-DYNPRO = '0110'.
WA_BDCDATA-DYNBEGIN = 'X'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'BDC_CURSOR'.
WA_BDCDATA-FVAL = 'LFA1-LAND1'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'BDC_OKCODE'.
WA_BDCDATA-FVAL = '=UPDA'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'LFA1-NAME1'.
WA_BDCDATA-FVAL = WA_VEN-NAME1.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'LFA1-SORTL'.
WA_BDCDATA-FVAL = WA_VEN-SORTL.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'LFA1-LAND1'.
WA_BDCDATA-FVAL = 'IN'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
CALL TRANSACTION 'XK01' USING IT_BDCDATA MODE 'A'.
REFRESH IT_BDCDATA.
ENDLOOP.

```



→ Develop a conversion program to upload the customer number data from flat file to SAP system by using BDC Call transaction method through XD01 transaction. The flat file contains the customer numbers, names, search terms & street.

**Do the recording:-** Execute SHDB. Click on new recording in the application tool bar. Provide recording name (ZXD01), transaction code (XD01). Enter. Provide the customer number (12501), account group (0004). Enter. Provide the name (GMR INFRA LTD), search term (GMR), street (Ameerpet), country (IN), language key (EN). Save.

## **Steps to create a transaction code for BDC program: -**

Execute **SE93**. Provide the transaction code [(zsbdc2) any name]. Click on create. Provide short description. Select the radio button program and selection screen. Enter. Provide the program name. Select the GUI check boxes. Click on save.

```

TYPES: BEGIN OF TY_CUS,
      KUNNR TYPE KNA1-KUNNR,
      NAME1 TYPE KNA1-NAME1,
      SORTL TYPE KNA1-SORTL,
      STRAS TYPE KNA1-STRAS,
      END OF TY_CUS.

DATA: WA_CUS TYPE TY_CUS,
      IT_CUS LIKE TABLE OF WA_CUS.

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

* UPLOAD THE DATA.

CALL FUNCTION 'UPLOAD'
  EXPORTING
    FILETYPE = 'DAT'
  TABLES
    DATA_TAB = IT_CUS.

LOOP AT IT_CUS INTO WA_CUS.

WA_BDCDATA-PROGRAM = 'SAPMF02D'.
WA_BDCDATA-DYNPRO = '0100'.
WA_BDCDATA-DYNBEGIN = 'X'.
APPEND WA_BDCDATA TO IT_BDCDATA.

CLEAR WA_BDCDATA.

WA_BDCDATA-FNAM = 'BDC_CURSOR'.
WA_BDCDATA-FVAL = 'RF02D-KTOKD'.
APPEND WA_BDCDATA TO IT_BDCDATA.

CLEAR WA_BDCDATA.

WA_BDCDATA-FNAM = 'BDC_OKCODE'.
WA_BDCDATA-FVAL = '/00'.
APPEND WA_BDCDATA TO IT_BDCDATA.

CLEAR WA_BDCDATA.

WA_BDCDATA-FNAM = 'RF02D-KUNNR'.
WA_BDCDATA-FVAL = WA_CUS-KUNNR.
APPEND WA_BDCDATA TO IT_BDCDATA.

CLEAR WA_BDCDATA.

WA_BDCDATA-FNAM = 'RF02D-KTOKD'.
WA_BDCDATA-FVAL = '0004'.
APPEND WA_BDCDATA TO IT_BDCDATA.

CLEAR WA_BDCDATA.

* 2nd screen details

WA_BDCDATA-PROGRAM = 'SAPMF02D'.
WA_BDCDATA-DYNPRO = '0110'.
WA_BDCDATA-DYNBEGIN = 'X'.
APPEND WA_BDCDATA TO IT_BDCDATA.

CLEAR WA_BDCDATA.

WA_BDCDATA-FNAM = 'BDC_CURSOR'.
WA_BDCDATA-FVAL = 'KNA1-SPRAS'.
APPEND WA_BDCDATA TO IT_BDCDATA.

CLEAR WA_BDCDATA.

WA_BDCDATA-FNAM = 'BDC_OKCODE'.
WA_BDCDATA-FVAL = '=UPDA'.
APPEND WA_BDCDATA TO IT_BDCDATA.

```

```

CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'KNA1-NAME1'.
WA_BDCDATA-FVAL = WA_CUS-NAME1.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'KNA1-SORTL'.
WA_BDCDATA-FVAL = WA_CUS-SORTL.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'KNA1-STRAS'.
WA_BDCDATA-FVAL = WA_CUS-STRAS.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'KNA1-LAND1'.
WA_BDCDATA-FVAL = 'IN'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'KNA1-SPRAS'.
WA_BDCDATA-FVAL = 'EN'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.

CALL TRANSACTION 'XD01' USING IT_BDCDATA MODE 'A'.
REFRESH IT_BDCDATA.
ENDLOOP.

```

### **Differences between call transaction method & session method**

#### **Call transaction**

1. Call transaction method can process only one transaction at a time.
2. In call transaction method we manually handle the errors.
3. This method is fast.
4. This is immediate data base updation.
5. This is suitable if the flat file contains less amount of data.
6. Background scheduling isn't possible in this method.
7. It returns the SY-SUBRC value.
8. Synchronous as well as asynchronous data base updation.
9. Synchronous process

#### **Session Method**

1. Session method can process any number of transactions at a time.
2. In this method an error log will be generated that will be handle the errors.
3. This method is slower.
4. In this method after processing the session through SM35 only data base is updated.
5. This is suitable if the flat file contains huge amount of data.
6. Background scheduling is possible in this session.
7. It can't return the SY-SUBRC value.
8. Synchronous data base updation
9. Asynchronous process

### **Steps to work with session method: -**

1. Do the Recording.
2. Prepare flat file.
3. Upload the data from flat file to internal table.
4. Create the session by using 'BDC\_OPEN\_GROUP' function module.

The input for the above function module is

- i. GROUP → Name of the session, which is used to process the session.
  - ii. KEEP → Re maintain the session. After processing the session (Activate = 'X').
  - iii. HOLDDATE → The session is locked, until it reaches the hold date.
  - iv. USER → Valid user.
5. Loop at <Data internal table>.
- 
- 

Call the transaction by using 'BDC\_INSERT' function module.

The input for the above function module is

- i. <TCODE>
- ii. <BDCDATA INTERNAL TABLE>

6. Close the session by using 'BDC\_CLOSE\_GROUP' function module.

**→ Develop a conversion program to upload the customer master data from flat file to SAP system by using BDC session method. The flat file contains customer numbers, names, search terms, street.**

Steps to process the session (after execution the program): -

Execute SM35. Select the session name. Click on process in the application tool bar. Click on process.

**TYPES: BEGIN OF TY\_CUS,**

```
KUNNR TYPE KNA1-KUNNR,  
NAME1 TYPE KNA1-NAME1,  
SORTL TYPE KNA1-SORTL,  
STRAS TYPE KNA1-STRAS,  
END OF TY_CUS.
```

**DATA: WA\_CUS TYPE TY\_CUS,**

```
IT_CUS LIKE TABLE OF WA_CUS.
```

**DATA: WA\_BDCDATA LIKE BDCDATA,**

```
IT_BDCDATA LIKE TABLE OF WA_BDCDATA.
```

**\* UPLOAD THE DATA.**

**CALL FUNCTION 'UPLOAD'**

**EXPORTING**

```
FILETYPE = 'DAT'
```

**TABLES**

```
DATA_TAB = IT_CUS.
```

**CALL FUNCTION 'BDC\_OPEN\_GROUP'**

**EXPORTING**

```
GROUP      = 'DARLING'
```

**\* HOLDDATE = FILLER8**

```
KEEP      = 'X'
```

```
USER      = SY-UNAME.
```

**LOOP AT IT\_CUS INTO WA\_CUS.**

```
WA_BDCDATA-PROGRAM = 'SAPMF02D'.
```

```
WA_BDCDATA-DYNPRO  = '0100'.
```

```
WA_BDCDATA-DYNBEGIN = 'X'.
```

```
APPEND WA_BDCDATA TO IT_BDCDATA.
```

```
CLEAR WA_BDCDATA.
```

```

WA_BDCDATA-FNAM = 'BDC_CURSOR'.
WA_BDCDATA-FVAL = 'RF02D-KTOKD'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'BDC_OKCODE'.
WA_BDCDATA-FVAL = '/00'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'RF02D-KUNNR'.
WA_BDCDATA-FVAL = WA_CUS-KUNNR.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'RF02D-KTOKD'.
WA_BDCDATA-FVAL = '0004'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
* 2nd screen details
WA_BDCDATA-PROGRAM = 'SAPMF02D'.
WA_BDCDATA-DYNPRO = '0110'.
WA_BDCDATA-DYNBEGIN = 'X'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'BDC_CURSOR'.
WA_BDCDATA-FVAL = 'KNA1-SPRAS'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'BDC_OKCODE'.
WA_BDCDATA-FVAL = '=UPDA'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'KNA1-NAME1'.
WA_BDCDATA-FVAL = WA_CUS-NAME1.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'KNA1-SORTL'.
WA_BDCDATA-FVAL = WA_CUS-SORTL.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'KNA1-STRAS'.
WA_BDCDATA-FVAL = WA_CUS-STRAS.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'KNA1-LAND1'.
WA_BDCDATA-FVAL = 'IN'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
WA_BDCDATA-FNAM = 'KNA1-SPRAS'.
WA_BDCDATA-FVAL = 'EN'.
APPEND WA_BDCDATA TO IT_BDCDATA.
CLEAR WA_BDCDATA.
CALL FUNCTION 'BDC_INSERT'

```

```

EXPORTING
  TCODE      = 'XD01'
  TABLES
    DYNPROTAB = IT_BDCDATA.
  REFRESH IT_BDCDATA.
ENDLOOP.

* Close the session
CALL FUNCTION 'BDC_CLOSE_GROUP'.

```

### BNKA (Bank Master Table)

BANKS → Bank country key

BANKL → Bank key

BANKA → Bank name

### Steps to create a bank: -

Execute FI01. Provide bank country key 'IN', bank key [any name (669111)]. Enter. Provide the bank name. Save.

**→ Develop a conversion program to upload the vendor and customer master data from two flat files to SAP system by using BDC session method. The vendor flat files contain vendor numbers, names & search term, customer flat file contains customer numbers, customer names, search terms & street.**

CALL FUNCTION 'UPLOAD' ← · · · · ·

- 'DAT'
- IT\_VEN.

CALL FUNCTION 'UPLOAD' ← · · · · ·

- 'DAT'
- IT\_CUS.

CALL FUNCTION 'BDC\_OPEN\_GROUP'

- GROUP = 'SPMT'
- KEEP = 'X'
- USER = SY-UNAME

LOOP AT IT\_VEN INTO WA\_VEN.

----- } Collect the screen and field details of vendor

CALL FUNCTION 'BDC\_INSERT'

- 'XK01'
- IT\_BDCDATA.

REFRESH IT\_BDCDATA.

→ ENDLOOP.

LOOP AT IT\_CUS INTO WA\_CUS.

----- } Collect the screen and field details of vendor

CALL FUNCTION 'BDC\_INSERT'

- 'XD01'
- IT\_BDCDATA.

REFRESH IT\_BDCDATA.

→ ENDLOOP.

CALL FUNCTION 'BDC\_CLOSE\_GROUP'.

DATA: BDCDATA LIKE BDCDATA OCCURS 0 WITH HEADER LINE.

```

TYPES: BEGIN OF TY_VEN,
       LIFNR TYPE LFA1-LIFNR,
       NAME1 TYPE LFA1-NAME1,
       SORTL TYPE LFA1-SORTL,
       END OF TY_VEN.
DATA: WA_VEN TYPE TY_VEN,
      IT_VEN TYPE TABLE OF TY_VEN.

TYPES: BEGIN OF TY_CUS,
       KUNNR TYPE KNA1-KUNNR,
       NAME1 TYPE KNA1-NAME1,
       SORTL TYPE KNA1-SORTL,
       STRAS TYPE KNA1-STRAS,
       END OF TY_CUS.
DATA: WA_CUS TYPE TY_CUS,
      IT_CUS TYPE TABLE OF TY_CUS.

START-OF-SELECTION.
  CALL FUNCTION 'UPLOAD'
    EXPORTING
      FILETYPE = 'DAT'
    TABLES
      DATA_TAB = IT_VEN.
  CALL FUNCTION 'UPLOAD'
    EXPORTING
      FILETYPE = 'DAT'
    TABLES
      DATA_TAB = IT_CUS.

  CALL FUNCTION 'BDC_OPEN_GROUP'
    EXPORTING
      GROUP      = 'DARLING2'
*     HOLDDATE = FILLER8
      KEEP      = 'X'
      USER      = SY-UNAME.
LOOP AT IT_VEN INTO WA_VEN.
  perform bdc_dynpro      using 'SAPMF02K' '0100'.
  perform bdc_field       using 'BDC_CURSOR'
                           'RF02K-KTOKK'.
  perform bdc_field       using 'BDC_OKCODE'
                           '/00'.
  perform bdc_field       using 'RF02K-LIFNR'
                           WA_VEN-LIFNR.
  perform bdc_field       using 'RF02K-KTOKK'
                           '0004'.
  perform bdc_dynpro      using 'SAPMF02K' '0110'.
  perform bdc_field       using 'BDC_CURSOR'
                           'LFA1-LAND1'.
  perform bdc_field       using 'BDC_OKCODE'
                           '=UPDA'.
  perform bdc_field       using 'LFA1-NAME1'

```

```

        WA_VEN-NAME1.
perform bdc_field      using 'LFA1-SORTL'
                                WA_VEN-SORTL.
perform bdc_field      using 'LFA1-LAND1'
                                'KW'.

CALL FUNCTION 'BDC_INSERT'
EXPORTING
TCODE      = 'XK01'
TABLES
DYNPROTAB = BDCDATA.
REFRESH BDCDATA.
ENDLOOP.

LOOP AT IT_CUS INTO WA_CUS.
perform bdc_dynpro      using 'SAPMF02D' '0100'.
perform bdc_field       using 'BDC_CURSOR' 'RF02D-KTOKD'.
perform bdc_field       using 'BDC_OKCODE' '/00'.
perform bdc_field       using 'RF02D-KUNNR' WA_CUS-KUNNR.
perform bdc_field       using 'RF02D-KTOKD' '0004'.
perform bdc_dynpro      using 'SAPMF02D' '0110'.
perform bdc_field       using 'BDC_CURSOR' 'KNA1-SPRAS'.
perform bdc_field       using 'BDC_OKCODE' '=UPDA'.
perform bdc_field       using 'KNA1-NAME1' WA_CUS-NAME1.
perform bdc_field       using 'KNA1-SORTL' WA_CUS-SORTL.
perform bdc_field       using 'KNA1-STRAS' WA_CUS-STRAS.
perform bdc_field       using 'KNA1-LAND1' 'IN'.
perform bdc_field       using 'KNA1-SPRAS' 'EN'.

CALL FUNCTION 'BDC_INSERT'
EXPORTING
TCODE      = 'XD01'
TABLES
DYNPROTAB = BDCDATA.
REFRESH BDCDATA.
ENDLOOP.

CALL FUNCTION 'BDC_CLOSE_GROUP'.

FORM BDC_DYNPRO USING PROGRAM DYNPRO.
CLEAR BDCDATA.
BDCDATA-PROGRAM = PROGRAM.
BDCDATA-DYNPRO  = DYNPRO.
BDCDATA-DYNBEGIN = 'X'.
APPEND BDCDATA.
ENDFORM.

FORM BDC_FIELD USING FNAM FVAL.
CLEAR BDCDATA.
BDCDATA-FNAM = FNAM.
BDCDATA-FVAL = FVAL.
APPEND BDCDATA.
ENDFORM.

```

→ Develop a conversion program to upload the bank details from flat file to SAP system by using BDC call transaction method through FI01 transaction. The flat file contains Bank country key, bank key & bank name.

Steps to do the recording:-

Execute SHDB. Click on new recording in the application tool bar. Provide recording name (ZSFI01), transaction code (FI01). Click on start recording. Provide bank country key, bank key. Enter. Provide the bank name. Save. Save the recording.

Steps to develop the program from recording:-

Execute SHDB. Select the recording name. click on program in the application tool bar. Provide the program name (ZSPR\_930AM\_BDC5). Select the radio button transfer from recording. Enter. Provide title. Select the status, application. Click on source code. Save in our own package.

```
DATA: BDCDATA LIKE BDCDATA OCCURS 0 WITH HEADER LINE.
```

```
TYPES: BEGIN OF TY_BANK,  
        BANKS TYPE BNKA-BANKS,  
        BANKL TYPE BNKA-BANKL,  
        BANKA TYPE BNKA-BANKA,  
        END OF TY_BANK.
```

```
DATA: WA_BANK TYPE TY_BANK,  
      IT_BANK TYPE TABLE OF TY_BANK.
```

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

```
  CALL FUNCTION 'UPLOAD'  
    EXPORTING  
      FILETYPE = 'DAT'  
    TABLES  
      DATA_TAB = IT_BANK.
```

```
LOOP AT IT_BANK INTO WA_BANK.
```

```
  perform bdc_dynpro      using 'SAPMF02B' '0100'.  
  perform bdc_field       using 'BDC_CURSOR'  
                            'BNKA-BANKL'.  
  perform bdc_field       using 'BDC_OKCODE'  
                            '/00'.  
  perform bdc_field       using 'BNKA-BANKS'  
                            WA_BANK-BANKS.  
  perform bdc_field       using 'BNKA-BANKL'  
                            WA_BANK-BANKL.  
  perform bdc_dynpro      using 'SAPMF02B' '0110'.  
  perform bdc_field       using 'BDC_CURSOR'  
                            'BNKA-BANKA'.  
  perform bdc_field       using 'BDC_OKCODE'  
                            '=UPDA'.  
  perform bdc_field       using 'BNKA-BANKA'  
                            WA_BANK-BANKA.
```

```
  CALL TRANSACTION 'FI01' USING BDCDATA MODE 'A'.  
  REFRESH BDCDATA.
```

```
ENDLOOP.
```

```
FORM BDC_DYNPRO USING PROGRAM DYNPRO.  
CLEAR BDCDATA.
```

```

BDCDATA-PROGRAM = PROGRAM.
BDCDATA-DYNPRO = DYNPRO.
BDCDATA-DYNBEGIN = 'X'.
APPEND BDCDATA.
ENDIFORM.

FORM BDC_FIELD USING FNAM FVAL.
CLEAR BDCDATA.
BDCDATA-FNAM = FNAM.
BDCDATA-FVAL = FVAL.
APPEND BDCDATA.
ENDIFORM.

```

**→ Develop a conversion program to upload the vendor city from flat file to SAP system by using BDC call transaction method through XK02 transaction. The flat file contains vendor numbers and city.**

**Steps to do the recording:** -

Execute SHDB. Click on new recording in the application tool bar. Provide the recording name, transaction code (XK02). Enter. Provide existing vendor number (V1011). Select the address checkbox. Enter. Provide the city. Save.

**Note:** - When ever we are working with update transaction then we must remove the other than flat file fields recording steps from recording by using minus (-) symbol in the application tool bar.

Here we remove the NAME1, SORTL, LAND1 steps.

```

DATA: BDCDATA LIKE BDCDATA OCCURS 0 WITH HEADER LINE.
TYPES: BEGIN OF TY_VC,
        LIFNR TYPE LFA1-LIFNR,
        ORT01 TYPE LFA1-ORT01,
        END OF TY_VC.
DATA: WA_VC TYPE TY_VC,
      IT_VC TYPE TABLE OF TY_VC.

start-of-selection.
  CALL FUNCTION 'UPLOAD'
    EXPORTING
      FILETYPE = 'DAT'
    TABLES
      DATA_TAB = IT_VC.

LOOP AT IT_VC INTO WA_VC.
  perform bdc_dynpro      using 'SAPMF02K' '0101'.
  perform bdc_field       using 'BDC_CURSOR'
                           'RF02K-D0110'.
  perform bdc_field       using 'BDC_OKCODE'
                           '/00'.
  perform bdc_field       using 'RF02K-LIFNR'
                           WA_VC-LIFNR.
  perform bdc_field       using 'RF02K-D0110'

```

```

          'X'.
perform bdc_dynpro      using 'SAPMF02K' '0110'.
perform bdc_field       using 'BDC_CURSOR'
                           'LFA1-ORT01'.
perform bdc_field       using 'BDC_OKCODE'
                           '=UPDA'.
perform bdc_field       using 'LFA1-ORT01'
                           'WA_VC-ORT01.
perform bdc_field       using 'LFA1-LAND1'
                           'IN'.

CALL TRANSACTION 'XK02' USING BDCDATA MODE 'A'.
REFRESH BDCDATA.
ENDLOOP.

FORM BDC_DYNPRO USING PROGRAM DYNPRO.
CLEAR BDCDATA.
BDCDATA-PROGRAM = PROGRAM.
BDCDATA-DYNPRO = DYNPRO.
BDCDATA-DYNBEGIN = 'X'.
APPEND BDCDATA.
ENDFORM.

FORM BDC_FIELD USING FNAM FVAL.
CLEAR BDCDATA.
BDCDATA-FNAM = FNAM.
BDCDATA-FVAL = FVAL.
APPEND BDCDATA.
ENDFORM.

```

#### Handle the error in call transaction method: -

1. By using 'FORMAT\_MESSAGE' function module
2. Handling the errors through SESSION method

FORMAT\_MESSAGE is the function module which is used to handle the errors in call transaction method. The input for the above function module is

1. Message id
2. Message number
3. Message1
4. Message2
5. Message3
6. Message4
7. language

The output for the above function module is ‘Meaningful message’.

#### Syntax of call transaction: -

Call transaction ‘<TCODE>’ using <BDCDATA Internal table> mode ‘A/N/E’ messages into <BDCMSGCOLL Internal table>.

The call transaction method written the success or failure information into BDCMSGCOLL Internal table.

#### Some of the fields in BDCMSGCOLL Internal table: -

1. MSGID → Message ID
2. MSGNR → Message number
3. MSGV1 → Message1
4. MSGV2 → Message2

5. MSGV3 → Message3
6. MSGV4 → Message4

**Note:** - In the DDIC, we have one structure. i.e. BDCMSGCOLL, which contains above fields. So we simply declare our internal table by referring BDCMSGCOLL structure.

**Note:** - When ever we are handling the errors in call transaction method then we must provide mode is 'N'.

**→ Develop a conversion program to upload the vendor master data from flat file to SAP system by using BDC call transaction method & also download the error messages (handling the errors). The flat file contains vendor numbers, account groups, name & search term.**

```

DATA:     BDCDATA LIKE BDCDATA      OCCURS 0 WITH HEADER LINE.
TYPES: BEGIN OF TY_VEN,
        LIFNR TYPE LFA1-LIFNR,
        KTOKK TYPE LFA1-KTOKK,
        NAME1 TYPE LFA1-NAME1,
        SORTL TYPE LFA1-SORTL,
        END OF TY_VEN.
DATA: WA_VEN TYPE TY_VEN,
      IT_VEN TYPE TABLE OF TY_VEN.
DATA: BEGIN OF WA_ERROR,
      LNO TYPE SYTABIX,
      MSG(100) TYPE C,
      END OF WA_ERROR.
DATA IT_ERROR LIKE TABLE OF WA_ERROR.

DATA WA_BMC LIKE BDCMSGCOLL,
      IT_BMC LIKE TABLE OF WA_BMC.
DATA V_MSG(100) TYPE C.

start-of-selection.
  CALL FUNCTION 'UPLOAD'
    EXPORTING
      FILETYPE = 'DAT'
    TABLES
      DATA_TAB = IT_VEN.

LOOP AT IT_VEN INTO WA_VEN.
  WA_ERROR-LNO = SY-TABIX.
  perform bdc_dynpro      using 'SAPMF02K' '0100'.
  perform bdc_field       using 'BDC_CURSOR'
                           'RF02K-KTOKK'.
  perform bdc_field       using 'BDC_OKCODE'
                           '/00'.
  perform bdc_field       using 'RF02K-LIFNR'
                           WA_VEN-LIFNR.
  perform bdc_field       using 'RF02K-KTOKK'
                           WA_VEN-KTOKK.
  perform bdc_dynpro      using 'SAPMF02K' '0110'.
  perform bdc_field       using 'BDC_CURSOR'
                           'LFA1-LAND1'.
  perform bdc_field       using 'BDC_OKCODE'

```

```

        '=UPDA'.
perform bdc_field      using 'LFA1-NAME1'
                           WA_VEN-NAME1.
perform bdc_field      using 'LFA1-SORTL'
                           WA_VEN-SORTL.
perform bdc_field      using 'LFA1-LAND1'
                           'IN'.

CALL TRANSACTION 'XK01' USING BDCDATA MODE 'N' MESSAGES INTO
IT_BMC.

IF SY-SUBRC <> 0.
LOOP AT IT_BMC INTO WA_BMC.

CALL FUNCTION 'FORMAT_MESSAGE'
EXPORTING
  ID      = WA_BMC-MSGID
  LANG    = SY-LANGU
  NO      = WA_BMC-MSGNR
  V1      = WA_BMC-MSGV1
  V2      = WA_BMC-MSGV2
  V3      = WA_BMC-MSGV3
  V4      = WA_BMC-MSGV4
IMPORTING
  MSG     = V_MSG.
IF SY-SUBRC = 0.
  WA_ERROR-MSG = V_MSG.
  APPEND WA_ERROR TO IT_ERROR.
  CLEAR WA_ERROR.
ENDIF.
ENDLOOP.
ENDIF.
REFRESH: BDCDATA, IT_BMC.
ENDLOOP.

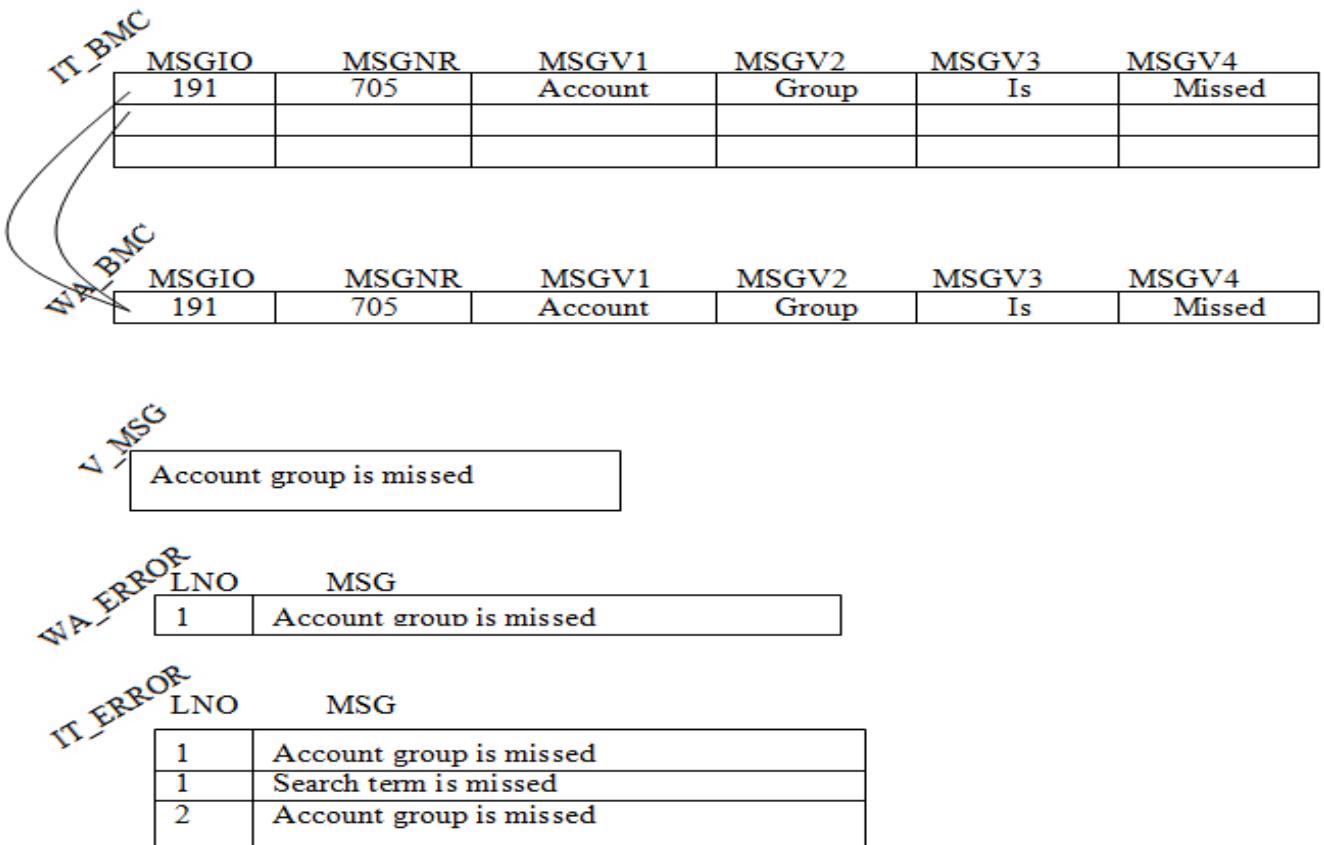
CALL FUNCTION 'DOWNLOAD'
EXPORTING
  FILETYPE = 'DAT'
TABLES
  DATA_TAB = IT_ERROR.

FORM BDC_DYNPRO USING PROGRAM DYNPRO.
CLEAR BDCDATA.
BDCDATA-PROGRAM = PROGRAM.
BDCDATA-DYNPRO   = DYNPRO.
BDCDATA-DYNBEGIN = 'X'.
APPEND BDCDATA.
ENDFORM.

```

C:\VEN\TXT

H5050	---	BIGBAZAR	---
H5051	004	CMR	CMR
H5052	---	RS BROTH	RS



#### Handling the errors through session method: -

Call function 'BDC\_OPEN\_GROUP'

- GROUP = 'HE'
- KEEP = 'X'
- USER = SY-UNAME.

LOOP AT IT\_VEN INTO WA\_VEN.

---

---

Call transaction 'XK01' using IT\_BDCDATA mode 'N'.

If SY-SUBRC <> 0.

CALL FUNCTION 'BDC\_INSERT'

- 'XK01'
- IT\_BDCDATA.

ENDIF.

REFRESH IT\_BDCDATA.

ENDLOOP.

CALL FUNCTION 'BDC\_CLOSE\_GROUP'.

After, we execute the program, we process the session 'HE' through 'SM35' in background & observe the error log.

```
DATA: BDCDATA LIKE BDCDATA      OCCURS 0 WITH HEADER LINE.
```

```
TYPES: BEGIN OF TY_VEN,
       LIFNR TYPE LFA1-LIFNR,
       KTOKK TYPE LFA1-KTOKK,
       NAME1 TYPE LFA1-NAME1,
       SORTL TYPE LFA1-SORTL,
     END OF TY_VEN.
```

```

DATA: WA_VEN TYPE TY_VEN,
      IT_VEN TYPE TABLE OF TY_VEN.

start-of-selection.
  CALL FUNCTION 'UPLOAD'
    EXPORTING
      FILETYPE = 'DAT'
    TABLES
      DATA_TAB = IT_VEN.

  CALL FUNCTION 'BDC_OPEN_GROUP'
    EXPORTING
      GROUP      = 'DARLING3'
*     HOLDDATE  = FILLER8
      KEEP       = 'X'
      USER       = SY-UNAME.

LOOP AT IT_VEN INTO WA_VEN.
  perform bdc_dynpro      using 'SAPMF02K' '0100'.
  perform bdc_field       using 'BDC_CURSOR'
                                'RF02K-KTOKK'.
  perform bdc_field       using 'BDC_OKCODE'
                                '/00'.
  perform bdc_field       using 'RF02K-LIFNR'
                                WA_VEN-LIFNR.
  perform bdc_field       using 'RF02K-KTOKK'
                                WA_VEN-KTOKK.
  perform bdc_dynpro      using 'SAPMF02K' '0110'.
  perform bdc_field       using 'BDC_CURSOR'
                                'LFA1-LAND1'.
  perform bdc_field       using 'BDC_OKCODE'
                                '=UPDA'.
  perform bdc_field       using 'LFA1-NAME1'
                                WA_VEN-NAME1.
  perform bdc_field       using 'LFA1-SORTL'
                                WA_VEN-SORTL.
  perform bdc_field       using 'LFA1-LAND1'
                                'IN'.

CALL TRANSACTION 'XK01' USING BDCDATA MODE 'N'.
IF SY-SUBRC <> 0.
  CALL FUNCTION 'BDC_INSERT'
    EXPORTING
      TCODE      = 'XK01'
    TABLES
      DYNPROTAB = BDCDATA.
ENDIF.
REFRESH BDCDATA.
ENDLOOP.
CALL FUNCTION 'BDC_CLOSE_GROUP'.

```

Session over view (SM35): -

**Analysis:** -

This is used to identify the number of transactions are available in the session and their status and also this is used to identify the screens & fields information.

**Process:** -

This is used to process the session either in foreground or background or error mode.

**Statistics:** -

This is used to identify the quick information of the session. I.e. how many transactions are successfully processed how many are deleted. How many are still to be process.

**Log:** -

This is used to identify the each & every step of entire session processing.

**Recording:**-

This is used to cal the SHDB transaction code.

**Delete:** -

This is used to delete the sessions from the session overview.

**Lock:** -

This is used to lock the session until a particular date.

**Unlock:** -

This is used to unlock the session which is already locked.

**Syntax of concatenate:** -

Concatenate <variable1> <variable2> ---- into <variable3> separated by '<delimiter>'.

**Ex:** -

```
Data A(10) type C value 'SJF'.
Data B(20) type C value 'TECH'.
Data C(30) type C.
Concatenate A B into C separated by ' '.
Write C.
O/P → SJF TECH
```

**Ex:** -

```
Data A(2) type C.
Data R(20) type C.
A = 01.
Concatenate 'SJF(' A ')' into R.
Write R.
O/P → SJFA(1)
```

**Note:** - Concatenate is only possible for character data types (C, N, D, T) not for numeric data type (I, F, P).

**Syntax of Split:** -

Split <variable1> at '<delimiter>' into <variable1> <variable2> -----

**Ex:** -

```
Data A(30) type C value 'SJF TECH'.
Data B(10) type C.
Data C(20) type C.
Split A at ' ' into B C.
Write:/ B, C.
```

O/P → SJF TECH

Ex: -

```
Data V(30) type C value '1000, TCS, HYD'.
Data: V1(4) type C,
      V2(4) type C,
      V3(4) type C.
Split V at ',' into V1 V2 V3.
Write:/ V1, V2, V3.
O/P → 1000 TCS HYD
```

→ Based on the given vendor numbers, display the vendor numbers, vendor names & cities as shown in the below. If the user clicks on update button then we update the vendor cities of selected checkbox by using BDC call transaction method through 'XK02' transaction method.

**UPDATE**

<input checked="" type="checkbox"/>	V1010	BIGBAZAR	<u>HYD</u>	PUN
<input type="checkbox"/>	V1011	RS BROTH	<u>CHE</u>	
<input checked="" type="checkbox"/>	V1012	CMR	<u>BAN</u>	MUM

```
DATA: BDCDATA LIKE BDCDATA      OCCURS 0 WITH HEADER LINE.
```

```
DATA V TYPE SYLINNO.
```

```
TABLES LFA1.
```

```
SELECT-OPTIONS S_LIFNR FOR LFA1-LIFNR.
```

```
DATA: A, B.
```

```
TYPES: BEGIN OF TY_VEN,
       LIFNR TYPE LFA1-LIFNR,
       NAME1 TYPE LFA1-NAME1,
       ORT01 TYPE LFA1-ORT01,
       END OF TY_VEN.
```

```
DATA: WA_VEN TYPE TY_VEN,
      IT_VEN TYPE TABLE OF TY_VEN.
```

```
DATA: BEGIN OF WA,
      LIFNR TYPE LFA1-LIFNR,
      ORT01 TYPE LFA1-ORT01,
      END OF WA.
```

```
DATA IT LIKE TABLE OF WA.
```

```
SELECT LIFNR NAME1 ORT01 FROM LFA1 INTO TABLE IT_VEN WHERE LIFNR IN
S_LIFNR.
```

```
LOOP AT IT_VEN INTO WA_VEN.
```

```
WRITE:/ A AS CHECKBOX, WA_VEN-LIFNR, WA_VEN-NAME1, WA_VEN-ORT01 INPUT.
```

```
ENDLOOP.
```

```
V = SY-LINNO.
```

```
SET PF-STATUS 'STAT'.
```

```
AT USER-COMMAND.
```

```
IF SY-UCOMM = 'UPD'.
```

```
DO V TIMES.
```

```
READ LINE SY-INDEX FIELD A INTO B
      WA_VEN-LIFNR INTO WA-LIFNR
      WA_VEN-ORT01 INTO WA-ORT01.
```

```

IF B = 'X'.
APPEND WA TO IT.
CLEAR WA.
ENDIF.
ENDDO.

LOOP AT IT INTO WA.
perform bdc_dynpro      using 'SAPMF02K' '0101'.
perform bdc_field        using 'BDC_CURSOR'  'RF02K-LIFNR'.
perform bdc_field        using 'BDC_OKCODE'   '/00'.
perform bdc_field        using 'RF02K-LIFNR'  WA-LIFNR.
perform bdc_field        using 'RF02K-D0110' 'X'.
perform bdc_dynpro      using 'SAPMF02K' '0110'.
perform bdc_field        using 'BDC_CURSOR'  'LFA1-ORT01'.
perform bdc_field        using 'BDC_OKCODE'   '=UPDA'.
perform bdc_field        using 'LFA1-ORT01'  WA-ORT01.

```

```

CALL TRANSACTION 'XK02' USING BDCDATA MODE 'A'.
REFRESH BDCDATA.
ENDLOOP.
ENDIF.

```

```

FORM BDC_DYNPRO USING PROGRAM DYNPRO.
=====
ENDFORM.

FORM BDC_FIELD USING FNAM FVAL.
=====
ENDFORM.

```

**Note:** - When ever we are working with conversion program if the flat file contains date field then we must consider as char 10 in the data internal table if the flat file contains quantity & amount fields then we must consider as char 15 in the data internal table.

**→ Develop a conversion program to upload the cost center master data from flat file to SAP system by using BDC call transaction method through KS01 transaction. The flat file contains controlling area, cost center number, from date, to date, name, person responsible.**

**Steps to do the recording:**

Execute SHDB. Click on new recording in the application toolbar. Provide the recording name (SKS01). Provide the transaction code (KS01). Enter. Provide controlling area (6000), provide the cost center (15440), provide the from date (26.01.2015), to date (31.01.2015). Enter. Provide the name (SJF TECH), person responsible (venkat). Select the cost center category (5), hierarchical area 1000, business area (0001), save. Enter. [CSKS (Cost center master data), CSKT (Cost center description table)].

Data: BDCDATA like BDCDATA OCCURS 0 WITH HEADERLINE.

Types: Begin of ty\_cost,

- KOKRS type CSKS-KOKRS,
- KOSTL type CSKS-KOSTL,
- VFD(10) type C,
- VTD(10) type C,
- KTEXT type CSKT-KTEXT,
- VERAK type CSKS-VERAK,

End of ty\_cost.  
 Data: wa\_cost type ty\_cost,  
     It\_cost type table of ty\_cost.  
 Start-of-selection.  
 Call function 'UPLOAD'  
 Exporting  
     FILETYPE = 'DAT'  
 Tables  
     DATA\_TAB = IT\_COST.  
 Loop at it\_cost into wa\_cost.  
 ======  
 Perform bdc\_field using 'CSKSZ-KOKRS' WA\_COST-KOKRS.  
 Perform bdc\_field using 'CSKSZ-KOSTL' WA\_COST-KOSTL.  
 Perform bdc\_field using 'CSXSZ-DATAB\_ANFO' WA\_COS-VFD.  
 Perform bdc\_field using 'CSKSZ-DATB1\_ANFO' WA\_COST-VTD.  
 ======  
 Perform bdc\_field using 'CSKSZ-KTEXT' WA\_COST-KTEXT.  
 Perform bdc\_field using 'CSKSZ-VERAK' WA\_COST-VERAK.  
 ======  
 Call transaction 'KS01' using BDCDATA mode 'A'.  
 Refresh BDCDATA.  
 Endloop.  
 Form BDC-DYNPRO using program DYNPRO.  
 ======  
 Endform.  
 Form BDC\_FIELD using FNAM FVAL.  
 ======  
 Endform.

### **LFBK (Vendor Bank Table)**

LIFNR → Vendor number  
 BANKS → Bank County Key  
 BANKL → Bank key  
 BANKN → Account Number

### **KNBK (Customer Bank Table)**

KUNNR → Customer  
 BANKS → Bank Country Key  
 BANKL → Bank Key  
 BANKN → Account number

**→ Upload the vendor & vendor bank details from a single flat file to two internal tables. The vendor & vendor bank details are differentiated with V & B. the vendor details are in the flat file vendor numbers, names & search terms. In the flat file bank details are vendor numbers bank country keys bank keys & bank names.**

```

DATA V1.
TYPES: BEGIN OF TY_VEN,
        LIFNR TYPE LFA1-LIFNR,
        NAME1 TYPE LFA1-NAME1,
        SORTL TYPE LFA1-SORTL,
      END OF TY_VEN.
  
```

```
DATA: WA_VEN TYPE TY_VEN,
      IT_VEN TYPE TABLE OF TY_VEN.
```

```
TYPES: BEGIN OF TY_BANK,
        LIFNR TYPE LFBK-LIFNR,
        BANKS TYPE LFBK-BANKS,
        BANKL TYPE LFBK-BANKL,
        BANKN TYPE LFBK-BANKN,
        END OF TY_BANK.
```

```
DATA: WA_BANK TYPE TY_BANK,
      IT_BANK TYPE TABLE OF TY_BANK.
```

```
DATA: BEGIN OF WA,
      ROW(100) TYPE C,
      END OF WA.
```

```
DATA IT LIKE TABLE OF WA.
```

```
CALL FUNCTION 'UPLOAD'
  EXPORTING
```

```
    FILETYPE = 'DAT'
```

```
  TABLES
```

```
    DATA_TAB = IT.
```

```
LOOP AT IT INTO WA.
```

```
IF WA-ROW+0(1) = 'V'.
```

```
  SPLIT WA-ROW AT ',' INTO V1 WA_VEN-LIFNR WA_VEN-NAME1 WA_VEN-SORTL.
```

```
  APPEND WA_VEN TO IT_VEN.
```

```
  CLEAR WA_VEN.
```

```
ELSE.
```

```
  SPLIT WA-ROW AT ',' INTO V1
```

```
WA_BANK-LIFNR WA_BANK-BANKS WA_BANK-
BANKL WA_BANK-BANKN.
```

```
  APPEND WA_BANK TO IT_BANK.
```

```
  CLEAR WA_BANK.
```

```
ENDIF.
```

```
ENDLOOP.
```

```
LOOP AT IT_VEN INTO WA_VEN.
```

```
  WRITE:/ WA_VEN-LIFNR, WA_VEN-NAME1, WA_VEN-SORTL.
```

```
ENDLOOP.
```

```
ULINE.
```

```
LOOP AT IT_BANK INTO WA_BANK.
```

```
  WRITE:/ WA_BANK-LIFNR, WA_BANK-BANKS, WA_BANK-BANKL, WA_BANK-BANKN.
```

```
ENDLOOP.
```

V1  
B

WA  
B, H9093, IN, 121212, 1010101010

V, H9091, BIGBAZAR, BB
V, H9092, CMR, CMR
V, H9093, RS BROTHERS, RS
B, H9091, IN, 121212, 100786116
B, H9091, IN, 121214, 421140015
B, H9093, IN, 121212, 1010101010

LIFNR	BANKS	BANKL	BANKN

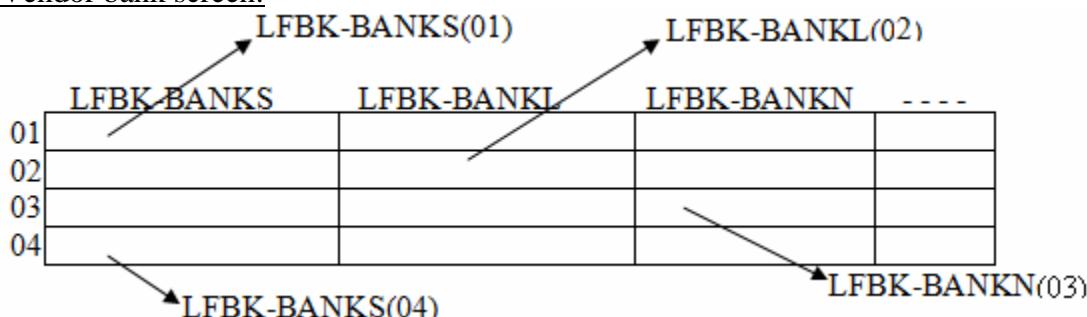
WA\_BANK

### BDC table control:-

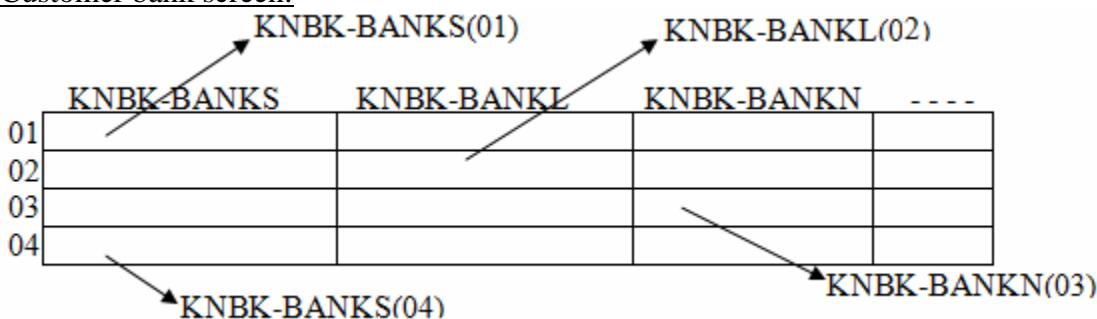
When ever we upload the multiple records in a tabular format for a single transaction then it's called bdc table control.

When ever we upload the vendor bank details, customer bank details then we go for bdc table control. Because one vendor having any number of bank details in a tabular format & customer also having any number of bank details in a tabular format.

Vendor bank screen: -



Customer bank screen: -



→ Develop a conversion program to upload the vendor bank details from flat file to sap system by using BDC call transaction method through XK01 transaction. Flat file contains vendor & vendor bank details with separation at V and B. the vendor details are vendor number, name & search terms. The bank details are vendor number, bank country key, bank key & account number.

Steps to do the recording: -

Execute SHDB. Click on new recording in the application tool bar. Provide recording name (SXK01), transaction code (XK01). Enter. Provide vendor number (H9080), account group (0004). Enter. Provide the name (power grid). Search term (pg), country (IN). Enter. Third screen we no need to provide information. Enter. In 4<sup>th</sup> screen provide the details.

CTRY BANK KEY BANK ACCOUNT

IN	111111	1234567890
IN	111112	9876543210

→ Must exist in BNKA table

Click on Save.

```
DATA: BDCDATA LIKE BDCDATA      OCCURS 0 WITH HEADER LINE.
DATA V2 TYPE N.
DATA V_FNAME(20) TYPE C.
DATA V1.
TYPES: BEGIN OF TY_VEN,
        LIFNR TYPE LFA1-LIFNR,
        NAME1 TYPE LFA1-NAME1,
        SORTL TYPE LFA1-SORTL,
        END OF TY_VEN.
DATA: WA_VEN TYPE TY_VEN,
      IT_VEN TYPE TABLE OF TY_VEN.

TYPES: BEGIN OF TY_BANK,
        LIFNR TYPE LFBK-LIFNR,
        BANKS TYPE LFBK-BANKS,
        BANKL TYPE LFBK-BANKL,
        BANKN TYPE LFBK-BANKN,
        END OF TY_BANK.
DATA: WA_BANK TYPE TY_BANK,
      IT_BANK TYPE TABLE OF TY_BANK.

DATA: BEGIN OF WA,
      ROW(100) TYPE C,
      END OF WA.
DATA IT LIKE TABLE OF WA.

START-OF-SELECTION.
  CALL FUNCTION 'UPLOAD'
    EXPORTING
      FILETYPE = 'DAT'
    TABLES
      DATA_TAB = IT.
  LOOP AT IT INTO WA.
    IF WA-ROW+0(1) = 'V'.
      SPLIT WA-ROW AT ',' INTO V1 WA_VEN-LIFNR WA_VEN-NAME1 WA_VEN-
      SORTL.
      APPEND WA_VEN TO IT_VEN.
      CLEAR WA_VEN.
    ELSE.
      SPLIT WA-ROW AT ',' INTO V1 WA_BANK-LIFNR WA_BANK-BANKS WA_BANK-
      BANKL WA_BANK-BANKN.
      APPEND WA_BANK TO IT_BANK.
      CLEAR WA_BANK.
    ENDIF.
  ENDLOOP.

* First screen
  LOOP AT IT_VEN INTO WA_VEN.
    perform bdc_dynpro      using 'SAPMF02K' '0100'.
    perform bdc_field       using 'BDC_CURSOR'
```

```

                    'RF02K-KTOKK'.
perform bdc_field      using 'BDC_OKCODE'
                    '/00'.
perform bdc_field      using 'RF02K-LIFNR'
                    WA_VEN-LIFNR.
perform bdc_field      using 'RF02K-KTOKK'
                    '0004'.

* SECOND SCREEN
perform bdc_dynpro    using 'SAPMF02K' '0110'.
perform bdc_field      using 'BDC_CURSOR'
                    'LFA1-LAND1'.
perform bdc_field      using 'BDC_OKCODE'
                    '/00'.
perform bdc_field      using 'LFA1-NAME1'
                    WA_VEN-NAME1.
perform bdc_field      using 'LFA1-SORTL'
                    WA_VEN-SORTL.
perform bdc_field      using 'LFA1-LAND1'
                    'IN'.

*THIRD SCREEN
perform bdc_dynpro    using 'SAPMF02K' '0120'.
perform bdc_field      using 'BDC_CURSOR'
                    'LFA1-KUNNR'.
perform bdc_field      using 'BDC_OKCODE'
                    '/00'.

* FOURTH SCREEN
perform bdc_dynpro    using 'SAPMF02K' '0130'.
perform bdc_field      using 'BDC_CURSOR'
                    'LFBK-BANKN(02)'.
perform bdc_field      using 'BDC_OKCODE'
                    '=UPDA'.

LOOP AT IT_BANK INTO WA_BANK WHERE LIFNR = WA_VEN-LIFNR.
V2 = V2 + 1.
CONCATENATE 'LFBK-BANKS(' V2 ')' INTO V_FNAM.

perform bdc_field      using V_FNAM WA_BANK-BANKS.

CONCATENATE 'LFBK-BANKL(' V2 ')' INTO V_FNAM.

perform bdc_field      using V_FNAM WA_BANK-BANKL.

CONCATENATE 'LFBK-BANKN(' V2 ')' INTO V_FNAM.

perform bdc_field      using V_FNAM WA_BANK-BANKN.
ENDLOOP.
CLEAR V2.
CALL TRANSACTION 'XK01' USING BDCDATA MODE 'A'.
REFRESH BDCDATA.
ENDLOOP.

FORM BDC_DYNPRO USING PROGRAM DYNPRO.

```

```

CLEAR BDCDATA.
BDCDATA-PROGRAM = PROGRAM.
BDCDATA-DYNPRO = DYNPRO.
BDCDATA-DYNBEGIN = 'X'.
APPEND BDCDATA.
ENDFORM.

FORM BDC_FIELD USING FNAM FVAL.
CLEAR BDCDATA.
BDCDATA-FNAM = FNAM.
BDCDATA-FVAL = FVAL.
APPEND BDCDATA.
ENDFORM.

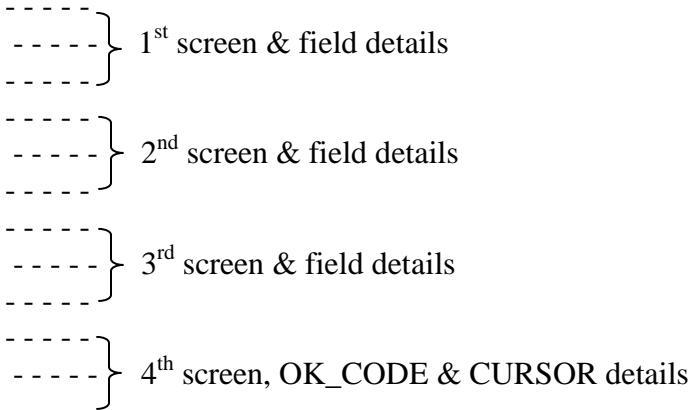
```

### Over view:

Data v2(2) type N.

Data v\_fnam(20) type c.

Loop at it\_ven into wa\_ven.



Loop at it\_bank into wa\_bank where lifnr = wa\_ven-lifnr.

V2 = V2 + 1.

Concatenate 'LFBK-BANKS(' V2 ') into V\_FNAM WA\_BANK-BANKS.

Concatenate 'LFBK-BANKL(' V2 ') into V\_FNAM WA\_BANK-BANKL.

Concatenate 'LFBK-BANKN(' V2 ') into V\_FNAM WA\_BANK-BANKN.

	LFBK-BANKS	LFBK-BANKL	LFBK-BANKN
01	IN	121212	1010101010
02			
03			
04			

Endloop.

Clear V2.

Call transaction 'XK01' using BDCDATA mode 'A'.

Refresh BDCDATA.

Endloop.

V_FNAM
LFBK-BANKN(01)

→ Develop a conversion program to upload the customer bank details flat file to SAP system by using 'XD01' transaction code. The flat file contains both customer & bank details with separation of C & B. In the flat file customer details are customer numbers, names, search term, street. In the flat file bank details are customer numbers, bank country key, bank key & account number.

Execute SHDB. Click on new recording. Provide recording name (SXD01\_TBC), transaction code (XD01). Enter. Provide the customer number (65432), account group (0004). Enter. Provide the name (Coal India), search term (CI), street (Ameerpet), country (IN), language (EN). Enter. Third screen we no need to provide any details. Enter. 4<sup>th</sup> screen we no need to enter any details. Enter. Provide the bank details.

Ctry	Bank key	Bank account
IN	121213	15151515
IN	121212	2525252525

KUNNR	NAME1	SORTL	STRAS
65432	GMR	G	SR NAGAR
65433	GVK	GVK	BG

Save. Click on save.

Loop at It\_cus into wa\_cus.

-----} First screen & field details.

-----} Second screen & field details.

-----} Third screen & field details.

-----} Fourth screen & field details.

-----} Fifty screen & CURSOR, OK\_CODE details.

KUNNR	NAME1	SORTL	STRAS

KUNN	BANKS	BANKL	BANKN
65432	IN	121212	1919191919
65433	IN	121212	2525151515
65433	IN	121213	1515152525

KUNNR	BANKS	BANKL	BANKN

Loop at it\_bank into wa\_bank where kunnr = wa\_cus-kunrn.

V2 = V2 + 1.

Concatenate 'KNBK-BANKS(' V2 ')' into V\_FNAM WA\_BANK-BANKS.

Concatenate 'KNBK-BANKL(' V2 ')' into V\_FNAM WA\_BANK-BANKL

Concatenate 'KNBK-BANKN' V2 ')' into V\_FNAM WA\_BANK-BANKN

	KNBK-BANKS	KNBK-BANKL	KNBK-BANKN
01	IN	121212	1515151515
02	IN	121212	2525252525
03			

Endloop.

Clear V2.

Call transaction 'XD01' using BDCDATA mode 'A'.

Refresh BDCDATA.

Endloop.

C,65432,GMR,G,SR NAGAR
C,65433,GVK,GV,BG
B,65432,IN,12121,1919191919
B,65433,IN,121212,2525151515
B,65433,IN,121213,1515152525

**Note:** - At the time of session processing through SM35 the following commands are used.

'/n' – It skips the current transaction from session processing.

**/BEND:** - It skips the entire session processing.

**/BDEL:** - It deletes the current transaction from the session processing.

### **Reasons to choose call transaction method:** -

1. My flat file contains fewer amounts of data (200 or 100)
2. My client requirement is immediate data base updation.
3. Call transaction method is faster.

### **Reasons to choose call Session method:** -

1. My flat file contains huge amount of data (1000 of records)
2. My client wants to run the program in background (scheduling)
3. Session method generates an error log. That will be handle errors.

### **Steps to execute the session method in background:** -

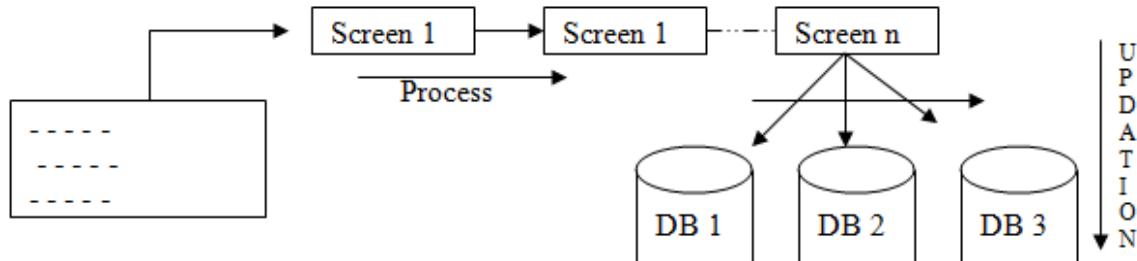
#### **Method 1:** -

Execute SM35. Select the session. Click on process. Select the radio button background. Click on process.

#### **Method 2:** -

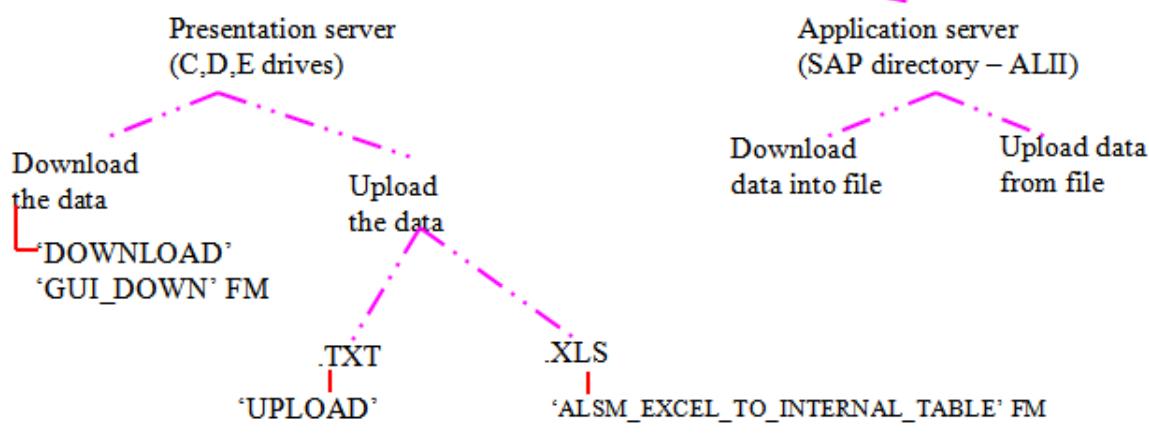
By using **RSBDCSUSB** Standard program, we can run the session method in background.

**Note:** - When ever we run the any conversion program in background then we must maintain the file in the application server directly (SAP directory AL11).



### **Syntax of call transaction:** -

Call transaction '<TCODE>' using '<BDCDATA internal table>' mode 'A/N/E' messages into '<BDCMSGCOLL internal table>' update 'A/S'. **Working with files**



### **UPLOAD:** -

Upload is the function module which is used to browse the file as well as upload the data from file to internal table. The input for the above function module is

1. File type → ‘DAT’
2. Data internal table which is similar as file.

```

TYPES: BEGIN OF TY_T001,
       BUKRS TYPE T001-BUKRS,
       BUTXT TYPE T001-BUTXT,
       ORT01 TYPE T001-ORT01,
       END OF TY_T001.
DATA: WA_T001 TYPE TY_T001,
      IT_T001 TYPE TABLE OF TY_T001.
CALL FUNCTION 'UPLOAD'
  EXPORTING
    FILETYPE = 'DAT'
  TABLES
    DATA_TAB = IT_T001.
LOOP AT IT_T001 INTO WA_T001.
  WRITE:/ WA_T001-BUKRS, WA_T001-BUTXT, WA_T001-ORT01.
ENDLOOP.

```

**GUI\_UPLOAD:** - It's the function module which is used to upload the data from file to internal table.  
The input for the above function module is

1. File name with extension.
- 2 . Field separator = ‘X’.
- 3 . Data internal table which is similar as file

```

Include ziiit001.
CALL FUNCTION 'GUI_UPLOAD'
  EXPORTING
    FILENAME          = 'C:\Users\Administrator\Desktop\Darling.txt'
    HAS_FIELD_SEPARATOR = 'X'
  TABLES
    DATA_TAB         = IT_T001.

LOOP AT IT_T001 INTO WA_T001.
  WRITE:/ WA_T001-BUKRS, WA_T001-BUTXT, WA_T001-ORT01.
ENDLOOP.

```

**Note:** - Now a day ‘upload’ function module is obsolete. So we use ‘GUI\_UPLOAD’ function module.  
For the ‘GUI\_UPLOAD’ function module file name is fixed. This isn’t allowed by client people and functional people.

Instead of UPLOAD function module we use F4\_FILENAME & GUI\_UPLOAD function module.  
F4\_FILENAME is the function module which is used to browse the file and GUI\_UPLOAD function module is used to upload the data from file to internal table. The output for the F4\_FILENAME is file path or file name.

```

DATA V_FILE TYPE STRING.
INCLUDE ZIIT001.
PARAMETER P_FILE LIKE IBIPPARMS-PATH.

AT SELECTION-SCREEN ON VALUE-REQUEST FOR P_FILE.
  CALL FUNCTION 'F4_FILENAME'

```

```

IMPORTING
  FILE_NAME = P_FILE.

START-OF-SELECTION.
  V_FILE = P_FILE.
  CALL FUNCTION 'GUI_UPLOAD'
    EXPORTING
      FILENAME          = V_FILE
      HAS_FIELD_SEPARATOR = 'X'
    TABLES
      DATA_TAB          = IT_T001.
LOOP AT IT_T001 INTO WA_T001.
  WRITE:/ WA_T001-BUKRS, WA_T001-BUTXT, WA_T001-ORT01.
ENDLOOP.

```

**Download:** - Download is the function module which is used to browse the file as well as download the data from internal table to file. The input for the above function module is

1. File type → 'DAT'
2. Data internal table

```

TABLES T001.
SELECT-OPTIONS S_BUKRS FOR T001-BUKRS.
INCLUDE ZIIT001.
SELECT BUKRS BUTXT ORT01 FROM T001 INTO TABLE IT_T001 WHERE BUKRS IN
S_BUKRS.
CALL FUNCTION 'DOWNLOAD'
  EXPORTING
    FILETYPE = 'DAT'
  TABLES
    DATA_TAB = IT_T001.

```

**GUI\_DOWNLOAD:** - It's the function module which is used to download the data from internal table to file. The input for the above function module is

1. File name with extension.
2. Field separation
3. Data internal table which data we want to download

```

TABLES T001.
SELECT-OPTIONS S_BUKRS FOR T001-BUKRS.
INCLUDE ZIIT001.
SELECT BUKRS BUTXT ORT01 FROM T001 INTO TABLE IT_T001 WHERE BUKRS IN
S_BUKRS.
CALL FUNCTION 'GUI_DOWNLOAD'
  EXPORTING
    FILENAME          = 'C:\Users\Administrator\Desktop\T002.TXT'
    WRITE_FIELD_SEPARATOR = 'X'
  TABLES
    DATA_TAB          = IT_T001.

```

**Note:** - Now a day 'DOWNLOAD' function module is absolute. So we use 'GUI\_DOWNLOAD' function module. In the 'GUI\_DOWNLOAD' function module the file name is fixed. This isn't allowed by functional people.

Instead of ‘DOWNLOAD’ we use ‘F4\_FILENAME’ function module with ‘GUI\_DOWNLOAD’ function module.

```

INCLUDE ZIIT001.
PARAMETER P_FILE TYPE IBIPPARMS-PATH.
DATA V_FILE TYPE STRING.

AT SELECTION-SCREEN ON VALUE-REQUEST FOR P_FILE.
  CALL FUNCTION 'F4_FILENAME'
    IMPORTING
      FILE_NAME = P_FILE.
START-OF-SELECTION.
  SELECT BUKRS BUTXT ORT01 FROM T001 INTO TABLE IT_T001.
  V_FILE = P_FILE.
  CALL FUNCTION 'GUI_DOWNLOAD'
    EXPORTING
      FILENAME          = V_FILE
      WRITE_FIELD_SEPARATOR = 'X'
    TABLES
      DATA_TAB          = IT_T001.

```

‘ALSM\_EXCEL\_TO\_INTERNAL\_TABLE’ is the function module which is used to upload the data from XL Sheet to internal table. The input for the above function module is

1. File name with extension
2. Begin column
3. Begin row
4. End column
5. End row

The output for the above function module is an internal table which contains 3 fields (Row, Col, Value)

```

PARAMETER: P_BC TYPE I,
           P_BR TYPE I,
           P_EC TYPE I,
           P_ER TYPE I.

```

```
INCLUDE ZIIT001.
```

```

DATA: IT TYPE TABLE OF ALSMEX_TABLINE,
      WA LIKE LINE OF IT.
CALL FUNCTION 'ALSM_EXCEL_TO_INTERNAL_TABLE'
  EXPORTING
    FILENAME      = 'C:\Users\Administrator\Desktop\88.xls'
    I_BEGIN_COL   = P_BC
    I_BEGIN_ROW   = P_BR
    I_END_COL     = P_EC
    I_END_ROW     = P_ER
  TABLES
    INTERN       = IT.
LOOP AT IT INTO WA.
  WRITE:/ WA-ROW, WA-COL, WA-VALUE.
ENDLOOP.

```

```

ULINE.
LOOP AT IT INTO WA.
  IF WA-COL = '0001'.

```

	0001	0002	0003	0004	-----
Row	1000	TCS	HYD		
	2000	IBM	CHE		
	3000	HCL	MUM		

IT	Row	Col	Value
	0001	0001	1000
	0001	0002	TCS
	0001	0003	HYD
	0002	0001	2000
	0002	0002	IBM
	0002	0003	CHE
	0003	0001	3000
	0003	0002	HCL
	0003	0003	MUM

WA	Row	Col	Value
	0001	0003	HYD

```

WA_T001-BUKRS = WA-VALUE.
ELSEIF WA-COL = '0002'.
  WA_T001-BUTXT = WA-VALUE.
ELSEIF WA-COL = '0003'.
  WA_T001-ORT01 = WA-VALUE.
ENDIF.
AT END OF ROW.
  APPEND WA_T001 TO IT_T001.
ENDAT.
ENDLOOP.
LOOP AT IT_T001 INTO WA_T001.
  WRITE:/ WA_T001-BUKRS, WA_T001-BUTXT, WA_T001-ORT01.
ENDLOOP.

```

IT_T001		
BUKRS	BUTXT	ORT01
1000	TCS	HYD
2000	IBM	CHE
3000	HCL	MUM

WA\_T001  
BUKRS BUTXT ORT01  
3000 HCL MUM

### Application Server: -

Application server is the SAP directory. The transaction code for application server is AL11. In the application server each file is called one data seg.

In the application server we can't create the file directly through program only we can create.

**Note:** - .Directory is the [.(DIR\_TEMP)] is the default directory in the application server.

### Steps to download the data into application server: -

1. Open the data set / file in write mode / output.
2. Loop at <data internal table> into <work area>.  
Transfers the data from <work area> to data set / file.  
Endloop.
3. Close the data set / file.

### Syntax of open data set: -

Open data set '<file name>' in text / binary mode for output / input encoding default - - .

### Steps to open the file in application server: -

Execute AL11. Double click on .(DIR\_TEMP). Identify the file (SPT). Double click on it. And absorb the data.

```

INCLUDE ZIIT001.
SELECT BUKRS BUTXT ORT01 FROM T001 INTO TABLE IT_T001 UP TO 10 ROWS.
* Download the data to application server
OPEN DATASET 'SPT' IN TEXT MODE FOR OUTPUT ENCODING DEFAULT.
LOOP AT IT_T001 INTO WA_T001.
  TRANSFER WA_T001 TO 'SPT'.
ENDLOOP.
CLOSE DATASET 'SPT'.

```

### Steps to upload the data from application server: -

1. Open the dataset / file in read mode
2. Do
  - Read the data set and placed into work area.
  - If sy-subrc = 0.
  - Append the data from work area to internal table.
  - Else.
  - Exit.
  - Endif.

Enddo.

3. Close the dataset / file

```
INCLUDE ZIIT001.
```

```
OPEN DATASET 'SPT' IN TEXT MODE FOR INPUT ENCODING DEFAULT.
```

```

DO.
  READ DATASET 'SPT' INTO WA_T001.
  IF SY-SUBRC = 0.
    APPEND WA_T001 TO IT_T001.
  ELSE.
    EXIT.
  ENDIF.
ENDDO.
CLOSE DATASET 'SPT'.
* Display the output.
LOOP AT IT_T001 INTO WA_T001.
  WRITE:/ WA_T001-BUKRS, WA_T001-BUTXT, WA_T001-ORT01.
ENDLOOP.

```

**Note:** - When ever we develop the conversion program to schedule in background then file must be in the application server.

In this case we need to develop one more separate program to upload the data from presentation server & download into application server. In the conversion program instead of GUI upload we upload the data from application server by using open data set and read dataset.

### → Develop a program for uploading one excel sheet into internal table and save that data into one table.

```

data a.
types: begin of ty_zmb_fc,
  mandt type mandt,
  WERKS type mard-werks,
  MATNR type mard-matnr,
  Z_MB_FC type zmb_fc-z_mb_fc,
  Z_CLASS type zmb_fc-z_class,
  Z_BULK type zmb_fc-z_bulk,
  SEQUENCE(10) TYPE C,
end of ty_zmb_fc.
DATA WA_zmb_fc TYPE ZMB_FC.
data wa_zmb_fc1 type zmb_fc.
DATA it_zmb_fc TYPE STANDARD TABLE OF zmb_fc.
DATA it_zmb_fc1 TYPE STANDARD TABLE OF zmb_fc.
DATA: fname TYPE ibiparms-path,
      file TYPE rlgrap-filename.
SELECTION-SCREEN: BEGIN OF BLOCK b1.
PARAMETERS : p_file TYPE ibiparms-path.
SELECTION-SCREEN: END OF BLOCK b1.
DATA trux TYPE truxs_t_text_data.
AT SELECTION-SCREEN ON VALUE-REQUEST FOR p_file.
  CALL FUNCTION 'F4_FILENAME'
    EXPORTING
      program_name = syst-cprog
    IMPORTING
      file_name      = fname.
  file = fname.
Start-of-selection.
  CALL FUNCTION 'TEXT_CONVERT_XLS_TO_SAP'
    EXPORTING
      i_line_header      = 'X'
      i_tab_raw_data     = trux
      i_filename         = file
    TABLES

```

```

    i_tab_converted_data = it_zmb_fc
EXCEPTIONS
    conversion_failed      = 1
    OTHERS                 = 2.
IF sy-subrc = 0.
LOOP AT it_zmb_fc INTO WA_zmb_fc.
    CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
        EXPORTING
            input = WA_zmb_fc-matnr
        IMPORTING
            output = WA_zmb_fc-matnr.
    CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
        EXPORTING
            input = WA_zmb_fc-z_bulk
        IMPORTING
            output = WA_zmb_fc-z_bulk.
    MODIFY IT_zmb_fc FROM WA_zmb_fc.
ENDLOOP.
MODIFY zmb_fc FROM TABLE it_zmb_fc.
COMMIT WORK.
IF sy-subrc = 0.
    MESSAGE 'Data Updated' TYPE 'S'.
ENDIF.
ENDIF.

```

### **Interview questions on BDC: -**

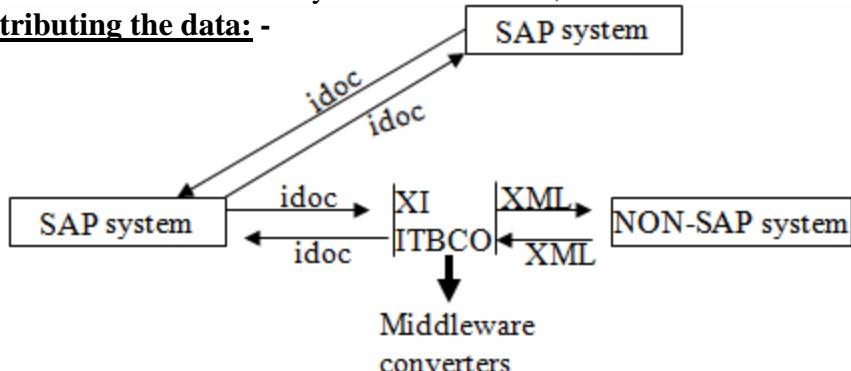
1. Why we go for BDC.
2. What are steps to work with BDC programming?
3. What are differences between DML commands & BDC?
4. What are differences between call transaction method & session method?
5. When we choose call transaction method.
6. When we choose Session method.
7. What are the types of update modes? (Synchronous & Asynchronous)
8. How can we handle the errors in call transaction method?
9. What are the function modules we used in session method?
10. How to process the session (SM35)
11. How we run the session in background (RSBDCSUB standard program).
12. How to do the recording (SHDB)
13. What is the last entry at screen? (BDC-OK\_CODE)
14. What are the fields in BDC data structure / dynpro components?
15. What is the use '/N', '/BEND', '/BDEL'.
16. How we hold/lock the session until particular date.
17. Can we process multiple transactions at a time? (By using session)
18. When we maintain the file in application server.
19. What is transaction code for application server? (AL11)
20. What are components of BDCMSCOLL structure & what is the use (it hold the Success / failure information).
21. Can we use call transaction method & session method in the same program (yes)
22. What is the use of FORMAT\_MESSAGE function module (handle the errors)

**Note:** - TEXT\_CONVERT\_XLS\_TO\_SAP is the FM to upload the data from excel sheet to SAP system. Here we use one type that is TRUX\_T\_TEXT\_DATA.

# CROSS APPLICATIONS

- Cross application is the concept to exchange the data among the systems.
- ALE (Application Link Enabling) is an SAP technology to support cross application.
- ALE uses IDOC to support the cross applications.
- IDOC is the carrier to carry the data from one system to another system.
- SAP can understand only the IDOC format, when it communicates with any other system.

## Distributing the data: -



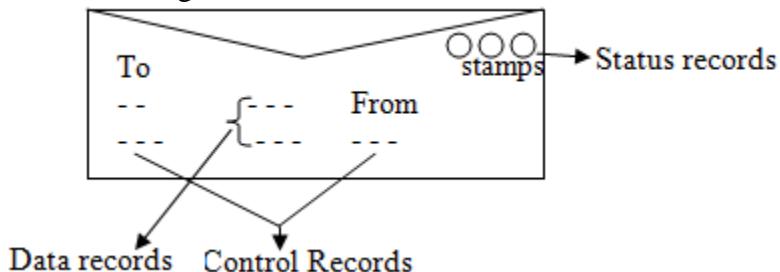
- Irrespective of the receivers, the ABAPER job in sender system is to generate the Idoc.
- The process of generating the Idoc is nothing but out bound process.
- Irrespective of the senders, the abaper job in receiver system is to collect the data from Idoc.
- The process of collecting the data from Idoc is nothing but in bound process.

## Runtime components of Idoc: -

1. It generates a unique Idoc number, which is 16 digit.

2. It generates 3 types of records.

- i. Control records
- ii. Data records
- iii. Status records



### 1. Control record : -

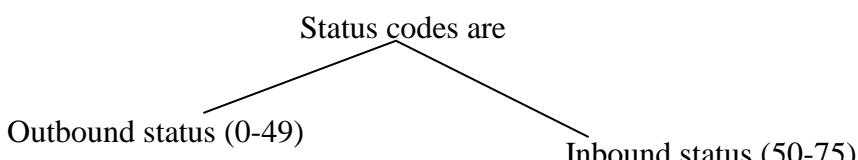
- i. Control record specifies the sender as well as receiver information.
- ii. It generates only one control record.
- iii. This information will be saved on EDIDC table

### 2. Data records: -

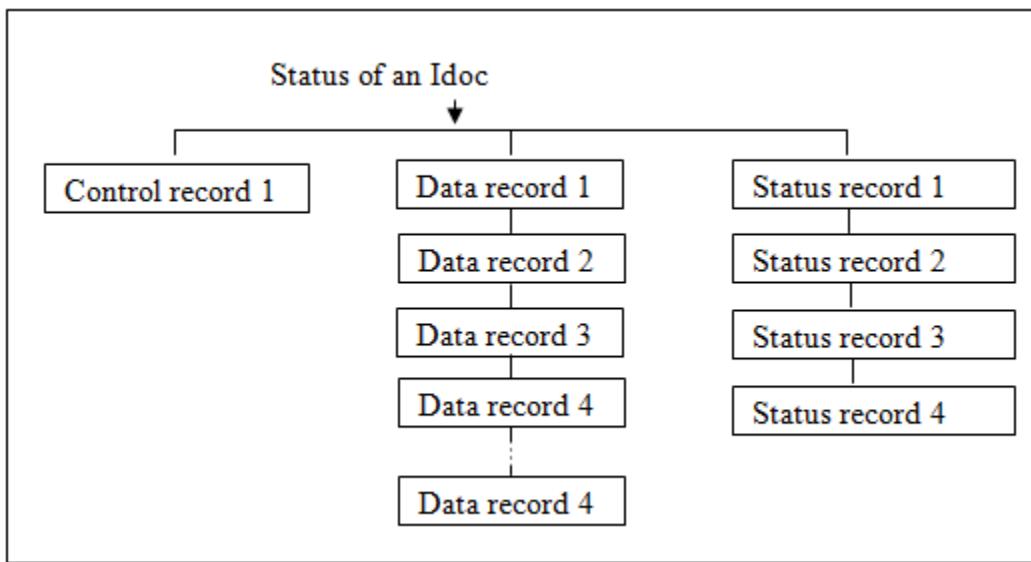
- i. It specify the data which is send by the senders system.
- ii. It generates any number of data records.
- iii. This information will be saved on EDIDD table.

### 3. Status records: -

- i. It generates status codes for each and every stage of transferring the data.
- ii. It generates any number of status records.
- iii. This information will be saved on EDIDS table.



**Note:** - The linking between EDIDC, EDIDD and EDIDS is the IDOC number.



### Types of IDOCS

1. BASIC type → Standard Idoc
2. Extension → Standard Idoc + Custom Idoc

#### Standard Idoc:

If you want to send as well as receives the standard data base table information then we go for standard idoc.

#### LFA1

LIFNR      *Send*  
NAME1      *Receive*  
ORT01

#### Custom Idoc –

If you want to send as well as receives the custom table information then we go for custom idoc.

Zhai      *Send*  
EID      *Receive*  
ENAME  
ESAL

#### Extension Idoc –

If you want to send as well as receives the additional fields information of standard data base table along with standard fields information then we go for extension Idoc.

#### LFA1

LIFNR  
NAME1  
ORT01

||

+

ZHAI  
EID  
ENAME  
ESAL

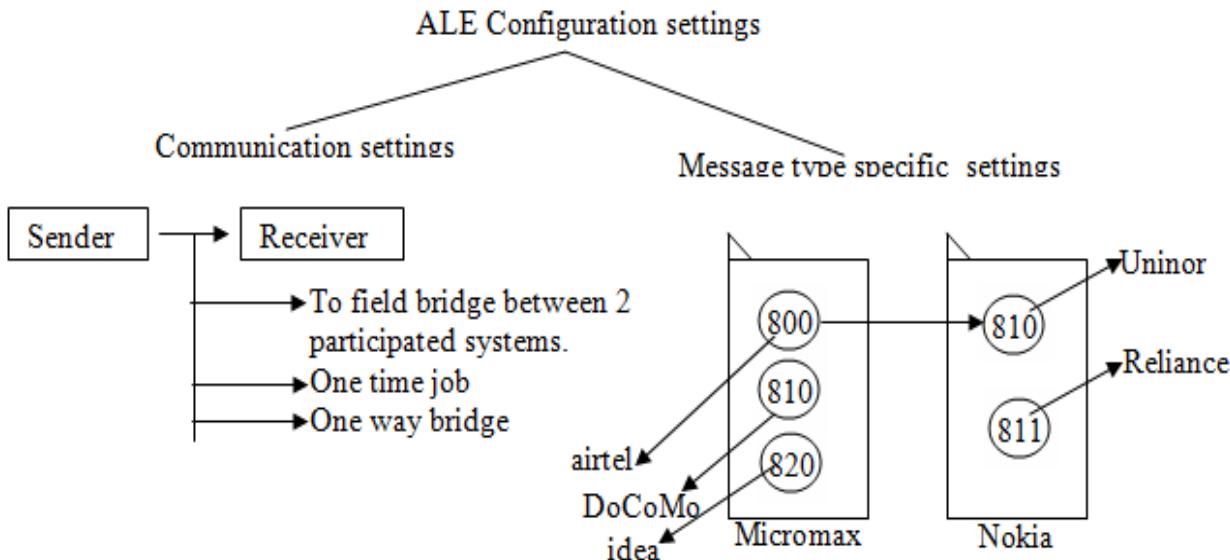
**Note:** - IDOC is the collection of segments. Each segment is the collection of fields.

### Characteristics of an IDOC: -

1. Name of the IDOC.
2. List of the segment
3. Hierarchy of the segments in the IDOC.
4. Optional (vs) mandatory for the segments
5. Provide parent & child relationship of the segments.
6. Each segment can carry up to 1000 bytes.
7. Provide the minimum and maximum number of repetitions for the segment.

Seg 1	F1	F2	F3
Seg 2	F4		F5
Seg 3	F6	F7	F8   F9
Seg 4	F10		F11

IDOC



Communication between one system to another system is nothing but communication between one client of sender system to another client of receiving system. Each participated system is called one logical system.

### Steps to establish the communication settings: -

1. Define logical system (SALE)
2. Assign client to logical system (SCC4)
3. Maintain RFC destination details (SM59)

**Note:** - In the real time communication settings are established by BASIS people.

If the receiver is available in following address & he wants vendor H7070 details.

Client	:	810
User	:	SAPUSER
Password	:	india123
Logical system:	:	SP810.

In this before sending the vendor details first we need to establish the communication settings from sender to receiver system.

### Steps to define logical system: -

Execute SALE. Expand basic settings. Expand logical systems. Execute define logical system. Enter. Click on new entries in the application tool bar. Provide sender, receiver logical system name & short description. Save.

SP800	Sender logical system
SP810	Receiver logical system

### Steps to assign client to logical system: -

Execute 'SCC4'. Click on change mode. Select the client. Click on details. Provide the logical system name SP800. Click on save.

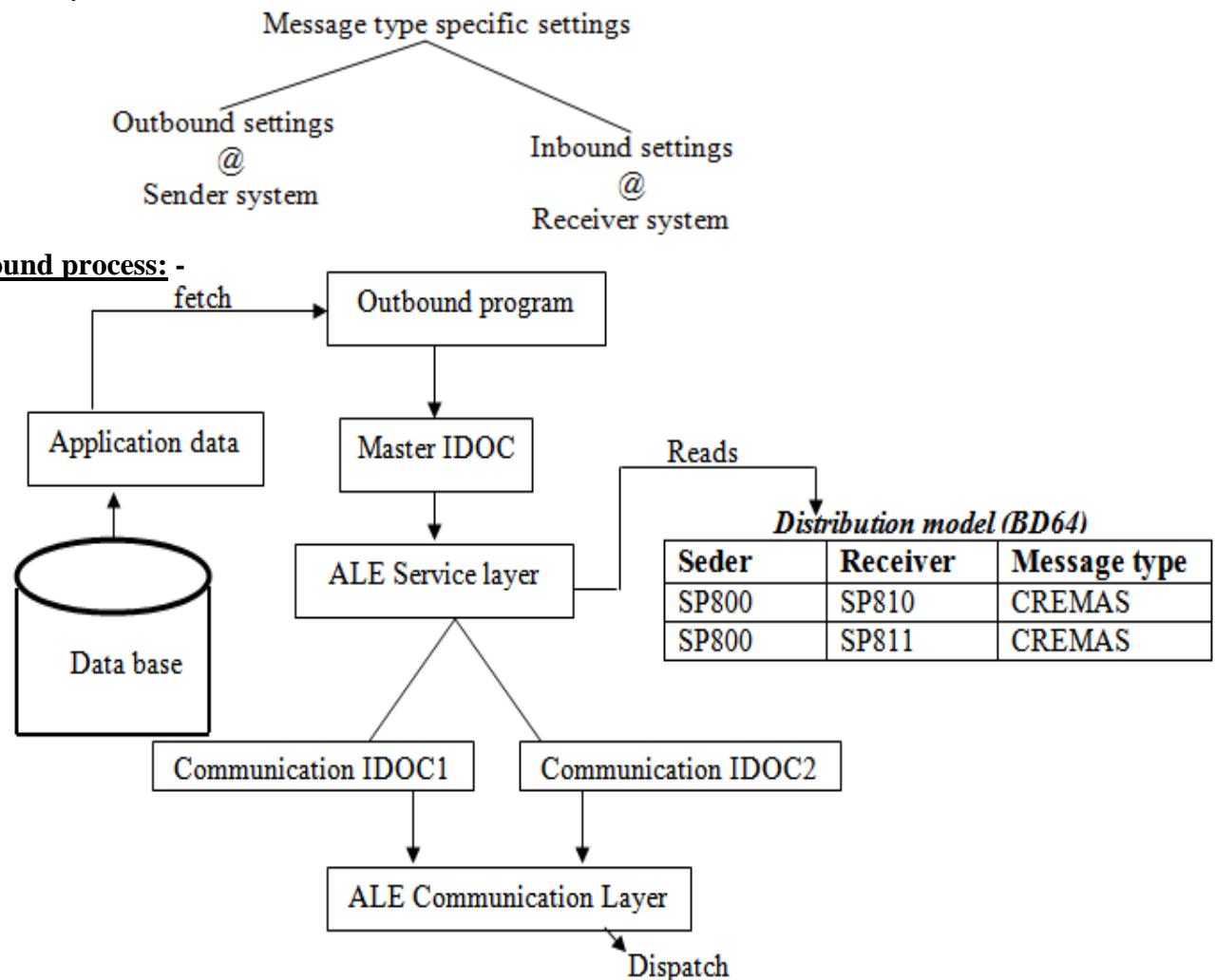
### **Steps to maintain RFC destination details: -**

Execute ‘SM59’. Select the ABAP connections. Click on create. Provide RFC destination (SP810). Provide short description (Sender to receiver). Click on logon & security tab. Provide the receive log on details.

Client : 810  
User : SAPUSER  
PW Status : is initial  
Password : india123.

Click on save. Click on connection test. Click on back. Click on remote logon in the application tool bar.

**Note:** – The table is cross-client. It means what ever the changes are made in one client those are automatically reflected into all other clients in same server.



- Based on the given input one outbound program will triggered & fetch the application data from the data base & generate the master IDOC.
- Master IDOC is nothing but data in internal table. Master IDOC won't save anywhere in SAP.
- ALE service layer reads the distribution model & identifies the interested receivers. Based on the receivers it generates the communication IDOC.
- Distribution model is the collection of senders, receivers, message type.
- Message type is used to identify the type of the application (vendor, customer, material, . . .).
- Communication IDOC is the physical IDOC which is receiver specific.
- ALE communication layer dispatch these communication IDOC to their relevant receiver systems.

**Note:** - If you get the zero master IDOCs, the reason is the given input having no data in the data base.

**Note:** - If you get the zero communication IDOCs the reasons are

1. There is no interested receivers are available in distribution model.
2. Communication settings problem.

Steps to identify the transaction code based on short description: -

Execute 'SDMO'. Provide short description (Send vendor). Execute. Then we identify the transaction codes.

**Some of the standard message types: -**

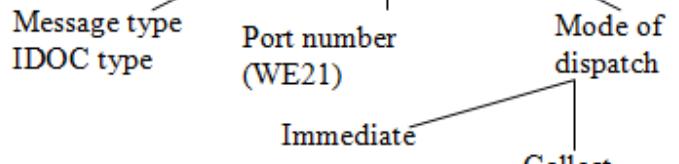
TCODE	MESSAGE TYPE	IDOC TYPE	SHORT DESCRIPTION
BD10	MATMAS	MATMAS01	
		MATMAS02	Send Material
		MATMAS05	
BD11	MATMAS	DEBMAS01	Get Material
		BEBMAS02	Send Customer
BD12	DEBMAS		
		BDEMAS06	
BD13	DEBMAS	CREMAS01	Get customer
		CREMAS02	Send Vendor
BD14	CREMAS		
		CREMAS05	
BD15	CREMAS		Get Vendor

#### ALE configuration steps for standard IDOC outbound

Create distribution mode (BD64)

Sender	Receiver	Message type

Create bound partner profile (WE20)



Steps to create distribution model: -

Execute 'BD64'. Click on change mode. Click on create model view in the application tool bar. Provide the short description (distribution), technical name (DBCM1). Enter. Select the distribution model which is in the last. Click on add message type in the application tool bar. Provide sender (SP800), receiver (SP810) and message type (CREMAS). Enter. Repeat the same step for all the receivers. Click on save.

Port number: -

Port number is used to specify the way of transferring the data. Types of ports are.

1. TRFC port → ALE/IDOC
2. File port → LSMW, EDI/IDOC
3. Internet port → Web Apps
4. XML port → Java

Steps to create the port number: -

Execute 'WE21'. Select the transactional RFC. Click on create (F7). Enter. The system automatically generates a port number (A000075). Provide short description (port number). Provide RFC destination (SP810). Click on save.

Steps to create outbound partner profile: -

Execute 'WE20'. Select the partner type LS. Click on create. Provide partner number (SP810). Click on save. Click on create out bound parameter plus (+) button. Provide the message type (CREMAS). Provide the port number (A000075). Select the radio button transfer IDOC immed. Select the basic type (CREMAS05). Click on save.

Steps to send the vendor details: -

Execute BD14. Provide the vendor number (H7070). Provide the message type (CREMAS), target system (SP810). Click on execute and absorb the master and communication IDOC s.

Steps to create vendor: -

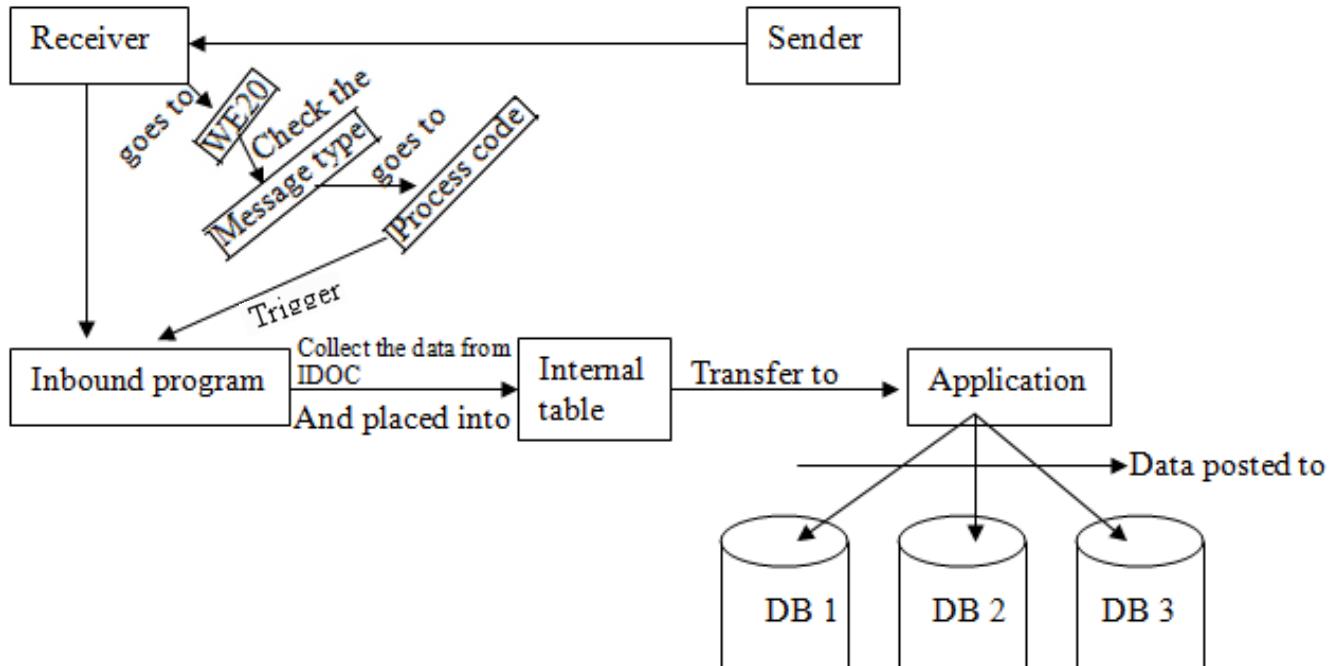
Execute 'XK01'. Provide the vendor number (H7070), Account group (0004). Enter. Provide the name (GMR COR), search term (GMR), country (IN). Click on save.

Steps to check or test the IDOC: -

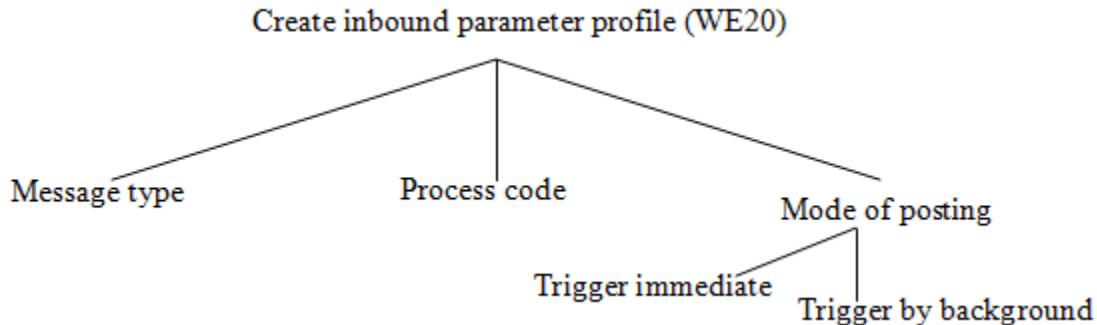
Execute 'WE02' or 'WE05'. Provide the logical message (CREMAS), partner number (SP810). Execute. & absorb the run time components (data control, status, data records).

**Note:** - After we get the status code 03, we execute 'RBDMOIND' standard program. If the status code 03 turn to 12 then the IDOC is successfully reached to destination. If the status code remains 03, then the IDOC is in the TRFC queue. If the status code turns to 11, then the IDOC is damaged in the queue.

Inbound process: -



**Note:** - Outbound program can be developed either through executable program or through function module where as inbound program must be developed through functional module only because the interface parameters are same for any message type (import, export, changing, ---).



At the first time, before creating the inbound partner profile we do following things.

1. Define logical systems (this isn't required, if we are working with the same server).
2. Assign client to logical system.

**Note:** - Process code is used to identify the function module.

In the receiver system 810 client: -

Steps to assign client to logical system: -

Execute SCC4. Click on change mode. Enter. Select the client (810). Click on details. Provide logical system (SP810). Save.

Steps to create inbound partner profile: -

Execute 'WE20'. Select the partner type LS. Click on create (F5). Provide partner number (SP800). Click on save. Click on create inbound parameter. Provide the message type (CREMAS). Select the process code (CRE1). Click on save.

Steps to test / check the idoc: -

Execute WE02/05. Provide the message type (CREMAS), partner number (SP800). Execute.

Steps to reprocess the idoc: -

Execute 'BD87'. Provide the idoc number (85001). Execute. select the error. Click on process.

Inbound process: -

After idoc reached to receiver system, then it goes to WE20 inbound partner profile & check the message type. If the message type is available, then it goes to process code. Against the process code it triggers the inbound program (function module). The inbound program collects the data from idoc & placed into internal table. From the internal table the data is transferred to their application. From the application the data is posted to their relevant data base tables.

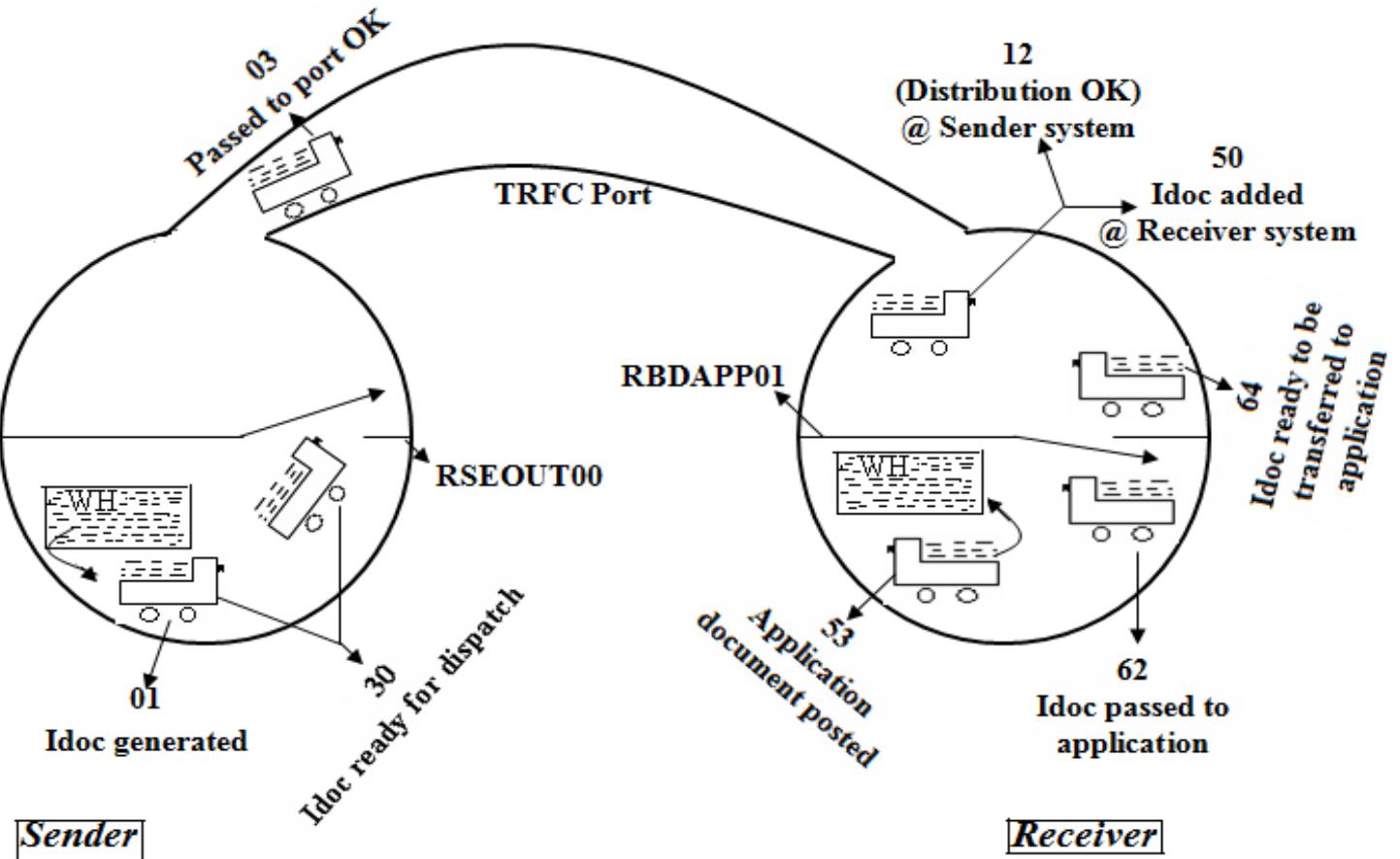
**Note:** - At the time of creating the outbound partner profile if we select the mode of dispatch is collect then we must execute '**RSEOUT00**' standard program then only the collect idocs will be dispatched to their relevant receiver system.

**Note:** - At the time of creating the inbound partner profile if you select the mode of posting is trigger by background (collect) then we must execute '**RBDAPP01**' standard program. Then only the collect idocs will be posted to their relevant data base table.

**Note:** - EDI stands for Electronic Data Interchange

**Note:** - IDOC stands for Intermediate Document.

**Note:** - MTART = Material Type, HALB = Semi finished product.



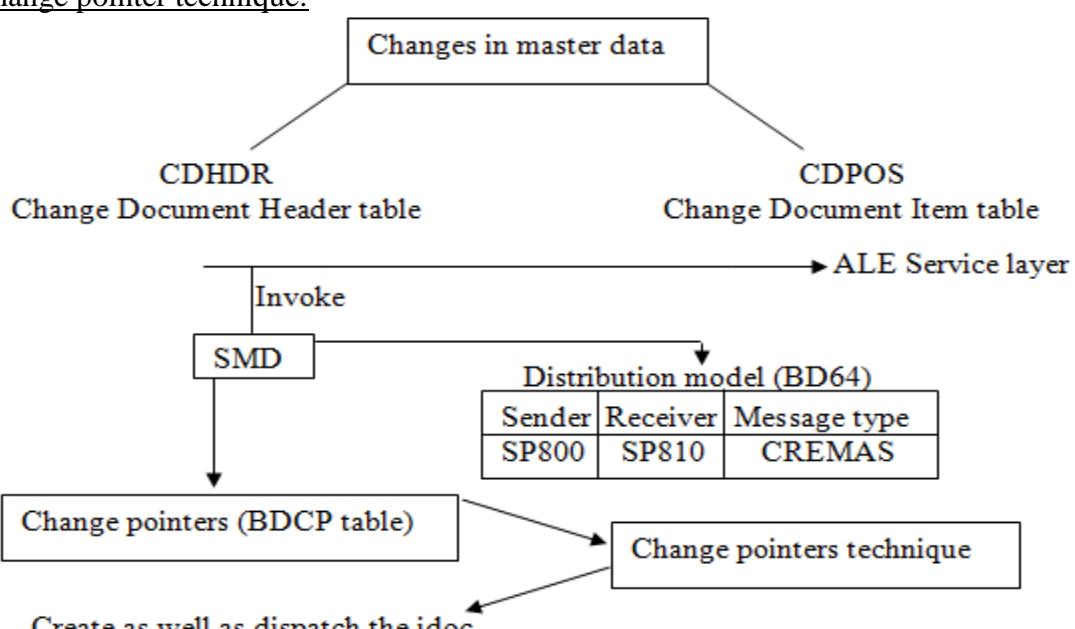
#### Types of distributing the data: -

1. Send entire information
2. Send changes only (change pointer technique)
3. Get entire information

#### Change pointer technique: -

Change pointer technique is used to send the changes of master data from sender to receiver system.

#### Process flow of change pointer technique: -



When ever the changes occurred in the master data the standard SAP itself prepare one change document. ALE service layer invokes the SMD (Shared Master Data) SMD reads the distribution model & identifies the interested receivers. If any receiver is available then it generates the change pointers for the change document. These change pointers are available in ‘BDCP’ table. The change pointer technique reads the change pointers & generates as well as dispatch the idoc to the receiver system.

#### **ALE configuration steps for change pointer technique: -**

##### **1. ALE Configuration steps for Idoc outbound.**

Create distribution model (BD64)      Create outbound partner profile (WE20)

2. Activate the change pointer technique (BD61)
3. Activate the message type (BD50)
4. Generate as well as dispatch the IDOC (BD21)

#### **→ Configure the ALE to send the changes of vendor master data from sender to receiver system**

##### Steps to activate change pointer technique: -

Execute BD61. Select the change box change points activated. Click on save.

##### Steps to activate message type: -

Execute ‘BD50’. Click on position. Provide the message type (CREMAS). Enter. Select the activate checkbox (CREMAS, CREMAS\_SUSMM). Click on save.

##### Steps to change the vendor: -

Execute ‘XK02’. Provide vendor number (H7071). Select the address check box. Enter. Provide the title (Company) change search to BIG instead of ‘B’. Click on save.

##### Steps to generate as well as dispatch for IDOC for change pointers: -

Execute ‘BD21’. Provide the message type (CREMAS). Execute.

Check the idoc in WE02 / WE05.

**Note:** – Change pointer technique always at senders system only for master data.

##### Steps to identify the old & new changes of any document: -

Execute SE11. Open the table CDHDR in display mode. Click on contents. Provide the object ID as your document number (H7071). Execute. Identify the object class & changes.

Open the CDPOS table in SE11. Click on contents. Provide the object class (KRED) object ID (H7071) changenr (592227). Execute & identify the old & new values of the document.

##### Some of the standard requesting message type: -

<b><u>Application</u></b>	<b><u>Message type</u></b>	<b><u>Requesting Message type</u></b>
VENDOR	CREMAS	CREFET
CUSTOMER	DEBMAS	DEBFET
MATERIAL	MATMAS	MATFET

**Note:** – ‘ALEREQ01’ is the IDOC type for any requesting message type.

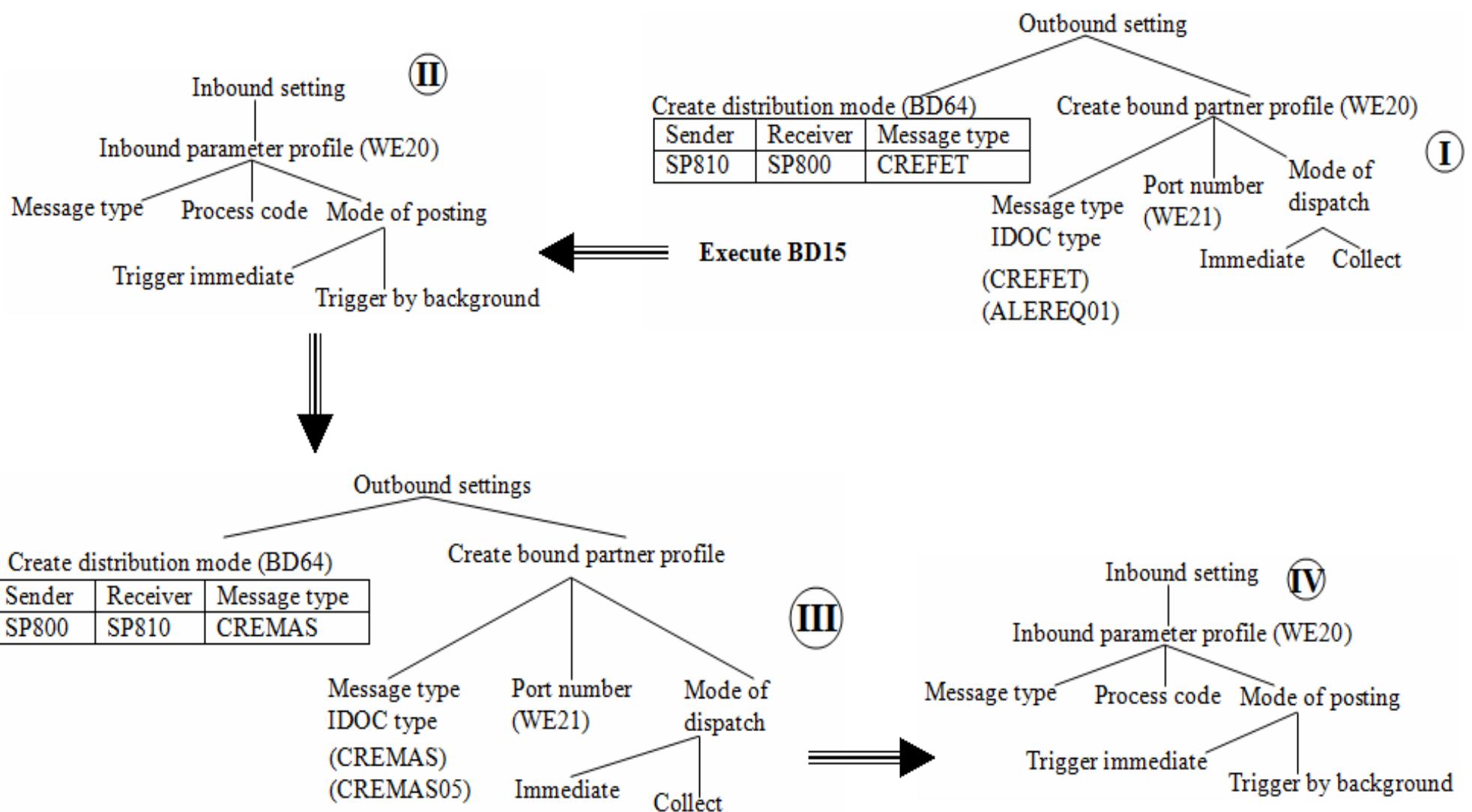
**Note:** – ‘EDIMSG’ is the standard data base table which contains all the message types, IDOC types and requesting message types.

#### **Getting entire information**

***Note:*** – Before configure the get entire information we must establish the communication settings from receiver to sender.

##### Steps to Establish the communication settings from receiver to sender: -

1. Define the logical system → it does not require if we are working with same server.
2. Assign client to logical system.
3. Maintain RFC destination details



In receiver system 810: -

Steps to maintain RFC destination details: -

Execute SM59. Select the ABAP connections. Click on create. Provide RFC destination (SP800). Provide short description (Receiver to sender). Click on logon & security tab. Provide sender logon details.

Client : 800

User : sapuser

Password : india123.

Click on save. Click on connection text. Come back. Click on remote log on.

Steps to create distribution model: -

Execute BD64. Click on change mode. Click on create model view. Provide short description, technical name. Enter. Select the distribution model which is in last. Click on add message type in the application tool bar. Provide sender (SP810), Receiver (SP800), message type (CREFET). Enter. Click on save.

Steps to create port number: -

Execute WE21. Select the transactional RFC. Click on create. Enter. Provide short description, port number, RFC destination (SP800). Click on save.

Steps to create outbound partner profile: -

Execute WE20. Select partner (SP800). Click on create outbound parameter [(+) symbol]. Provide message type (CREFET), port number (A000018). Click on transfer IDOC immediate. Select the basic type (ALEREQ01). Save.

Steps to get the vendor: -

Execute BD15. Provide the vendor number which vendor you want (H8091). Message type (CREMAS). Execute.

Steps to check the IDOC: -

Execute WE02. Provide the message type (CREFET), partner number (SP800). Execute.

In sender system 800 client: -

Steps to create inbound partner profile: -

Execute WE20. Select the partner (SP810). Click on create inbound parameter. Provide the message type (CREFET). Select process code (CREF). Save.

Steps to check the idoc: -

Execute 'WE05'. Provide the message type (CREFET). Provide the partner number (SP810). Execute.

**Note:** - In the receiver requested data isn't available. Then we get the status 51. (No object for requested Idoc selected for sending).

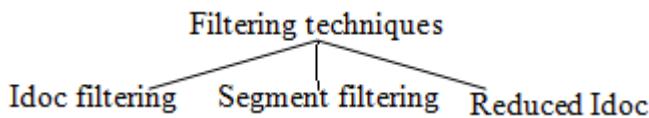
**Note:** - The name of the standard segment start with 'E'. The name of the custom segment starts with 'Z1'. The definition name of the standard segment starts with 'EZ'. The definition of custom segment starts with 'Z2'.

**Note:** - The segment definition is useful when we communicate with non-SAP system. Based on the segment definition only the middleware converters convert the sender's format to receiver's format.

### Filtering technique: -

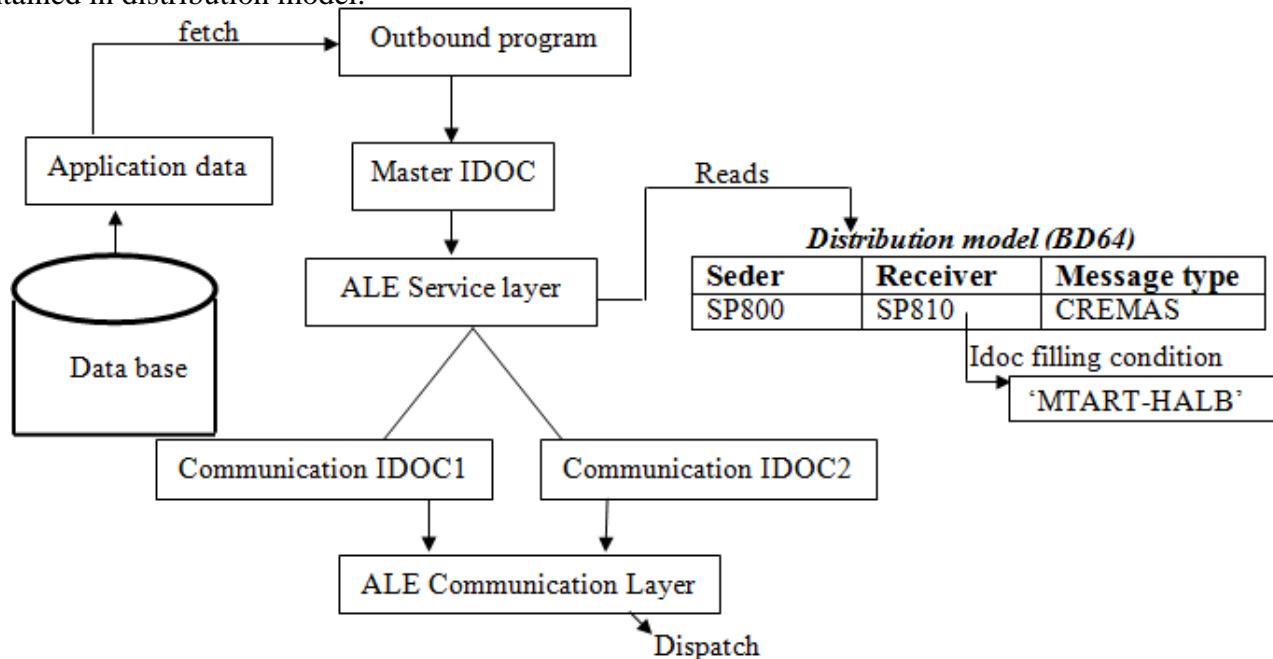
Filtering techniques are used to generate the idoc based on the conditions. Filtering techniques are always at sender system.

There are 3 types of filtering techniques.



### Idoc filtering: -

Idoc filtering is used to drop the idoc at run time. Idoc filtering conditions are placed or maintained in distribution model.



Before generating the communication idocs ALE service layer reads the distribution model & identifies the interested receivers. If any receiver is available then it checks the given input satisfies the filtering conditions or not. If it satisfies the filtering condition then only it generates the communication idoc. Otherwise it won't generate any communication idoc.

### **→ Configure the ALE to send only semi finished products to the receiver system**

In this object we provide the idoc filtering conditions in the distribution model.

#### Steps to create distribution model: -

Execute 'BD64'. Click on change mode. Click on create model view in the application tool bar. Provide the short description & technical name (DBMATMAS). Enter. Select the distribution model which is in last. Click on add message type. Provide sender, receiver, and message type (MATMAS). Enter. Expand the distribution model until 'No filter set'. Double click on it. Click on create filter group in the bottom. Expand the data filtering. Expand the filter group. Double click on our requirement (material type). Click on insert row button (+). Select the value (HALB). Enter. Click on ok. Save the distribution. Create outbound partner profile WE20.

#### Steps to send the material: -

Execute 'BD10'. Provide the material number (100-110), message type (MATMAS), logical system (SA810). Execute. Here 100 – 110 is a raw material. So it won't generate any communication idoc.

#### Segment filtering: -

It's used to drop the segments permanently to the particular receiver system. The transaction code for segment filtering is 'BD56'.

**Note:** - 'WE30' is the transaction code to create as well as display the idoc.

**Note:** - ‘WE31’ is the transaction code to create as well as display the segment.

**Note:** - When ever we are working with segment filtering then we must create dummy partner profile for sender system itself.

Steps to create dummy partner profile for sender itself: -

Execute WE20. select the partner type LS. Click on create. Provide the partner number SP800. Save.

**→ Configure the segment filtering to drop the ‘E1LFA1A’ segment information permanently to the SP810 receiver.**

Steps to work with segment filtering: -

Execute BD56. Provide the message type. Click on new entries (F5) in the application tool bar. Provide the sender type (LS) sender (SP800). Receiver type (LS), Receiver (SP810) segment type (E1LFA1A). Repeat the same steps for all which are dropped. Click on save.

**Reduced Idoc:** -

Reduced idoc is used to drop the segments as well as fields permanently to the particular receiver system. We can't drop the mandatory segments as well as mandatory fields which are in green colour. The transaction code for reduced idoc is ‘BD53’.

In the reduced idoc we create the new message type based on the existing message type. Based on the new message type we configure the ALE. (Create distribution model, outbound partner profile).

Steps to work with reduced idoc: -

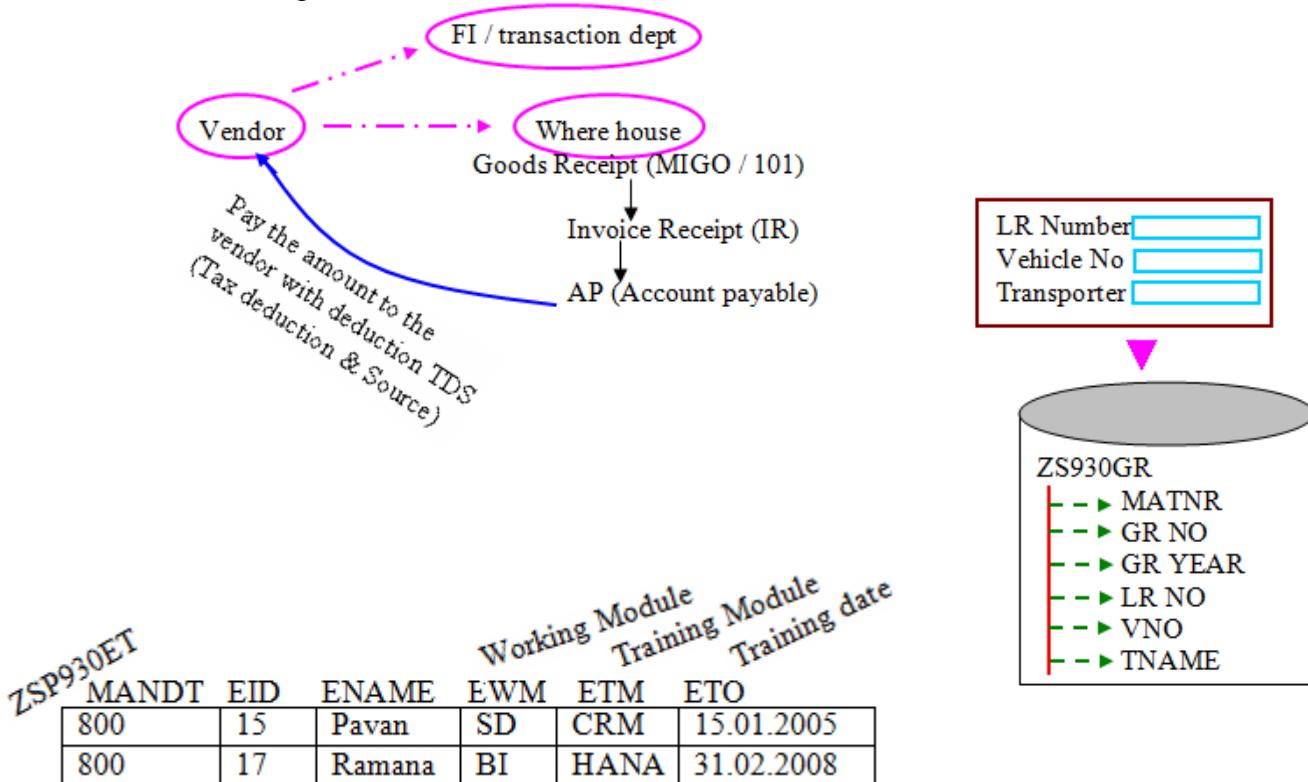
Execute BD53. Provide the reduced message type (DEBMAS). Click on create. Provide the reference message type (DEBMAS). Enter. Provide short description (Reduced message of customer). Select the required segments by placing the cursor on the segment. Click on select button in the application tool bar. Double click on the segment. Select the required fields. Click on select. Enter. Repeat the same steps for all other segments. Click on save. Based on this new message type (ZSDEBMAS) we create the distribution model & outbound partner profile.

**Differences between segment filtering and reduced idoc:** -

<u>Segment filtering</u>	<u>Reduced Idoc</u>
<ol style="list-style-type: none"><li>1. Segment filtering is used to drop the segments permanently to the receiver system.</li><li>2. In this the selected segment only dropped.</li><li>3. The transaction code of this is BD56.</li><li>4. We no need to configure the ALE once again</li></ol>	<ol style="list-style-type: none"><li>1. Reduced idoc is used to drop the segments as well as fields information permanently to the particular receiver system.</li><li>2. In this idoc the selected segments only send.</li><li>3. The transaction for this is BD53.</li><li>4. Based on the new message type we configure the ALE once again.</li></ol>

Against vendor supplies the where house department create a GR [Goods Receipt] with ‘MIGO’ transaction with 101 moment type. Against GR the Finance people physically verify the stock & prepare the invoice receipt (IR). Against IR document the account payable department pay the amount to the vendor with deducts of TDS amount. After few days the vendor asks the form 16 to the tax department of the company. When ever the tax department or Finance department generate the form 16 then need to enter the LR number. Vehicle number, transporter name etc. This information is not captured in the entire MM life cycle. Here design one screen with those fields & attached to ‘MIGO’ transaction by using ‘BADI’. When ever the where house people create the GR then they also maintain these additional screen

information. These information stored in separated ‘Ztable’. If you want to send or receive this ‘Ztable’ information then we go for custom idoc.



Each batch has their own characteristics (Classification) that information is maintained in separate ‘ZTABLE’. If you want to send as well as receive that ‘ZTABLE’ information then we go for custom idoc.

The diagram shows the ZSP930BC table structure with annotations for classification fields:

**Annotations:**

- Manufactured by**: Points to the WERKS column.
- Supplied by**: Points to the CHARG column.
- Storage location**: Points to the SLCON column.
- Condition**: Points to the DIA column.
- Dimensions**: Points to the ASSAY and WATER columns.

MANDT	MATNR	WERKS	CHARG	Mfg By	SUP BY	SLCON	DIA	ASSAY	WATER

If you want to open the batch classification details then execute ‘MSC3N’. Provide the material, plant & batch. Click on enter. Click on classification tab. Identify the classification details. ‘MCHA’ is the standard data base table which contains material batch details.

**→ Configure the ALE to send as well as receives the GR additional information which is stored in the following table.**

Fieldname	Key	Data element	Domain	Data type	Length	Short description
MANDT	✓	As per SAP standard field	MANDT			
GRNO	✓	As per SAP standard field	MBLNR			
GRYEAR	✓	As per SAP standard field	MJAHR			
LRNO		ZS930LRNO	YS930LRNO	CHAR	15	Lorry Receipt Number
VNO		ZS930VNO	YS930VNO	CHAR	15	Vendor Number
TNAME		As per SAP standard field	NAME1			

## ALE Configuration steps for custom idoc

### outbound: -

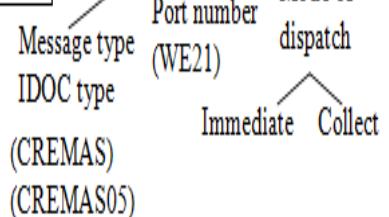
1. Create the segments (WE31).
2. Create the IDOC (WE30).
3. Create the Message type (WE81)
4. Link the message type to idoc type (WE82).
5. Create the port number (WE21).
6. Create the outbound partner profile (WE20)
7. Distribution model isn't required if we pass control record information in the program (Sender, Receiver, Message type - - -).

### ALE Configuration steps for standard IDOC Outbound

#### Create distribution mode (BD64)

Sender	Receiver	Message type
SP800	SP810	CREMAS

#### Create bound partner profile (WE20)



### Steps to create segment: -

Execute 'WE31'. Provide the segment name (Z1SGRS1). Click on create. Provide short description. Provide field names, data elements.

Save the segment. Repeat the steps for all the segments.

**Note:** - When ever we create the segment then automatically an equality structure is create in the data dictionary & also segment definition is create.

### Steps to create the IDOC: -

Execute 'WE30'. Provide the IDOC name (ZS930GRI). Click on create. Provide short description. Enter. Select the IDOC name. Click on create segment. Provide the segment name. If the segment is mandatory then select the check box. Provide minimum & maximum number of repetitions (1 & 99). Enter. Repeat the same steps for all the segments in the IDOC. Click on save.

### Steps to create Message type: -

Execute 'WE81'. Click on change mode. Enter. Click on new entries in the application tool bar. Provide message type (S930GRMAS), short description (GR ADDITIONAL INFORMATION MESSAGE TYPE). Save.

### Steps to link the message type to idoc type: -

Execute 'WE82'. Click on change mode. Click on new entries. Provide the message type (S930GRMAS), basic type (ZS930GRI) & release (700). Save.

### Steps to identify the release: -

Execute 'SE11'. Open the data base table 'EDIMSG'. Click on contents. Execute. Select the released field & click on sort descending order. Identify the latest released.

### Steps to create port number: -

Execute 'WE21'. Select the transactional RFC. Click on create. Select the radio button own port name. Provide the name (S930GRP). Enter. Provide short description. Provide RFC destination (SP810).

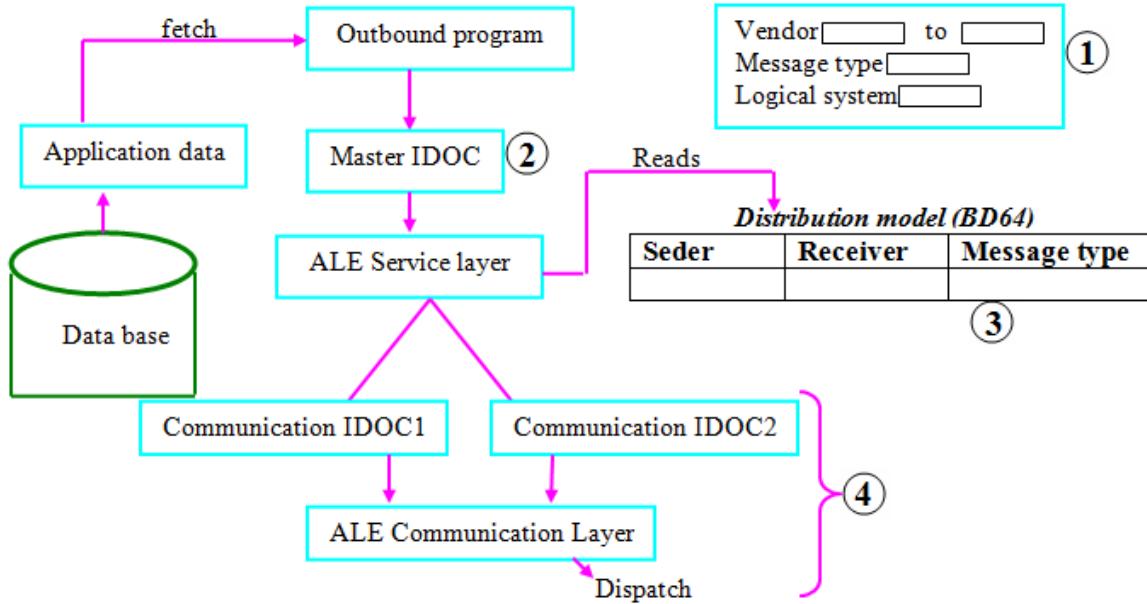
### Steps to create outbound partner profile: -

Execute 'WE20'. Select the partner if it's already exist. Click on create outbound parameter. Provide the message type (S930GRMAS), provide the port number (S930GRP). Select the radio button transfer immediate. Select basic type. Save.

### Steps to develop the custom Idoc outbound program: -

1. Design the selection-screen as shown in the below.
2. Generate the master IDOC (Based on the given input, we fetch the application data from data base and placed into internal table).
3. Collect the control record information (Sender, Receiver, Message type, Idoc type, - - ).
4. Generate as well as dispatch the communication Idoc.

Position	Fieldname	Data element
1	GRNo	
2	GRYEAR	
3	LRNO	ZS930LRNO
4	VNO	ZS930VNO
5	TNAME	NAME1



### Steps to identify the field name & data element of message type & logical system: -

Execute 'BD14' or any known transaction. Place the cursor on message type input field. Click on F1. Click on technical information. Identify the data element (EID\_MESTYP) & field name (MSGTYP). Place the cursor on target system input field. Click on F1. Click on technical information. Identify the data element (LOGSYS) & field.

#### Step 1: -

Tables ZS930GR.

Selection-screen begin of block A with frame.

Select-options S\_GRNO for ZS930GR-GRNO.

Parameter: P\_MESTYP type EDI\_MESTYPE obligatory,

P\_LOGSYS type LOGSYS.

Selection-screen end of block A.

GR Number [ ] to [ ]
Message type [ ] ✓
Logical system [ ]

#### Step 2: -

Generate the master IDOC is nothing but fill an internal table which contains two fields.

$\begin{array}{c} \text{Segment name} \\ \text{(SEGNAM)} \end{array}$        $\begin{array}{c} \text{Segment data} \\ \text{(SDATA)} \end{array}$

**Note:** - In the data dictionary we have one structure or a table I.e. EDIDD which contains the above fields. So we simply declare our internal table by referring EDIDD structure or table.

**Note:** - When ever we are working with custom idoc then we must declare 3 internal tables. One is for data. One is for control & one is for communication & also need to declare one work area & internal table for each segment in the idoc.

#### Procedure to fill master idoc: -

Based on the given input we'll fetch the data from data base & fill the each segment internal table. Loop the each segment internal data. Fill the master idoc.

Data: WA\_DATA like EDIDD,

IT\_DATA like table of WA\_DATA.  
 Data: WA\_SEG1 like Z1SGRS1,  
 IT\_SEG1 like table of WA\_SEG1.

Select GRNO GRYEAR LRNO VNO TNAME from ZS930GR into table IT\_SEG1 where GRNO in S\_GRNO.

Loop at IT\_SEG1 into WA\_SEG1.

WA\_DATA-SEGNAM = 'Z1SGRS1'.

WA\_DATA-SDATA = WA\_SEG1.

Append WA\_DATA to IT\_DATA.

Clear WA\_DATA.

Endloop.

GR Number 300000 to 300001

Message type

Logical system

GRNO	GRYEAR	LRNO	VNO	TNAME
300001	2014	LR98765	AP04 4304	SANDEEP

GRNO	GRYEAR	LRNO	VNO	TNAME
300001	2014	LR98765	AP04 4304	SANDEEP
300002	2014	LR12345	AP04 1234	KISHORE

SEGNAM	SDATA
Z1SGRS1	WA_SEG1
300002	2014

SEGNAM	SDATA
ZS1GRS1	WA_SEG1 300002 2014
ZS1SGRS2	WA_SEG2 300001 2014

### Step 3: -

Collect the control record information is nothing but fill an internal table which contains following fields.

- SNDPRT → Sender Partner Type
- SNDPRN → Sender Partner Number
- RCVPRT → Receiver Partner Type
- RCVPRN → Receiver Partner Number
- RCVPOR → Receiver Port Number
- DOCTYP → IDOC Number
- MESTYP → Message type

In the dictionary we have one structure or a table that is EDIDC which contains above fields. So we simply declare our internal table by referring EDIDC structure.

Data: WA\_CONT like EDIDC,

IT\_CONT like table of WA\_CONT.

- WA\_CONT-SNDPRT = 'LS'.
- WA\_CONT-SNDPRN = 'SP800'.
- WA\_CONT-RCVPRT = 'LS'.
- WA\_CONT-RCVPRN = 'SP810'.
- WA\_CONT-RCVPOR = 'S930GRP'.
- WA\_CONT-DOCTYP = 'ZS930GRI'.

WA\_CONT-MESTYP = ‘S930GRMAS’.

Append WA\_CONT to IT\_CONT.

Repeat the same steps for all the receivers.

#### **Step 4:-**

For each receiver in the control record internal table we need to generate as well as dispatch the communication idoc.

**Note:** - ‘MASTER\_IDOC\_DISTRIBUTE’ is the function module which is used to generate as well as dispatch the communication idoc. The input for the above function module is control record work area, data internal table. The output for the above function module is communication idoc internal table.

\*Declare communication idoc internal table

Data: WA\_COMM like EDIDC,

IT\_COMM like table of WA\_COMM.

\* For each receiver generate & dispatch communication idoc.

Loop at it\_cont into wa\_cont.

Call function ‘MASTER\_IDOC\_DISTRIBUTE’

EXPORTING

MASTER\_IDOC\_CONTROL = WA\_CONT

TABLES

COMMUNICATION\_IDOC\_CONTROL = IT\_COMM

MASTER\_IDOC\_DATA = IT\_DATA.

Commit work.

Endloop.

\* Release the idocs for queue.

Call function ‘DEQUEUE\_ALL’.

\*Display the communication idoc to know the status.

Call function ‘REUSE\_ALV\_GRID\_DISPLAY’

EXPORTING

I\_STRUCTURE\_NAME = ‘EDIDC’.

TABLE

T\_OUTTAB = IT\_COMM.

#### **Steps to create TCODE:-**

Execute ‘SE93’. Provide transaction code (ZS930GRS). Click on create. Provide short description. Select the radio button program and selection screen. Enter. Provide the program name (ZSJF\_930AM\_CUS\_IDOC\_OB). Select the GUI checkboxes. Save.

#### **ALE configuration steps for custom IDOC inbound: -**

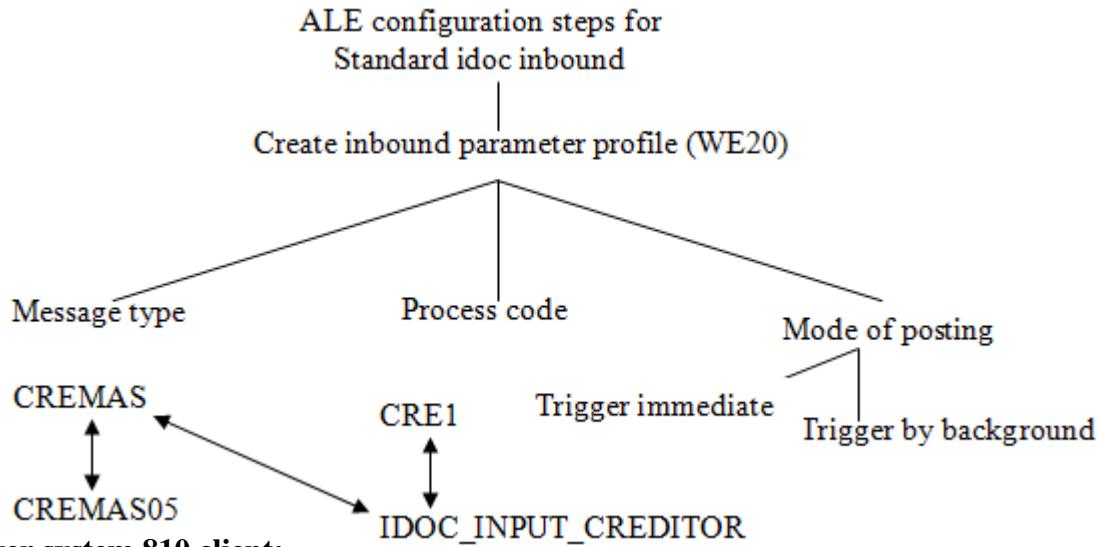
1. Create the segments (WE31).
2. Create the IDOC (WE30).
3. Create the message type (WE81).
4. Link the message type to idoc type (WE82).
5. Create the function module (SE37).

**Note:** - In the real time we never create our own function module. We always copy the existing function module because interface parameters are same for any message type (import, export, - - ).

6. Link the message type to function module & Idoc (WE57).
7. Create the mode of posting (BD51).
8. Create the process code (WE42).

9. Link the process code to function module (WE42).

10. Create inbound partner profile (WE20).



Against the process code it triggers the ZS930\_IDOC\_INPUT\_GRTOR function module & stores the sender sending information in idoc data internal table. Here our business logic is read the segments & their information from IDOC\_DATA internal table & posted to their respective data base tables.

**Function module:** ZS930\_DOC\_INPUT\_GRTOR.

Data: wa\_data like line of idoc\_data,

Wa\_reseg1 like z1sgrs1,

Wa like zs930gr,

Wa\_stat like line of idoc\_status.

Loop at idoc\_data into wa\_data.

If wa\_data-segnam = 'Z1SGRS1'.

Wa\_rseg1 = wa\_data-sdata.

Move-corresponding wa\_rseg1 to wa.

Insert zs930gr from wa.

If sy-subrc = 0.

Wa\_stat-docnum = wa\_data-docnum.

Wa\_stat-status = '53'.

Append wa\_stat to idoc\_status.

Else.

Wa\_stat-docnum = wa\_data-docnum.

Wa\_stat-status = '51'.

Append wa\_stat to idoc\_status.

Endif.

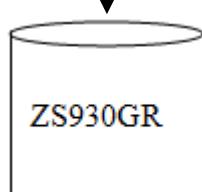
Endif.

Endloop.

SEGNAM	SDATA	DOCNUM		KISHORE
ZS1GRS1	WA_SEG1	300002   2014	LR12345   AP04 1234	
ZS1SGRS2	WA_SEG1	300001   2014	LR98765   AP04 4304	SANDEEP

SEGNAM	SDATA	DOCNUM		KISHORE
Z1SGRS1	WA_SEG1	300002   2014	LR12345   AP04 1234	

MANDT	GRNO	GRYEAR	LRNO	VNO	TNAME
	300001	2014	LR98765	AP04 4304	SANDEEP



### **Steps to test the inbound program by using inbound test tool: -**

Execute 'WE19'. Select the radio button via message type. Provide message type (S930GRMAS). Click on execute. Double click on the segment. Provide the sample data (300001, 2014, LR98765, AP04 4304, SANDEEP). Enter. Click on inbound function module in the application tool bar. Select the check box call in debugging mode. Select the radio button in four ground. Enter. In the menu bar click on debugging → classic debugger. Continuously click on F5 button, absorb the process flow.

### **In sender system 800 client**

### **Steps to send or generate IDOC through test transaction: -**

Execute WE19, select the Radio button via message type. Provide the message type. Execute. Double click on the segment provide sample data. (60000, 2015, LR2222, AP20222, PAVAN). Enter. In the menu bar click on edit control record. Provide receiver port (S930GRP), partner number (SP810), port type (LS), sender partner number (SP800), partner type (LS). Enter. In the menu bar click on IDOC → test outbound idoc. Enter.

**→ Configure the ALE to send as well as receives the employee training details as shown in the below.**

Fieldname	Key	Data element	Domain	Data type	Length	Short description
MANDT	✓	As per SAP standard field MANDT				
EID	✓	ZS930EID	YS930EID	CHAR	10	EMPLOYEE ID
EWM	✓	ZS930EWM	YS930EWM	CHAR	15	EMPLOYEE WORKING MODULE
ETM	✓	ZS930ETM	YS930ETM	CHAR	15	EMPLOYEE TRAINING MODULE
ENAME		As per SAP standard field NAME1				
ETD		ZS930ETD	YS930ETD	CHAR	10	EMPLOYEE TRAINING DATE

Segment name: Z1ETS1

Idoc name : ZETI

Message type : SETMAS

Port number : ETPORT

\* Design the selection screen

\* Tables ZS930ET.

Selection-screen begin of block A with frame.

Select-options S\_EID for ZS930ET-EID.

Parameter: P\_MESTYP type EDI\_MESTYP obligatory,  
P\_LOGSYS type LOGSYS.

Selection-screen end of block A.

Data: WA\_DATA like EDIDD,

IT\_DATA like table of WA\_DATA.

Data: WA\_SEG1 like Z1SGRS1,

IT\_SEG1 like table of WA\_SEG1.

Select EID EWM ETM ENAME ETD from ZS930ET into table IT\_SEG1 where EID in S\_EID.

Loop at IT\_SEG1 into WA\_SEG1.

WA\_DATA-SEGNAM = 'Z1ETS1'.

WA\_DATA-SDATA = WA\_SEG1.

Append WA\_DATA to IT\_DATA.

Clear WA\_DATA.

Endloop.

Data: WA\_CONT like EDIDC,

IT\_CONT like table of WA\_CONT.

WA\_CONT-SNDPRT = ‘LS’.  
 WA\_CONT-SNDPRN = ‘SP800’.  
 WA\_CONT-RCVPRT = ‘LS’.  
 WA\_CONT-RCVPRN = ‘SP810’.  
 WA\_CONT-RCVPOR = ‘ETPORT’.  
 WA\_CONT-DOCTYP = ‘ZETI’.  
 WA\_CONT-MESTYP = P\_MESTYPE.

Append WA\_CONT to IT\_CONT.

\*Declare communication idoc internal table

Data: WA\_COMM like EDIDC,  
IT\_COMM like table of WA\_COMM.

\* For each receiver generate & dispatch communication idoc.

Loop at IT\_CONT into WA\_CONT.

Call function ‘MASTER\_IDOC\_DISTRIBUTE’

**EXPORTING**

MASTER\_IDOC\_CONTROL = WA\_CONT

**TABLES**

COMMUNICATION\_IDOC\_CONTROL = IT\_COMM

MASTER\_IDOC\_DATA = IT\_DATA.

Commit work.

Endloop.

\* Release the idocs for queue.

Call function ‘DEQUEUE\_ALL’.

\*Display the communication idoc to now the status.

Call function ‘REUSE\_ALV\_GRID\_DISPLAY’

**EXPORTING**

I\_STRUCTURE\_NAME = ‘EDIDC’.

**TABLE**

T\_OUTTAB = IT\_COMM.

**In receiver system 810 client: -**

Function Module → ZS930\_IDOC\_INPUT\_ETTOR.

Process code → ET1.

**Process flow: -**

After IDOC reached to receiver system then it goes inbound partner profile WE20 & check the message type ‘SETMAS’ is available or not. It’s available. So it goes to process code ET1. Against this process code it triggers the function module. In the function module tables tab we have IDOC\_DATA internal table is available in that the sender data is stored.

**Stored login: -**

We read the data from IDOC\_DATA internal table & posted the data into their data base tables.

Data: WA\_DATA like line of IDOC\_DATA,

WA\_RSEG1 like Z1ETS1,

WA like ZS930ET,

WA\_STAT like line of IDOC\_STATUS.

Loop at IDOC\_DATA into WA\_DATA.

If WA\_DATA-SEGNAM = ‘Z1ET1’.

WA\_RSEG1 = WA\_DATA-SDATA.

Move-corresponding WA\_RSEG1 to WA.

Insert ZS930ET from WA.

If SY-SUBRC = 0.  
 WA\_STAT-DOCNUM = WA\_DATA-DOCNUM.  
 WA\_STAT-STATUS = '53'.  
 Append WA\_STAT to IDOC\_STATUS.  
 Else.  
 WA\_STAT-DOCNUM = WA\_DATA-DOCNUM.  
 WA\_STAT-STATUS = '51'.  
 Append WA\_STAT to IDOC\_STATUS.  
 Endif.

SEGNAM	SDATA	DOCNUM	WM	PRADEEP	15.01.2015
Z1ES1	WA_SEG1 SP001	MM			
Z1ES1	WA_SEG1 SP002	SD	CRM	PAVAN	10.05.2014

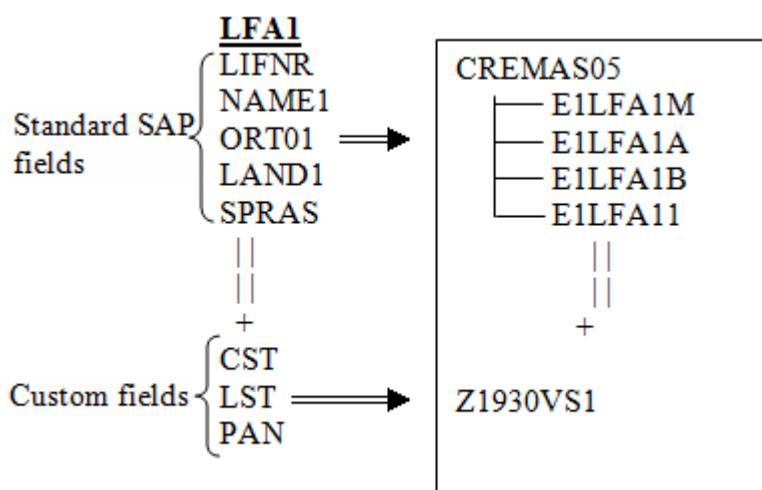
SEGNAM	SDATA	DOCNUM	WM	PRADEEP	15.01.2015
Z1ES1	WA_SEG1 SP001	MM	WM	PRADEEP	15.01.2015

EID	EUM	ETM	ENAME	ETD
SP001	MM	WM	PRADEEP	15.01.2015

MANDT	EID	EUM	ETM	ENAME	ETD
	SP001	MM	WM	PRADEEP	15.01.2015

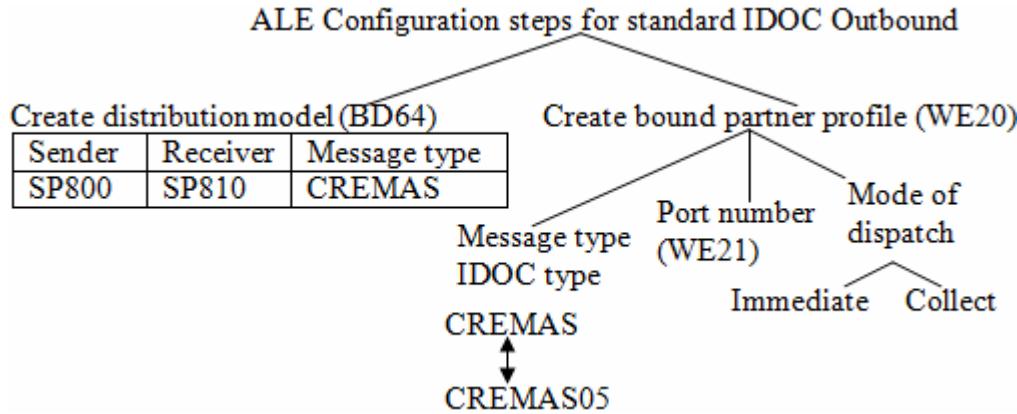
#### Extension idoc: -

It's the collection of standard idoc + custom segment. As per client requirement if we add the CST number (Central Sales Tax), LST Number (Local Sales Tax), PAN Number to the LFA1table through append structure. If you want to send as well as receives the custom fields information along with standard field information then we go for extension IDOC.



### **ALE Configuration steps for extension IDOC outbound: -**

1. Create the additional segments (WE31)
2. Create the extension IDOC (WE30)
3. Link the message type to extension IDOC (WE82)
4. Create the port number (WE21)
5. Create the outbound partner profile (WE20)
6. Create the distribution model (BD64)



Before configure the ALE the following fields are added to LFA1 table through append structure.

<b>Field</b>	<b>Data element</b>	<b>Data type</b>	<b>Length</b>
CSTNO	ZSPCSTNO	CHAR	10
LSTNO	ZSPLSTNO	CHAR	10
PANNO	ZSPPANNO	CHAR	10

#### **Steps to create additional segments: -**

Execute ‘WE31’. Provide the segment name (Z1930VS1). Click on create. Provide short description. Provide the field names as well as data elements.

- 1 CSTNO ZSPCSTNO
- 2 LSTNO ZSPLSTNO
- 3 PANNO ZSPPANNO

Save, repeat the same steps for all segments.

#### **Steps to create extension idoc: -**

Execute ‘WE30’. Provide the extension idoc name (Z930EI). Select the radio button extension. Click on create. Provide the linked basic type (CREMAS05). Provide short description (Vendor extension idoc). Enter. Select the any one of the segment (E1LFA1M) (Reference segment), click on create segment. Enter. Provide the segment type (Z1930VS1). Provide minimum (1), maximum (99). Enter. Save.

#### **Steps to link the message type to extension idoc: -**

Execute WE82. Click on change mode. Enter. Click on new entries in the application tool bar. Provide the message type (CREMAS), basic type (CREMAS05), extension (Z930EI), release (700). Click on save. Create the port number.

#### **Steps to create outbound partner profile:-**

Execute WE20. Select the partner if the message type is already available then select message type. Click on detail. Provide the extension. Click on save.

If the message type isn’t available then click on create outbound parameter. Provide the message type (CREMAS), port number (A000075). Select the radio button transfer idoc immediate. Provide the basic type (CREMAS05), extension (Z930EI). Save.

Create distribution model.

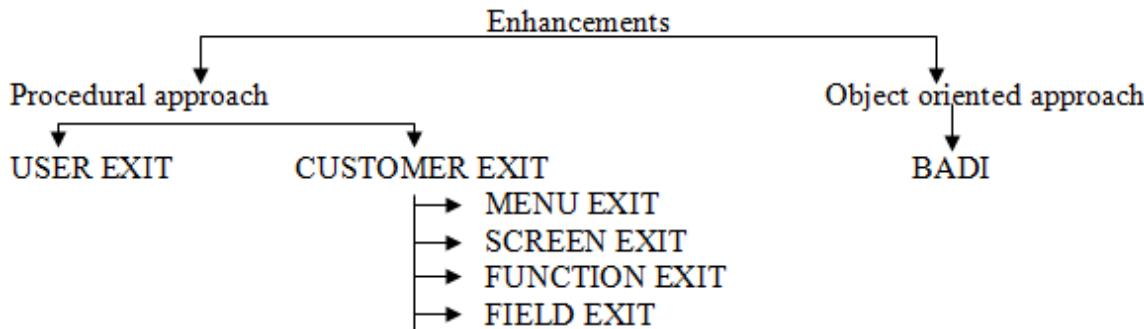
**Note:** – Developing a extension idoc outbound program is nothing but fill the additional segment information only. The standard segments are filled by standard program.

Here we add the filling of additional segments information to the standard program BD14.

Adding some additional functionality to the standard functionality without disturbing the standard functionality is always through enhancements.

### **Enhancements**

Enhancement is nothing but adding some additional functionality to the standard functionality without disturbing the standard functionality.



**USER EXIT:** - User exits are used to adding some additional functionality to the standard functionality is always through sub routines (form, end form).

These are implemented in the form of subroutines and hence are also known as FORM EXITS. The user exits are generally collected in includes and attached to the standard program by the SAP. User Exits are developed for the SD module. User-exits are empty subroutines that SAP Developers have provided for you. You can fill them with your own source code. Technically this is a modification.

All User exits start with the word USEREXIT\_...

```
FORM USEREXIT_XXXX....  
ENDFORM.
```

### **Some of the scenarios in user exit**

#### **Scenario 1: -**

In real time if we maintain the key information of the sales order in a separate 'Z' table at the time of creating sales order. At the time of cancel the sales order then we must remove the sales order key information from the 'Z' table by using user exit.

Program Name: MV45AFZZ

User Exit: USEREXIT\_DELETE\_DOCUMENT

#### **Scenario 2: -**

We can create Sales Order through VA01. That user only can be modified through VA02 by using user exit.

User Exit: USEREXIT\_CHECK\_VBAK

Program: MV45AFZB

#### **Scenario 3: -**

In the real time SD functional consultant provide the number ranges for sales order, delivery & billing document. Based on the sales organization & company we split the number ranges into smaller ranges by using user exit.

Program Name: MV45AFZZ

User Exit: USEREXIT\_NUMBER\_RANGE

#### **Scenario 4: -**

As per client requirement if we add some additional fields to standard SD tables if you want to assign these fields values at the time of creating the sales documents then we use user exit.

Program Name: MV45AFZZ

User Exit: USEREXIT\_MOVE\_FIELD\_TO\_VBAK

#### **Scenario 5: -**

In the real time at the time of creating the sales order some times the end user select the same tax conditions more than once. To avoid this by using user exit.

Program Name: MV45AFZZ

User exit Name: USEREXIT\_SAVE\_DOCUMENT\_PREPARE

#### **Steps to identify the user exit: -**

Execute SE93 transaction. Provide the transaction code. Click on display. Identify the package of the transaction & copy that package. Execute SMOD transaction. Place the cursor on Enhancement. Click on F4 button. Click on Information System button. Provide the package of the transaction. Press enter. You'll get list of User Exits with short description.

**→ Delete the sales order key information from the ‘Z’ table at the time of cancel the sales document by using user exit.**

#### **Steps to implement the user exit: -**

Execute SE38. Provide the program name MV45AZZ. Click on display. Click on find function key. Provide the user exit name (USEREXIT\_DELETE\_DOCUMENT). Enter. Double click on form or user exit. identify the place where we implement the logic. Click on enhance ( ) symbol in the application tool bar. In the menu bar click on edit → enhancement operations → show implicit enhancement options. It'll provide yellow line for all the forms. Select our user exit yellow line. Right click → enhancement implementation → create. Click on code. Click on create enhancement implementation. Provide the implementation name (ZS10AM\_UEI), short description (Delete sales document). Enter. Save in our own package or local object. Select the implementation name. enter. Provide the logic.

Delete from ZS10AM\_SKI where VBELN = VBAK-VBELN. Save, check, activate.

**→ Split the sales order number ranges into smaller ranger based on the sales organization.**

When ever we are working with number ranges then we must create one ‘Z’ table which contains based on which fields we want to split the number ranges and from value, to value and current value.

Table: ZS10NR

MANDT	VKORG	BUKRS_VF	FVAL	TVAL	CVAL
800	0001	1000	1000000	1999999	1000000
800	0001	2000	2000000	2999999	2000000
800	0005	1000	3000000	3999999	3000000
800	0005	2000	4000000	4999999	4000000

USEREXIT\_NUMBER\_RANGE: - Data WA like ZS10NR.  
Select single \* from ZS10NR into WA where VKORG = VBAK-VKORG and BUKRS\_VF = VBAK-BUKRS\_VF.

#### **USEREXIT\_SAVE\_DOCUMENT: -**

Update ZS10NR set CVAL = VBAK-VBELN where VKORG = VBAK-VKORG and BUKRS\_VF = VBAK-BUKRS\_VF.

**CUSTOMER EXIT: -** It's used to add some additional functionality to the standard functionality is always through function modules. These are one type of enhancements that are available in some specific programs, screens and menus within standard SAP Applications. These are Function Modules with a custom empty include program, you can add your own functionality in these include programs.

Customer Exits are available in all SAP modules, where as user exits are only available in SD module.

All customer exits ( Function Modules) starts with CALL CUSTOMER word.

**Note:** - Customer exit is either menu exit or screen exit or function exit or field exit.

### **Advantages:**

- They do not affect standard SAP source code
- They do not affect software updates

### **Disadvantage:**

- Customer exits are not available for all programs and screens found in the SAP System. You can only use customer exits if they already exist in the SAP System.

**MENU EXIT:** - It's used to adding some additional menus to the standard program.

**SCREEN EXIT:** - It's used to adding some additional sub screens to the standard program.

### **Working with Screen exit:-**

Screen exit is used to adding some additional sub screens to the standard transaction code. Screen exit isn't possible for all the transaction codes. Some of the transaction codes which contains screen exit.

**VX11:-** Create financial document.

**CO01:-** Create production document.

**CJ01:-** Create work break down structure.

**Note:** - When ever we are working with screen exit first we add the screen fields to the standard data base table through append structure or create the 'Z' table with those fields. Based on the table fields we design the screen (screen number is provided by SAP people).

### **Steps to work with screen exit: -**

Identify the package of the transaction. Based on the package we identify the customer exit. Identify the right customer exit which contains screen exit. Implement the customer exit through CMOD transaction.

#### **→ Implement the screen exit for the VX11 transaction**

##### **Steps to identify the package of the transaction:-**

Execute SE93. Provide the transaction code VX11. Click on display. Identify the package.

##### **Steps to identify the customer exit based on the package:-**

Execute SMOD transaction. Click on one find function key. Provide the package. Execute. Identify the all customer exits.

##### **Steps to identify the right customer exit which contains screen exit: -**

Double click on each customer exit. Identify the right customer exit which contains screen exit. (RVEEXAKK1).

##### **Steps to implement the screen exit: -**

Execute 'CMOD'. Provide the project name. Click on create. Provide short description. Click on save. Click on enhancement assignments in the application tool bar. Provide the customer exit name. Enter. Click on save. Click on components. Click on back to initial screen. Select the radio button components. Double click on the first screen exit. Enter. Provide short description. Click on sub screen radio button. Click on save. Click on layout. Design the screen from table fields (AKKP). Save, check, activate the screen. Click on back. Activate the components. Back. Activate the project.

##### **Steps to check the screen exit: -**

Execute VX11. Provide the input. Enter.

**FUNCTION EXIT:** - Function exit play a major role in the real time. Because when ever we are working with menu exit & screen exit & their functionality is implemented through function exit.

**FIELD EXIT:** - It's used to perform the additional validations on the field. Now a days field exits are outdated.

In this extension idoc we identify the right function exit to add the filling of additional segments logic to the standard program 'BD14'. If you want to identify the function exit, first we need to identify the customer exit. Because function exit is key part of customer exit.

### **Scenario for customer exit**

Validate vendor master application for postal code. Some times users will not enter postal code, city, tax code etc in the application which are optional but important for clients. Provide validation such that without entering postal code the transaction shouldn't be created.

**Note:** - Customer exit always identified through package of the transaction code.

### **Steps to identify the package of transaction:**

Execute SE93. Provide the transaction code ('BD14') for which transaction function we need to identify the package. Click on display. Identify the package (CGV).

### **Steps to identify the customer exit based on the package:**

Execute 'SMOD'. Click on find function key. Provide the package (CGV). Click on execute. Identify the customer exits (VSV00002).  
VSV00002 means VSV00001 as well as VSV00002.

### **Steps to identify the function exit based on the customer exit:**

Execute 'SMOD'. Provide the enhancement as customer exit name. Click on display. Click on components in the application tool bar. Read the short description of each & every function exit & identify the right function exit.

### **Steps to implement the customer exit:**

Execute CMOD. Provide the project name (ZS10AMME) [any name]. Click on create. Provide short description. Save. Click on enhancement assignment. Provide the enhancement name as customer exit name (RVEXAKK2). Enter. Click on save. Click on components. Back to initial screen. Select the radio button components. Click on change. Double click on each menu exit. Provide the short description (Go to SE16). Second button is Go to SE11. Enter. Double click on function exit. Double click on include. Implement the logic.

If SUCOMM = '+CS1'.

CALL TRANSACTION 'SE16'.

ELSEIF SUCOMM = '+CS2'.

CALL TRANSACTION 'SE11'.

ENDIF.

Save, check, activate the include. Back, activate the function module. Back. Activate the components. Back. Activate the project.

**Note:** - Some times we can't identify the right function exit based on the short description. So we always identify the right function exit through break points.

**Note:** - Outbound exit will be triggered after filling of each & every standard segment.

**Note:** - Customer exit is always implemented through project i.e. 'CMOD' transaction.

### **Steps to identify the right function exit based on the break point: -**

Execute ‘CMOD’. Provide the project name (Z930VE). Click on create. Provide short description. Click on save. Click on enhancement assignments in the application tool bar. Provide the enhancement as customer exit names. Enter. Save. Click on components in the application tool bar. Double click on each function exit. Place the cursor on include. Click on set / delete session break point. Activate. Back. Click on change mode. Click on activate. Click on back. Activate the project. Now execute ‘BD14’. Provide the input (S9090), message type, target system. Execute. Based on the given input outbound program will be triggered & fetch the application data from database & fill the first standard segment. After filling the each & every standard segment it goes to the right place then the cursor will stop at right place due to break point & identify the right exit.

FUNCTION EXIT\_SAPLKD01\_001.

### **Syntax rules of an IDOC: -**

1. The data for the segment must exist, if it specified as mandatory.
2. We shouldn't exceed maximum number of repetitions for the segment.
3. The data for the segments must exist in the same physical sequence of the segments in the idoc.
4. The data for the child segment can't exist without having the data in parent segment.

Project: Z930VE

FUNCTION EXIT: EXIT\_SAPLKD01\_001

Data: wa\_data like line of IDOC\_DATA,

Wa\_seg1 like Z1930VS1,

It\_seg1 like table of wa\_seg1,

Wa like E1LFA1M.

If segment-name = ‘E1LFA1M’.

Read table idoc\_data into wa\_data index 1.

Wa = wa\_data-sdata.

Select CSTNO LSTNO PANNO from LFA1 into table it\_seg1 where lifnr = wa-lifnr.

Loop at it\_seg1 into wa\_seg1.

Wa\_data-segnam = ‘Z1930VS1’.

Wa\_data-sdata = wa\_seg1.

Append wa\_data to idoc\_data.

Endloop.

Endif.

IDOC\_CIMTYPE = ‘Z930EI’.

WA	LIFNR	NAME1	ORT01	SORTL	LAND1	---
	S9090	COAL IN		COAL	DE	

*IDOC\_DATA*

SEGNAM	SDATA	DOCNUM
E1LFA1M	S9090 C123 L123 P123	
Z1930VS1	C123 L123 P123	
E1LFA1A		

*WA DATA*

SEGNAM	SDATA	DOCNUM
Z1930VS1	C123 L123 P123	

*IT\_SEG1*

CSTNO	LSTNO	PANNO
C123	L123	P123

*WA\_SEG1*

CSTNO	LSTNO	PANNO
C123	L123	P123

### **Steps to send the vendor: -**

Execute BD14. Provide vendor number (S9090), message type (CREMAS), target system (SP810). Execute. Now it sends the standard segment information & custom segment information. Go to WE05/02 to absorb the all the segments.

### **Extension idoc inbound: -**

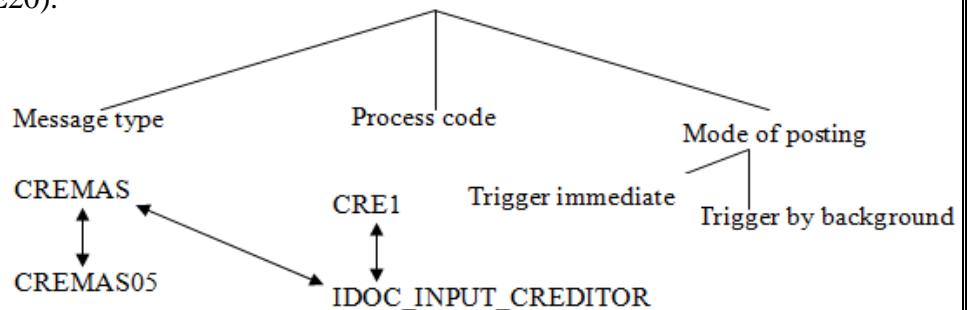
Writing an extension idoc inbound is nothing but read & post the additional segment information only. The standard segments are read & posted by standard function module.

## ALE configuration steps for extension IDOC inbound: -

1. Create the additional segment (WE31).
2. Create the extension idoc (WE30)
3. Link the message type to extension idoc (WE82)
5. Create inbound partner profile (WE20).
6. Link the message type to FM.

### ALE configuration steps for Standard idoc inbound

Create inbound parameter profile (WE20)



## In the receiver system

### Steps to link the message type to function module and extension idoc: -

Execute 'WE57'. Click on change mode. Enter. Click on new entries in the application tool bar. Provide function module name IDOC\_INPUT\_CREDITOR. Select the function type as function module. Provide basic type as CREMAS05, extension as Z930EI, message type as CREMAS. Select the direction is inbound. Save.

### Create inbound partner profile: -

**Note:** - Inbound exit will be triggered after it reaches the each & every custom segment. Outbound exit is '1'. Inbound exit is '2'.

Here inbound exit is EXIT\_SAPLKD02\_001

### Procedure: -

After IDOC reached to receiver system then it goes to inbound partner profile WE20. & check the message type CREMAS is available or not. CREMAS message type is available. So it goes to process code CRE1. Against the process code, it identifies or triggers the inbound function module IDOC\_INPUT\_CREDITOR. This function module collects the first segment data E1LFA1M & Posted to data base. After it goes to 2<sup>nd</sup> segment Z1930VS1. This is the custom segment. So it goes to inbound ext. in this exit we develop the logic of custom segments data reading & posted to data base.

Project: Z930VE

EXIT NAME: EXIT\_SAPLKD02\_001.

Data: wa\_data like line of idoc\_data,

Wa\_rseg1 like z1930vs1,

Rwa like e1lfa1m,

Wa\_stat like line of idoc\_status.

Loop at idoc\_data into wa\_data.

If wa\_data-segnam = 'E1LFA1M'.

Rwa = wa\_data-sdata.

Elseif wa\_data-segnam = 'E1LFA1M'.

Wa\_rseg1 = wa\_data-sdata.

Update lfa1 set CSTNO = wa\_rseg1-cstno

LSTNO = wa\_rseg1-lstno

PANNO = wa\_rseg1-panno

Where LIFNR = rwa-lifnr.

If sy-ucomm = 0.

Wa\_stat-docnum = wa\_data-docnum.

Wa\_stat-status= '53'.

Append wa\_stat to idoc\_status.

SEGNAM	SDATA	DOCNUM
E1LFA1M	S9090 C123 L123 P123	
Z1930VS1	C123 L123 P123	
E1LFA1A		

SEGNAM	SDATA	DOCNUM
Z1930VS1	C123 L123 P123	

CSTNO	LSTNO	PANNO
C123	L123	P123

Else.  
 Wa\_stat-docnum = wa\_data-docnum.  
 Wa\_stat-status= '51'.  
 Append wa\_stat to idoc\_status.  
 Endif.  
 Endif.  
 Endloop.

RWA	LIFNR	NAME1	ORT01	SORTL	LAND1	---
	S9090	COAL IN		COAL	DE	

### **Serialization:** -

Serialization is used to send as well as receive the collected message type information in a sequence.

#### **EX:** -

Material contains classification. I.e. first you need to send material & then classification.

If you send classification first then it waits until material is sent.

→ **Create the serialization to send the material (MATMAS) & CLFMAS information.**

**Note:** - CLFMAS → Material classification

### **ALE configuration steps for serialization outbound:** -

1. Create the serialization group (SALE)

Sender	Receiver	Message type
SP800	SP810	MATMAS
SP800	SP810	CLFMAS
SP800	SP810	SERDAT

2. Create the distribution model with related message types and SERDAT message type.
3. Create the outbound partner profile with related message types & SERDAT

message type.

#### **Steps to create serialization group:** -

Execute SALE. Expand modeling & implementing business process. Expand master data distribution. Expand serialization for sending & receiving data. Expand serialization using message type. Expand define serialization groups. Click on new entries in the application tool bar. Provide the serialization group (S930SG), short description. Click on save, (warning). Enter. Select the serialization group. Double click on assignment of logical message to serial group. Click on new entries. Provide the message type sequence number

MATMAS	1
CLFMAS	2

Click on save. Enter.

#### **Create the distribution model with MATMAS CLFMAS and SERDAT:** -

Execute BD64. Change, create model view.

Model view	DBMS
Sender	SP800
Receiver	SP810
Message	MATMAS

Click on filter model display.

SP800
SP810
MATMAS

Select that one delete.

Similar for CLFMAS, SERDAT.

#### **Create the outbound partner profile with MATMAS CLFMAS SERDAT message type:** -

Execute WE20 → LS → SP810 → Create outbound

CLFMAS	PORNUMBER
BASIC type	CLFMAS02

→ SERDAT, Port number, Basic type.

### **SDMO (for identifying the transaction based on short description)**

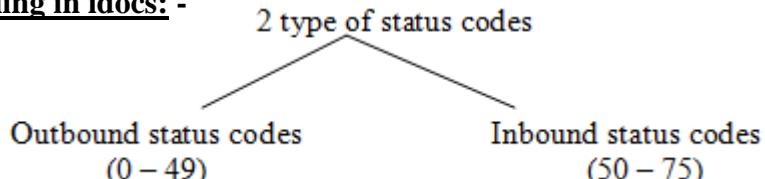
BD93 → send → classification.

**Note:** - Whenever we send the material details through BD10 & classification details through BD93 then the system check the serialization sequence & sent to the receiver system in same server.

### **ALE configuration steps for serialization inbound:** -

1. Create the serialization group (SALE)
2. Create the inbound partner profile with related message types and SERDAT message type (WE20).

### **Working with status codes / error handling in idocs:** -



**Note:** - WE47 is the transaction code which contains all the status codes & their short description.

#### **Outbound status codes:** -

Status code 03 (Data passed to port ok). After we get the status 03, we execute RBDMOIND standard program. If the status code 03 turns 12 then the idoc successfully reached to destination. If the status code 03 remains 03 same, then the idoc is in the TRFC (TRFC) port. If the status code 03 turned to 11 then the idoc is damaged in the TRFC port.

#### **Status code 30 (Idoc ready for dispatch):** -

If the status code remains 30 then must check mode of dispatch in the outbound partner profile (WE20). If the mode of dispatch is collect then we must execute RSEOUT00 standard program. Then only the collect idoc will be departed to their relevant receiver systems. If the mode of dispatch is immediate that is due to traffic. It'll reach with in 5 minutes.

#### **Status code 29 (error in ALE server):** -

→ An entry in the outbound table is missed (ABAPer job)

    Outbound partner profile isn't available

→ Data miss matched (functional people job)

    Ex: The given company 1000 isn't available

→ Synchronization and configuration problem (BASIS people job)

#### **Status code 26 (error during syntax check the idoc):** -

When ever we aren't follow the syntax rules, then we get the syntax error i.e. 26.

#### **Status code 42 (idoc was created by test transaction):** -

When ever we generate the idoc through test transaction WE19 then we get the status code 42.

#### **Status code 01 (idoc generated):** -

When ever the communication idoc is generated then we get the status code '01'.

**Note:** - Status code 00 only used in R/2 system not in R/3 system.

#### **Status code 64 (idoc ready to be transferred to application):** -

If the status code remains 64 then must check the mode of posting in the inbound partner profile (WE20). If the mode of posting is trigger by background or collects then we must execute the RBDAPP01 standard program then only the collect IDOCS will be posted to their relevant data base tables. If the mode of posting is trigger immediate that's due to traffic.

#### **Status code 51 (Application document not posted):** -

→ Data mismatched (functional people)

    Ex:- The given country 'IN' is not defined

→ Synchronization & Configuration problem (BASIS people job)

### **Status code 56 (idoc with error added):-**

- An ALE in the inbound table isn't found (ABAPer job)
  - (Inbound partner profile is not available)

### **Status code 53 (Application document posted):-**

When ever the data is posted from the application to their relevant data base table successfully then we get the status code 53.

### **Status code 50 (idoc added): -**

When ever the idoc reached to receiver system then we get the status code 50.

### **Status code 62(Idoc passed to application):-**

After idoc reached to receiver system then the inbound function module collect the data from idoc & transferred to particular application if it's success then we get the status code 62.

### **Status code 74 (Idoc was created by test transaction): -**

When ever we test the inbound program through inbound test tool WE19 then we get the status code 74 instead of 50.

### **Archiving idoc: -**

Archiving idocs are used to move or transverse the idocs information from SAP system to temporary file in the presentation server. This is used to improve the performance of the system. Before archiving the idocs the BASIS people create the physical file path in the presentation server & logical file path in the SAP system & link the logical file path to physical file path through file transaction.

**Note:** - We can't archive the error idocs which status code is 29 & 51 directly. First we need to convert the error status code through some other status codes by using RC1\_IDOC\_SET\_STATUS standard group. Later we archive the idoc.

### **Steps to convert the idoc status code: -**

Execute SE38. Provide the program name. RC1\_IDOC\_SET\_STATUS. Click on execute. Provide the idoc number. Old status code (51). New status code (68). Remove the check box test. Execute.

29 into 31

51 into 68

### **Steps to archive the idocs:-**

Execute SARA. Provide the archive idoc is provide is 'IDOC'. Click on enter. Click on write. Click on maintain. Provide the variant name (ZSV1). Click on create. Select the radio button for all selection view. Enter. Provide the inputs as per client requirement. Click on attributes. Provide the short description. Save. Click on back. Click on start data. Click on immediate or provide date & time save. Here the idocs are transferred or archived into physical file in the presentation server which is created by BASIS people.

**Note:** - 'RSEXARCA' is the standard program which is used to archive the IDOCS.

**Note:** - 'RSEXARCD' is the standard program which is used delete the archive idoc.

**Note:** - 'RSEXARCL' is the standard program which is used reload the archive idocs.

**Note:** - 'WE11' is the transaction code which is used to delete the idocs.

### **Steps to download the idoc information: -**

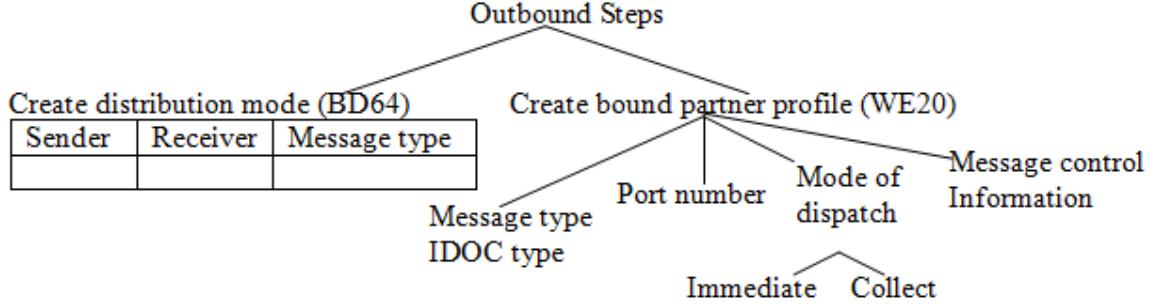
Execute 'IDOC' transaction. Select the radio button analyse idoc field values. Execute (F8). Provide the idoc number (798746). Select the checkbox 'Also output empty fields'. Execute. In the menu bar click on system → list → save → local file. Select the radio button spread sheet. Enter. Browse the file. Save. Generate.

### **Working with transactional data: -**

By using message control technique we can send the transactional data. When ever the functional people or end user create & save the transactional data then it automatically generate as well as dispatch the idoc.

### **ALE configuration steps for transactional data: -**

1. Provide the output medium as ALE for the application
- 2.



→ **Configure the ALE to send as well as receives the purchase order information**

#### **Steps to provide output medium as ALE for purchase order application: -**

Execute 'NACE'. Select the purchase order application (EF). Click on output types. Select the output type which is provided by functional people. Double click on processing routines in the left panel & absorb the medium. If the ALE medium isn't available, click on change mode, select the EDI medium. Click on copy in the application toolbar. Select the transaction medium as 'Distribution (ALE)' instead of EDI. Click on enter.

Create the distribution model.

Sender	Receiver	Message type
SP800	SP810	ORDERS

#### **Steps to create outbound partner profile with message control information:-**

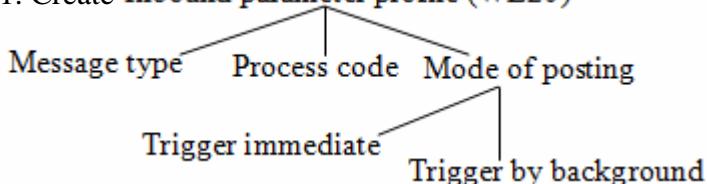
Execute WE20. Select the partner. Click on create outbound parameter. Provide the message type (ORDERS), port number (A000075). Select the radio button 'Transfer IDOC immediate', select the basic type (ORDERS05). Click on message control tab, click on insert row (+) button. Select the application (EF), select the message type (NEU). Select the process code (ME10).

If you want to send the changes also then click on insert row (+) button once again. Provide the application, same message type, same process code. Select the check box change message. Click on save. After completion of configuration the function people or end user perform the following activities.

Execute ME21N. Create the purchase order. Click on messages in the application tool bar. Provide the output type 'NEU', medium as 'ALE', partner type as 'LS' & provide partner number. Click on further data in the application tool bar. Enter & select the dispatch time is send immediately (when saving the application). Click on back. Save. Then automatically one idoc is generated & dispatched to receiver system.

#### **ALE configuration steps for transactional data for inbound: -**

1. Create Inbound parameter profile (WE20)



#### **Steps to create inbound partner profile in receiver system: -**

Execute WE20. Select the partner. Click on create inbound parameter. Select the message type 'ORDERSP'. Select the process code (ORDR). Click on save.

**Note:** - If you want to send the purchase order details to vendor then we must create outbound partner profile under partner type LI (in WE20). If you want to send the sales order details to the customer then you must create outbound partner profile under partner type KU. If you want to send the payment details to the bank then we must create outbound partner profile under partner type B.

# OOABAP

## Different types of programming structures

1. Unstructured Programming
2. Procedural Programming
3. Object Oriented Programming

### **1. Unstructured programming:** -

- The entire program contains only one mail.
- The same set of statements is placed in multiple locations of the same program.
- It's very difficult to maintain if the program becomes very large.

### **2. Procedural programming:** -

- The entire program is splitted into smaller programs.
- The same set of statements is placed in a procedure (Subroutines or Function Module) & later we call the same procedure from different locations of the same program.
- All the subroutines & function modules can access the global declarations.
- It take little bit extra time to enhance the existing functionality.

### **3. Object Oriented Programming:** -

- The entire program is visualized in terms of class & objects
- All the methods can't access the global declarations.
- It takes very less time to enhance the existing functionality.

### **Key features of object oriented programming:** -

1. Better programming structure
2. Most stress on data security & access
3. Reduce the redundancy of code
4. Data abstraction & encapsulation.
5. Inheritance and polymorphism

### **Class and Object:** -

Class is the blueprint or template of an object. Object is the real one.

**EX:** - If you want to build a form house then we take a plan from engineer to build the form house. It's nothing but class. Based on the plan constructed house is an object.

**Note:** - Based on one class we can create any number of objects.

There are two types of classes.

1. Local Class
2. Global class

Differences between Local & Global classes

#### Local

1. Local class name starts with any letter
2. It's created through SE38 transaction.
3. We can access the local class with in the program only.
4. Local class is stored in the memory of ABAP program.

#### Global

- 1) Global class name must start with 'Y' or 'Z'.
- 2) It's created through SE24 transaction.
- 3) We can access the global class from any where in the SAP.
- 4) Global class is stored in the class repository.

A class contains two sections.

1. Class Definition
2. Class implementation

### **Class definition:** -

Class definition is nothing but declaring the all the components of the class & any one of the visibility section.

### **Components of a class:** -

1. Attributes
2. Methods
3. Events
4. Interfaces

**Attributes:** - Attributes are used to declare the variables, work areas, internal tables which are needed to implement the logics.

**Methods:** - Method is the collection of statements which perform the particular activity. Methods are coding blocks of a class, which can provide some business functionality

**Events:** - Event is an action which is performed at run time. Events are used to provide the dynamic features at run time. Events are used to handle the methods of some other class.

**Interface:** - Interface is the collection of methods which are defined & not implemented.

There are three types of visibility sections.

1. Public Section
2. Protected Section
3. Private Section

**1. Public Section:** - We can access the public components within the class as well as outside the class.

**2. Protected section:** - We can access the protected components within the class as well as derived or child class.

**3. Private section:** - We can access the private components within the class only.

**Note:** - In ABAP we haven't default visibility section.

### **Syntax of class definition:** -

Class <Class name> definition.

-----  
----- } components of class  
-----

Endclass.

### **Class Implementation:** -

Class implementation is nothing but implementing the methods which are defined in the class definition.

### **Syntax of class implementation:** -

Class <class name> implementation.

Method <method1>

-----  
-----  
-----

Endmethod.

Method <method1>

-----  
-----  
-----

Endmethod.

||  
||

Endclass.

**Note:** - We can access the components of the class is always through class object.

### **Syntax of creating the object for the class: -**

This is two step procedure.

1. Create the reference to the class
2. Create the object based on reference

### **Syntax of creating the reference to the class:-**

Data <reference name> type ref to <class name>

### **Syntax of creating the object based on reference: -**

Create object <reference name>.

**Note:** - Class object is always created under start-of-selection event only.

### **Syntax of declaring the method: -**

Methods <method name> importing <IV1> type <DT>

<IV2> type <DT>

||

Exporting <EV1> type <DT>

<EV2> type <DT>

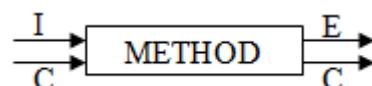
||

||

Changing <CV1> type <DT>

<CV2> type <DT>

||



### **Syntax of implementing the method: -**

If the method is declared in class

### **Syntax**

Method <method name>.

```
-----
----- logic
-----
Endmethod.
```

If the method is declared in interface.

### **Syntax**

Method <interface>~<method>.

```
-----
----- logic
Endmethod.
```

### **Syntax of calling the method: -**

Call method <object name of the class> -> <method name> exporting

<IV1> = <Value1>

<IV2> = <Value2>

Importing

<EV1> = <variable1>

<EV2> = <Variable2>

||

||

<object name of the class> -> <method name> (Exporting

-----

-----

Importing

-----

→ Perform the addition of two numbers by using OOABAP.

Parameter: P1 type I,  
P2 type I.

Class C1 definition.

Public section.

Data: A type I,

B type I,

C type I.

Methods: Add1 importing M type I N type I,  
Display.

Enclass.

Class C1 implementation.

Method Add1.

A = M.

B = N.

=====

C = A + B.

Endmethod.

Method display.

Write C.

Endmethod.

Endclass.

→ Based on the given company code, display the comp code, comp name, city by using OOABAP.

PARAMETER P\_BUKRS TYPE T001-BUKRS .

DATA: BEGIN OF WA\_T001 ,  
      BUKRS TYPE T001-BUKRS ,  
      BUTXT TYPE T001-BUTXT ,  
      ORT01 TYPE T001-ORT01 ,  
      END OF WA\_T001 .

CLASS C1 DEFINITION .

  PUBLIC SECTION .

    METHODS: GET\_DATA IMPORTING  
              I\_BUKRS TYPE T001-BUKRS ,  
              DISPLAY .

  ENDCLASS .

CLASS C1 IMPLEMENTATION .

  METHOD GET\_DATA .

    SELECT SINGLE BUKRS BUTXT ORT01 FROM T001 INTO WA\_T001 WHERE BUKRS  
    = I\_BUKRS .

  ENDMETHOD .

  METHOD DISPLAY .

    WRITE:/ WA\_T001-BUKRS , WA\_T001-BUTXT , WA\_T001-ORT01 .

  ENDMETHOD .

  ENDCLASS .

START-OF-SELECTION .

  DATA RC1 TYPE REF TO C1 .

  CREATE OBJECT RC1 .

  CALL METHOD RC1->GET\_DATA( EXPORTING I\_BUKRS = P\_BUKRS ) .

  CALL METHOD RC1->DISPLAY .

Each class contains two types of components

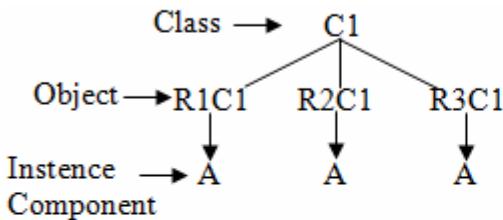
1. Instance components.

2. Static components

Differences between Instance & Static

### Instance

- i. Instance components exists for each object of the class.



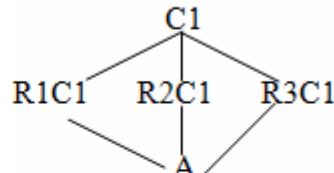
- ii. Instance components are declared with

- Data
- Methods
- Events
- Interfaces

- iii. Instance components always accessed through class objects.

### Static

- i. Static components exists for only once for all objects at class.



- ii. Static components are declared with

- Class-Data
- Class-Methods
- Class-Events
- Class-Interfaces

- iii. Static components always accessed through class object or class name.

```

CLASS c1 DEFINITION.
  PUBLIC SECTION.
    CLASS-DATA: v1 TYPE i.
    DATA v2 TYPE i.
    CLASS-METHODS: static_method.
    METHODS instance_method.
ENDCLASS.
  
```

```

CLASS c1 IMPLEMENTATION.
  
```

```

  METHOD static_method.
    v1 = 10.
*    v2 = 20.
  ENDMETHOD.

  METHOD instance_method.
    v1 = 20.
    v2 = 30.
  ENDMETHOD.
ENDCLASS.
  
```

```

START-OF-SELECTION.
  
```

```

    DATA: ob1 TYPE REF TO c1.
    CREATE OBJECT ob1.
    DATA: ob2 TYPE REF TO c1.
    CREATE OBJECT ob2.
  
```

```

    CALL METHOD ob1->static_method.
    CALL METHOD ob1->instance_method.
    CALL METHOD ob2->static_method.
    CALL METHOD ob2->instance_method.
  
```

→ Manually filling the company code, company name & city internal tables by using OOABAP.

TYPES: BEGIN OF TY\_T001,  
 BUKRS TYPE T001-BUKRS,

एम एन सतीष कुमार रेडि

```

BUTXT TYPE T001-BUTXT,
ORT01 TYPE T001-ORT01,
END OF TY_T001.
DATA: WA_T001 TYPE TY_T001,
      IT_T001 TYPE TABLE OF TY_T001.
CLASS C1 DEFINITION.
PUBLIC SECTION.
METHODS: GET_DATA IMPORTING
         I_BUKRS TYPE BUKRS
         I_BUTXT TYPE BUTXT
         I_ORT01 TYPE ORT01.
ENDCLASS.
CLASS C1 IMPLEMENTATION.
METHOD GET_DATA.
  WA_T001-BUKRS = I_BUKRS.
  WA_T001-BUTXT = I_BUTXT.
  WA_T001-ORT01 = I_ORT01.
  APPEND WA_T001 TO IT_T001.
  CLEAR WA_T001.
ENDMETHOD.
ENDCLASS.

START-OF-SELECTION.
DATA RC1 TYPE REF TO C1.
CREATE OBJECT RC1.

CALL METHOD RC1->GET_DATA
EXPORTING
  I_BUKRS = '1000'
  I_BUTXT = 'HCL'
  I_ORT01 = 'HYD'.
CALL METHOD RC1->GET_DATA
EXPORTING
  I_BUKRS = '2000'
  I_BUTXT = 'IBM'
  I_ORT01 = 'CHE'.

LOOP AT IT_T001 INTO WA_T001.
  WRITE:/ WA_T001-BUKRS,WA_T001-BUTXT,WA_T001-ORT01.
ENDLOOP.

```

### → Calling a method from another method

```

CLASS LC DEFINITION.
PUBLIC SECTION.
METHODS:M1 IMPORTING I_ENO TYPE I
         IENAME TYPE C,
         M2 .
PROTECTED SECTION.
DATA: EMPNO(20) TYPE C,
      EMPNAME(30) TYPE C.
ENDCLASS.

```

```

CLASS LC IMPLEMENTATION.
METHOD M1.
  EMPNO = I_ENO.
  EMPNAME = IENAME.
  CALL METHOD M2.
ENDMETHOD.
METHOD M2.
  WRITE:/ EMPNO, EMPNAME .
ENDMETHOD.
ENDCLASS.
DATA OBJ TYPE REF TO LC.
START-OF-SELECTION.
  CREATE OBJECT OBJ.

PARAMETER: P_ENO TYPE I,
            PENAME(30) TYPE C.

CALL METHOD OBJ->M1
  EXPORTING
    I_ENO = P_ENO
    IENAME = PENAME.

```

**→ Display the customer details based on customer number**

```

CLASS LC DEFINITION.
  PUBLIC SECTION.
    METHODS GET_DET IMPORTING I_KUNNR TYPE KNA1-KUNNR
      EXPORTING E_NAME1 TYPE KNA1-NAME1
                  E_LAND1 TYPE KNA1-LAND1.
  ENDCLASS.
CLASS LC IMPLEMENTATION.
  METHOD GET_DET.
    SELECT SINGLE NAME1 LAND1
      FROM KNA1 INTO (E_NAME1, E_LAND1)
      WHERE KUNNR = I_KUNNR.
  ENDMETHOD.
ENDCLASS.

DATA OBJ TYPE REF TO LC.
START-OF-SELECTION.
  CREATE OBJECT OBJ.
  PARAMETER P_KUNNR TYPE KNA1-KUNNR.
  DATA: D_NAME1 TYPE KNA1-NAME1,
        D_LAND1 TYPE KNA1-LAND1.
  CALL METHOD OBJ->GET_DET
    EXPORTING
      I_KUNNR = P_KUNNR
    IMPORTING
      E_NAME1 = D_NAME1
      E_LAND1 = D_LAND1.
  WRITE:/ 'CUST NAME: ', D_NAME1.
  WRITE:/ 'CUST CITY: ', D_LAND1.

```

**Constructor:-** Constructor is a special method to initialize the attributes at runtime. Constructor contains two types. They are Instance constructor and Static Constructor.  
Differences between Normal methods and Constructors.

normal methods	special methods (constructors)
1. Can be declared in any of the sections	1. can be declared only in public section
2. Should be called explicitly	2. Will be called implicitly
3. Can be called any no. of times using the same object	3. inst. const will be called only once in the lifetime of every object , static const. will be called only once in the lifetime of class
4. can contain any type of parameters (import, export, changing, returning)	4. instance const. can contain only import parameters and static const. cannot contain any parameters
5. can return any no. of values	5. Never returns values

→ Write a simple program to differentiate between Instance constructor and Static constructor

CLASS SR DEFINITION.

  PUBLIC SECTION.

    METHODS CONSTRUCTOR.

      CLASS-METHODS CLASS\_constructor.

  ENDCLASS.

CLASS SR IMPLEMENTATION.

  METHOD CONSTRUCTOR.

    WRITE:/ 'INSIDE THE INSTANCE CONSTRUCTOR'.

  ENDMETHOD.

  METHOD CLASS\_CONSTRUCTOR.

    WRITE:/ 'INSIDE THE STATIC CONSTRUCTOR'.

  ENDMETHOD.

  ENDCLASS.

DATA OBJ TYPE REF TO SR.

DATA OBJ2 TYPE REF TO SR.

DATA OBJ3 TYPE REF TO SR.

START-OF-SELECTION.

  WRITE:/ 'FIRST OBJECT'.

  CREATE OBJECT OBJ.

  ULINE.

  WRITE:/ 'SECOND OBJECT'.

```

CREATE OBJECT OBJ2 .
ULINE .
WRITE:/ 'THIRD OBJECT' .
CREATE OBJECT OBJ3 .

```

The output will be displayed like this. Here Instance constructor is executed in all the objects. But Static constructor is executed in only first object. After that it's not executed. Because Instance constructor is execute only one time in the life time of the object. So there are 3 types of objects. So 3 times it's executed. Static constructor will be exected only once in the life time of the class. There are 3 types of different objects, but those all are in the single class. So It's executed only once in this program.

FIRST OBJECT INSIDE THE STATIC CONSTRUCTOR INSIDE THE INSTANCE CONSTRUCTOR
SECOND OBJECT INSIDE THE INSTANCE CONSTRUCTOR
THIRD OBJECT INSIDE THE INSTANCE CONSTRUCTOR

#### → Develop a Global class & method to display the all purchasing document details based on the given purchasing document number.

Execute 'SE24'. Provide the object type as class name (ZSJF\_10AM\_GC1). Click on create. Enter. Provide short description. Click on save click on local object. Provide the method name (GET\_PO\_DETAILS). Select the level (INSTANCE Method). Select the visibility as public, provide short description. Click on parameters.

<u>Parameter</u>	<u>Type</u>	<u>Typing message</u>	<u>Associated type</u>
I_EBELN	IMPORTING TYPE		EKKO-EBELN
E_WA	EXPORTING TYPE		EKKO

Click on save. Click on code icon beside exceptions. Click on signature in the application tool bar & identifies the input output parameters & implement the logic.

METHOD GET\_PO\_DETAILS.

SELECT SINGLE \* FROM EKKO INTO E\_WA WHERE EBELN = I\_EBELN.

ENDMETHOD.

Save, check, activate the method. Click on back. Repeat the same steps for all other methods. Save, check, activate the class.

#### Steps to create the object for Global class: -

Place the cursor in the program where we want to create the object. Click on pattern in the application tool bar. Select the radio button ABAP objects patterns. Enter. Select the radio button create object. Provide the instance name as reference name (RGC). Provide the Global class name (ZSJF\_10AM\_GC1). Enter.

**Note:** – Constructer is the one special type of method in the class. We no need to call the constructer externally by using call method. When ever we create the object for the class then automatically constructer will be triggered & asked the input output parameters.

#### Steps to call the Global method: -

Place the cursor where we want to call the method in the program. Click on pattern in the application tool bar. Select the ABAP objects patterns. Enter. Select the radio button call method. Provide the instance name as reference name (RGC). Provide the class name (ZSJF\_10AM\_GC1), method name (GET\_PO\_DETAILS). Enter.

```

PARAMETER P_EBELN TYPE EKKO-EBELN .
DATA: WA_EKKO LIKE EKKO ,
      IT_EKKO LIKE TABLE OF WA_EKKO .
* CREATE THE REFERENCE TO THE GLOBAL CLASS .
DATA RGC TYPE REF TO ZSJF_10AM_GC1 .

```

```

* CREATE THE OBJECT.
START-OF-SELECTION.
  CREATE OBJECT RGC.
  CALL METHOD RGC->GET_PO_DETAILS
    EXPORTING
      I_EBELN = P_EBELN
    IMPORTING
      E_WA      = WA_EKKO.
    APPEND WA_EKKO TO IT_EKKO.
* Display the output.
  CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
    EXPORTING
      I_STRUCTURE_NAME = 'EKKO'
    TABLES
      T_OUTTAB          = IT_EKKO.

```

**Note:** - If you want to declare the WA with some of the fields in the global method & class then we must create one structure with those fields in the data dictionary & after we refer the structure in global method & class.

**Note:** - If you want to declare the IT with some of the fields in the global class & method then in data dictionary we have to create table type & later we refer the table type in the global class & method.

**Note:** - If you want to declare the select-options in global class & method then we must create one table type with sign, option, low, high in the data dictionary & later we refer the table type in the global method & class.

→ Based on the given sales document number, display the sales document number, document date & customer number by using global class & method.

In this object we create one structure with VBELN, AUDAT, KUNNR & later we refer the structure in the global method.

Structure : ZSJF\_10AM\_GCS

<u>Component</u>	<u>Component type</u>
VBELN	VBELN_VA
AUDAT	AUDAT
KUNNR	KUNAG

Global class: ZSJF\_10AM\_GC2

Method : GET\_SO\_DETAILS

<u>Parameter</u>	<u>Type</u>	<u>Typing message</u>	<u>Associated type</u>
I_VBELN	IMPORTING	TYPE	VBAK-VBELN
E_WA	EXPORTING	TYPE	ZSJF_10AM_GCS

Code: -

```

SELECT SINGLE VBELN AUDAT KUNNR FROM VBAK INTO E_WA WHERE VBELN =
I_VBELN.

```

```
PARAMETER P_VBELN TYPE VBAK-VBELN.
```

```
DATA WA_VBAK TYPE ZSJF_10AM_GCS.
```

\* Create the reference

```
DATA RGC TYPE REF TO ZSJF_10AM_GC2.
```

```

* Create the object
START-OF-SELECTION.
  CREATE OBJECT RGC.

* Call the method
  CALL METHOD RGC->GET_SO_DETAILS
    EXPORTING
      I_VBELN = P_VBELN
    IMPORTING
      E_WA = WA_VBAK.
  WRITE:/ WA_VBAK-VBELN, WA_VBAK-AUDAT, WA_VBAK-KUNNR.

```

→ Based on the given material numbers, display the material numbers, material types, plant numbers, plant descriptions by using global class & method.

In this object we create two table types one is for select-options, one is for display the output. Creation of table type is two step procedure.

1. Create the structure with those fields.
2. Based on the structure we create the table type.

Structure: ZSJF\_10AM\_GCSI

Sign	Char	1
Option	Char	2
Low	Char	18
High	Char	18

Table type name: ZSJF\_10AM\_TTI.

Line type : ZSJF\_10AM\_GCSI.

Structure name: ZSJF\_10AM\_GCSO

MATNR	MATNR
MTART	MTART
WERKS	WERKS_D
NAME1	NAME1

Table type: ZSJF\_10AM\_TTO

Line type: ZSJF\_10AM\_GCSO

Class name: ZSJF\_10AM\_GC3

Method : GET\_MAT\_DETAILS

I_SMATNR	Importing	type	ZSJF_10AM_TTI
E_IT	Exporting	type	ZSJF_10AM_TTO

Code: -

```

SELECT MARA~MATNR MARA~MTART MARC~WERKS T001W~NAME1 INTO TABLE E_IT
FROM MARA INNER JOIN MARC ON MARA~MATNR = MARC~MATNR INNER JOIN T001W
ON MARC~WERKS = T001W~WERKS WHERE MARA~MATNR IN I_SMATNR.

```

TABLES MARA.

SELECT-OPTIONS S\_MATNR FOR MARA-MATNR.

DATA IT\_FINAL TYPE ZSJF\_10AM\_TTO.

\* Create the reference to the class

DATA RGC TYPE REF TO ZSJF\_10AM\_GC3.

\* Create the object

START-OF-SELECTION.

CREATE OBJECT RGC.

```

* Call the method
CALL METHOD RGC->GET_MAT_DETAILS
  EXPORTING
    I_SMATNR = S_MATNR[]
  IMPORTING
    E_IT      = IT_FINAL.

* Display the output
CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
  EXPORTING
    I_STRUCTURE_NAME = 'ZSJF_10AM_GCSO'
  TABLES
    T_OUTTAB        = IT_FINAL.

```

→ Based on the given company code, display the vendors, vendor company details (BUKRS, LIFNR, AKONT) by using global class and method

#### Steps to create the global class method: -

Execute SE24. Provide the class name (ZSJF\_10AM\_GC4). Click on create. Enter. Provide short description. Enter. Click on types tab. Provide the type name.

Type name : TY\_LFB1

Visibility : Public

Select the visibility and click on back (against type). Click on yes. Remove the types. Declare the data types.

Types: Begin of ty\_lfb1,

Bukrs type lfb1-bukrs,

Lifnr type lfb1-lifnr,

Akont type lfb1-akont,

End of ty\_lfb1.

Types TY\_T\_LFB1 type table of ty\_lfb1.

Save, check, activate. Click on back. Click on methods tab. Provide the method name (GET\_VENDOR).

Select the level is (Instance method). Select the visibility is public. Provide description. Click on parameters tab.

<u>Parameter</u>	<u>Type</u>	<u>Typing message</u>	<u>Associated type</u>
I_bukrs	importing	type	lfb1-bukrs
E_it	exporting	type	ty_t_lfb1

Click on save. Click on code.

Select bukrs lifnr akont from lfb1 into table e\_it where bukrs = i\_bukrs.

Save, check, activate the method. Click on back. Save, check, activate the class.

**Inheritance:** - Inheritance is nothing but creating new object based on existing one. The new class is called child class / derived class / sub class. The existing class is called super class / parent class.

The sub class can access all the components of super class which are defined under public or protected section only not under the private section. Through super class object we can access the components of the super class only. Through sub class object we can access the components of sub class as well as super class also.

### Syntax of defining the subclass: -

Class <sub class name> definition inheriting from <super class name>.

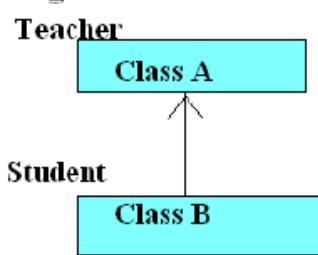
Public / protected / private section.

-----  
----- } components.  
-----

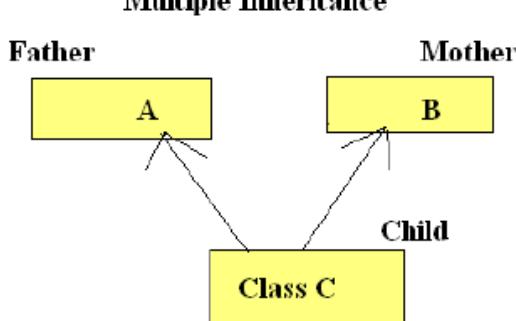
Endclass.

### Types of Inheritance

#### Single Inheritance



#### Multiple Inheritance



Actually Multiple inheritance isn't possible. If we want Multiple Inheritance then we can do through interfaces only. Directly it's not possible.

→ Based on given company code display the company code, company name & city by using OOABAP which inheritance concept.

PARAMETER P\_BUKRS TYPE T001-BUKRS.

DATA: BEGIN OF WA\_T001,

```

      BUKRS TYPE T001-BUKRS,
      BUTXT TYPE T001-BUTXT,
      ORT01 TYPE T001-ORT01,
    END OF WA_T001.
  
```

CLASS C1 DEFINITION.

PUBLIC SECTION.

```

METHODS GET_DATA IMPORTING
      I_BUKRS TYPE BUKRS.
  
```

ENDCLASS.

CLASS C1 IMPLEMENTATION.

METHOD GET\_DATA.

```

      SELECT SINGLE BUKRS BUTXT ORT01 FROM T001 INTO WA_T001 WHERE BUKRS
      = I_BUKRS.
    ENDMETHOD.
  
```

```

ENDCLASS.
CLASS C2 DEFINITION INHERITING FROM C1.
  PUBLIC SECTION.
    METHODS DISPLAY.
ENDCLASS.
CLASS C2 IMPLEMENTATION.
  METHOD DISPLAY.
    WRITE:/ WA_T001-BUKRS, WA_T001-BUTXT, WA_T001-ORT01.
  ENDMETHOD.
ENDCLASS.
START-OF-SELECTION.
  DATA: RC1 TYPE REF TO C1,
        RC2 TYPE REF TO C2.
  CREATE OBJECT: RC1, RC2.
  CALL METHOD RC2->GET_DATA
    EXPORTING
      I_BUKRS = P_BUKRS.
  CALL METHOD RC2->DISPLAY.

```

We can declare the classes in two ways. Global classes can declare in SE24 transaction We can maintain some methods in global classes, we can use those methods in local classes. Local classes are declared in SE38 transaction.

There are few types of classes. Those are

Usual ABAP class

Abstract class

Final class

Persistent class

Friend class

Singleton class

Usual ABAP class is nothing but normal class. A class which contain at least one or more than one abstract method, then it's called as Abstract class. A class which don't allow the Inheritance concept that class is called Final class. Persistence service is used to form a bridge between your relational database and ABAP objects. Friend class is used to access the properties of another class which are defined in Private section also. A class which can allow to create only one instance (object), it's called as Singleton class.

### **Method Overriding:**

Method overriding means We can declare one method in super class. We have some values in this method. I'm using method redefinition in the sub class. In sub class I'm giving the different values. At the time of calling the method from subclass object, it'll display the values which are defined in the subclass method.

### **Method Over loading:**

Method over loading means a class can contain more than one methods with same name and different variables, it's called as method overloading. But ABAP doesn't support Over loading.

```

CLASS SR DEFINITION.
  PUBLIC SECTION.
    METHODS: M1,
              M2 FINAL.
    CLASS-METHODS M3.

```

```

ENDCLASS.
CLASS SR IMPLEMENTATION.
METHOD M1.
  WRITE:/ 'UNDER SUPER CLASS INSTANCE METHOD'.
ENDMETHOD.
METHOD M2.
  WRITE:/ 'UNDER SUPER CLASS INSTANCE FINAL METHOD'.
ENDMETHOD.
METHOD M3.
  WRITE:/ 'UNDER SUPER CLASS STATIC METHOD'.
ENDMETHOD.
ENDCLASS.
CLASS KS DEFINITION INHERITING FROM SR.
PUBLIC SECTION.
  METHODS M1 REDEFINITION.
  METHODS M4.
  CLASS-METHODS M5.
ENDCLASS.

CLASS KS IMPLEMENTATION.
METHOD M1.
  WRITE:/ 'REDEFINED M1 METHOD IN SUB CLASS WITH OBJ2'.
ENDMETHOD.

METHOD M4.
  WRITE:/ 'UNDER SUB CLASS INSTANSE METHOD'.
ENDMETHOD.
METHOD M5.
  WRITE:/ 'UNDER SUB CLASS STATIC METHD'.
ENDMETHOD.
ENDCLASS.

DATA OBJ1 TYPE REF TO SR.
DATA OBJ2 TYPE REF TO KS.

START-OF-SELECTION.
CREATE OBJECT OBJ1.
CREATE OBJECT OBJ2.

WRITE:/ 'M1 METHOD FROM OBJ2'.
CALL METHOD OBJ2->M1.
ULINE.
WRITE:/ 'M1 METHOD FROM OBJ1'.
CALL METHOD OBJ1->M1.
ULINE.
WRITE:/ 'M2 METHOD FROM OBJ2'.
CALL METHOD OBJ2->M2.
ULINE.
WRITE:/ 'M3 METHOD FROM OBJ2'.
CALL METHOD OBJ2->M3.
ULINE.

```

```

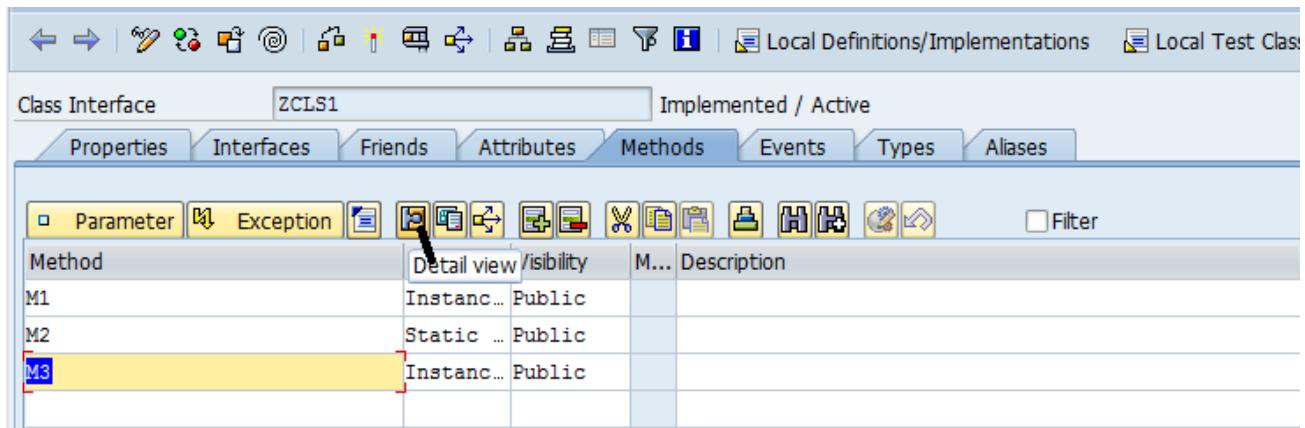
WRITE:/ 'M4 METHOD FROM OBJ2'.
CALL METHOD OBJ2->M4.
ULINE.
WRITE:/ 'M5 METHOD FROM OBJ2'.
CALL METHOD OBJ2->M5.

```

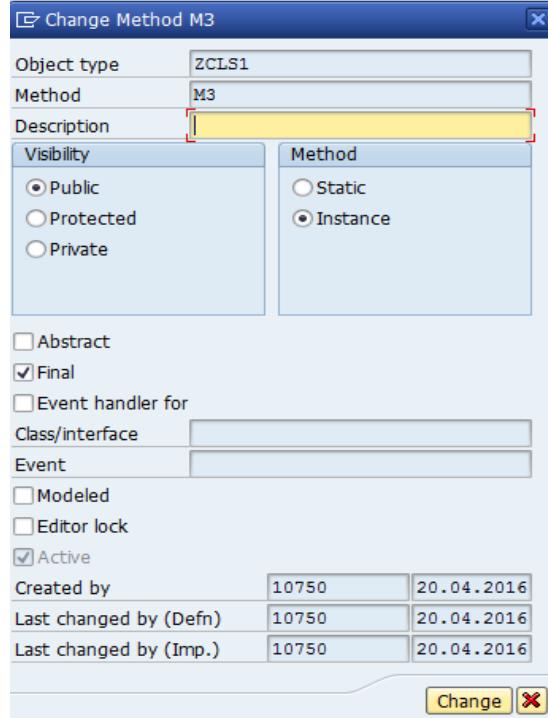
**Note:** - We can't redefine Static methods, Final methods in the Sub class.

In SE24 also we can create Final method like this.

First select the method. Click on Detail view icon which is beside code icon.

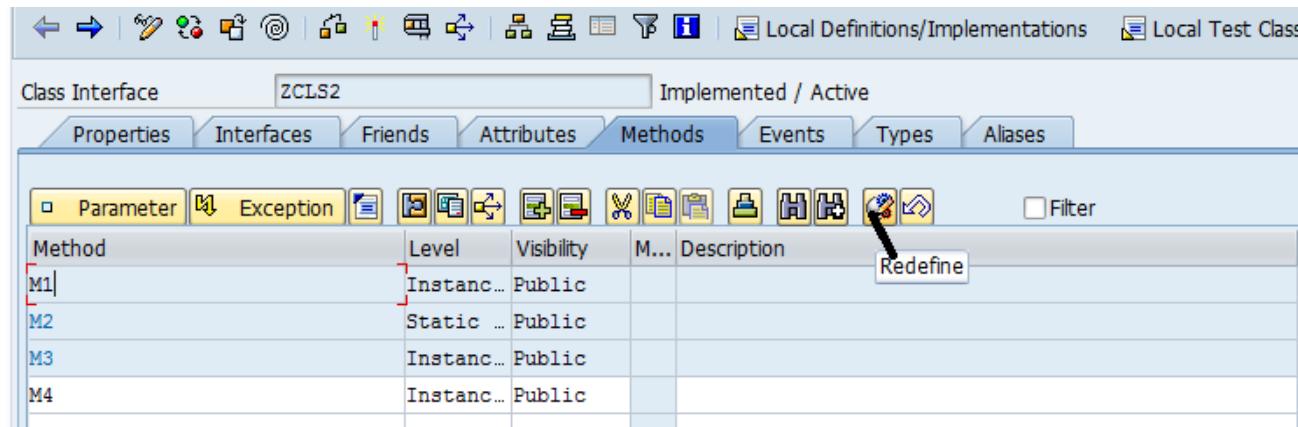


After click on Detail view, you select 'FINAL' check box as shown in the beside picture. Next click on change. Save, check, activate the class.



If you want redefine the method in sub class. Then you have to do like this.

Click on the method which method you want to redifne. Click on Redefine Icon. Then you can write the code in that method.



**Abstract:** - A class which contain atleast one or more than one abstract method then it's called as Abstract class. These are used to define some common functionalities in Abstract class (Super-class) and those can be used in derived classes (Sub classes).

**Note:** - When ever we are working with Abstract class in SE38, then you have define 'ABSTRACT' as keyword externally as given below syntax.

```
class SR DEFINITION ABSTRACT.
```

**Note:** - We can't create the object for the Abstract class.

```
CLASS SR DEFINITION ABSTRACT.  
  PUBLIC SECTION.  
    METHODS SANJAYRAMASWAMI ABSTRACT.  
    METHODS SATHISH.  
    METHODS HARSHA.  
  ENDCLASS.  
  
CLASS SR IMPLEMENTATION.  
  METHOD SATHISH.  
    WRITE:/ 'SATHISH = SANJAYRAMASWAMI'.  
  ENDMETHOD.  
  METHOD HARSHA.  
    WRITE:/ 'HARSHA = SATHISH'.  
  ENDMETHOD.  
ENDCLASS.  
  
CLASS KS DEFINITION INHERITING FROM SR.  
  PUBLIC SECTION.  
    METHODS SANJAYRAMASWAMI REDEFINITION.  
    METHODS KALPANA.  
  ENDCLASS.
```

```

CLASS KS IMPLEMENTATION.
METHOD KALPANA.
  WRITE:/ 'SANJAYRAMASWAMI = KALPANA'.
ENDMETHOD.
METHOD SANJAYRAMASWAMI.
  WRITE:/ 'SANJAY RAMASWAMI CHARACTER ACTS AS MAIN ROAL IN MY LIFE'.
ENDMETHOD.
ENDCLASS.

DATA OBJ TYPE REF TO SR.
DATA OBJ1 TYPE REF TO KS.

START-OF-SELECTION.
* CREATE OBJECT OBJ. "we can't create instances of the abstract class
  CREATE OBJECT OBJ1.

*
  CALL METHOD OBJ1->SATHISH.
*
  CALL METHOD OBJ1->HARSHA.
*
  CALL METHOD OBJ1->KALPANA.
*
  CALL METHOD OBJ1->SANJAYRAMASWAMI.

OBJ = OBJ1.
CALL METHOD OBJ->SANJAYRAMASWAMI .
CALL METHOD OBJ->SATHISH .

```

**Interface:** - Interface is the collection of Methods which are defined but not implemented. These all are implemented through another class.

```

INTERFACE MEN.
  METHODS: FATHER,
           BROTHER.
ENDINTERFACE.

INTERFACE WOMEN.
  METHODS: MOTHER,
           SISTER.
ENDINTERFACE.

CLASS MAN DEFINITION.
  PUBLIC SECTION.
    INTERFACES: MEN,
                WOMEN.
ENDCLASS.

CLASS MAN IMPLEMENTATION.
  METHOD MEN~FATHER.
    WRITE:/ 'FATHER GENDER IS MALE UNDER MEN GROUP'.
  ENDMETHOD.
  METHOD MEN~BROTHER.
    WRITE:/ 'BROTHER IS MALE UNDER MEN GROUP'.
  ENDMETHOD.

```

```

METHOD WOMEN~MOTHER.
  WRITE:/ 'MOTHER IS FEMAL UNDER WOMEN GROUP'.
ENDMETHOD.
METHOD WOMEN~SISTER.
  WRITE:/ 'SISTER IS FEMAL UNDER WOMEN GROUP'.
ENDMETHOD.
ENDCLASS.

DATA OBJ TYPE REF TO MAN.

START-OF-SELECTION.
  CREATE OBJECT OBJ.

*           CALL METHOD OBJ->MEN~FATHER.

DATA OBJ1 TYPE REF TO MEN.
DATA OBJ2 TYPE REF TO WOMEN.

OBJ1 = OBJ.
OBJ2 = OBJ.
CALL METHOD OBJ1->FATHER.
CALL METHOD OBJ2->MOTHER.

```

ABSTRACT CLASSES	INTERFACES
1. Can contain both abstract and non-abstract methods	1. Can contain only abstract methods
2. Explicitly needs to declare the method as 'abstract'	2. By default, all the methods are 'abstract'
3. Abstract methods can be declared in public and protected sections	3. By default , all the components of interface are public
4. A Class Can inherit only one abstract class	4. A Class Can Implement any no. of interfaces
5. Abstract class inherited components can be referred directly in sub classes	5. Outside the interface, interface components must be prefixed with interface name

**Encapsulation:** - Binding or wrapping the code and data in a single unit is called Encapsulation. CLASS is the best example for the Encapsulation. In a class we can write methods, events, variables all as single unit. So, Class is the best example for Encapsulation.

**Polymorphism:** - A single entity or method behaves in multiple forms then it's called Polymorphism. For example take VBELN. It acts as Sales document number in VBAK table. It acts as Delivery number in LIKP table. It acts as billing number in VBRK table. VBELN is only one variable acts as 3 types in 3 different tables.

**Exceptions:** - Exception is nothing but run time error. We handle this exceptions by using TRY & CATCH blocks.

**Note:** - CX\_ROOT is used to store the error.

```
CLASS SR DEFINITION.  
  PUBLIC SECTION.  
    METHODS DEV IMPORTING I_X TYPE I  
             I_Y TYPE I  
             EXPORTING E_Z TYPE I.  
  ENDCLASS.  
  
CLASS SR IMPLEMENTATION.  
  METHOD DEV.  
    TRY.  
      E_Z = I_X / I_Y.  
    CATCH CX_ROOT.  
      MESSAGE 'CAN NOT DEVISIBLE BY ZERO' TYPE 'I'.  
    ENDTRY.  
  ENDMETHOD.  
ENDCLASS.  
  
DATA OBJ TYPE REF TO SR.  
  
START-OF-SELECTION.  
  CREATE OBJECT OBJ.  
  
  PARAMETER: P_X TYPE I,  
            P_Y TYPE I.  
  DATA RES TYPE I.  
  
  CALL METHOD OBJ->DEV  
    EXPORTING  
      I_X = P_X  
      I_Y = P_Y  
    IMPORTING  
      E_Z = RES.  
    WRITE:/ RES.
```

**Casting:** - Casting is noting but switching one object from one type to another type. This is two types. 1> Narraow casting 2> Wide casting.

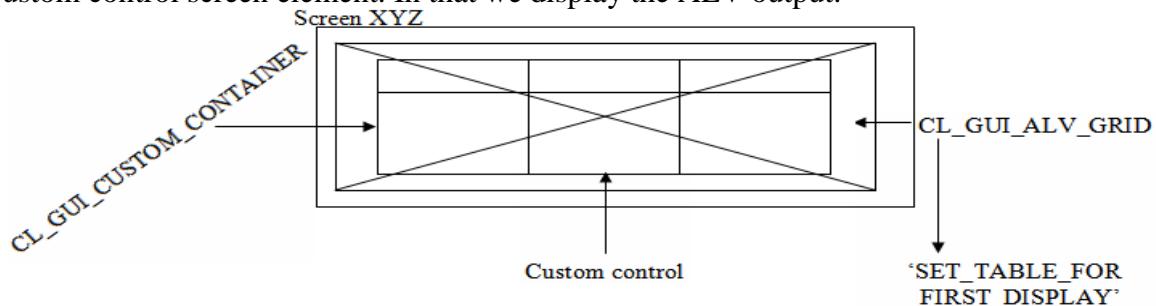
Narrow casting is nothing but switching one object from more detailed view to less detailed view.

Wide casting is nothing but switching one object from less detailed view to more detailed view.

**Note:** - When ever we are working with class, we have to implement all methods in implementation part. Other wise it'll display error. If you are working with Abstract class, we may have abstract methods & normal methods. We have to implement the normal methods in parent class & implement the abstract methods in child class. Otherwise it'll show error. If you are working with Inherintance and redefinition method, then the visibility must me same in super, sub classes of that methods.

# OOPS ALV

When ever we are working with OOPS ALV, then we must design one custom screen other than '1000' and draw the custom control screen element. In that we display the ALV output.



**CL\_GUI\_CUSTOM\_CONTAINER:** - The Global class which refers the custom control screen element.

**CL\_GUI\_ALV\_GRID:** - The global class which refers the ALV Grid.

**SET\_TABLE\_FOR\_FIRST\_DISPLAY:** - This is the Global method under grid class which is used to display the output in an OOPS ALV. The input for the above method is two internal tables.

1. Data internal table
2. Field catalog internal table

If you want to call this method then we must create object for gird class. When ever we create the object for gird class then automatically one constructor is triggered and asks the input as object name of container class.

When ever we create the object for container class then automatically one constructor is triggered and asks the input as container name.

**Note:** - When ever we are working with OOPS ALV then we

must design one custom screen

## **Steps to work with OOPS ALV:-**

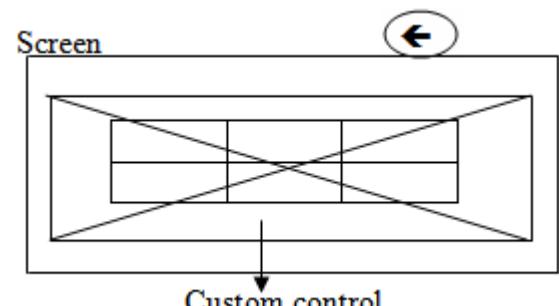
1. Create the selection-screen / input fields.
2. Declare the data internal table and field catalog internal table
3. Create the reference to the container and grid class start-of-selection.
4. Call screen '<screen number>' & place custom control component in it.

## **PBO of the screen: -**

1. Design the back button
2. Create the object to the container and gird class
3. Fill the data internal table
4. Fill the field catalog
5. Call the SET\_TABLE\_FOR\_FIRST\_DISPLAY method

## **PAI of the screen**

1. Logic of back button.



## **Filling the field catalog**

If we are working with all the fields from any one of the data base table / structure, then we no need to prepare the filed catalog. We simply pass i\_structure\_name as data base table name / structure name.

Manually filling the field catalog

By using  
'LVC\_FIELDCATALOG\_MERGE' function module

### **Some of the fields in field catalog: -**

1. Field name → Name of the field
2. Col\_pos → Column position
3. Coltext → Column heading
4. Emphasize → Color
5. Outputlen → Length of the displayed field
6. No-zero → Remove the leading zero's
7. No-sign → Remove the leading sign
- 8 No-out → Hide the displayed field
9. Hotspot → Handle symbol
10. Edit → Changeable mode
11. Do\_sum → Calculate the total

**→ Based on the given purchasing document numbers to display the all the purchasing document header details by using OOPS ALV.**

REPORT ZOOPS7.

TABLES EKKO.

SELECT-OPTIONS S\_EBELN FOR EKKO-EBELN.

\* Declare the data internal table

DATA IT LIKE TABLE OF EKKO.

\* Create the reference to the container and grid

DATA RC TYPE REF TO CL\_GUI\_CUSTOM\_CONTAINER.

DATA RG TYPE REF TO CL\_GUI\_ALV\_GRID.

START-OF-SELECTION.

CALL SCREEN '2000'.

MODULE STATUS\_2000 OUTPUT.

SET PF-STATUS 'STATUS'.

\* Create the object for container and grid

CREATE OBJECT RC

EXPORTING

CONTAINER\_NAME = 'CC'.

CREATE OBJECT RG

EXPORTING

I\_PARENT = RC.

\* Filling the data internal table

SELECT \* FROM EKKO INTO TABLE IT WHERE EBELN IN S\_EBELN.

\* Display the output

CALL METHOD RG->SET\_TABLE\_FOR\_FIRST\_DISPLAY

EXPORTING

I\_STRUCTURE\_NAME = 'EKKO'

CHANGING

IT\_OUTTAB = IT.

ENDMODULE.

MODULE USER\_COMMAND\_2000 INPUT.

IF SY-UCOMM = 'BACK'.

LEAVE TO SCREEN 0.

ENDIF.

ENDMODULE.

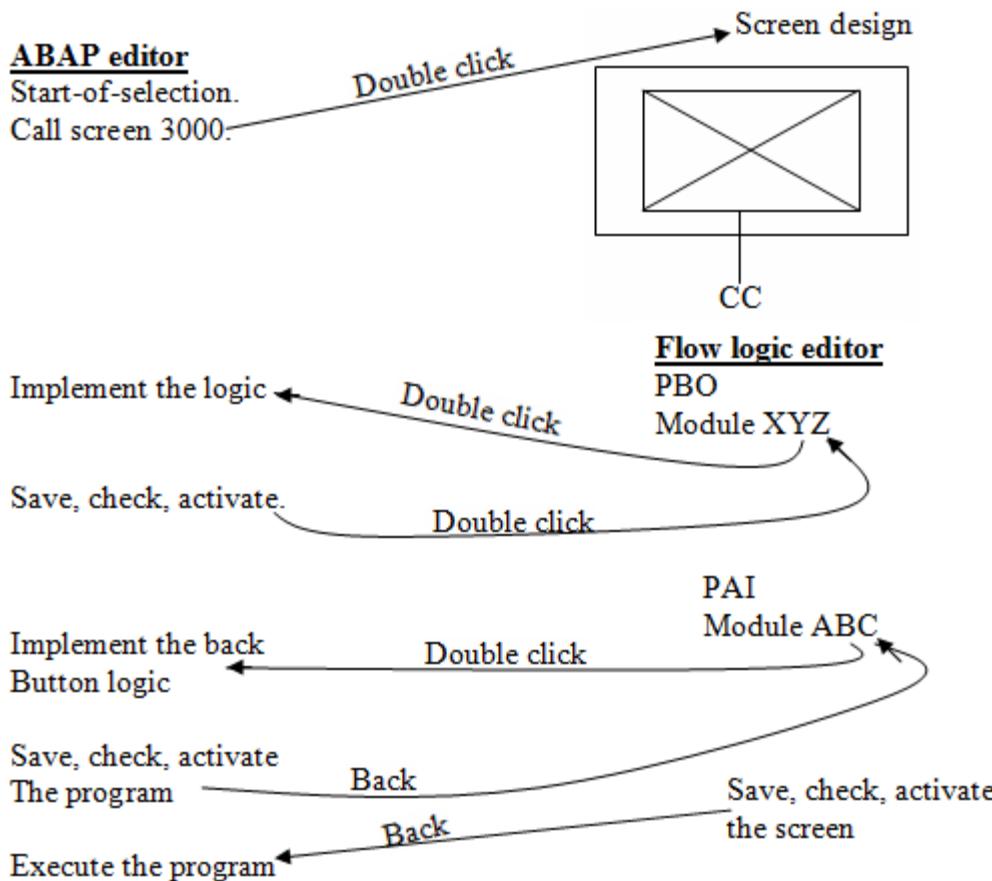
### **Flow logic of 2000 screen**

PROCESS BEFORE OUTPUT.

```

MODULE STATUS_2000.
PROCESS AFTER INPUT.
MODULE USER_COMMAND_2000.

```



→ Based on the given customer numbers to display the customer numbers, names & cities by using OOPS ALV and also display the customer numbers with green color, name with hotspot, city with edit.

```

TABLES KNA1.
SELECT-OPTIONS S_KUNNR FOR KNA1-KUNNR.
* Declare the data internal table
TYPES: BEGIN OF TY_KNA1,
         KUNNR TYPE KNA1-KUNNR,
         NAME1 TYPE KNA1-NAME1,
         ORT01 TYPE KNA1-ORT01,
         END OF TY_KNA1.
DATA: WA_KNA1 TYPE TY_KNA1,
      IT_KNA1 TYPE TABLE OF TY_KNA1.
* Declare the field catalog
DATA: IT_FCAT TYPE LVC_T_FCAT,
      WA_FCAT LIKE LINE OF IT_FCAT.
* Create the references
DATA: RC TYPE REF TO CL_GUI_CUSTOM_CONTAINER,
      RG TYPE REF TO CL_GUI_ALV_GRID.

START-OF-SELECTION.
  CALL SCREEN '2000'.

```

```

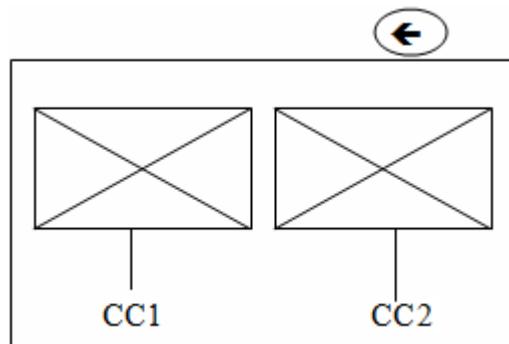
MODULE STATUS_2000 OUTPUT.
  SET PF-STATUS 'STAT'.
  CREATE OBJECT RC
    EXPORTING
      CONTAINER_NAME = 'CC'.
  CREATE OBJECT RG
    EXPORTING
      I_PARENT = RC.
* Filling the data internal table
  SELECT KUNNR NAME1 ORT01 FROM KNA1 INTO TABLE IT_KNA1 WHERE KUNNR IN
S_KUNNR.
* Filling the field catalog
  WA_FCAT-FIELDNAME = 'KUNNR'.
  WA_FCAT-COL_POS = '1'.
  WA_FCAT-COLTEXT = 'CUSTOMER'.
  WA_FCAT-EMPHASIZE = 'C501'.
  WA_FCAT-NO_ZERO = 'X'.
  APPEND WA_FCAT TO IT_FCAT.
  CLEAR WA_FCAT.
  WA_FCAT-FIELDNAME = 'NAME1'.
  WA_FCAT-COL_POS = '2'.
  WA_FCAT-COLTEXT = 'CUST.NAME'.
  WA_FCAT-HOTSPOT = 'X'.
  APPEND WA_FCAT TO IT_FCAT.
  CLEAR WA_FCAT.
  WA_FCAT-FIELDNAME = 'ORT01'.
  WA_FCAT-COL_POS = '3'.
  WA_FCAT-COLTEXT = 'CITY'.
  WA_FCAT-EDIT = 'X'.
  APPEND WA_FCAT TO IT_FCAT.
  CLEAR WA_FCAT.
* Display the output
  CALL METHOD RG->SET_TABLE_FOR_FIRST_DISPLAY
    CHANGING
      IT_OUTTAB = IT_KNA1
      IT_FIELDCATALOG = IT_FCAT.
ENDMODULE.

MODULE USER_COMMAND_2000 INPUT.
  IF SY-UCOMM = 'BACK'.
    LEAVE TO SCREEN 0.
  ENDIF.
ENDMODULE.

```

→ Based on the given sales order numbers, display the sales document header details (VBELN, AUDAT, KUNNR) & sales doc item details (VBELN, POSNR, KWMENG, MEINS, NETWR) side by side by using OOPS ALV.

TABLES VBAK.  
 SELECT-OPTIONS S\_VBELN FOR VBAK-VBELN.  
 TYPES: BEGIN OF TY\_VBAK,  
 VBELN TYPE VBAK-VBELN,



```

AUDAT TYPE VBAK-AUDAT,
KUNNR TYPE VBAK-KUNNR,
END OF TY_VBAK.

DATA: WA_VBAK TYPE TY_VBAK,
      IT_VBAK TYPE TABLE OF TY_VBAK.

TYPES: BEGIN OF TY_VBAP,
       VBELN TYPE VBAP-VBELN,
       POSNR TYPE VBAP-POSNR,
       KWMENG TYPE VBAP-KWMENG,
       MEINS TYPE VBAP-MEINS,
       NETWR TYPE VBAP-NETWR,
       END OF TY_VBAP.

DATA: WA_VBAP TYPE TY_VBAP,
      IT_VBAP TYPE TABLE OF TY_VBAP.

* Declare the field catalogs
DATA: IT_FCAT1 TYPE LVC_T_FCAT,
      WA_FCAT1 LIKE LINE OF IT_FCAT1.

DATA: IT_FCAT2 TYPE LVC_T_FCAT,
      WA_FCAT2 LIKE LINE OF IT_FCAT2.

* Create the references
DATA: RC1 TYPE REF TO CL_GUI_CUSTOM_CONTAINER,
      RG1 TYPE REF TO CL_GUI_ALV_GRID.
DATA: RC2 TYPE REF TO CL_GUI_CUSTOM_CONTAINER,
      RG2 TYPE REF TO CL_GUI_ALV_GRID.

START-OF-SELECTION.
  CALL SCREEN '2000'.
MODULE STATUS_2000 OUTPUT.
  SET PF-STATUS 'STAT'.
  CREATE OBJECT RC1
    EXPORTING
      CONTAINER_NAME = 'CC1'.
  CREATE OBJECT RG1
    EXPORTING
      I_PARENT = RC1.
  CREATE OBJECT RC2
    EXPORTING
      CONTAINER_NAME = 'CC2'.
  CREATE OBJECT RG2
    EXPORTING
      I_PARENT = RC2.

* Filling the data internal tables
  SELECT VBELN AUDAT KUNNR FROM VBAK INTO TABLE IT_VBAK WHERE VBELN IN
S_VBELN.
  SELECT VBELN POSNR KWMENG MEINS NETWR FROM VBAP INTO TABLE IT_VBAP
WHERE VBELN IN S_VBELN.

* Filling the field catalogs
  WA_FCAT1-FIELDNAME = 'VBELN'.
  WA_FCAT1-COL_POS = '1'.
  WA_FCAT1-COLTEXT = 'SALES.DOC'.
  APPEND WA_FCAT1 TO IT_FCAT1.

```

```

CLEAR WA_FCAT1.
WA_FCAT1-FIELDNAME = 'AUDAT'.
WA_FCAT1-COL_POS = '2'.
WA_FCAT1-COLTEXT = 'DOC.DATE'.
APPEND WA_FCAT1 TO IT_FCAT1.
CLEAR WA_FCAT1.
WA_FCAT1-FIELDNAME = 'KUNNR'.
WA_FCAT1-COL_POS = '3'.
WA_FCAT1-COLTEXT = 'CUSTOMER'.
APPEND WA_FCAT1 TO IT_FCAT1.
CLEAR WA_FCAT1.

WA_FCAT2-FIELDNAME = 'VBELN'.
WA_FCAT2-COL_POS = '1'.
WA_FCAT2-COLTEXT = 'SALES.DOC'.
APPEND WA_FCAT2 TO IT_FCAT2.
CLEAR WA_FCAT2.
WA_FCAT2-FIELDNAME = 'POSNR'.
WA_FCAT2-COL_POS = '2'.
WA_FCAT2-COLTEXT = 'ITEM'.
APPEND WA_FCAT2 TO IT_FCAT2.
CLEAR WA_FCAT2.
WA_FCAT2-FIELDNAME = 'KWMENG'.
WA_FCAT2-COL_POS = '3'.
WA_FCAT2-COLTEXT = 'QTY'.
APPEND WA_FCAT2 TO IT_FCAT2.
CLEAR WA_FCAT2.
WA_FCAT2-FIELDNAME = 'MEINS'.
WA_FCAT2-COL_POS = '4'.
WA_FCAT2-COLTEXT = 'UOM'.
APPEND WA_FCAT2 TO IT_FCAT2.
CLEAR WA_FCAT2.
CALL METHOD RG1->SET_TABLE_FOR_FIRST_DISPLAY
  CHANGING
    IT_OUTTAB = IT_VBAK
    IT_FIELDCATALOG = IT_FCAT1.
CALL METHOD RG2->SET_TABLE_FOR_FIRST_DISPLAY
  CHANGING
    IT_OUTTAB = IT_VBAP
    IT_FIELDCATALOG = IT_FCAT2.
ENDMODULE.

MODULE USER_COMMAND_2000 INPUT.
  IF SY-UCOMM = 'BACK'.
    LEAVE TO SCREEN 0.
  ENDIF.
ENDMODULE.

```

### Some of the fields in layout work area: -

1. CWIDTH\_OPT → Compress the displayed fields.
2. ZEBRA → Stripped pattern
3. INFO\_FNAME → color field

**Note:** - In the DDIC we have one structure. That is ‘LVC\_S\_LAYO’ which contains above fields. So we simply declare our layout work area by referring ‘LVC\_S\_LAYO’.

**→ Based on the given purchasing document numbers, display the document numbers, item numbers, quantity, UOM & net price & also display the item details in a yellow color if the amount is more than ‘1000’ by using OOPS ALV.**

### Steps to work with row color:

1. Declare the additional color field in the data internal table, which is char & length is 4.
2. Fill the data internal table based on given input
3. Modify the color field based on client requirement
4. pass the color field name into layout work area (INFO\_FNAME field)
5. Fill the field catalog
6. Display the output by using SET\_TABLE\_FOR\_FIRST\_DISPLAY method by passing data internal table, field catalog internal table, layout work area.

**TABLES EKPO.**

**SELECT-OPTIONS S\_EBELN FOR EKPO-EBELN.**

**\* Declare the data internal table**

**TYPES: BEGIN OF TY\_EKPO,**

```

    EBELN TYPE EKPO-EBELN,
    EBELP TYPE EKPO-EBELP,
    MENGE TYPE EKPO-MENGE,
    MEINS TYPE EKPO-MEINS,
    NETPR TYPE EKPO-NETPR,
    CF(4) TYPE C,
  
```

**END OF TY\_EKPO.**

**DATA: WA\_EKPO TYPE TY\_EKPO,**  
 **IT\_EKPO TYPE TABLE OF TY\_EKPO.**

**\* Declare the field catalog**

**DATA: IT\_FCAT TYPE LVC\_T\_FCAT,**  
 **WA\_FCAT LIKE LINE OF IT\_FCAT.**

**\* Declare the layout work area**

**DATA WA\_LAYOUT TYPE LVC\_S\_LAYO.**

**\* Create the reference**

**DATA: RC TYPE REF TO CL\_GUI\_CUSTOM\_CONTAINER,**  
 **RG TYPE REF TO CL\_GUI\_ALV\_GRID.**

**START-OF-SELECTION.**

**CALL SCREEN '2000'.**

**MODULE STATUS\_2000 OUTPUT.**

**SET PF-STATUS 'STAT'.**

**\* Create the object**

**CREATE OBJECT RC**

**EXPORTING**

**CONTAINER\_NAME = 'CC'.**

**CREATE OBJECT RG**

**EXPORTING**

*IT\_EKKO*

EBELN	EBELP	MENGE	MEINS	NETPR	CF
30004	01	10	KG	570.00	
30004	02	2	PCS	1130.00	
30005	01	5	LT	1500.00	
30005	02	7	BOX	250.00	

**WA\_EKKO-CF =**  
**‘C310’.**

**Modify it\_ekpo from**

*WA\_EKKO*

EBELN	EBELP	MENGE	MEINS	NETPR	CF
30004	01	10	KG	570.00	

```

I_PARENT = RC.

* Filling the data internal table
SELECT EBELN EBELP MENGE MEINS NETPR FROM EKPO INTO TABLE IT_EKPO
WHERE EBELN IN S_EBELN.

* Modify the color field.
WA_EKPO-CF = 'C310'.
MODIFY IT_EKPO FROM WA_EKPO TRANSPORTING CF WHERE NETPR > 1000.

* Pass the color field into layout work area.
WA_LAYOUT-INFO_FNAME = 'CF'.

* Filling the field catalog
WA_FCAT-FIELDNAME = 'EBELN'.
WA_FCAT-COL_POS = '1'.
WA_FCAT-COLTEXT = 'PUR.DOC'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME = 'EBELP'.
WA_FCAT-COL_POS = '2'.
WA_FCAT-COLTEXT = 'ITEM'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME = 'MENGE'.
WA_FCAT-COL_POS = '3'.
WA_FCAT-COLTEXT = 'QTY'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME = 'MEINS'.
WA_FCAT-COL_POS = '4'.
WA_FCAT-COLTEXT = 'UNITS'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

WA_FCAT-FIELDNAME = 'NETPR'.
WA_FCAT-COL_POS = '5'.
WA_FCAT-COLTEXT = 'PRICE'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

CALL METHOD RG->SET_TABLE_FOR_FIRST_DISPLAY
EXPORTING
  IS_LAYOUT      = WA_LAYOUT
CHANGING
  IT_OUTTAB      = IT_EKPO
  IT_FIELDCATALOG = IT_FCAT.

ENDMODULE.

MODULE USER_COMMAND_2000 INPUT.
  IF SY-UCOMM = 'BACK'.
    LEAVE TO SCREEN 0.
  ENDIF.
ENDMODULE.

```

**Note:** - If you want subtotals in OOPs ALV, Declare one Internal table with type compatible LVC\_T\_SORT. Declare one work area also. In this one fill the fields as WA\_SORT-FIELDNAME = 'EBELN', WA\_SORT-SUBTOT = 'X'.

### **Working with events:**

Events are raised as well as handled within the class & also handle through some other class by using methods. Events have only exporting parameters. It doesn't contain importing parameters. Actually Events are used to provide the dynamic values to the application.

**Note:** - When ever we are working with events then we must register the events before call the methods.

### **Syntax of declaring the events:**

Events <event name> exporting <EP1> <EP2> - - -

### **Syntax of raise the event:**

Raise event <event name>

### **Syntax of declaring event handler method:**

Methods <event handler method name> for event <event name> of <class name>.

Which class contains events

### **Syntax of register the events:**

SET handler <object name of class>-><event handler method> for <object name of class>

Which class contains event declaration

Which class contains event handler method

**→ Based on the given customer number, display the customer number, name, city by using OOABAP & also print the error if the input is invalid by using events.**

```

PARAMETER P_KUNNR TYPE KNA1-KUNNR.
DATA: BEGIN OF WA_KNA1,
      KUNNR TYPE KNA1-KUNNR,
      NAME1 TYPE KNA1-NAME1,
      ORT01 TYPE KNA1-ORT01,
      END OF WA_KNA1.

CLASS C1 DEFINITION.
  PUBLIC SECTION.
    METHODS: GET_DATA IMPORTING I_KUNNR TYPE KNA1-KUNNR,
              DISPLAY.
    EVENTS NODATA.
  ENDCCLASS.
CLASS C1 IMPLEMENTATION.
  METHOD GET_DATA.
    SELECT SINGLE KUNNR NAME1 ORT01 FROM KNA1 INTO WA_KNA1 WHERE KUNNR
    = I_KUNNR.
    IF SY-SUBRC <> 0.
      RAISE EVENT NODATA.
    ENDIF.
  ENDMETHOD.
  METHOD DISPLAY.
    WRITE:/ WA_KNA1-KUNNR, WA_KNA1-NAME1, WA_KNA1-ORT01.
  ENDMETHOD.
ENDCLASS.
CLASS C2 DEFINITION.
  PUBLIC SECTION.
    METHODS EHM FOR EVENT NODATA OF C1.
  ENDCCLASS.
CLASS C2 IMPLEMENTATION.
  METHOD EHM.

```

```

MESSAGE E000(ZHAI11) WITH 'INVALID CUSTOMER'.
ENDMETHOD.
ENDCLASS.

```

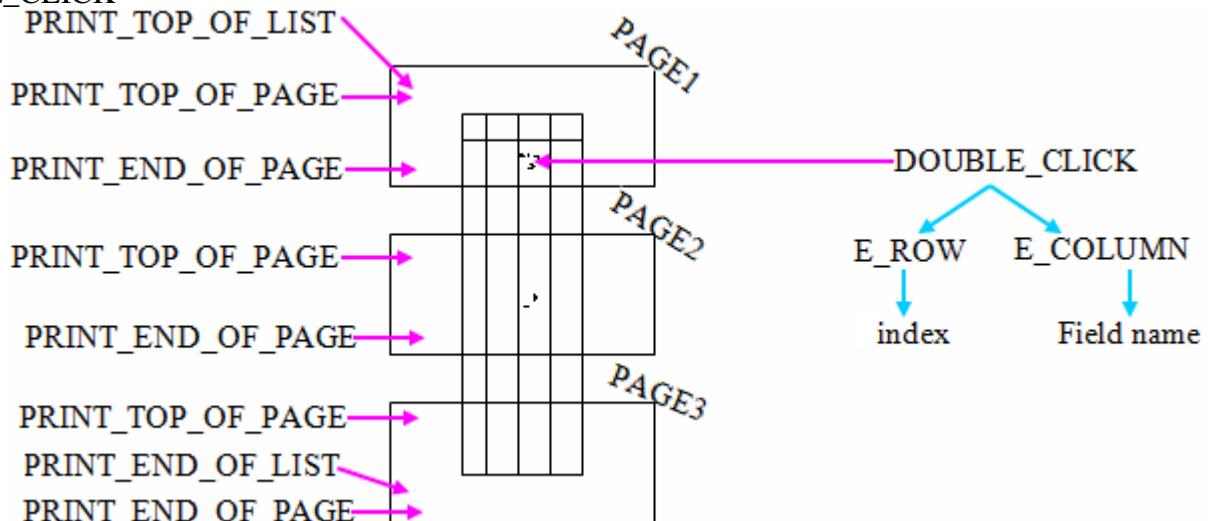
```

START-OF-SELECTION.
DATA: RC1 TYPE REF TO C1,
      RC2 TYPE REF TO C2.
CREATE OBJECT: RC1, RC2.
SET HANDLER RC2->EHM FOR RC1.
CALL METHOD RC1->GET _ DATA
  EXPORTING
    I_KUNNR = P_KUNNR.
CALL METHOD RC1->DISPLAY.

```

### Some of the events in OOPS ALV: -

1. PRINT\_TOP\_OF\_PAGE
2. PRINT\_TOP\_OF\_LIST
3. PRINT\_END\_OF\_PAGE
4. PRINT\_END\_OF\_LIST
5. DOUBLE\_CLICK



**PRINT\_TOP\_OF\_PAGE**: - It's an event which is triggered at the top of each page.

**PRINT\_TOP\_OF\_LIST**: - It's an event which is triggered at the end of the displayed output list.

**PRINT\_END\_OF\_PAGE**: - It's an event which is triggered at the end of each page.

**PRINT\_END\_OF\_LIST**: - It's an event which is triggered at the end of displayed output list.

**DOUBLE\_CLICK**: - It's an event which is triggered at the time of user clicks on any record of any list. It export two parameters (E\_ROW, E\_COLUMN).

E\_ROW contains index number of selected record.

E\_COLUMN contains field name which is clicked by the user.

→ Based on the given sales document number, display the sales document numbers, document dates & customer numbers by using OOPS ALV& also display the TOP\_OF\_PAGE as 'These are sales order details'.

```

REPORT zoops_test1.
DATA: cc          TYPE REF TO cl_gui_custom_container,
      gd          TYPE REF TO cl_gui_alv_grid,
      o_splitter  TYPE REF TO cl_gui_splitter_container,

```

```

o_parent_grid TYPE REF TO cl_gui_container,
o_parent_top  TYPE REF TO cl_gui_container,
o_html_cntrl  TYPE REF TO cl_gui_html_viewer,
o_dyndoc_id   TYPE REF TO cl_dd_document.

TABLES vbak.
SELECT-OPTIONS s_vbeln FOR vbak-vbeln.

DATA: it_fcat      TYPE lvc_t_fcat,
      it_fcat_sub TYPE lvc_t_fcat,
      wa_fcat      LIKE LINE OF it_fcat,
      wa_fcat_sub  LIKE LINE OF it_fcat.

CLASS lcl_application DEFINITION.
  PUBLIC SECTION.
    METHODS:
      top
        FOR EVENT top_of_page OF cl_gui_alv_grid
        IMPORTING e_dyndoc_id table_index.
  ENDCLASS.                                     "lcl_application DEFINITION

CLASS lcl_application IMPLEMENTATION.

  METHOD top.
    PERFORM event_top_of_page USING o_dyndoc_id.
  ENDMETHOD.

ENDCLASS.

START-OF-SELECTION.
  CALL SCREEN 1234.
*&-----*
*&     Module STATUS_1234 OUTPUT
*&-----*
*     text
*-----*
MODULE status_1234 OUTPUT.
  SET PF-STATUS 'STAT'.

  SELECT vbeln, audat, kunnr FROM vbak INTO TABLE @DATA(it_vbak) WHERE vbeln IN
@s_vbeln.
  wa_fcat-fieldname = 'VBELN'.
  wa_fcat-coltext = 'SALES DOC'.
  wa_fcat-col_pos = '1'.
  APPEND wa_fcat TO it_fcat.
  CLEAR wa_fcat.

  wa_fcat-fieldname = 'AUDAT'.
  wa_fcat-coltext = 'DOC DT'.
  wa_fcat-col_pos = '2'.
  APPEND wa_fcat TO it_fcat.
  CLEAR wa_fcat.

  wa_fcat-fieldname = 'KUNNR'.
  wa_fcat-coltext = 'CUSTOMER'.
  wa_fcat-col_pos = '3'.
  APPEND wa_fcat TO it_fcat.
  CLEAR wa_fcat.

```

```

CREATE OBJECT cc
  EXPORTING
    container_name = 'CUSTOM_CONTAINER'.

CREATE OBJECT o_dyndoc_id
  EXPORTING
    style = 'ALV_GRID'.

CREATE OBJECT o_splitter
  EXPORTING
    parent = cc
    rows   = 2
    columns = 1.

CALL METHOD o_splitter->get_container
  EXPORTING
    row      = 1
    column   = 1
  RECEIVING
    container = o_parent_top.

CALL METHOD o_splitter->get_container
  EXPORTING
    row      = 2
    column   = 1
  RECEIVING
    container = o_parent_grid.

CALL METHOD o_splitter->set_row_height
  EXPORTING
    id      = 1
    height  = 6.

CREATE OBJECT gd
  EXPORTING
    i_parent = o_parent_grid.

DATA obj TYPE REF TO lcl_application.
CREATE OBJECT obj.

SET HANDLER obj->top FOR gd.
CALL METHOD gd->set_table_for_first_display
*  EXPORTING
*    is_layout           =
  CHANGING
    it_outtab      = it_vbak
    it_fieldcatalog = it_fcat.

* Processing events
CALL METHOD gd->list_processing_events
  EXPORTING
    i_event_name = 'TOP_OF_PAGE'
    i_dyndoc_id  = o_dyndoc_id.
ENDMODULE.

*-----*
*-----* Module USER_COMMAND_1234 INPUT
*-----*
*-----* text
*-----*

MODULE user_command_1234 INPUT.
  IF sy-ucomm = 'BACK'.

```

```

        LEAVE TO SCREEN 0.
ENDIF.
ENDMODULE.

FORM event_top_of_page USING dg_dyndoc_id TYPE REF TO cl_dd_document.
DATA :
  dl_text(255) TYPE c.  "Text

  CALL METHOD dg_dyndoc_id->add_text
    EXPORTING
      text      = 'Sales Details'
      sap_style = cl_dd_area=>heading
      sap_fontsize = cl_dd_area=>small
*      sap_emphasis = cl_dd_area=>heading
      sap_color   = cl_dd_area=>list_heading_int.

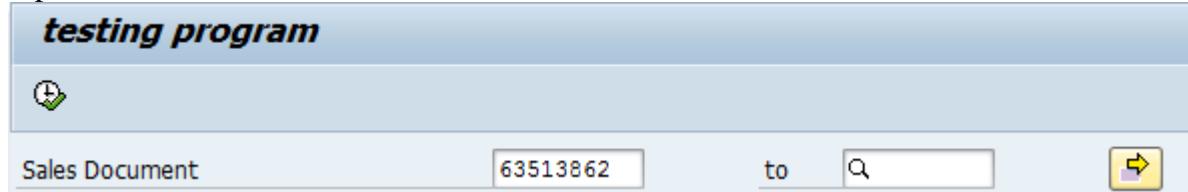
  CALL METHOD dg_dyndoc_id->add_gap
    EXPORTING
      width = 100.

  PERFORM display.
ENDFORM.

FORM display.
* Creating html control
  IF o_html_cntrl IS INITIAL.
    CREATE OBJECT o_html_cntrl
      EXPORTING
        parent = o_parent_top.
  ENDIF.
  CALL METHOD o_dyndoc_id->merge_document.
  o_dyndoc_id->html_control = o_html_cntrl.
* Display document
  CALL METHOD o_dyndoc_id->display_document
    EXPORTING
      reuse_control = 'X'
      parent        = o_parent_top
    EXCEPTIONS
      html_display_error = 1.
  IF sy-subrc NE 0.
    MESSAGE 'Error in displaying top-of-page' TYPE 'I'.
  ENDIF.
ENDFORM.

```

### Input screen



If you execute, then TOP\_OF\_PAGE event will trigger, the output will display as the below screen.

Sales Details		
SALES DOC	DOC DT	CUSTOMER
00635138...	16.10.2...	0000118175

**Note:** - LVC\_FIELDCATALOG\_MERGE is the function module which is used to fill the field catalog internal table. The input for the above function module is data structure which data we want to display (Based on the displayed fields we create structure in SE11). The output for the above function module is field catalog internal table. Instead of passing the structure to the merge function module we can directly pass the structure to SET\_TABLE\_FOR\_FIRST\_DISPLAY method. So we can't use merge function module to prepare the field data

#### **Syntax of call transaction:** -

Call transaction '<TCODE>'.

#### **Syntax of set the value before calling the transaction:** -

Set parameter id '<idname>' field '<value>'.

#### **Steps to identify the parameter id:** -

Execute required transaction (XK03). Place the cursor on input field. Click on F1. clickon technical information. Identify the parameter id.

Parameter p\_lifnr type lfa1-lifnr,

Set parameter id 'LIF' field p\_lifnr.

Call transaction 'XK03'.

#### **Syntax of get the current document which is opened:** -

Get parameter id <ID NAME> field variable.

#### **EX:** -

Data V type EKKO-EBELN.

Get parameter id 'BES' field V.

Write V.

GET & SET are called SAP memory. IMPORT, EXPORT, CHANGING, -- are called ABAP memory.

→ Based on the given sales document numbers, display the sales document numbers, document dates & customer numbers by using OOPS ALV. If the user clicks on any sales document number only then we display the sales order details through VA03 transaction. If the user clicks on any customer number only then we display the customers details through XD03 transaction.

```

TABLES VBAK.
SELECT-OPTIONS S_VBELN FOR VBAK-VBELN.
TYPES: BEGIN OF TY_VBAK,
        VBELN TYPE VBAK-VBELN,
        AUDAT TYPE VBAK-AUDAT,
        KUNNR TYPE VBAK-KUNNR,
        END OF TY_VBAK.
DATA: WA_VBAK TYPE TY_VBAK,
      IT_VBAK TYPE TABLE OF TY_VBAK.
DATA: IT_FCAT TYPE LVC_T_FCAT,
      WA_FCAT LIKE LINE OF IT_FCAT.

```

```

DATA: RC TYPE REF TO CL_GUI_CUSTOM_CONTAINER,
      RG TYPE REF TO CL_GUI_ALV_GRID.

CLASS C2 DEFINITION.
  PUBLIC SECTION.
    METHODS EHDC FOR EVENT DOUBLE_CLICK OF CL_GUI_ALV_GRID
          IMPORTING E_ROW E_COLUMN.
  ENDCLASS.

CLASS C2 IMPLEMENTATION.
  METHOD EHDC.
    READ TABLE IT_VBAK INTO WA_VBAK INDEX E_ROW-INDEX.
    IF E_COLUMN-FIELDNAME = 'VBELN'.
      SET PARAMETER ID 'AUN' FIELD WA_VBAK-VBELN.
      CALL TRANSACTION 'VA03'.
    ELSEIF E_COLUMN-FIELDNAME = 'KUNNR'.
      SET PARAMETER ID 'KUN' FIELD WA_VBAK-KUNNR.
      CALL TRANSACTION 'XD03'.
    ENDIF.
  ENDMETHOD.

ENDCLASS.

START-OF-SELECTION.
  CALL SCREEN 2000.

MODULE STATUS_2000 OUTPUT.
  SET PF-STATUS 'HAI'.
  CREATE OBJECT RC
    EXPORTING
      CONTAINER_NAME = 'CC'.
  CREATE OBJECT RG
    EXPORTING
      I_PARENT = RC.

  SELECT VBELN AUDAT KUNNR FROM VBAK INTO TABLE IT_VBAK WHERE VBELN IN
  S_VBELN.
  WA_FCAT-FIELDNAME = 'VBELN'.
  WA_FCAT-COL_POS = '1'.
  WA_FCAT-COLTEXT = 'SALES.DOC'.
  APPEND WA_FCAT TO IT_FCAT.
  CLEAR WA_FCAT.
  WA_FCAT-FIELDNAME = 'AUDAT'.
  WA_FCAT-COL_POS = '2'.
  WA_FCAT-COLTEXT = 'DATE'.
  APPEND WA_FCAT TO IT_FCAT.
  CLEAR WA_FCAT.
  WA_FCAT-FIELDNAME = 'KUNNR'.
  WA_FCAT-COL_POS = '3'.
  WA_FCAT-COLTEXT = 'CUSTOMER'.
  APPEND WA_FCAT TO IT_FCAT.
  CLEAR WA_FCAT.
  DATA RC2 TYPE REF TO C2.

```

```

CREATE OBJECT RC2 .
SET HANDLER RC2->EHDC FOR RG .

CALL METHOD RG->SET_TABLE_FOR_FIRST_DISPLAY
  CHANGING
    IT_OUTTAB      = IT_VBAK
    IT_FIELDCATALOG = IT_FCAT .
ENDMODULE .

MODULE USER_COMMAND_2000 INPUT .
  IF SY-UCOMM = 'BACK' .
    LEAVE TO SCREEN 0 .
  ENDIF .
ENDMODULE .

```

→ Prepare the SOD Analytics report as below. You have to use HOTSPOT\_CLICK , TOOLBAR , USER\_COMMAND events in this object.

INPUT SCREEN

If you select SOD Analytics radio button the below output will display.

OUTPUT SCREEN

SKU Material	SKU Descri...	Depot	Factory	Bulk	BULK Descr...	Basic mater...	Material Gr...	Material Gr...	M
00000000...	NER SYN C...	D607	F105			REFINISH	1411	NSCV	P
00000000...	NER SYN C...	D814	F105			REFINISH	1411	NSCV	P
00000000...	1K PU INT...	D607	F105			REFINISH	1402	Wood Filler	P
00000000...	NEROLAC ...	B430	F430			AUTO	1600	INDUSTRIA...	N
00000000...	TSA BC PR...	D932	F430			AUTO	1660	STOVINGT...	P

If you click on PRODUCTION button the below screen will display

SKU Material	SKU Descri...	Factory	Bulk	BULK Descr...	Basic mater...	Material Gr...	Material Gr...	TOTAL LA...
00000000...	NER SYN C...	F105			REFINISH	1411	NSCV	2.800
00000000...	1K PU INT...	F105			REFINISH	1402	Wood Filler	0.200
00000000...	NEROLAC ...	F430			AUTO	1600	INDUSTRIA...	20.000
00000000	TSA BC PR	F430			AUTO	1660	STOVINGT	0.000

If you select BULK Strategy radio button the below output will display.

SKU Material	SKU Descri..	Depot	Factory	Bulk	BULK Descr..	Basic mater..	TOTAL LA..	TOTAL LA..	Frequency	Bulk Streate..	Check	STATUS	Comment
00000000..	NER SYN C..	D607	F105		REFINISH	0.000	0.600		1		<input type="checkbox"/>	APPROVED	✓
00000000..	NER SYN C..	D814	F105		REFINISH	2.200	2.200		2		<input type="checkbox"/>	APPROVED	✓
00000000..	1K PU INT..	D607	F105		REFINISH	0.000	0.200		1		<input type="checkbox"/>	APPROVED	✓
00000000..	NEROLAC ...	B430	F430		AUTO	0.000	20.000		1		<input type="checkbox"/>	APPROVED	✓
00000000..	TSA BC PR..	D932	F430		AUTO	0.000	0.000		0		<input type="checkbox"/>	APPROVED	✓

If you select the check box (no 3) then status field will be in enable mode. For that Status field we have to apply one drop down button. In that dropdown button we have to put some values. If we click on Save button whatever the data changed in the ALV screen those data have to save in the database.

This document is used to learn about events (HOTSPOT\_CLICK , TOOLBAR , USER\_COMMAND ) by using oops and how to provide drop down to the output fields.

REPORT zsod\_analytics .

```

INCLUDE zsod_analytics_dec.          "DECLARATION
INCLUDE zsod_analytics_ss.          "SELECTION SCREEN.
INCLUDE zsod_local_class.          "LOCAL CLASS

AT SELECTION-SCREEN.
  PERFORM at_sel_scr.

START-OF-SELECTION.
  CALL SCREEN 1234.

*-----*
*&      Module   STATUS_1234    OUTPUT
*-----*
*      text
*-----*

MODULE status_1234 OUTPUT.
  SET PF-STATUS 'STAT'.

CREATE OBJECT cc
  EXPORTING
    container_name           = 'CUSTOM_CONTAINER'
  EXCEPTIONS
    cntl_error                = 1
    cntl_system_error          = 2
    create_error               = 3
    lifetime_error             = 4
    lifetime_dynpro_dynpro_link = 5
    OTHERS                     = 6.
  IF sy-subrc <> 0.
    MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
      WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
  ENDIF.

CREATE OBJECT gd
  EXPORTING
    i_parent                 = cc
  EXCEPTIONS
    error_cntl_create        = 1
    error_cntl_init           = 2
    error_cntl_link           = 3
    error_dp_create            = 4
    OTHERS                     = 5.
  IF sy-subrc <> 0.
    MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno

```

```

        WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
ENDIF.

INCLUDE zsod_analytics_fd.                                     "FETCH DATA

DATA rc2 TYPE REF TO event_class.
CREATE OBJECT rc2.

IF sod_an = 'X'.
  SET HANDLER rc2->toolbar FOR gd.
  SET HANDLER rc2->uc FOR gd.

  CALL METHOD gd->set_table_for_first_display
    EXPORTING
      is_layout                      = wa_layout1
    CHANGING
      it_outtab                      = it_final
      it_fieldcatalog                = it_fcat
    EXCEPTIONS
      invalid_parameter_combination = 1
      program_error                  = 2
      too_many_lines                 = 3
      OTHERS                         = 4.

ELSE.

  SET HANDLER rc2->ehm FOR gd.
  SET HANDLER rc2->toolbar_save FOR gd.
  SET HANDLER rc2->uc FOR gd.

  CALL METHOD gd->set_table_for_first_display
    EXPORTING
      is_layout                      = wa_layout
    CHANGING
      it_outtab                      = it_final_bulk_st
      it_fieldcatalog                = it_fcat
    EXCEPTIONS
      invalid_parameter_combination = 1
      program_error                  = 2
      too_many_lines                 = 3
      OTHERS                         = 4.

ENDIF.

ENDMODULE.

*&-----*
*&     Module   USER_COMMAND_1234  INPUT
*&-----*
*     text
*-----*
MODULE user_command_1234 INPUT.
  IF sy-ucomm = 'BACK'.
    LEAVE TO SCREEN 0.
  ENDIF.
ENDMODULE.

*&-----*
*&     Form   AT_SEL_SCR
*&-----*
*     text
*-----*
* --> p1      text

```

```

*   <-- p2          text
*-----*
FORM at_sel_scr .
  IF s_wrkst-low IS NOT INITIAL.
    IF s_wrkst-low = 'DECO'
    OR s_wrkst-low = 'deco'
    OR s_wrkst-low = 'Deco'
    OR s_wrkst-low = 'SOLDIERS'
    OR s_wrkst-low = 'soldiers'
    OR s_wrkst-low = 'Soldiers' .
      MESSAGE 'Please give inputs other than DECO & SOLDIERS' TYPE 'E'.
    ENDIF.
  ENDIF.

  IF s_wrkst-high IS NOT INITIAL.
    IF s_wrkst-high = 'DECO'
    OR s_wrkst-high = 'deco'
    OR s_wrkst-high = 'Deco'
    OR s_wrkst-high = 'SOLDIERS'
    OR s_wrkst-high = 'soldiers'
    OR s_wrkst-high = 'Soldiers' .
      MESSAGE 'Please give inputs other than DECO & SOLDIERS' TYPE 'E'.
    ENDIF.
  ENDIF.
ENDFORM.

*-----*
*&-----*
*&  Include           ZSOD_ANALYTICS_DEC
*&-----*

DATA: cc          TYPE REF TO cl_gui_custom_container,
      gd          TYPE REF TO cl_gui_alv_grid,
      g_grid      TYPE REF TO cl_gui_alv_grid,
      o_splitter  TYPE REF TO cl_gui_splitter_container,
      o_parent_grid  TYPE REF TO cl_gui_container,
      o_parent_top  TYPE REF TO cl_gui_container,
      o_html_cntrl  TYPE REF TO cl_gui_html_viewer,
      o_dyndoc_id  TYPE REF TO cl_dd_document.

TABLES mara.
***** types declaration *****
TYPES : BEGIN OF ty_ce1gnpl,
         matnr TYPE mara-matnr,
         werks TYPE marc-werks,
         vvvvol TYPE ce1gnpl-absmsg,
       END OF ty_ce1gnpl.

TYPES: BEGIN OF ty_final_bulk_st,
        mandt          TYPE mandt,
        matnr          TYPE mara-matnr,
        werks          TYPE marc-werks,
        factory        TYPE werks_d,
        maktx          TYPE makt-maktx,
        bulk           TYPE zupdate_table-idnrk,
        bulk_des       TYPE maktx,
        wrkst          TYPE wrkst,
        last_15m_sales TYPE ce1gnpl-vvvvol,
        last_3m_sales  TYPE ce1gnpl-vvvvol,
        bulk_strategy(20) TYPE c,
        frequency      TYPE i,
        check(1)        TYPE c,
        status(15)      TYPE c,
      END OF ty_final_bulk_st.

```

```

comment(25)          TYPE c,
field_style          TYPE lvc_t_styl,
END OF ty_final_bulk_st.

TYPES : BEGIN OF ty_ce1gnpl1,
    matnr  TYPE mara-matnr,
    werks  TYPE marc-werks,
    budat  TYPE celgnpl-budat,
    vvvvol  TYPE celgnpl-absmsg,
END OF ty_ce1gnpl1.

TYPES : BEGIN OF ty_ce1gnpl2,
    matnr  TYPE mara-matnr,
    werks  TYPE marc-werks,
    month(3)  TYPE c,
    vvvvol  TYPE celgnpl-absmsg,
    freq   TYPE i,
END OF ty_ce1gnpl2.

TYPES: BEGIN OF ty_zscpc_db_fdp_new,
    matnr  TYPE mara-matnr,
    werks  TYPE marc-werks,
    cb_dis  TYPE zscpc_db_fdp_new-cb_dis,
END OF ty_zscpc_db_fdp_new.

TYPES: BEGIN OF ty_zscpc_db_fdp_new1,
    matnr  TYPE mara-matnr,
    werks  TYPE marc-werks,
    budat_mkpf  TYPE zscpc_db_fdp_new-budat_mkpf,
    cb_dis  TYPE zscpc_db_fdp_new-cb_dis,
END OF ty_zscpc_db_fdp_new1.

TYPES: BEGIN OF ty_zscpc_db_fdp_new2,
    matnr  TYPE mara-matnr,
    werks  TYPE marc-werks,
    month(3)  TYPE c,
    cb_dis  TYPE zscpc_db_fdp_new-cb_dis,
END OF ty_zscpc_db_fdp_new2.

TYPES: BEGIN OF ty_mseg,
    matnr  TYPE mara-matnr,
    werks  TYPE marc-werks,
    menge  TYPE mseg-menage,
END OF ty_mseg.

TYPES: BEGIN OF ty_mseg1,
    matnr  TYPE mara-matnr,
    werks  TYPE marc-werks,
    budat_mkpf  TYPE budat,
    menge  TYPE mseg-menage,
END OF ty_mseg1.

TYPES: BEGIN OF ty_mseg2,
    matnr  TYPE mara-matnr,
    werks  TYPE marc-werks,
    month(3)  TYPE c,
    menge  TYPE mseg-menage,
END OF ty_mseg2.

TYPES: BEGIN OF ty_final,
    matnr          TYPE mara-matnr,

```

```

maktx          TYPE makt-maktx,
matkl          TYPE mara-matkl,
wgbez          TYPE t023t-wgbez,
wrkst          TYPE mara-wrkst,
werks          TYPE marc-werks,
last_15m_sales TYPE celgnpl-vvvol,
m1             TYPE celgnpl-vvvol,
m2             TYPE celgnpl-vvvol,
m3             TYPE celgnpl-vvvol,
m4             TYPE celgnpl-vvvol,
m5             TYPE celgnpl-vvvol,
m6             TYPE celgnpl-vvvol,
m7             TYPE celgnpl-vvvol,
m8             TYPE celgnpl-vvvol,
m9             TYPE celgnpl-vvvol,
m10            TYPE celgnpl-vvvol,
m11            TYPE celgnpl-vvvol,
m12            TYPE celgnpl-vvvol,
m13            TYPE celgnpl-vvvol,
m14            TYPE celgnpl-vvvol,
m15            TYPE celgnpl-vvvol,
sales_frequency TYPE i,
avg_sales      TYPE celgnpl-vvvol,
mat_group_flag(15) TYPE c,
sales_frequency_flag(10) TYPE c,
bulk_strategy(20)  TYPE c,
avg_sales_band(5)  TYPE c,
factory         TYPE werks_d,
bulk            TYPE zupdate_table-idnrk,
bulk_des        TYPE maktx,
last_15m_prd   TYPE zscpc_db_fdp_new-cb_dis,
prd_frequency  TYPE i,
avg_prd         TYPE zscpc_db_fdp_new-cb_dis,
pr_m1           TYPE zscpc_db_fdp_new-cb_dis,
pr_m2           TYPE zscpc_db_fdp_new-cb_dis,
pr_m3           TYPE zscpc_db_fdp_new-cb_dis,
pr_m4           TYPE zscpc_db_fdp_new-cb_dis,
pr_m5           TYPE zscpc_db_fdp_new-cb_dis,
pr_m6           TYPE zscpc_db_fdp_new-cb_dis,
pr_m7           TYPE zscpc_db_fdp_new-cb_dis,
pr_m8           TYPE zscpc_db_fdp_new-cb_dis,
pr_m9           TYPE zscpc_db_fdp_new-cb_dis,
pr_m10          TYPE zscpc_db_fdp_new-cb_dis,
pr_m11          TYPE zscpc_db_fdp_new-cb_dis,
pr_m12          TYPE zscpc_db_fdp_new-cb_dis,
pr_m13          TYPE zscpc_db_fdp_new-cb_dis,
pr_m14          TYPE zscpc_db_fdp_new-cb_dis,
pr_m15          TYPE zscpc_db_fdp_new-cb_dis,
pr_m1_band(5)   TYPE c,
pr_m2_band(5)   TYPE c,
pr_m3_band(5)   TYPE c,
pr_m4_band(5)   TYPE c,
pr_m5_band(5)   TYPE c,
pr_m6_band(5)   TYPE c,
pr_m7_band(5)   TYPE c,
pr_m8_band(5)   TYPE c,
pr_m9_band(5)   TYPE c,
pr_m10_band(5)  TYPE c,
pr_m11_band(5)  TYPE c,
pr_m12_band(5)  TYPE c,
pr_m13_band(5)  TYPE c,

```

```

pr_m14_band(5)          TYPE c,
pr_m15_band(5)          TYPE c,
END OF ty_final.

***** internal tables & work area declarations
DATA: it_ce1gnpl          TYPE STANDARD TABLE OF ty_ce1gnpl,
      it_ce1gnpl1         TYPE STANDARD TABLE OF ty_ce1gnpl1,
      it_ce1gnpl2         TYPE STANDARD TABLE OF ty_ce1gnpl2,
      wa_ce1gnpl          TYPE ty_ce1gnpl,
      wa_ce1gnpl1         TYPE ty_ce1gnpl1,
      wa_ce1gnpl2         TYPE ty_ce1gnpl2,
      it_final             TYPE STANDARD TABLE OF ty_final,
      it_final_prd         TYPE STANDARD TABLE OF ty_final,
      it_final_prd1        TYPE STANDARD TABLE OF ty_final,
      wa_final             TYPE ty_final,
      wa_final_prd         TYPE ty_final,
      wa_final_prd1        TYPE ty_final,
      it_zscpc_db_fdp_new  TYPE STANDARD TABLE OF ty_zscpc_db_fdp_new,
      it_zscpc_db_fdp_new1 TYPE STANDARD TABLE OF ty_zscpc_db_fdp_new1,
      it_zscpc_db_fdp_new2 TYPE STANDARD TABLE OF ty_zscpc_db_fdp_new2,
      wa_zscpc_db_fdp_new  TYPE ty_zscpc_db_fdp_new,
      wa_zscpc_db_fdp_new1 TYPE ty_zscpc_db_fdp_new1,
      wa_zscpc_db_fdp_new2 TYPE ty_zscpc_db_fdp_new2.

DATA: wa_final_bulk_st    TYPE ty_final_bulk_st,
      it_final_bulk_st    TYPE STANDARD TABLE OF ty_final_bulk_st,
      wa_final_bulk_st1   TYPE zsod_analytics.

DATA ls_stylerow TYPE lvc_s_styl .                                "WORKAREA
DATA lt_styletab TYPE TABLE OF lvc_s_styl . "LVC_T_STYL .       "INTERNAL TABLE

DATA: it_fcat      TYPE lvc_t_fcat,
      it_fcat_prd  TYPE lvc_t_fcat,
      wa_fcat       TYPE lvc_s_fcat,
      wa_fcat_prd  TYPE lvc_s_fcat.

DATA: it_mseg      TYPE STANDARD TABLE OF ty_mseg,
      it_mseg1     TYPE STANDARD TABLE OF ty_mseg1,
      it_mseg2     TYPE STANDARD TABLE OF ty_mseg2,
      wa_mseg      TYPE ty_mseg,
      wa_mseg1     TYPE ty_mseg1,
      wa_mseg2     TYPE ty_mseg2.

***** variables declaration

DATA: month1       TYPE sy-datum,
      month2       TYPE sy-datum,
      month3       TYPE sy-datum,
      month4       TYPE sy-datum,
      month5       TYPE sy-datum,
      month6       TYPE sy-datum,
      month7       TYPE sy-datum,
      month8       TYPE sy-datum,
      month9       TYPE sy-datum,
      month10      TYPE sy-datum,
      month11      TYPE sy-datum,
      month12      TYPE sy-datum,
      month13      TYPE sy-datum,
      month14      TYPE sy-datum,
      month15      TYPE sy-datum,
      count_m1     TYPE i,

```

```

count_m2      TYPE i,
count_m3      TYPE i,
count_m4      TYPE i,
count_m5      TYPE i,
count_m6      TYPE i,
count_m7      TYPE i,
count_m8      TYPE i,
count_m9      TYPE i,
count_m10     TYPE i,
count_m11     TYPE i,
count_m12     TYPE i,
count_m13     TYPE i,
count_m14     TYPE i,
count_m15     TYPE i,
sno          TYPE i,
sno_prd      TYPE i,
sales_m15(20) TYPE c,
sales_m14(20) TYPE c,
sales_m13(20) TYPE c,
sales_m12(20) TYPE c,
sales_m11(20) TYPE c,
sales_m10(20) TYPE c,
sales_m9(20)  TYPE c,
sales_m8(20)  TYPE c,
sales_m7(20)  TYPE c,
sales_m6(20)  TYPE c,
sales_m5(20)  TYPE c,
sales_m4(20)  TYPE c,
sales_m3(20)  TYPE c,
sales_m2(20)  TYPE c,
sales_m1(20)  TYPE c,
count_pr_m1   TYPE i,
count_pr_m2   TYPE i,
count_pr_m3   TYPE i,
count_pr_m4   TYPE i,
count_pr_m5   TYPE i,
count_pr_m6   TYPE i,
count_pr_m7   TYPE i,
count_pr_m8   TYPE i,
count_pr_m9   TYPE i,
count_pr_m10  TYPE i,
count_pr_m11  TYPE i,
count_pr_m12  TYPE i,
count_pr_m13  TYPE i,
count_pr_m14  TYPE i,
count_pr_m15  TYPE i,
prd(15)       TYPE c,
factory       TYPE werks_d.

DATA: v_first_day    TYPE sy-datum,
      v_last_day     TYPE sy-datum,
      lv_month(2)    TYPE c,
      lv_month2(2)   TYPE c,
      lv_month3(2)   TYPE c,
      lv_month4(2)   TYPE c,
      lv_month5(2)   TYPE c,
      lv_month6(2)   TYPE c,
      lv_month7(2)   TYPE c,
      lv_month8(2)   TYPE c,
      lv_month9(2)   TYPE c,
      lv_month10(2)  TYPE c,

```

```

lv_month11(2) TYPE c,
lv_month12(2) TYPE c,
lv_month13(2) TYPE c,
lv_month14(2) TYPE c,
lv_month15(2) TYPE c,
lv_year(4) TYPE c,
lv_last_month TYPE sy-datum.

*-----*
*&-----*
*& Include          ZSOD_ANALYTICS_SS
*&-----*


SELECTION-SCREEN BEGIN OF BLOCK A WITH FRAME TITLE TEXT-001.
  SELECT-OPTIONS: S_MATNR FOR MARA-MATNR,
                  S_WRKST FOR MARA-WRKST.
  SELECTION-SCREEN skip 2.
  parameters: SOD_AN RADIobutton GROUP a,
              BULK_ST RADIobutton GROUP a.
SELECTION-SCREEN END OF BLOCK A.

*-----*
*&-----*
*& Include          ZSOD_LOCAL_CLASS
*&-----*


CLASS event_class DEFINITION.
  PUBLIC SECTION.
    DATA: l_valid      TYPE c,
          c_refresh   TYPE c,
          c_refres    TYPE c VALUE 'X'.

  METHODS: ehm
            FOR EVENT hotspot_click OF cl_gui_alv_grid
            IMPORTING e_row_id e_column_id,

            toolbar
              FOR EVENT toolbar OF cl_gui_alv_grid
              IMPORTING e_object e_interactive,

            uc
              FOR EVENT user_command OF cl_gui_alv_grid
              IMPORTING e_ucomm,

            toolbar_save
              FOR EVENT toolbar OF cl_gui_alv_grid
              IMPORTING e_object e_interactive.
ENDCLASS.

CLASS event_class IMPLEMENTATION.

  METHOD ehm.
    CALL METHOD gd->check_changed_data
      IMPORTING
        e_valid    = l_valid
      CHANGING
        c_refresh = c_refres.

    READ TABLE it_final_bulk_st INTO wa_final_bulk_st INDEX e_row_id-index.
    IF sy-subrc = 0.
      IF wa_final_bulk_st-
check = ' '." The value for checkbox will not be stored with hotspot link, th
ats why the check on the previous value.
      wa_final_bulk_st-check = 'X'.
  ENDMETHOD.

```

```

ls_stylerow-fieldname = 'STATUS'.
ls_stylerow-style = cl_gui_alv_grid->mc_style_enabled.
"set field to disabled
DELETE wa_final_bulk_st-field_style WHERE fieldname = 'STATUS'.
APPEND ls_stylerow TO wa_final_bulk_st-field_style.
MODIFY it_final_bulk_st FROM wa_final_bulk_st INDEX e_row_id-index.
CLEAR ls_stylerow.
ELSE.
CLEAR wa_final_bulk_st-check.
ls_stylerow-fieldname = 'STATUS'.
ls_stylerow-style = cl_gui_alv_grid->mc_style_disabled.
DELETE wa_final_bulk_st-field_style WHERE fieldname = 'STATUS'.
APPEND ls_stylerow TO wa_final_bulk_st-field_style.
MODIFY it_final_bulk_st FROM wa_final_bulk_st INDEX e_row_id-index.
CLEAR ls_stylerow.
ENDIF.
ENDIF.

CALL METHOD gd->refresh_table_display.
ENDMETHOD.

METHOD toolbar.
DATA: ls_toolbar TYPE stb_button.
* append a separator to normal toolbar
CLEAR ls_toolbar.
MOVE 3 TO ls_toolbar-butn_type.
APPEND ls_toolbar TO e_object->mt_toolbar.
* append an icon to show booking table
CLEAR ls_toolbar.
MOVE 'PRODUCTION' TO ls_toolbar-function.
* MOVE icon_employee TO ls_toolbar-icon.
MOVE 'Production Details'(111) TO ls_toolbar-quickinfo.
MOVE 'PRODUCTION'(112) TO ls_toolbar-text.
MOVE '' TO ls_toolbar-disabled.
APPEND ls_toolbar TO e_object->mt_toolbar.
ENDMETHOD.

METHOD toolbar_save.
DATA: ls_toolbar1 TYPE stb_button.
* append a separator to normal toolbar
CLEAR ls_toolbar1.
MOVE 3 TO ls_toolbar1-butn_type.
APPEND ls_toolbar1 TO e_object->mt_toolbar.
* append an icon to show booking table
CLEAR ls_toolbar1.
MOVE 'SAVE' TO ls_toolbar1-function.
* MOVE icon_employee TO ls_toolbar-icon.
MOVE 'SAVE'(111) TO ls_toolbar1-quickinfo.
MOVE 'SAVE'(112) TO ls_toolbar1-text.
MOVE '' TO ls_toolbar1-disabled.
APPEND ls_toolbar1 TO e_object->mt_toolbar.
ENDMETHOD.

METHOD uc.
CASE e_ucomm.
WHEN 'PRODUCTION'.
CALL SCREEN 1235.
WHEN 'SAVE'.

CALL METHOD gd->check_changed_data.

```

```

CLEAR : wa_final_bulk_st1, wa_final_bulk_st.
LOOP AT it_final_bulk_st INTO wa_final_bulk_st.
  wa_final_bulk_st1-matnr = wa_final_bulk_st-matnr.
  wa_final_bulk_st1-werks = wa_final_bulk_st-werks.
  wa_final_bulk_st1-factory = wa_final_bulk_st-factory.
  wa_final_bulk_st1-maktx = wa_final_bulk_st-maktx.
  wa_final_bulk_st1-bulk1 = wa_final_bulk_st-bulk.
  wa_final_bulk_st1-bulk_des = wa_final_bulk_st-bulk_des.
  wa_final_bulk_st1-wrkst = wa_final_bulk_st-wrkst.
  wa_final_bulk_st1-last_15m_sales = wa_final_bulk_st1-last_15m_sales.
  wa_final_bulk_st1-last_3m_sales = wa_final_bulk_st-last_3m_sales.
  wa_final_bulk_st1-bulk_strategy = wa_final_bulk_st-bulk_strategy.
  wa_final_bulk_st1-frequency = wa_final_bulk_st-frequency.
  wa_final_bulk_st1-check1 = wa_final_bulk_st-check.
  wa_final_bulk_st1-status1 = wa_final_bulk_st-status.
  wa_final_bulk_st1-comment1 = wa_final_bulk_st-comment.
  MODIFY zsod_analytics FROM wa_final_bulk_st1.
  CLEAR wa_final_bulk_st1.
ENDLOOP.

ENDCASE.
ENDMETHOD.
ENDCLASS.

*&-----*
*&      Module   STATUS_1235   OUTPUT
*&-----*
*      text
*-----*

MODULE status_1235 OUTPUT.
SET PF-STATUS 'STAT_2'.
REFRESH it_final_prd1.
it_final_prd1 = it_final.
SORT it_final_prd1 BY matnr factory.
SORT it_final_prd BY matnr factory.
DELETE ADJACENT DUPLICATES FROM it_final_prd1 COMPARING matnr factory.
LOOP AT it_final_prd1 INTO wa_final_prd1.
  wa_final_prd-matnr = wa_final_prd1-matnr .
  wa_final_prd-maktx = wa_final_prd1-maktx .
  wa_final_prd-matk1 = wa_final_prd1-matk1 .
  wa_final_prd-wgbez = wa_final_prd1-wgbez .
  wa_final_prd-wrkst = wa_final_prd1-wrkst .
  wa_final_prd-factory = wa_final_prd1-factory .
  wa_final_prd-bulk = wa_final_prd1-bulk .
  wa_final_prd-bulk_des = wa_final_prd1-bulk_des .
  wa_final_prd-last_15m_prd = wa_final_prd1-last_15m_prd .
  wa_final_prd-prd_frequency = wa_final_prd1-
pr_frequency .
  wa_final_prd-avg_prd = wa_final_prd1-avg_prd .
  wa_final_prd-pr_m1 = wa_final_prd1-pr_m1 .
  wa_final_prd-pr_m2 = wa_final_prd1-pr_m2 .
  wa_final_prd-pr_m3 = wa_final_prd1-pr_m3 .
  wa_final_prd-pr_m4 = wa_final_prd1-pr_m4 .
  wa_final_prd-pr_m5 = wa_final_prd1-pr_m5 .
  wa_final_prd-pr_m6 = wa_final_prd1-pr_m6 .
  wa_final_prd-pr_m7 = wa_final_prd1-pr_m7 .
  wa_final_prd-pr_m8 = wa_final_prd1-pr_m8 .
  wa_final_prd-pr_m9 = wa_final_prd1-pr_m9 .
  wa_final_prd-pr_m10 = wa_final_prd1-pr_m10 .
  wa_final_prd-pr_m11 = wa_final_prd1-pr_m11 .
  wa_final_prd-pr_m12 = wa_final_prd1-pr_m12 .

```

```

wa_final_prd-pr_m13 = wa_final_prd1-pr_m13 .
wa_final_prd-pr_m14 = wa_final_prd1-pr_m14 .
wa_final_prd-pr_m15 = wa_final_prd1-pr_m15 .
wa_final_prd-pr_m1_band = wa_final_prd1-pr_m1_band .
wa_final_prd-pr_m2_band = wa_final_prd1-pr_m2_band .
wa_final_prd-pr_m3_band = wa_final_prd1-pr_m3_band .
wa_final_prd-pr_m4_band = wa_final_prd1-pr_m4_band .
wa_final_prd-pr_m5_band = wa_final_prd1-pr_m5_band .
wa_final_prd-pr_m6_band = wa_final_prd1-pr_m6_band .
wa_final_prd-pr_m7_band = wa_final_prd1-pr_m7_band .
wa_final_prd-pr_m8_band = wa_final_prd1-pr_m8_band .
wa_final_prd-pr_m9_band = wa_final_prd1-pr_m9_band .
wa_final_prd-pr_m10_band = wa_final_prd1-pr_m10_band .
wa_final_prd-pr_m11_band = wa_final_prd1-pr_m11_band .
wa_final_prd-pr_m12_band = wa_final_prd1-pr_m12_band .
wa_final_prd-pr_m13_band = wa_final_prd1-pr_m13_band .
wa_final_prd-pr_m14_band = wa_final_prd1-pr_m14_band .
wa_final_prd-pr_m15_band = wa_final_prd1-pr_m15_band .

LOOP AT it_final INTO wa_final WHERE matnr = wa_final_prd-
matnr AND factory = wa_final_prd-factory .
wa_final_prd-last_15m_sales = wa_final-last_15m_sales + wa_final_prd-
last_15m_sales .
wa_final_prd-m1 = wa_final-m1 + wa_final_prd-m1 .
wa_final_prd-m2 = wa_final-m2 + wa_final_prd-m2 .
wa_final_prd-m3 = wa_final-m3 + wa_final_prd-m3 .
wa_final_prd-m4 = wa_final-m4 + wa_final_prd-m4 .
wa_final_prd-m5 = wa_final-m5 + wa_final_prd-m5 .
wa_final_prd-m6 = wa_final-m6 + wa_final_prd-m6 .
wa_final_prd-m7 = wa_final-m7 + wa_final_prd-m7 .
wa_final_prd-m8 = wa_final-m8 + wa_final_prd-m8 .
wa_final_prd-m9 = wa_final-m9 + wa_final_prd-m9 .
wa_final_prd-m10 = wa_final-m10 + wa_final_prd-m10 .
wa_final_prd-m11 = wa_final-m11 + wa_final_prd-m11 .
wa_final_prd-m12 = wa_final-m12 + wa_final_prd-m12 .
wa_final_prd-m13 = wa_final-m13 + wa_final_prd-m13 .
wa_final_prd-m14 = wa_final-m14 + wa_final_prd-m14 .
wa_final_prd-m15 = wa_final-m15 + wa_final_prd-m15 .
wa_final_prd-sales_frequency = wa_final-sales_frequency + wa_final_prd-
sales_frequency .
wa_final_prd-avg_sales_band = wa_final_prd-avg_sales_band + wa_final-
avg_sales_band .
CLEAR wa_final .
ENDLOOP .
IF wa_final_prd-sales_frequency NE 0 .
wa_final_prd-avg_sales = wa_final_prd-last_15m_sales / wa_final_prd-
sales_frequency .
ENDIF .

APPEND wa_final_prd TO it_final_prd .
CLEAR wa_final_prd .
ENDLOOP .

sno_prd = sno_prd + 1 .
wa_fcat_prd-fieldname = 'MATNR' .
wa_fcat_prd-coltext = 'SKU Material' .
wa_fcat_prd-col_pos = sno_prd .
APPEND wa_fcat_prd TO it_fcat_prd .
CLEAR wa_fcat_prd .

sno_prd = sno_prd + 1 .

```

```

wa_fcat_prd-fieldname = 'MAKTX'.
wa_fcat_prd-coltext = 'SKU Description'.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'FACTORY'.
wa_fcat_prd-coltext = 'Factory'.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'BULK'.
wa_fcat_prd-coltext = 'Bulk'.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'BULK DES'.
wa_fcat_prd-coltext = 'BULK Description'.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'WRKST'.
wa_fcat_prd-coltext = 'Basic material (MBD)'.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'MATKL'.
wa_fcat_prd-coltext = 'Material Group'.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'WGBEZ'.
wa_fcat_prd-coltext = 'Material Group Des'.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'LAST_15M_SALES'.
wa_fcat_prd-coltext = 'TOTAL LAST 15M Sales Volume'.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'SALES_FREQUENCY'.
wa_fcat_prd-coltext = 'Sales frequency'.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.

```

```

CLEAR wa_fcat_prd.

sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'AVG_SALES'.
wa_fcat_prd-coltext = 'Average sales'.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'SALES' month15+4(2) '.' month15+0(4) INTO sales_m15.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'M15'.
wa_fcat_prd-coltext = sales_m15.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'SALES' month14+4(2) '.' month14+0(4) INTO sales_m14.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'M14'.
wa_fcat_prd-coltext = sales_m14.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'SALES' month13+4(2) '.' month13+0(4) INTO sales_m13.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'M13'.
wa_fcat_prd-coltext = sales_m13.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'SALES' month12+4(2) '.' month12+0(4) INTO sales_m12.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'M12'.
wa_fcat_prd-coltext = sales_m12.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'SALES' month11+4(2) '.' month11+0(4) INTO sales_m11.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'M11'.
wa_fcat_prd-coltext = sales_m11.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'SALES' month10+4(2) '.' month10+0(4) INTO sales_m10.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'M10'.
wa_fcat_prd-coltext = sales_m10.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'SALES' month9+4(2) '.' month9+0(4) INTO sales_m9.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'M9'.
wa_fcat_prd-coltext = sales_m9.

```

```

wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'SALES' month8+4 (2) '.' month8+0 (4) INTO sales_m8.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'M8'.
wa_fcat_prd-coltext = sales_m8.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'SALES' month7+4 (2) '.' month7+0 (4) INTO sales_m7.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'M7'.
wa_fcat_prd-coltext = sales_m7.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'SALES' month6+4 (2) '.' month6+0 (4) INTO sales_m6.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'M6'.
wa_fcat_prd-coltext = sales_m6.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'SALES' month5+4 (2) '.' month5+0 (4) INTO sales_m5.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'M5'.
wa_fcat_prd-coltext = sales_m5.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'SALES' month4+4 (2) '.' month4+0 (4) INTO sales_m4.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'M4'.
wa_fcat_prd-coltext = sales_m4.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'SALES' month3+4 (2) '.' month3+0 (4) INTO sales_m3.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'M3'.
wa_fcat_prd-coltext = sales_m3.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'SALES' month2+4 (2) '.' month2+0 (4) INTO sales_m2.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'M2'.
wa_fcat_prd-coltext = sales_m2.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'SALES' month1+4 (2) '.' month1+0 (4) INTO sales_m1.

```

```

sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'M1'.
wa_fcat_prd-coltext = sales_m1.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'AVG_SALES_BAND'.
wa_fcat_prd-coltext = 'Average sales BAND'.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'LAST_15M_PRD'.
wa_fcat_prd-coltext = 'Total L15 Production Volume '.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PRD_FREQUENCY'.
wa_fcat_prd-coltext = 'Production frequency '.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'AVG_PRD'.
wa_fcat_prd-coltext = 'Average Production'.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'PRD' month15+4(2) '.' month15+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M15'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'PRD' month14+4(2) '.' month14+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M14'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'PRD' month13+4(2) '.' month13+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M13'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'PRD' month12+4(2) '.' month12+0(4) INTO prd.
sno_prd = sno_prd + 1.

```

```

wa_fcat_prd-fieldname = 'PR_M12'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'PRD' month11+4(2) '.' month11+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M11'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'PRD' month10+4(2) '.' month10+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M10'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'PRD' month9+4(2) '.' month9+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M9'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'PRD' month8+4(2) '.' month8+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M8'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'PRD' month7+4(2) '.' month7+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M7'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'PRD' month6+4(2) '.' month6+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M6'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'PRD' month5+4(2) '.' month5+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M5'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

```

```

CONCATENATE 'PRD' month4+4(2) '.' month4+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M4'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'PRD' month3+4(2) '.' month3+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M3'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'PRD' month2+4(2) '.' month2+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M2'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'PRD' month1+4(2) '.' month1+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M1'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'Prd BAND' month15+4(2) '.' month15+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M15_BAND'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'Prd BAND' month14+4(2) '.' month14+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M14_BAND'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'Prd BAND' month13+4(2) '.' month13+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M13_BAND'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'Prd BAND' month12+4(2) '.' month12+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M12_BAND'.
wa_fcat_prd-coltext = prd.

```

```

wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'Prd BAND' month11+4(2) '.' month11+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M11_BAND'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd. CLEAR wa_fcat_prd.

CONCATENATE 'Prd BAND' month10+4(2) '.' month10+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M10_BAND'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'Prd BAND' month9+4(2) '.' month9+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M9_BAND'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'Prd BAND' month8+4(2) '.' month8+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M8_BAND'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'Prd BAND' month7+4(2) '.' month7+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M7_BAND'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'Prd BAND' month6+4(2) '.' month6+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M6_BAND'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'Prd BAND' month5+4(2) '.' month5+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M5_BAND'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'Prd BAND' month4+4(2) '.' month4+0(4) INTO prd.
sno_prd = sno_prd + 1.

```

```

wa_fcat_prd-fieldname = 'PR_M4_BAND'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'Prd BAND' month3+4(2) '.' month3+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M3_BAND'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'Prd BAND' month2+4(2) '.' month2+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M2_BAND'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

CONCATENATE 'Prd BAND' month1+4(2) '.' month1+0(4) INTO prd.
sno_prd = sno_prd + 1.
wa_fcat_prd-fieldname = 'PR_M1_BAND'.
wa_fcat_prd-coltext = prd.
wa_fcat_prd-col_pos = sno_prd.
APPEND wa_fcat_prd TO it_fcat_prd.
CLEAR wa_fcat_prd.

DATA: ccl TYPE REF TO cl_gui_custom_container,
      gd1 TYPE REF TO cl_gui_alv_grid.

CREATE OBJECT ccl
  EXPORTING
    container_name = 'CONTAINER1'
EXCEPTIONS
  cntl_error      = 1
  cntl_system_error = 2
  create_error    = 3
  lifetime_error  = 4
  lifetime_dynpro_dynpro_link = 5
  others          = 6
.

IF sy-subrc <> 0.
MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
      WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
ENDIF.

CREATE OBJECT gd1
  EXPORTING
    i_parent = ccl
EXCEPTIONS
  error_cntl_create = 1
  error_cntl_init   = 2
  error_cntl_link   = 3
  error_dp_create   = 4
  others            = 5
.

IF sy-subrc <> 0.
MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO

```

```

        WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
ENDIF.

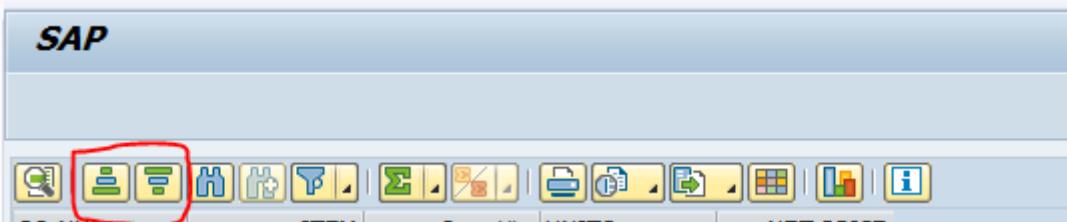
CALL METHOD gd1->set_table_for_first_display
  CHANGING
    it_outtab      = it_final_prd
    it_fieldcatalog = it_fcat_prd
EXCEPTIONS
  invalid_parameter_combination = 1
  program_error      = 2
  too_many_lines     = 3
  others             = 4 .
ENDMODULE.

*&-----*
*&      Module   USER_COMMAND_1235  INPUT
*&-----*
*      text
*-----*

MODULE user_command_1235 INPUT.
  IF sy-ucomm = 'BACK'.
    LEAVE TO SCREEN 0.
  ENDIF.
ENDMODULE.

```

→ Develop a sample report. In that report I don't want to display sorting buttons. (Hide Standard Application tool bar buttons)



```

PROGRAM ztest12345.
DATA: cc TYPE REF TO cl_gui_custom_container,
      gd TYPE REF TO cl_gui_alv_grid.
DATA: it_fcat TYPE lvc_t_fcat,
      wa_fcat LIKE LINE OF it_fcat.

TABLES vbak.
DATA : ls_exclude TYPE ui_func,
      pt_exclude TYPE ui_functions.

SELECTION-SCREEN BEGIN OF BLOCK a WITH FRAME.
SELECT-OPTIONS s_vbeln FOR vbak-vbeln.
SELECTION-SCREEN END OF BLOCK a.

START-OF-SELECTION.
  CALL SCREEN 2000.

```

```

*&-----*
*&      Module   STATUS_2000  OUTPUT
*&-----*
*      text
*-----*

MODULE status_2000 OUTPUT.

```

```

SET PF-STATUS 'STAT'.
* SET TITLEBAR 'STAT'.

CREATE OBJECT cc
EXPORTING
  container_name = 'CUSTOM_CONTAINER'.

CREATE OBJECT gd
EXPORTING
  i_parent = cc .

SELECT vbeln, posnr, kwmeng, meins, netwr FROM vbap INTO TABLE @DATA(it_vbap)
WHERE vbeln IN @s_vbeln.
REFRESH it_fcat.
wa_fcat-fieldname = 'VBELN'.

wa_fcat-col_pos = '1'.
wa_fcat-coltext = 'SO NUM'.
APPEND wa_fcat TO it_fcat.
CLEAR wa_fcat.
wa_fcat-fieldname = 'POSNR'.
wa_fcat-col_pos = '2'.
wa_fcat-coltext = 'ITEM'.
APPEND wa_fcat TO it_fcat.
CLEAR wa_fcat.
wa_fcat-fieldname = 'KWMENG'.
wa_fcat-col_pos = '3'.
wa_fcat-coltext = 'Quantity'.
APPEND wa_fcat TO it_fcat.
CLEAR wa_fcat.
wa_fcat-fieldname = 'MEINS'.
wa_fcat-col_pos = '4'.
wa_fcat-coltext = 'UNITS'.
APPEND wa_fcat TO it_fcat.
CLEAR wa_fcat.
wa_fcat-fieldname = 'NETWR'.
wa_fcat-col_pos = '5'.
wa_fcat-coltext = 'NET PRICE'.
APPEND wa_fcat TO it_fcat.
CLEAR wa_fcat.

-----
ls_exclude = cl_gui_alv_grid->mc_fc_sort_asc .
APPEND ls_exclude TO pt_exclude.
ls_exclude = cl_gui_alv_grid->mc_fc_sort_dsc .
APPEND ls_exclude TO pt_exclude.

CALL METHOD gd->set_table_for_first_display
EXPORTING
  i_save           = 'A'
  i_default        = 'X'
  it_toolbar_excluding = pt_exclude
CHANGING
  it_outtab        = it_vbap
  it_fieldcatalog = it_fcat.

ENDMODULE.

MODULE user_command_2000 INPUT.
  IF sy-ucomm = 'BACK'.
    LEAVE TO SCREEN 0.

```

ENDIF.  
ENDMODULE .

## Working with OOPS BDC

1. Do the recording
2. Prepare the file
3. Upload the data from file to internal table / BDC program
4. For each record in the internal table collect the screen and field details.
5. For each record in internal table, call the transaction.

**Note:** - In this OOPS BDC we need to declare 3 methods.

1. Upload data.
2. Fill screen details
3. Fill field details

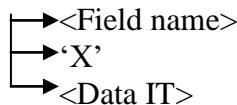
### **Syntax of declaring the method for upload data:** -

Methods <method name> importing <file name> type string.

### **Syntax of implementation:** -

Method <method name>

Call function 'GUI\_UPLOAD'



Endmethod.

### **Declaring the fill screen method:** -

Methods fill\_screen importing

```
I_prog type bdc_prog  
I_dynpro type bdc_dynr  
I_dynbegin type bdc_start.
```

### **Implementation**

Method fill\_screen.

```
Wa_bdcdata-program =      i_prog.  
Wa_bdcdata-dynpro =      i_dynpro.  
Wa_bdcdata-dynbegin =    i_dynbegin  
Append wa_bdcdata to it_bdcdata.
```

Clear wa\_bdcdata.

### **Declaring the fill field method:** -

Method fill\_field importing

```
I_fnam type fnam____4  
I_fval type c.
```

### **Implementation**

Method fill\_field.

```
Wa_bdcdata-fam =      i_fnam.  
Wa_bdcdata-fval =      i_fval.  
Append wa_bdcdata to it_bdcdata.
```

Clear wa\_bdcdata.

Endmethod.

**→ Develop a conversion program to upload the bank details from flat file to SAP system by using BDC call transaction method by using OOPS ABAP.**

The flat file contains bank country key, bank key & bank name.

### **Steps to do the recording:**

Execute SHDB. Click on new recording in the application tool bar. Provide the recording name, transaction code (FI01). Enter. Provide country (IN), bank key (554141). Enter. Provide the bank name (HDFC Bank). Save.

```

REPORT ZOOABAPBDC.

TYPES: BEGIN OF TY_BANK,
        BANKS TYPE BNKA-BANKS,
        BANKL TYPE BNKA-BANKL,
        BANKA TYPE BNKA-BANKA,
      END OF TY_BANK.

DATA: WA_BANK TYPE TY_BANK,
      IT_BANK TYPE TABLE OF TY_BANK.

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

CLASS C1 DEFINITION.

PUBLIC SECTION.

METHODS: UPLOAD_DATA IMPORTING I_FILE TYPE STRING,
          FILL_SCREEN IMPORTING
                  I_PROG TYPE BDC_PROG
                  I_DYNPRO TYPE BDC_DYNR
                  I_DYNBEGIN TYPE BDC_START,
          FILL_FIELD IMPORTING
                  I_FNAM TYPE FNAM_____4
                  I_FVAL TYPE C.

ENDCLASS.

CLASS C1 IMPLEMENTATION.

METHOD UPLOAD_DATA.
CALL FUNCTION 'GUI_UPLOAD'
  EXPORTING
    FILENAME           = I_FILE
    HAS_FIELD_SEPARATOR = 'X'
  TABLES
    DATA_TAB          = IT_BANK.
ENDMETHOD.

METHOD FILL_SCREEN.
  WA_BDCDATA-PROGRAM = I_PROG.
  WA_BDCDATA-DYNPRO   = I_DYNPRO.
  WA_BDCDATA-DYNBEGIN = I_DYNBEGIN.
  APPEND WA_BDCDATA TO IT_BDCDATA.
  CLEAR WA_BDCDATA.
ENDMETHOD.

METHOD FILL_FIELD.
  WA_BDCDATA-FNAM = I_FNAM.
  WA_BDCDATA-FVAL = I_FVAL.
  APPEND WA_BDCDATA TO IT_BDCDATA.
  CLEAR WA_BDCDATA.
ENDMETHOD.

```

```

ENDCLASS.

START-OF-SELECTION.
  DATA RC1 TYPE REF TO C1.
  CREATE OBJECT RC1.
  CALL METHOD RC1->UPLOAD_DATA
    EXPORTING
      I_FILE = 'C:\Users\Administrator\Desktop\22.TXT'.

  LOOP AT IT_BANK INTO WA_BANK.

    CALL METHOD RC1->FILL_SCREEN
      EXPORTING
        I_PROG      = 'SAPMF02B'
        I_DYNPRO    = '0100'
        I_DYNBEGIN = 'X'.

    CALL METHOD RC1->FILL_FIELD
      EXPORTING
        I_FNAM = 'BDC_CURSOR'
        I_FVAL = 'BNKA-BANKL'.

    CALL METHOD RC1->FILL_FIELD
      EXPORTING
        I_FNAM = 'BDC_OKCODE'
        I_FVAL = '/00'.

    CALL METHOD RC1->FILL_FIELD
      EXPORTING
        I_FNAM = 'BNKA-BANKS'
        I_FVAL = WA_BANK-BANKS.

    CALL METHOD RC1->FILL_FIELD
      EXPORTING
        I_FNAM = 'BNKA-BANKL'
        I_FVAL = WA_BANK-BANKL.

    CALL METHOD RC1->FILL_SCREEN
      EXPORTING
        I_PROG      = 'SAPMF02B'
        I_DYNPRO    = '0110'
        I_DYNBEGIN = 'X'.

    CALL METHOD RC1->FILL_FIELD
      EXPORTING
        I_FNAM = 'BDC_CURSOR'
        I_FVAL = 'BNKA-BANKA'.

    CALL METHOD RC1->FILL_FIELD
      EXPORTING

```

```

I_FNAM = 'BDC_OKCODE'
I_FVAL = '=UPDA'.

CALL METHOD RC1->FILL_FIELD
EXPORTING
I_FNAM = 'BNKA-BANKA'
I_FVAL = WA_BANK-BANKA.

CALL TRANSACTION 'FI01' USING IT_BDCDATA MODE 'A'.
REFRESH IT_BDCDATA.

ENDLOOP.

```

**Interface:** -Interface is the collection of methods which are defined not implemented. These are implemented through class implementation. Interfaces are reusable components. We can declare the same interface in any number of classes under public section only. Interface components are always access through object.

We can't create the interface object directly. First we create the interface reference & later assign the class object to interface reference. Then automatically interface object is created.

#### Syntax of declaring the interface: -

Interface <interface name>

-----} method declaration  
-----}-----}  
-----}

Endinterface.

#### Syntax of access the interface in class definition: -

Interfaces <interface name>.

→ Based on the given company code, display the customers under company details (BUKRS KUNNR AKONT) by using OOABAP with interface concept.

```

REPORT ZOOPS1199.
PARAMETER P_BUKRS TYPE KNB1-BUKRS .
INTERFACE IF1.
  METHODS DISPLAY.
ENDINTERFACE.
CLASS C1 definition.
  PUBLIC SECTION.
    TYPES: BEGIN OF TY_KNB1,
      BUKRS TYPE KNB1-BUKRS ,
      KUNNR TYPE KNB1-KUNNR ,
      AKONT TYPE KNB1-AKONT ,
      END OF TY_KNB1 .
    DATA: WA_KNB1 TYPE TY_KNB1 ,
      IT_KNB1 TYPE TABLE OF TY_KNB1 .
    METHODS: GET_DATA IMPORTING I_BUKRS TYPE BUKRS .
      INTERFACES IF1 .
ENDCLASS .
CLASS C1 IMPLEMENTATION .
  METHOD GET_DATA .
    SELECT BUKRS KUNNR AKONT FROM KNB1 INTO TABLE IT_KNB1 WHERE BUKRS =
I_BUKRS .
  ENDMETHOD .

```

```

METHOD IF1~DISPLAY.
LOOP AT IT_KNB1 INTO WA_KNB1.
  WRITE:/ WA_KNB1-BUKRS, WA_KNB1-KUNNR, WA_KNB1-AKONT.
ENDLOOP.
ENDMETHOD.
ENDCLASS.

```

```

START-OF-SELECTION.
DATA RC1 TYPE REF TO C1.
DATA RFC1 TYPE REF TO IF1.
CREATE OBJECT RC1.
RFC1 = RC1.
CALL METHOD RC1->GET_DATA
  EXPORTING
    I_BUKRS = P_BUKRS.
CALL METHOD RFC1->DISPLAY.

```

There are two types of interfaces like a class.

1. Local class
2. Global class

#### Abstract class: -

If the class isn't fully implemented then the class is called abstract class or if the class contains at least one abstract method then the class is called abstract class. These abstract methods are implemented through derived class or child class.

We can't create the object for abstract class.

→ Based on the given company code, display the company code, company name, city by using OOABAP with abstract class concept.

```

REPORT ZOOPS1100.
PARAMETER P_BUKRS TYPE T001-BUKRS.
INCLUDE ZIIT001.
CLASS C1 DEFINITION ABSTRACT.
  PUBLIC SECTION.
    METHODS: GET_DATA IMPORTING I_BUKRS TYPE BUKRS,
              DISPLAY ABSTRACT.
  ENDCLASS.
  CLASS C1 IMPLEMENTATION.
    METHOD GET_DATA.
      SELECT SINGLE BUKRS BUTXT ORT01 FROM T001 INTO WA_T001 WHERE BUKRS
      = I_BUKRS.
    ENDMETHOD.
  ENDCLASS.
  CLASS C2 DEFINITION INHERITING FROM C1.
    PUBLIC SECTION.
      METHODS DISPLAY REDEFINITION.
    ENDCLASS.
    CLASS C2 IMPLEMENTATION.
      METHOD DISPLAY.
        WRITE:/ WA_T001-BUKRS, WA_T001-BUTXT, WA_T001-ORT01.
      ENDMETHOD.
    ENDCLASS.

```

```

ENDCLASS.

START-OF-SELECTION.
  DATA RC2 TYPE REF TO C2.
  CREATE OBJECT RC2.
  CALL METHOD RC2->GET_DATA
    EXPORTING
      I_BUKRS = P_BUKRS.
  CALL METHOD RC2->DISPLAY.

```

**Deferred class:** -

If you want to access the class before declaring & implementing the class then we must specify the class as deferred.

**Syntax** –

Class <class name> definition deferred.

Ex: - class c1 definition deferred.

**Friend class:-**

Through friends class we can access the private components of some other class.

```

REPORT ZOOABAP1111.
class c2 definition deferred.
class c1 definition friends c2.
  public section.
    methods M1.
  private section.
    methods M2.
endclass.
class c1 implementation.
  method M1.
    write 'SJF Technologies'.
  endmethod.
  method M2.
    write:/ 'S.R Nagar, Hyderabad'.
  endmethod.
endclass.
class c2 definition.
  public section.
    methods M3.
endclass.
class c2 implementation.
  method M3.
    data rc1 type ref to c1.
    CREATE OBJECT rc1.
    CALL METHOD rc1->M1.
    CALL METHOD rc1->M2.
  endmethod.
endclass.

start-of-selection.
  data rc2 type ref to c2.
  CREATE OBJECT rc2.
  CALL METHOD rc2->M3.

```

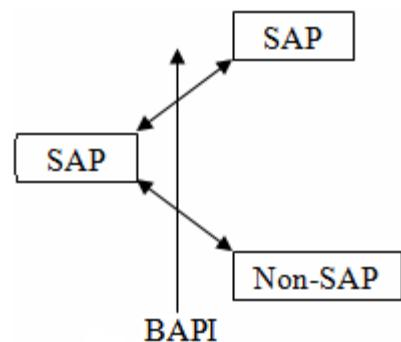
## BAPI

### (Business Application Programming Interface)

BAPI's are used to connecting from SAP to SAP as well as SAP to NON-SAP.

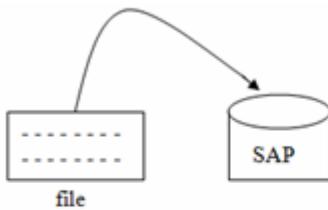
BAPI's are defined as an API method (Application Programming Interface) of SAP objects. These methods & their objects are stored as well maintained in BOR (Business Object Repository). BOR taking care about version change management (when ever the version is changed if any changes required in BAPI those changes is done by BOR).

BAPI's are also called as one of the remote enable function module.



#### **BAPI**

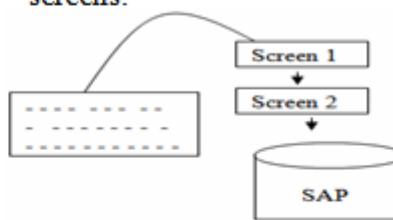
- 1) BAPI is used to upload the data from file to SAP system directly.



- 2) BAPI is faster
- 3) BAPI never cause to terminate the program when ever error occur in the BAPI it simply written those errors through written parameters.
- 4) BAPI perform their own authorization checks to validate the user.
- 5) In the BAPI the flat file fields are varying
- 6) When ever the version is changed we no need to change the code.

#### **BDC**

- 1) BDC is used to upload the data from file to SAP system through screens.



- 2) BDC is slower
- 3) Some times BDC open group BDC insert BDC close group may cause to terminates the program
- 4) In BDC we provide the authority check to validate the user.  
Authority - check <object name> id  
<name> active <value>.
- 5) In BDC the flat files fields are fixed.
- 6) When ever the version is changed some times we need to change the code.

#### Some of the important points related to BAPI: -

- 1) BAPI mustn't contain call transaction.
- 2) BAPI mustn't invoke commit work instead of commit work we call the BAPI\_TRANSACTION\_COMMIT function module.
- 3) BAPI structure mustn't contains includes.
- 4) There is no functional dependences between any two BAPI's.
- 5) BAPI never calls to terminate the program. Whenever an error occurred in BAPI it writes those errors through written parameter.
- 6) BAPI must not contain dialogue programs.
- 7) BAPI mustn't contain Submit reports.

#### Steps to create our own custom BAPI

1. Create the BAPI structure (SE11)

2. Create the BAPI function module (SE37)

1. Which is remote enable.
2. Must contains RETURN parameter
3. All parameters are pass by value

3. Create the object and methods for BAPI function module (SWO1)

4. Release the object and method (SWO1).

**→ Develop a custom BAPI to display the purchasing doc number, doc date & vendor number based on the given purchasing doc number.**

**Steps to create the BAPI structure:** -

Execute SE11. Select the radio button data type. Provide the BAPI structure name (ZBAPI\_10AM\_STR). Click on create. Select the radio button structure. Enter. Provide short description (BAPI Structure). Provide the component & component type.

EBELN EBELN

BEDAT BEDAT

LIFNR LIFNR

Save in own package. Check, activate.

**Steps to create function group:** -

Execute SE37. In the menu bar click on goto → function groups → create group. Provide the function group name (ZBAPI\_10AM\_FG), short description. Save in our own package.

**Steps to activate the function group:** -

In the menu bar click on environment → inactive objects. Expand the function group under transportable objects. Select the function group. Right click → activate. Enter.

**Steps to create function module:** -

Execute 'SE37'. Provide the function module name. Click on create. Provide the function group (ZBAPI\_10AM\_FG), short description. Enter. Click on attributes tab. Select the radio button remote-enabled module. Click on import tab.

**Import**

Parameter	type	associated type	pass by value
I_EBELN	TYPE	ZBAPI_10AM_STR-EBELN	<input checked="" type="checkbox"/>

**Export**

Parameter	type	associated type	pass by value
E_WA	TYPE	ZBAPI_10AM_STR	<input checked="" type="checkbox"/>
RETURN	TYPE	BAPIRET2	<input checked="" type="checkbox"/>

**Source code**

Select single ebeln bedat lifnr from ekko into wa\_ekko where ebeln = i\_ebeln.

Save, check, activate. Click on back. In the menu bar click on function module. Release -> Release.

Function module is released.

**Steps to release the object & method for BAPI function module:** -

Execute SWO1. Provide the object name (ZBAPI\_10AM). Click on create. Provide the object name (ZBAPI\_10AM). Click on create. Provide the same to all. Select the application (M). Enter. Click on enter. Save in our package. Place the cursor on methods. In the menu bar click on utilities. API methods → Add method. Provide function module name (ZBAPI\_10AM\_FM). Click on enter. Click on next step (CTRL + SHIFT + F11). Click on next. Click on yes. Click on save.

**Steps to release the object & method:** -

Place the cursor on object (ZBAPI\_10AM). In the menu bar click on edit → change release status → object type → to modeled (to implemented, to release also). Select the method. In the menu bar, click on edit → change release status → object type component → to modeled (to implemented, to released also). Click on generate (CTRL + F3).

**Steps to check BAPI is available or not in BOR:**-

Execute BAPI transaction. Click on alphabetical tab & absorb our BAPI.

**Note:** - In the real time we never create our own custom BAPIs. We always use existing BAPIs.

**Note:** - In the real time we always use the BAPIs like conversion program or report.

### **Steps to identify the standard BAPIs: -**

#### Method 1:-

Execute 'SE93'. Provide the related transaction code (ME23N). Click on display. Click on display object list (CTRL + SHIFT + F5) in the application tool bar. Expand the package. Expand the business engineering. Expand the business object types. Double click on our required business object. Expand the method. Double click on our required method (Get details). Click on ABAP tab. Identify the BAPI name (BAPI\_PO\_GETDETAIL).

#### Method 2:-

Execute BAPI transaction. Click on Alphabetical tab. Expand the Object (Sales Order). Click on method (CreateFromDat2). You can find the function module here.

#### Method 3:-

By using BAPI methods we can identify the standard BAPI.

### **Some of the BAPI methods are**

1. Get list ()
2. Get details ()
3. Get status ()
4. Existence check ()
5. Create ()
6. Change ()
7. Delete ()

### **Steps to identify the BAPI based on BAPI method name: -**

Execute 'SE37'. Provide BAPI <Method name>. Click on F4. We get the list of BAPIs related to that method & identify our required method.

**Note:** - When ever we are working with create, change, delete BAPIs, then we must committed the data base by using 'BAPI\_TRANSACTION\_COMMIT' function module. Other wise the data base isn't updated.

### **Steps to create cost center group manually: -**

Execute 'KSH1'. Provide the Controlling area (6000). Enter. Provide the cost center group name (ZSPG). Enter. Provide short description. Click on cost center in the application tool bar. Provide the from to (1000 – 3000), (5000 – 7000) cost centers. Enter. Click on save.

#### **→ Create the cost center group by using BAPI.**

```
BAPI: - BAPI_COSTCENTERGROUP_CREATE
PARAMETER P_CAREA LIKE BAPICO_GROUP-CO_AREA.
DATA WA_RETURN LIKE BAPIRET2.
DATA: IT_HNODE LIKE TABLE OF BAPISET_HIER,
      WA_HNODE LIKE LINE OF IT_HNODE.
DATA: IT_HVAL LIKE TABLE OF BAPI1112_VALUES,
      WA_HVAL LIKE LINE OF IT_HVAL.
WA_HNODE-GROUPNAME = 'ZSPG'.
WA_HNODE-HIERLEVEL = '0'.
WA_HNODE-VALCOUNT = '2'.
WA_HNODE-DESCRIPT = 'COSTCENTER GROUP1'.
APPEND WA_HNODE TO IT_HNODE.
WA_HVAL-VALFROM = '0000001000'.
WA_HVAL-VALTO = '0000002000'.
APPEND WA_HVAL TO IT_HVAL.
WA_HVAL-VALFROM = '0000003000'.
```

```

WA_HVAL-VALTO      = '0000004000'.
APPEND WA_HVAL TO IT_HVAL.
CALL FUNCTION 'BAPI_COSTCENTERGROUP_CREATE'
  EXPORTING
    CONTROLLINGAREAIMP = P_CAREA
  IMPORTING
    RETURN             = WA_RETURN
  TABLES
    HIERARCHYNODES    = IT_HNODE
    HIERARCHYVALUES   = IT_HVAL.

CALL FUNCTION 'BAPI_TRANSACTION_COMMIT'.
IF WA_RETURN IS INITIAL.
  WRITE 'COSTCENTER GROUP CREATED'.
ELSE.
  WRITE WA_RETURN-MESSAGE.
ENDIF.

```

Steps to create the bank details manually: -

Execute 'FI01'. Provide bank country key (IN). Bank key (15151619). Enter. Provide the bank name (HDFC Bank). Save.

#### → Create the bank details by using BAPI.

**BAPI NAME: BAPI\_BANK\_CREATE**

```

PARAMETER P_CTRY LIKE BAPI1011_KEY-BANK_CTRY.
DATA WA_RETURN LIKE BAPIRET2.
DATA WA_ADD LIKE BAPI1011_ADDRESS.
WA_ADD-BANK_NAME  = 'PANJAB BANK'.
WA_ADD-REGION     = '01'.
WA_ADD-STREET     = 'AMEERPET'.
WA_ADD-CITY       = 'HYD'.
CALL FUNCTION 'BAPI_BANK_CREATE'
  EXPORTING
    BANK_CTRY      = P_CTRY
    BANK_KEY       = '15161718'
    BANK_ADDRESS  = WA_ADD
  IMPORTING
    RETURN         = WA_RETURN.

```

```
CALL FUNCTION 'BAPI_TRANSACTION_COMMIT'.
```

```
IF WA_RETURN IS INITIAL.
```

```
  WRITE 'BANK CREATED'.
```

```
ELSE.
```

```
  WRITE WA_RETURN-MESSAGE.
```

```
ENDIF.
```

#### → Update the bank city & street using BAPI.

Steps to update the bank details manually: -

Execute FI02. provide bank country key (IN), bank key (112233). Enter. Provide the street (SR Nagar), city (HYD). Save.

```

PARAMETER: P_CTRY LIKE BAPI1011_KEY-BANK_CTRY,
            P_BKEY LIKE BAPI1011_KEY-BANK_KEY.

```

```

DATA: WA_ADD LIKE BAPI1011_ADDRESS,
      WA_ADDX LIKE BAPI1011_ADDRESSSX.
DATA WA_RETURN LIKE BAPIRET2.
WA_ADD-STREET = 'MOOSAPET'.
WA_ADD-CITY = 'HYD'.
WA_ADDX-STREET = 'X'.
WA_ADDX-CITY = 'X'.
CALL FUNCTION 'BAPI_BANK_CHANGE'
  EXPORTING
    BANKCOUNTRY = P_CTRY
    BANKKEY = P_BKEY
    BANK_ADDRESS = WA_ADD
    BANK_ADDRESSSX = WA_ADDX
  IMPORTING
    RETURN = WA_RETURN.

CALL FUNCTION 'BAPI_TRANSACTION_COMMIT'.
IF WA_RETURN IS INITIAL.
  WRITE / 'BANK UPDATED'.
ELSE.
  WRITE WA_RETURN-MESSAGE.
ENDIF.

```

→ Develop a conversion program to update the bank city & street from flat file to sap system by using BAPI. The flat file contains bank country key, bank key, street & city.

```

DATA: BEGIN OF WA_BANK,
      BCTRY LIKE BAPI1011_KEY-BANK_CTRY,
      BKEY LIKE BAPI1011_KEY-BANK_KEY,
      STREET TYPE STRAS_GP,
      CITY TYPE ORT01_GP,
      END OF WA_BANK.
DATA IT_BANK LIKE TABLE OF WA_BANK.
DATA: WA_ADD LIKE BAPI1011_ADDRESS,
      WA_ADDX LIKE BAPI1011_ADDRESSSX,
      WA_RETURN LIKE BAPIRET2.

```

```

CALL FUNCTION 'UPLOAD'
  EXPORTING
    FILETYPE = 'DAT'
  TABLES
    DATA_TAB = IT_BANK.
LOOP AT IT_BANK INTO WA_BANK.
  WA_ADD-STREET = WA_BANK-STREET.
  WA_ADD-CITY = WA_BANK-CITY.
  WA_ADDX-STREET = 'X'.
  WA_ADDX-CITY = 'X'.
CALL FUNCTION 'BAPI_BANK_CHANGE'
  EXPORTING
    BANKCOUNTRY = WA_BANK-BCTRY
    BANKKEY = WA_BANK-BKEY
    BANK_ADDRESS = WA_ADD
    BANK_ADDRESSSX = WA_ADDX

```

BCTRY	BKEY	STREET	CITY
IN	111111	SR NAGAR	HYD
IN	111112	KOTI	HYD
IN	111113	BEGUMPET	HYD
IN	111114	KPHB	HYD

WA_BANK	BCTRY	BKEY	STREET	CITY

WA_ADD	WA_BANK_NAME	REGION	STREET	CITY	----

WA_ADDX	BANK_NAME	REGION	STREET	CITY	----

```

IMPORTING
  RETURN      = WA_RETURN.
CALL FUNCTION 'BAPI_TRANSACTION_COMMIT'.
IF WA_RETURN IS INITIAL.
  WRITE / 'BANK UPDATED'.
ELSE.
  WRITE WA_RETURN-MESSAGE.
ENDIF.
ENDLOOP.

```

### Application of BAPI

- BAPIs are used to connect with R/3 to new SAP components BI/BW, APO, CRM, SCM etc.
- BAPIs are used to connect with R/3 to internet by using IACS (Internet Application Components).
- BAPIs are used to connect with distributed system with help of ALE.
- BAPIs are used to connect with VB as a front end to R/3 system.
- BAPIs are used to connect with R/3 to business partners own developments that is vendor portals, customer portals etc.
- BAPIs are used to connect with SAP to legacy system.
- BAPIs are used to connect with SAP to non-sap system.

### ➔ Upload the Bank details from flat file to SAP system by using BAPI

```

DATA: BEGIN OF WA_BANK,
  BCTRY LIKE BAPI1011_KEY-BANK_CTRY,
  BKEY LIKE BAPI1011_KEY-BANK_KEY.
  INCLUDE STRUCTURE BAPI1011_ADDRESS.
DATA END OF WA_BANK.

```

```

DATA IT_BANK LIKE TABLE OF WA_BANK.
DATA WA_ADD LIKE BAPI1011_ADDRESS.
DATA WA_RETURN LIKE BAPIRET2.
CALL FUNCTION 'UPLOAD'

```

```

  EXPORTING
    FILETYPE = 'DAT'
  TABLES
    DATA_TAB = IT_BANK.
LOOP AT IT_BANK INTO WA_BANK.
MOVE-CORRESPONDING WA_BANK TO WA_ADD.
CALL FUNCTION 'BAPI_BANK_CREATE'

```

```

  EXPORTING
    BANK_CTRY      = WA_BANK-BCTRY
    BANK_KEY       = WA_BANK-BKEY
    BANK_ADDRESS   = WA_ADD

```

```

  IMPORTING
    RETURN      = WA_RETURN.
CALL FUNCTION 'BAPI_TRANSACTION_COMMIT'.
IF WA_RETURN IS INITIAL.
  WRITE / 'BANK CREATED'.
ELSE.
  WRITE / WA_RETURN-MESSAGE.
ENDIF.
ENDLOOP.

```

MCTRY	BKEY	bank name	Reginon	Street	city
IN	54520	CITY Bank	1	KOTI	HYD

MCTRY	BKEY	bank name	Reginon	Street	city
IN	54520	CITY Bank	1	KOTI	HYD
IN	54522	AXIS Bank			BAN
IN	54523			D.NA	HYD

Bank name	Reginon	Street	City
CITY Bank	1	KOTI	HYD

→ Based on the given comp code vendor number & date, display the vendor open item details by using BAPI.

```
PARAMETER: P_BUKRS LIKE BAPI3008_1-COMP_CODE,
            P_LIFNR LIKE BAPI3008_1-VENDOR,
            P_DAT LIKE BAPI3008-KEY_DATE.
DATA WA_RETURN LIKE BAPIRETURN.
DATA T_OT LIKE TABLE OF BAPI3008_2.
CALL FUNCTION 'BAPI_AP_ACC_GETOPENITEMS'
  EXPORTING
    COMPANYCODE = P_BUKRS
    VENDOR      = P_LIFNR
    KEYDATE     = P_DAT
  IMPORTING
    RETURN      = WA_RETURN
  TABLES
    LINEITEMS   = T_OT.
IF WA_RETURN IS INITIAL.
  CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
    EXPORTING
      I_STRUCTURE_NAME = 'BAPI3008_2'
    TABLES
      T_OUTTAB       = T_OT.
ELSE.
  WRITE WA_RETURN-MESSAGE.
ENDIF.
```

→ Based on the given company code, customer number, from date & to date display the customer cleared items by using BAPI.

```
PARAMETER: P_BUKRS LIKE BAPI3007_1-COMP_CODE,
            P_KUNNR LIKE BAPI3007_1-CUSTOMER,
            P_FD LIKE BAPI3007-FROM_DATE,
            P_TD LIKE BAPI3007-TO_DATE.
DATA WA_RETURN LIKE BAPIRETURN.
DATA T_CI LIKE TABLE OF BAPI3007_2.
CALL FUNCTION 'BAPI_AR_ACC_GETBALANCEDITEMS'
  EXPORTING
    COMPANYCODE = P_BUKRS
    CUSTOMER    = P_KUNNR
    DATE_FROM   = P_FD
    DATE_TO     = P_TD
  IMPORTING
    RETURN      = WA_RETURN
  TABLES
    LINEITEMS   = T_CI.
IF WA_RETURN IS INITIAL.
  CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
    EXPORTING
      I_STRUCTURE_NAME = 'BAPI3007_2'
    TABLES
      T_OUTTAB       = T_CI.
ELSE.
  WRITE WA_RETURN-MESSAGE.
```

ENDIF.

→ Based on the given purchasing document number display the purchasing document item details & history details by using BAPI.

BAPI: BAPI\_PO\_GETDETAIL

```
PARAMETER P_EBELN LIKE BAPIEKKO-PO_NUMBER.  
DATA: IT_ITEMS LIKE TABLE OF BAPIEKPO,  
      WA_ITEMS LIKE LINE OF IT_ITEMS.  
DATA: IT_HIS LIKE TABLE OF BAPIEKBE,  
      WA_HIS LIKE LINE OF IT_HIS.  
DATA: IT_RETURN LIKE TABLE OF BAPIRETURN,  
      WA_RETURN LIKE LINE OF IT_RETURN.  
CALL FUNCTION 'BAPI_PO_GETDETAIL'  
  EXPORTING  
    PURCHASEORDER = P_EBELN  
    ITEMS        = 'X'  
    HISTORY       = 'X'  
  TABLES  
    PO_ITEMS      = IT_ITEMS  
    PO_ITEM_HISTORY = IT_HIS  
    RETURN        = IT_RETURN.
```

ME23N

```
IF IT_RETURN IS INITIAL.  
  LOOP AT IT_ITEMS INTO WA_ITEMS.  
    WRITE:/ WA_ITEMS-PO_NUMBER, WA_ITEMS-PO_ITEM.  
  ENDLOOP.  
  ULINE.  
  LOOP AT IT_HIS INTO WA_HIS.  
    WRITE:/ WA_HIS-MAT_DOC, WA_HIS-DOC_YEAR.  
  ENDLOOP.  
ELSE.  
  LOOP AT IT_RETURN INTO WA_RETURN.  
    WRITE:/ WA_RETURN-MESSAGE.  
  ENDLOOP.  
ENDIF.
```

→ Based on the given purchasing document numbers, display the purchasing document item, service & history details by using BAPI.

```
REPORT ZBAPI993.  
TYPE-POOLS SLIS.  
DATA V1 LIKE BAPIEKKO-PO_NUMBER.  
SELECT-OPTIONS S_EBELN FOR V1.  
DATA: BEGIN OF WA,  
      EBELN TYPE EKKO-EBELN,  
      END OF WA.  
DATA IT LIKE TABLE OF WA.  
  
DATA: IT_ITEMS LIKE TABLE OF BAPIEKPO,  
      WA_ITEMS LIKE LINE OF IT_ITEMS,  
      IT_ITEMS1 LIKE IT_ITEMS.  
DATA: IT_HIS LIKE TABLE OF BAPIEKBE,
```

```

        WA_HIS LIKE LINE OF IT_HIS,
        IT_HIS1 LIKE IT_HIS.
DATA: IT_SER LIKE TABLE OF BAPIESLL,
      WA_SER LIKE LINE OF IT_SER,
      IT_SER1 LIKE IT_SER.
DATA: IT_RETURN LIKE TABLE OF BAPIRETURN,
      WA_RETURN LIKE LINE OF IT_RETURN,
      IT_RETURN1 LIKE IT_RETURN,
      WA_RETURN1 LIKE LINE OF IT_RETURN1.

DATA: IT_FCAT1 TYPE SLIS_T_FIELDCAT_ALV,
      WA_FCAT1 LIKE LINE OF IT_FCAT1.
DATA: IT_FCAT2 TYPE SLIS_T_FIELDCAT_ALV,
      WA_FCAT2 LIKE LINE OF IT_FCAT2.
DATA: IT_FCAT3 TYPE SLIS_T_FIELDCAT_ALV,
      WA_FCAT3 LIKE LINE OF IT_FCAT3.

DATA: IT_EVENT1 TYPE SLIS_T_EVENT,
      IT_EVENT2 TYPE SLIS_T_EVENT,
      IT_EVENT3 TYPE SLIS_T_EVENT.

DATA: WA_LAYOUT1 TYPE SLIS_LAYOUT_ALV,
      WA_LAYOUT2 TYPE SLIS_LAYOUT_ALV,
      WA_LAYOUT3 TYPE SLIS_LAYOUT_ALV.

```

SELECT EBELN FROM EKKO INTO TABLE IT WHERE EBELN IN S\_EBELN.

```

LOOP AT IT INTO WA.
  CALL FUNCTION 'BAPI_PO_GETDETAIL'
    EXPORTING
      PURCHASEORDER      = WA_EBELN
      ITEMS              = 'X'
      HISTORY            = 'X'
      SERVICES           = 'X'
    TABLES
      PO_ITEMS          = IT_ITEMS
      PO_ITEM_HISTORY   = IT_HIS
      PO_ITEM_SERVICES = IT_SER.
  APPEND LINES OF IT_ITEMS TO IT_ITEMS1.
  APPEND LINES OF IT_HIS TO IT_HIS1.
  APPEND LINES OF IT_SER TO IT_SER1.
  APPEND LINES OF IT_RETURN TO IT_RETURN1.
ENDLOOP.

```

```

IF IT_RETURN IS INITIAL.
  WA_FCAT1-FIELDNAME = 'PO_NUMBER'.
  WA_FCAT1-COL_POS   = '1'.
  WA_FCAT1-SELTEXT_M = 'PO NUMBER'.
  APPEND WA_FCAT1 TO IT_FCAT1.
  CLEAR WA_FCAT1.
  WA_FCAT1-FIELDNAME = 'PO_ITEM'.

```

```

WA_FCAT1-COL_POS      = '2'.
WA_FCAT1-SELTEXT_M    = 'ITEM NUMBER'.
APPEND WA_FCAT1 TO IT_FCAT1.
CLEAR WA_FCAT1.

WA_FCAT2-FIELDNAME   = 'MAT_DOC'.
WA_FCAT2-COL_POS     = '1'.
WA_FCAT2-SELTEXT_M   = 'MATERIAL DOCUMENT'.
APPEND WA_FCAT2 TO IT_FCAT2.
CLEAR WA_FCAT2.

WA_FCAT2-FIELDNAME   = 'DOC_YEAR'.
WA_FCAT2-COL_POS     = '2'.
WA_FCAT2-SELTEXT_M   = 'DOCUMENT YEAR'.
APPEND WA_FCAT2 TO IT_FCAT2.
CLEAR WA_FCAT2.

WA_FCAT3-FIELDNAME   = 'PCKG_NO'.
WA_FCAT3-COL_POS     = '1'.
WA_FCAT3-SELTEXT_M   = 'PACKAGE NUMBER'.
APPEND WA_FCAT3 TO IT_FCAT3.
CLEAR WA_FCAT3.

WA_FCAT3-FIELDNAME   = 'SUBPCKG_NO'.
WA_FCAT3-COL_POS     = '2'.
WA_FCAT3-SELTEXT_M   = 'SUB PACKAGE NUMBER'.
APPEND WA_FCAT3 TO IT_FCAT3.
CLEAR WA_FCAT3.

CALL FUNCTION 'REUSE_ALV_BLOCK_LIST_INIT'
  EXPORTING
    I_CALLBACK_PROGRAM = SY-CPROG.
CALL FUNCTION 'REUSE_ALV_BLOCK_LIST_APPEND'
  EXPORTING
    IS_LAYOUT      = WA_LAYOUT1
    IT_FIELDCAT   = IT_FCAT1
    I_TABNAME     = 'IT_ITEMS1'
    IT_EVENTS      = IT_EVENT1
  TABLES
    T_OUTTAB      = IT_ITEMS1.
CALL FUNCTION 'REUSE_ALV_BLOCK_LIST_APPEND'
  EXPORTING
    IS_LAYOUT      = WA_LAYOUT2
    IT_FIELDCAT   = IT_FCAT2
    I_TABNAME     = 'IT_HIS1'
    IT_EVENTS      = IT_EVENT2
  TABLES
    T_OUTTAB      = IT_HIS1.
CALL FUNCTION 'REUSE_ALV_BLOCK_LIST_APPEND'
  EXPORTING
    IS_LAYOUT      = WA_LAYOUT3
    IT_FIELDCAT   = IT_FCAT3
    I_TABNAME     = 'IT_SER1'

```

```

IT_EVENTS      = IT_EVENT3
TABLES
T_OUTTAB      = IT_SER1.
CALL FUNCTION 'REUSE_ALV_BLOCK_LIST_DISPLAY'.

ELSE.
LOOP AT IT_RETURN1 INTO WA_RETURN1.
  WRITE:/ WA_RETURN1-MESSAGE.
ENDLOOP.
ENDIF.

```

Enhanced BAPIs: -

Based on the client requirement if you add additional fields to the standard data base table. If you want to update these fields information by using BAPI then we go for BAPI enhanced.

#### **Steps to work with enhanced BAPI:** -

Open the BAPI in SE37 & identify the extension in parameter & double click on their associated type & add our additional fields to the structure by using append structure.

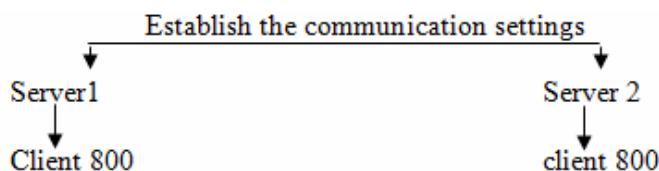
At the time of calling the BAPI we need to pass the additional fields information. Those information is automatically updated into standard data base table.

If you want get the additional fields information through BAPI then we add these additional fields to the extension out parameter in the BAPI.

#### ***Note: - Difference b/w BAPI & RFC (Remote Function Call)?***

BAPI is remote enable FM available for public to interact with SAP system. RFC is protocol used to call a FM from one system to another system. BAPI is defined in BOR as methods and implemented as RFC enabled FM. BAPI doesn't handle the exceptions. The calling program has to handle the exceptions. RFC handle the exceptions. In BAPI there is no direct system call while RFC are direct system call. BAPI and RFC have the same functionality except that BAPI hits the BOR via methods & RFC not necessary to hit BOR.

A BAPI function is a function module that can be called remotely using the RFC technology.



#### **Function module definition**

'ZSPRAO\_10AM\_ADD'

#### **IMPORT**

A type I

B type I

#### **EXPORT**

C type I

#### **Source code**

C = A + B

Call function 'ZSPRAO\_10AM\_ADD' destination LS1-800

Logical system name of the server client where the definition is available.

#### **EXPORTING**

A =

B =

#### **IMPORTING**

C =

## BADI (Business ADD-INS)

BADIs are used to adding some additional functionality to standard functionality without disturbing the standard functionality. BADIs are introduced from 4.6C version onwards. BADI contains 2 sections.

1. BADI Definition
2. BADI Implementation

**Badi Definition:-** Each BADI definition contains one adapter class and interface. Adapter class taking care the version change management (When ever the version is changed if any changes required in the BADI those are done by adapter class). Interface is the collection of methods which are defined not implemented. Those are implemented through BADI implementation based on client requirement.

The transaction code for BADI definition is **SE18**.

**BADI Implementation:** - BADI Implementation is nothing but implementing the methods which are defined in BADI definition. The transaction code for BADI implementation is **SE19**.

### Differences between Customer exit (Enhancement) and BADI.

#### Customer Exit (Enhancement)

1. We can implement the customer exit only once.
2. Customer exit is the procedural approach. So it takes some extra time to enhancing.
3. In the screen exit the screen number is fixed which is provide by SAP.
4. By using CMOD transaction we can implement the customer exit.

#### BADI

1. We can implement the same BADI any number of times.
2. BADIs are object oriented approach. So it takes very less time to enhancing.
3. In the screen BADI we provide our own screen number.
4. By using **SE19** transaction we can implement the BADI.

**Note:** - In the real time we never create our own BADI. We always use existing BADIs.

### Types of BADI'S

There are 4 types of BADI's

1. Single Implementation BADI
2. Multiple implementation BADI
3. Filter BADI
4. Custom BADI.

#### 1. Single implementation BADI:

- This BADI can allow implementing only once. This type of BADI is called single implementation BADI.
- If the BADI definition does not contain 'multiple use' check box then the BADI is called as single implementation BADI.

#### 2. Multiple implementation BADI:

- It can allow implementing the same BADI any number of times.
- If the BADI Definition contains 'multiple use' check boxes then the BADI is called as Multiple Implementation BADI.

### **3. Custom Badi:**

- These are created by us (Technical people)

### **4. Filter Badi:**

- It is type of BADI which has a filter value so that only those implementations which satisfy the filter value are executed. The remaining implementations are not executed this type of BADI is called a filter BADI.

Filter Badi is used to implement the same BADI multiple times with different conditions. If we want to implement the BADI based on country specific then we go for filter BADI.

When ever we are working with cost center and profit center related BADIs then we must provide filter type as controlling area.

When ever we are working with GL Account related BADIs then we must provide filter type as chart-of-accounts.

When ever we are working with vendor & customer screen BADIs then we must provide filter type as screen group.

General Data	
Package	ME
Language	DE German
Name of Generated BAdI Class	
CL_EX_ME_PROCESS_PO	
Type	
<input checked="" type="checkbox"/> Within SAP <input checked="" type="checkbox"/> Multiple Use  <input type="checkbox"/> Filter-Depend.	
BAdI migrates to enhancement spot ME_PROCESS_PO	
Filter Type SAPNuts.com	
<input type="checkbox"/> Enhanceable	

- If WITHIN SAP checkbox is selected then this BADI is only used by sap.
- If multiple use checkbox is selected then it is a multiple implementation BADI.
- If multiple use checkbox is not selected then it is a single implementation BADI.
- If filter dependent checkbox is selected then it is filter dependent BADI. We need to specify the filter type such as land1,bukrs,werks.for the filter BADI.

### **Some of the scenarios: -**

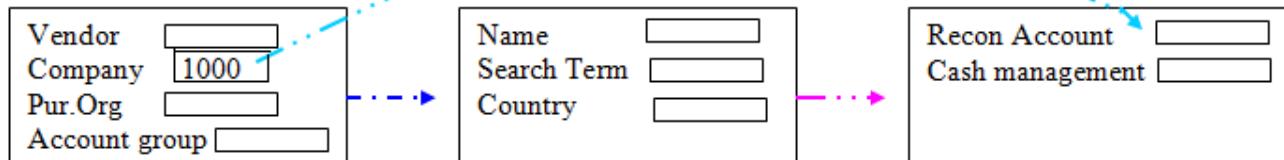
#### **Scenario 1: -**

In the real time MDM (Master Data Management) people create the vendor and customer 'XK01' & 'XD01' & later they maintain the key information in a excel sheet at end of the month they take a printout. They put a sign & their respective people also put a sign. This is required for audit companies. We

implement a BADI to maintain the key information in a excel sheet at the time of create & save the vendor & customer.

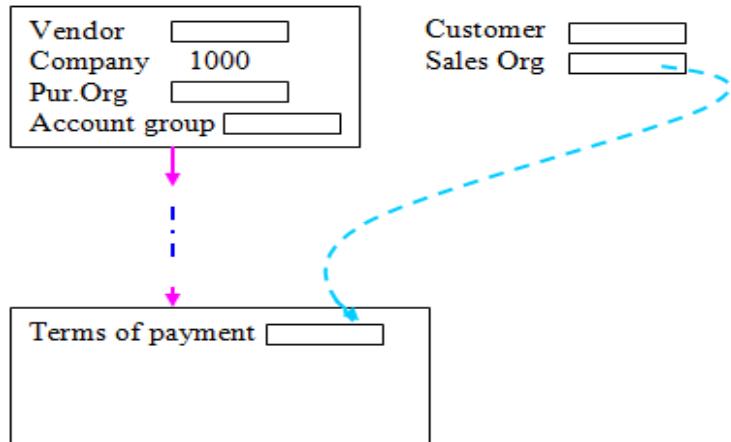
### Scenario 2: -

At the time of creating the vendor and customer some times MDM people provide the incorrect recon account and cash management group. To avoid this we implement the BADI to preset the recon account and cash management group based on given company code.



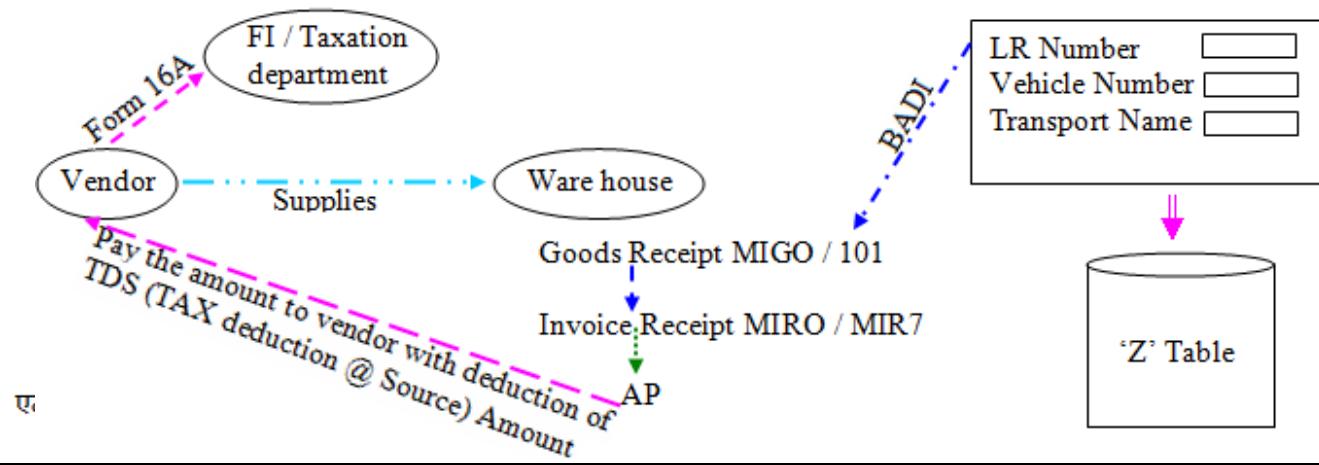
### Scenario 3: -

At the time of creating the vendor and customer some times MDM people provide the invalid terms of payment to avoid this we implement the BADI to preset the terms of payment based on the purchase organization or sales organization.



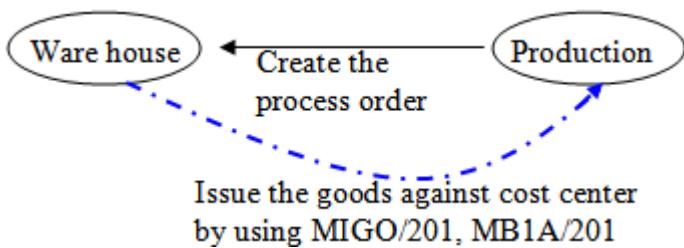
### Scenario 4: -

Against the vendor supplies, the warehouse department create a goods receipt with MIGO transaction 101 movement type. Against GR the finance people physically verify the stock & prepares the IR document. Against IR document the account payable department pay the amount to the vendor with deducts of TDS amount. After few days the vendor asks the Form 16 to finance people at the time of generating form 16, the finance department must provide LR Number, Vehicle number, Transport name. These information isn't captured in the entire MM life cycle. Here we implement the BADI to attach additional subscreen which contains above fields to MIGO transaction.



### **Scenario 5: -**

The warehouse people issue the goods against cost center by using MIGO or MB1A transaction code with 201 movement type. Sometimes they entered invalid cost center. To avoid this we implement the BADI to throw an error message if they pass invalid cost center or cost center not belongs to that plant.



### **Steps to identify the standard BADIs: -**

#### **Method 1: -**

Execute ‘SE24’. Provide the object type as ‘CL\_EXITHANDLER’. Click on display. Double click on GET\_INSTANCE method. Place the break-point on first call method. Now we execute our required transaction (which transaction BADIs we want) in a separate session. Execute XK01. Enter. In the menu bar click on debugger → switch to classic debugger. Provide the field name as EXIT\_NAME. Enter. Identify the BADI (VENDOR\_ADD\_DATA). Continuously click on F8 button. Identify the all the BADIs & maintained in Excel sheet. After delete the break point. Now open the each & every BADI in ‘SE18’. (BADI Name: VENDOR\_ADD\_DATA). Click on display. Click on interface tab. Double click on each & every method. Identify the input, output parameters. In the menu bar click on Goto → documentation → To component. Read the documentation if it satisfies our requirement then we implement this method through SE19.

**Note:** – In the real time we always identify the BADIs with the help of functional people.

#### **Method 2: -**

Execute ‘SE93’. Provide transaction code (XK01). Click on display. Click on display object list (Ctrl + Shift + F5). Expand the package in the left panel (FBK). Expand the enhancements. Expand the classic BADIs definition. Identify the all the BADIs. Open the each and every BADI in SE18. Click on interface tab. Double click on each method. Identify the input, output parameters. In the menu bar click on goto → documentation → to component. Read the documentation. If it satisfies client requirement then we implement this method through SE19.

#### **Method 3: -**

Execute **SE93** transaction. Provide the transaction code(for which transaction Badi we want like XK01/XD01). Click on Display. Identify the package. Go to **SE84** transaction, expand enhancements, expand Business Add-ins and select definition. Provide package name and execute, you will find list of BADI's.

**→ Implement the BADI to maintain the key information in an excel sheet at the time of create & save the vendor. The key information is vendor number, company code, account group, name, search term, street, city, postal code, country, created by, created date.**

BADI Name : VENDOR\_ADD\_DATA

Method Name: CHECK\_ALL\_DATA

### **Steps to implement the BADI: -**

Execute SE19. Select the Classic BADI Radio button in the implementation block. Provide BADI name (VENDOR\_ADD\_DATA). Click on create implementation. Provide implementation name (ZS10AM\_VEN\_IMP). Enter. Provide short description (Maintain key info in excel sheet). Click on save. Enter. Click on create enhancement implementation. Provide the same implementation name (ZS10AM\_VEN\_IMP). Provide short description (Maintain key info in an excel sheet). Enter. Save in local object. Select the implementation name. Enter. Click on interface tab. Double click on CHECK\_ALL\_DATA method. Click on signature. Identify the input output parameters & implement the logic. Save, check, activate the method. Click on back. Save, check, activate the BADI.

```
TYPES : BEGIN OF TY_VEN,
         LIFNR TYPE LFA1-LIFNR,
         BUKRS TYPE LFB1-BUKRS,
         KTOKK TYPE LFA1-KTOKK,
         NAME1 TYPE LFA1-NAME1,
         SORTL TYPE LFA1-SORTL,
         STRAS TYPE LFA1-STRAS,
         ORT01 TYPE LFA1-ORT01,
         PSTLZ TYPE LFA1-PSTLZ,
         LAND1 TYPE LFA1-LAND1,
         CBY TYPE SYUNAME,
         CDT(10) TYPE C,
         END OF TY_VEN.

DATA: WA_VEN TYPE TY_VEN,
      IT_VEN TYPE TABLE OF TY_VEN.

WA_VEN-LIFNR = I_LFA1-LIFNR.
WA_VEN-BUKRS = I_LFB1-BUKRS.
WA_VEN-KTOKK = I_LFA1-KTOKK.
WA_VEN-NAME1 = I_LFA1-NAME1.
WA_VEN-SORTL = I_LFA1-SORTL.
WA_VEN-STRAS = I_LFA1-STRAS.
WA_VEN-ORT01 = I_LFA1-ORT01.
WA_VEN-PSTLZ = I_LFA1-PSTLZ.
WA_VEN-LAND1 = I_LFA1-LAND1.
WA_VEN-CBY = SY-UNAME.
WA_VEN-CDT = SY-DATUM.
APPEND WA_VEN TO IT_VEN.
CLEAR WA_VEN.
CALL FUNCTION 'GUI_DOWNLOAD'
  EXPORTING
    FILENAME = 'C:\Users\Administrator\Desktop\HAI.xls'
    APPEND   = 'X'
  TABLES
    DATA_TAB = IT_VEN.
REFRESH IT_VEN.
```

**→ Implement the BADI to preset the Recon Account & cash management group based on the given company code**

AKONT → Recon Account  
 FDGRV → Cash Management Group

When ever we are working with Preset BADI's then we must create one 'Z' table with preset fields and based on which we want to preset the values and later the table data is updated by functional people. Based on this table data we preset the values. In this object we must create one 'Z' table with the following fields.

1. MANDT → Client
2. BUKRS → Company code
3. AKONT → Recon Account
4. FDGRV → Cash Management

Input field  
 (Based on this, we preset the values)

And later this table data is provided by functional people based on this data we preset the values.

Table name: ZS10AM\_PCV

MANDT	BUKRS	AKONT	FDGRV
800	1000	0000031000	A1
800	2000	0000160000	A6
800	3000	0000031000	A6

BADI Name: VENDOR\_ADD\_DATA

Method: PRESET\_VALUES\_CCODE

Implement the BADI through SE19.

```

DATA WA TYPE ZS10AM_PCV.
SELECT SINGLE * FROM ZS10AM_PCV INTO WA WHERE BUKRS = E_LFB1-
BUKRS.
IF SY-SUBRC = 0.
  E_LFB1-AKONT = WA-AKONT.
  E_LFB1-FDGRV = WA-FDGRV.
ENDIF.
```

#### Steps to identify the all the implementations of BADI: -

Execute 'SE18'. Provide the BADI name (VENDOR\_ADD\_DATA). Click on display. In the menu bar click on implementation → Display. Identify the all the implementations of BADI. Yellow color Badis are active and Blue color Badis are inactive.

#### → Implement the BADI to preset terms of payment based on purchase organization.

In this object we must create one 'Z' table with the following fields.

1. MANDT → Client
2. EKORG → Purchase organization (Input field)
3. ZTERM → Terms of Payment (Preset field)

Table Name: (ZS10AM\_PPV)

MANDT	EKORG	ZTERM
800	0001	0001
800	0005	0003
800	1000	0002

BADI Name: VENDOR\_ADD\_DATA

Method : PRESET\_VALUES\_PORG

Implement the BADI through SE19.

```
DATA WA TYPE ZS10AM_PPV.  
SELECT SINGLE * FROM ZS10AM_PPV INTO WA WHERE EKORG = E_LFM1-EKORG.  
IF SY-SUBRC = 0.  
E_LFM1-ZTERM = WA-ZTERM.  
ENDIF.
```

#### Steps to work with Screen BADI: -

1. The Screen fields are added to standard data base table through append structure or create a table with screen fields.
2. Based on this table fields we create the Module Pool program & sub screen.
3. This module pool program and sub screen is attached to standard transaction code by using Badi.

#### **→ Add the following subscreen to the MIGO transaction by using BADI**

Here we add the above fields to EKBE table through Append structure or we create the 'Z' table with these fields based on this table fields. We create the module pool program and subscreen.

This screen and program is attached to MIGO transaction by using BADI.

SCREEN 2222

LR Number	<input type="text"/>
Vehicle Number	<input type="text"/>
Trans. Name	<input type="text"/>

#### Steps to create module pool program & screen: -

Execute 'SE38'. Provide the program name (ZS10AM\_MIGO\_SCREEN). Click on create. Provide title. Select type as module pool. Save. Click on local object. Click on display object list. Select the program in left panel. Right click → create → screen. Provide screen number (2222). Provide short description. Select the radio button sub screen. Click on save. Click on layout. Click on dictionary or program fields (F6). Provide the table name which table contains these fields. Select the required fields. Enter. Save, check, activate the screen. Click on back. Double click on the program. Right click → activate.

BADI Name: MB\_MIGO\_BADI

Method Name: PBO\_DETAIL

Badi is implemented through SE19.

```
E_CPROG = 'ZSCREEN990'.
```

```
E_DYNNR = '2222'.
```

```
E_HEADING = 'GR ADITIONAL SCREEN'.
```

Now we execute MIGO transaction and absorb the additional screen in the tab.

#### Steps to attach the additional subscreen to the vendor and customer: -

1. Create the screen-group
2. Activate the screen group
3. Attach the program & subscreen to the screen group.

#### **→ Steps to attach the additional subscreen to the vendor and customer**

In this object the above screen fields are added to LFA1 table through append structure or create a 'Z' table with these fields. Based on these table fields we create a Module Pool Program & design a subscreen. This program & subscreen is attached to screen group.

The following fields are added to LFA1 table through append structure.

```
CSTNO      ZSPCSTNO  CHAR 10  
LSTNO      ZSPCSTNO  CHAR 10  
PANNO      ZSPPANNO  CHAR 10
```

Based on these fields we create the module pool program & subscreen.

Sub screen 1011

CST Number	<input type="text"/>
LST Number	<input type="text"/>
PAN Number	<input type="text"/>

**Module Pool Program:** ZS10AM\_XK01\_SCREEN.

**Screen:** 1011

**Steps to create screen group: -**

Execute ‘SPRO’. Click on SAP reference IMG. Expand logistics – General. Expand business partner. Expand vendors. Expand control. Expand adoption of customers own master data fields. Execute prepare modification – free enhancement of vendor master data. Enter. Click on new entries. Provide the screen group (SP), short description (Screen group). Save. Select the screen group. Double click on label tab pages. Click on new entries in the application tool bar. Provide the number (111), function code (SPF), description (Tax details). Save, back to initial group.

**Steps to activate the screen group: -**

Execute processing at Master data enhancement. Select any one of the implementation. Click on create (F5). Provide implementation name (ZS10AM\_ASGROUP). Enter. Provide short description. Save. Click on create enhancement implementation. Provide the same implementation name (ZS10AM\_ASGROUP). Short description. Enter. Select the implementation name. Enter. Click on interface tab. Double click on check\_add\_on\_activate method. Click on signature in the application tool bar. Identify the input output parameters and implement the logic.

```
If I_SCREEN_GROUP = 'SP'.
E_ADD_ON_ACTIVE = 'X'.
Endif.
```

Save, check, activate the method. Click on back. Save, check, activate the BADI.

**Steps to attach program & subscreen to screen group: -**

Execute customer subscreens. Select any one of the implementation. Click on create. Provide the implementation name (ZS10AM\_ASSCREEN). Enter. Provide short description. Click on attributes tab. Click on insert row (Plus button +). Provide the screen group (SP). Click on save. Click on create. Enhancement implementation. Provide the same implementation name (ZS10AM\_ASSCREEN), short description. Enter. Select the implementation name. enter. Click on interface tab. Double click on GET\_TAXI\_SCREEN method. Click on signature in the application tool bar. Identify the input output parameters & implement the logic.

```
If I_TAXI_FCODE = 'SPF'.
E_SCREEN = '1011'.
E_PROGRAM = 'ZS10AM_XK01_SCREEN'.
E_HEADERSCREEN_LAYOUT = ''.
Endif.
```

Save, check, activate the method. Click on back. Save, check, activate the BADI.

**Steps to check whether the screen is added to vendor transaction or not: -**

Execute ‘XK01’. Provide the vendor number, account group. Enter and absorb the screen group in the application tool bar.

**➔ Implement the BADI to throw an error message if they pass invalid cost center at the time of goods issue against cost center.**

In the real time they issue the goods against cost center in two ways.

1. MIGO transaction code with 201 movement type
2. MB1A transaction with 201 movement type

In the client first four digit of cost center is plant number.

1. BWART → Movement type
2. KOSTL → Cost Center
3. WERKS → Plant

#### **Logic:** -

If BWART = '201'.

If WERKS <> KOSTL + 0(4).

MESSAGE E000 (ZMESSAGE1) WITH 'COST CENTER ISN'T BELONGS TO THIS PLANT'.

ENDIF.

ENDIF.

#### **MIGO Transaction with 201 movement type**

BADI Name : MB\_MIGO\_BADI

Method Name: PUBLISH\_MATERIAL\_ITEM

```
IF LS_GOITEM-BWART = '201'.
  IF LS_GOITEM-WERKS <> LS_GOITEM-KOSTL+0(4).
    MESSAGE E000(ZMESSAGE1) WITH 'COSTCENTER IS NOT BELONGS TO THIS
PLANT'.
  ENDIF.
ENDIF.
```

#### **MB1A WITH 201 Movement type**

BADI Name : MB\_QUAN\_CHECK\_BADI

Method Name: CHECK\_ITEM\_DATA

```
IF IS_MSEG-BWART = '201'.
  IF IS_MSEG-WERKS <> IS_MSEG-KOSTL+0(4).
    MESSAGE E000(ZMESSAGE1) WITH 'COSTCENTER IS NOT BELONGS TO
THIS PLANT'.
  ENDIF.
ENDIF.
```

#### **Enhancement Spot:** -

Enhancement Spot is the collection of BADI definitions. If you want to create a BADI then first we create the enhancement spot. In that we create the BADI definition.

**Note:** - We can't create the transaction code for enhancement.

#### **Enhancement frame work:** -

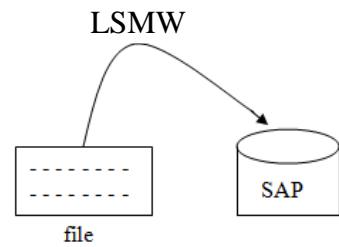
Enhancement frame work is the collection of implicit enhancement & explicit enhancement. Now-a-days user exits are called implicit enhancements. In the old version if you want to implement the user exits then we must provide the access key (access key is provided by BASIS people). Now-a-days we implement the user exits through implicit enhancement.

The Enhancement Framework enables you to add functionality to standard SAP software without actually changing the original repository objects, and to organize these enhancements as effectively as possible. With this new technology you can enhance global classes, function modules, Web Dynpro ABAP components, and all source code units using implicit enhancement options provided by the system. An application developer can also define additional explicit enhancement options for source code plug-ins and new kernel-based BAdIs, which are also integrated in this new framework.

# LSMW

(Legacy Systems Migration Workbench)

LSMW is used to upload the data from file to SAP system.



Differences between LSMW and BDC

## LSMW

1. LSMW is purely designed for functional people who don't do much coding.
2. In LSMW Mapping is done by system itself.
3. LSMW offers 4 techniques
  - a) Direct input recording
  - b) Batch input recording
  - c) BAPI
  - d) IDOC
4. In LSMW the flat file must be .TXT file.
5. LSMW is only possible for standard transaction code.

## BDC

- 1) BDC is designed for technical people who knows much coding.
- 2) In BDC mapping is done by technical people
- 3) BDC offers two techniques
  - a) Call Transaction Method
  - b) Session Method
- 4) In BDC the flat file is either .TXT file or Excel sheet.
- 5) BDC is possible for any transaction code.

**Note:** - The transaction code for LSMW is 'LSMW'.

At the time of creating the LSMW we need to provide project (Module name), sub project (application), object (task).

→ **Develop a conversion program to upload the vendor master data from flat file to SAP system by using LSMW (Batch input recording method). The flat file contains vendor numbers, names and search terms.**

### Steps to work with LSMW: -

Execute LSMW. Provide the project name (ZSMM), sub project name (ZSVEN) object name (ZSVC). Click on create in the application tool bar. Provide short description of project. Enter. Provide short description of sub project. Enter. Provide short description of object. Enter. Click on execute.

**STEP 1 Maintain object attributes:** Click on execute (Ctrl + F8). Click on change mode. Select the radio button Batch Input Recording. Click on over view icon in the right side. Place the cursor on recordings. Click on create recording. Provide the recording name & short description. Enter. Provide the transaction code (XK01). Enter. Provide the vendor number (H2020), account group (0004). Click on enter. Provide the name (BIG C), search terms (BC), country (KW). Save. Click on default all in the application tool bar. Save. Back, come back. Provide the recording name (SXK01). Save, back.

**STEP 2 Maintain source structures:** Click on execute. Click on change mode. Place the cursor on source structure. Click on create structure. Provide source structure is internal table name and provide short description. Enter. Save, back.

**STEP 3 Maintain source fields:** Click on execute. Click on change mode. Place the cursor on internal table name. Click on change mode. Click on table Maintainece icon in the application tool bar. Provide field names. (Data types & lengths which is similar as file).

LIFNR	C	10	VENDOR
NAME1	C	35	NAME
SORTL	C	10	SEARCH TERM

Save, back. Save, back.

**STEP 4 Maintain structure relations:** Click on execute. Click on change mode. Save, back.

**STEP 5 Main field mapping and conversion rules:** Click on execute. Click on change mode. Select the each field. If the field is coming from flat file then click on source field. Otherwise click on rule in the application tool bar. Select the radio button constant. Enter. Provide the constant value. Save, back.

**STEP 6 Specify files:** Click on execute. Click on change mode. Select the legacy data on the PC (Frontend). Click on create or adding on top. Browse the filename. Provide short description. Select the delimiter is comma, enter. Save, back.

**STEP 7 Assign files:** Click on execute. Click on change mode. Save, back.

**STEP 8 Read data:** Click on execute. Click on execute. Back, click on back.

**STEP 9 Display read data:** Click on execute. Enter. Click on back.

**STEP 10 Convert data:** Click on execute. Click on execute. Back, click on back.

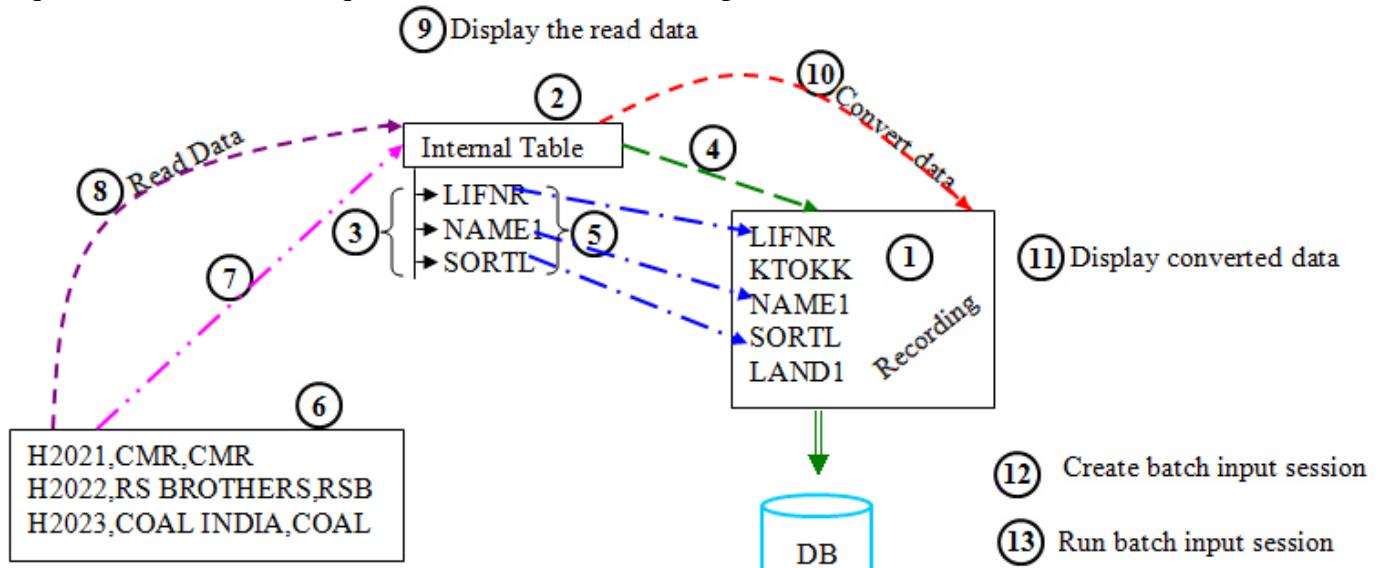
**STEP 11 Display converted data:** Click on execute. Enter. Back.

**STEP 12 Create batch input session:** Click on execute. Select the check box keep. Execute. Enter.

**STEP 13 Run batch input session:** Click on execute. Select the session name. Process. Click on process.

Next on wards the end user execute the same LSMW from 6<sup>th</sup> step on wards for each flat file.

**Steps to create the request number to LSMW:** - In the menu bar click on extras → Generate change request. Click on create request. Provide the short description. Enter.



→ Develop a conversion program to upload the vendor master data from flat file to SAP system by using LSMW. (Direct input method). The flat file contains vendor numbers, names & search terms.

**Steps:** Execute LSMW. Provide project name, sub project name & object name. Provide short description. Execute.

**STEP 1 Main Object Attributes:** Click on execute. Click on change mode. Select the radio button. Standard batch/ direct input method. Select the object – (0040), select the method (0001) [standard]. Save, back.

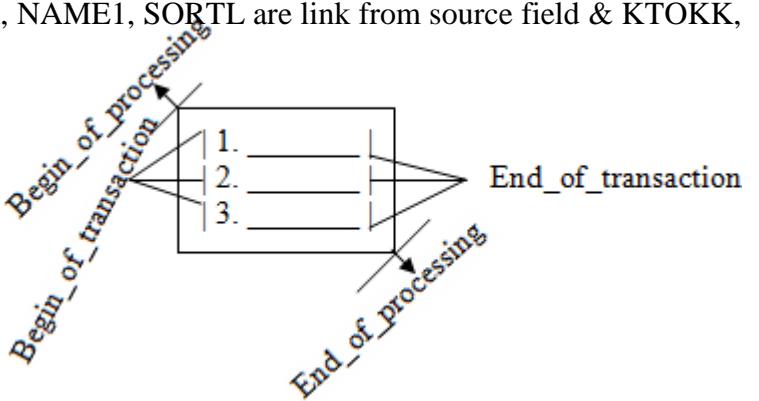
**STEP Maintain Structure Relations:** Click on execute. Click on change mode. Select our required structure & click on relationship. Here we establish the relationship to BLA1. Save. Back.

**STEP Maintain field mapping and conversion rules:** Click on execute. Click on change mode. Select the transaction code field. Click on rule. Select the constant. Enter. Provide the transaction code (XK01). Enter. Provide the rest of the mandatory (LIFNR, NAME1, SORTL are link from source field & KTOKK, LAND1 are link from Rule). Save, back.

Rest of the steps are same as previous method.

#### Events in LSMS:

1. Global Data
2. Begin\_of\_processing
3. Begin\_of\_transaction
4. End\_of\_transaction
5. End\_of\_processing



**Global data:** - It's an event which is used to declare the variables, work areas & internal tables which are needed to implement the logics.

**Begin\_of\_processing:** - It's an event which triggered at the time of entire flat file processing start.

**Begin\_of\_transaction:** - It's an event which is triggered at the time of each record processing start.

**End\_of\_transaction:** - It's an event which is triggered after completion of each record processing.

**End\_of\_processing:** - It's an event which is triggered after completion of entire flat file processing.

**Note:** - LSMW events are available in 5<sup>th</sup> step. That field mapping & conversion rules.

**→ Develop a conversion program to update the vendor city from flat file to SAP system by using LSMW (batch input recording method). Through XK02 transaction. The flat file contains vendor numbers & cities & also download the error vendor which aren't created.**

Execute LSMW. Provide the project name, sub project name, object name. Click on create. Provide short descriptions. Execute Maintain object attributes. Execute. Change mode. Select the radio button batch input recording. Click on over view icon in the right side. Select the recording. Click on create. Provide the recording name (SXK02). Short description. Enter. Provide transaction code (XK02). Enter. Provide the vendor which is already created. Select the address check box. Enter & update the city. Click on save. Click on default all.

**Note:** - When ever we are working with update transaction then we must remove the other than flat file fields recording steps from recording.

Here we remove the NAME1, SORTL, LAND1 recording steps by using ( - ) screen field in the application tool bar. Save, back. Click on back. Provide the recording name. Save, back.

In main field mapping and conversion rules: execute. Click on change mode. Provide the mapping. (LIFNR ORT01 from source field, D0110 from rule constant as X). Click on layout (Ctrl + F7) in the application tool bar. Select the Global data definitions, Processing time check boxes. Enter. Then we get the events. Double click on global data. Declare the work area & internal tables.

Data: Begin of wa\_error,

    LIFNR type lfa1-lifnr,

    End of wa\_error.

Data it\_error like table of wa\_error.

Data V type lfa1-lifnr.

Click on save. Check, back. Click on save. Double click on begin of transaction. Implement validation logic.

Select single lifnr from lfa1 into V where lifnr = itab1-lifnr.

If sy-subrc <> 0.

Wa\_error-lifnr = itab1-lifnr.

Append wa\_error to it\_error.

Skip\_transaction.

Endif.

Save, check, back. Double click on end of processing.

Call function 'GUI\_DOWNLOAD'

EXPORTING

    FILE NAME = 'C:\VIDYA.TXT'

    WRITE\_FIELD\_SEPERATOR = 'X'

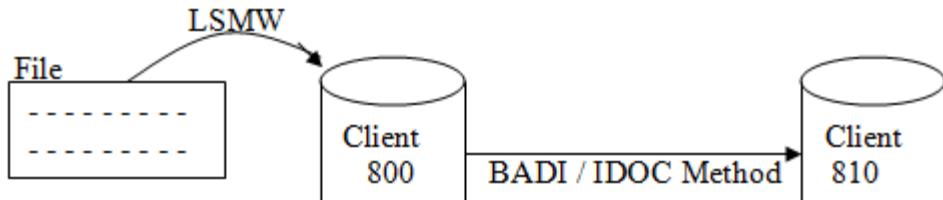
TABLES

    DATA\_TAB = IT\_ERROR.

Save, check, back. Save, back. Rest of the steps are same as previous one.

**Note:** - When ever we are working with BAPI methods & idoc methods then we must create a file port in WE21 transaction. In the real time file ports are provided by BASIS people.

**Note:** - In the LSMW BAPI and Idocs methods acts like inbound.



**Steps to create file port:** - Execute WE21. Expand the file. Select the SUBSYSTEM file port. Click on copy in the application tool bar. Provide the to port (SPT). Enter. Select the check box Unicode format in system settings. Click on save.

**→ Develop a conversion program to upload bank details from flat file to SAP system by using LSMW (BAPI method) the flat file contains bank country key, bank key & bank name.**

**STEPS** Execute LSMW. Provide project name (ZSFI), sub project name (ZSBAN), object name (ZSBC). Click on create. Provide short description. In the menu bar click on settings → IDOC inbound processing. Provide the file port (SPT). Partner type (LS), partner number (SP800). Click on activate idoc inbound processing. Click on yes. Save, back, execute.

**Maintain object attributes:** -Execute. Click on change mode. Select the radio button BAPI. Select the business object of Bank (BUS1011), method (Create), save, back.

**STEP** Start idoc generation: execute, click on execute.

**STEP** Start IDOC processing: Execute. Click on Execute & absorb the status. If the status is '53', it's success.

**→ Develop a conversion program to upload customer master data from flat file to SAP system by using LSMW (Idoc method). The flat file contains customer number, names, search terms & street.**

**STEP** Execute LSMW. Provide project (ZSSD), subproject (ZSCUS), object. Click on create. Provide short descriptions. In the menu bar click on settings → idoc inbound processing. Provide the file port (SPT), partner type (LS), partner number (SP800). Click on activate Idoc inbound processing. Click on back. Execute.

**STEP** Maintain object attributes: Execute. Click on change mode. Select the radio button IDOC. Provide the message type (DEBMAS), idoc type (DEBMAS06). Save, back.

### **Performance issues / coding standards: -**

Whenever we read the single record from the internal table based on the condition, then we must use binary search because binary search is faster than the linear search.

**Note:** - Before applying the binary search we must sort the internal table based on the condition field.

#### **Syntax: -**

Read table <internal table> into <work area> with key <condition> binary search.

Ex: -

```
SORT IT_EKKO BY EBELN.  
LOOP AT IT_EKPO INTO WA_EKPO.
```

```
-----  
-----  
READ TABLE IT_EKKO INTO WA_EKKO WITH KEY EBELN = WA_EKPO-EBELN BINARY SEARCH.
```

```
-----  
-----  
ENDLOOP.
```

#### **Avoid the dead code: -**

After completion of the program we perform the extended program check to avoid the unnecessary declarations of the program.

#### **Steps to identify the unnecessary declarations: -**

Open the program in SE38. In the menu bar click on program → check → extend program. Check. Execute. Identify the warnings & errors. Double click on it. Absorb the unnecessary declarations from the code.

We must maintain the order of the fields in the work area as well as order of the fields in the select query must be the similar order of the fields in the table.

BUTXT ORT01 BUKRS

X Select butxt ort01 bukrs from t001 into table it\_t001.

(In this case we aren't follow the order of the fields in the table. So it takes extra time to execute).

✓ Select bukrs butxt ort01 from t001 into table it\_t001.

BUKRS BUTXT ORT01

#### **Steps to identify the order of the field in the table: -**

Execute SE11. Open the table (KNA1). Click on contents. In the menu bar click on settings → format list → choose fields. Click on deselect all. Select the required fields check box by using find function key. Enter. Provide the minimum number of hits (2). Execute. Identify the order of the fields.

Instead of Inner join we always use for all entries when we fetch the data from more than two tables. Inner join picks the data, based on the 'ON' condition first, next it based on where condition. For all entries pick the data based on where condition first, next it based on 'ON' condition.

**Note:** - At the time of implementing the SAP the data bases contains fewer amounts of data then the performance of inner join & for all entries is same. Day by day the size of the data base table is increased. The performance of the inner join is decreased & for all entries performance is same.

Whenever we are working with for all entries, then we must consider following things.

- Check the higher-level internal table having the data or not.
- Sort the higher level internal table
- Delete adjacent duplicates from higher level internal table, if it's required.
- consider all the possible conditions in the where condition of the next level select query

1.If higher level internal table having no data then it'll fetch entire data of data base for the next level.

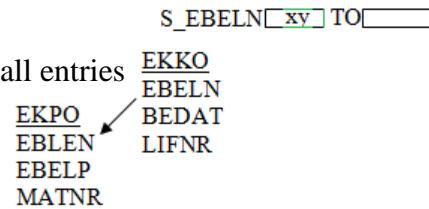
Select ebeln bedat lifnr from ekko into table it\_ekko where ebeln in s\_ebeln.

If not it\_ekko is initial.

Select ebeln ebelp - - - from ekpo into table it\_ekpo for all entries

In it\_ekko where ebeln = it\_ekko-ebeln.

Endif.



In the For all entries when ever we fetch the master data based on the transactional data or When ever we fetch the header data based on the item data then we must apply delete adjacent duplicate in the higher level internal table.

**Note:** - Before using delete adjacent duplicates we must sort the internal table based on comparing fields.

Select EBELN BEDAT LIFNR from EKKO into table it\_ekko where Ebeln in s\_ebeln.

It\_ekko1 = it\_ekko.

Sort it\_ekko1 by lifnr.

Delete adjacent duplicates from it\_ekko1 comparing lifnr.

If it\_ekko1 is not initial.

Select lifnr name1 ort01 from lfa1 into table it\_lfa1 for all entries in it\_ekko1 where lifnr = it\_ekko1-lifnr.

endif.

IT_EKKO			IT_EKKO1			IT_EKKO1			MAKT		
EBELN	BEDAT	LIFNR	EBELN	BEDAT	LIFNR	EBELN	BEDAT	MAKTX	MAKT	POSNR	MATNR
30003	01.08.2000	1910	30003	01.08.2000	1910	30003	01.08.2000				VBELN
30004	02.07.2002	2520	30004	02.07.2002	2520	30005	03.09.2004				VBELN
30005	03.09.2005	1910	30005	03.09.2005	1910	30007	08.09.2011	1910			MAKT
30006	07.09.2009	2520	30006	07.09.2009	2520	30004	02.07.2002	2520			POSNR
30007	08.09.2011	1910	30007	08.09.2011	1910	30006	07.09.2009	2520			MATNR

**EX:** -

Select vbeln posnr matnr kwmeng meins netwr from vbap into table it\_vbap where vbeln in s\_vbeln.

It\_vbap1 = it\_vbap.

Sort it\_vbap1 by matnr.

Delete adjacent duplicates from it\_vbap1 comparing matnr.

If it\_vbap1 is not initial.

Select matnr maktx from makt into table it\_makt for all entries in it\_vbap1 where matnr = it\_vbap1-matnr and spras = sy-langu.

Endif.

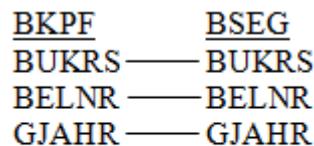
**Consider all the possible conditions in the next level select query:** -

If not it\_bkpf is initial.

Select BUKRS BELNR GJAHR - - - from BSEG into table it\_BSEG

For all entries in it\_bkpf where bukrs = wa\_bseg-bukrs and

Belnr = wa\_bseg-belnr and gjahr = wa\_bseg-gjahr.



6. We always use select single if you know entire primary key combination of data base table.

Select single bukrs kunnr akont from knb1 into table it\_knb1 where bukrs = \_\_\_\_\_ and kunnr = \_\_\_\_\_ .

We always use select up to 1 rows. If you know part of the primary key combination of data base table.

Select bukrs kunnr akont from knb1 into table it\_knb1 up to 1 rows where bukrs = \_\_\_\_ .  
Endselect.

**Note:** - Select single is faster than select up to 1 rows. Because select singe hits the data base once & up to 1 rows hits twice.

7. We never use ranges, instead of ranges we use select-options with no display.

**X** Tables t001.

Ranges s\_bukrs for t001-bukrs.

**✓** Tables t001.

Select-options s\_bukrs for t001-bukrs.

8. We always use list display instead of grid display. Because list display is faster than grid display.

9. We never write select \*. Instead of this we use select fields (field 1, field 2, - - field n).

**X** Select \* from ekko into table it\_ekko where Ebeln in s\_ebeln.

**✓** Select Ebeln bedat lifnr - - - from ekko into table it\_ekko where Ebeln in s\_ebeln.

10. We never write a select query with out where condition.

**X** Select bukrs butxt ort01 from t001 into table it\_t001.

11. We never write a select query into corresponding fields.

**X** Select butxt ort01 bukrs from t001 into corresponding fields of table it\_t001 where bukrs in s\_bukrs.

**✓** Select bukrs butxt ort01 from t001 into table it\_t001 where bukrs in s\_bukrs.

12. We never write a select query with in a select query. (we never write nested selected query).

**X** Select bukrs butxt from t001 into wa\_t001 upto 1 rows where bukrs = p\_bukrs.

Select bukrs kunnr from knb1 into wa\_knb1 upto 1 rows where bukrs = wa\_t001-bukrs.

Endselect.

Endselect.

Output:-

BUKRS	BUTXT	ORT01

IT\_T001

1000 TCS 241  
1000 TCS 291  
2000 IBM 116

13. We never write a select query within a loop.

Loop at it\_vbap into wa\_vbap.

Wa\_final-vbeln = wa\_vbap-vbeln.

Wa\_final-matnr = wa\_vbap-matnr.

**X** Select single matnr maktx from makt into wa\_makt where matnr = wa\_vbap-matnr and spras = sy-langu.

Wa-final-maktx = wa\_makt-maktx.

Append wa\_final to it\_final.

Endloop.

#### 14. Instead of nested loops we use parallel cursor technique / model

In the real time some times we must use loop inside a loop to fill the final internal table. If the internal table data is increased then the performance of the system is decreased or it takes more time to execute the program. To avoid this or to over come this we use parallel cursor method.

Sort it\_ekpo by Ebeln.

Loop at it\_ekko into wa\_ekko with key Ebeln = wa\_ekko-ebeln.

read tabe it\_ekpo into wa\_ekpo with key ebeln = wa\_ekko-ebeln.

If sy-subrc = 0.

V\_no = sy-tabix.

Loop at it\_ekpo into wa\_ekpo from v\_no.

If wa\_ekko-ebeln <> wa\_ekpo-ebeln.

Exit.

Else

----

----

Endif.

Endloop.

Endif.

Endloop.

IT\_EKKO

EBELN	BEDAT	LIFNR
30004	01.05.2000	1191
30005	07.07.2007	1192
3006	08.09.2010	1193

IT_EKKO	EBELN	BEDAT	LIFNR
WA_EKKO	30004	01.05.2000	1191
	30005	07.07.2007	1192
	3006	08.09.2010	1193

IT\_EKPO

EBELN	EBELP	MENGE	MEINS	NETPR
300004	01	15	KG	100.00
300004	02	02	PCS	200.00
300005	01	10	BOX	500.00
300005	02	20	KG	200.00
300005	03	10	LIT	400.00
30006	01	05	KG	200.00

WA\_EKPO

EBELN	EBELP	MENGE	MEINS	NETPR

#### 15. We always use assignment (=) operator instead of move corresponding.

X Move-corresponding wa to wa1.

WA

BUKRS	BUTXT	ORT01
1000	TCS	HYD

✓ Wa1-bukrs = wa-bukrs.  
Wa1-ort01 = wa-ort01.

WA1

BUKRS	ORT01	KUNNR
1000	HYD	

Note: - We never use the sort inside the loop.

Loop at it\_ekko into wa\_ekko.

---

X Sort it\_lfa1 by lifnr.

Read table it\_lfa1 into wa\_lfa1 with key lifnr = wa\_ekko-lifnr binary search.

Endloop.

#### 16. We always use the events. We always avoid the (LDBs).

We always avoid the fetching the data from LDB s (Logical Database). LDB is the collection of data base tables. LDB s are used in HR ABAP not in ABAP. LDB s are created through SE36.

17. We always use ‘WHILE’ instead of ‘DO’ because ‘WHILE’ is faster as well as clearer than ‘DO’.
18. We always use case instead of nested if. Because case is faster than ‘IF’.
19. Before writing the select query its better to use ‘SAPGUI\_PROGRESS\_INDICATOR’ function module to know the execution process.

EX: -

Data it\_bseg like table of it\_bseg.

Call function ‘SAPGUI\_PROGRESS\_INDICATOR’

Exporting

Percentage = ‘100’.

Text = ‘Fetching data from BSEG tale’.

Select \* from BSEG into table it\_bseg.

Call function ‘SAPGUI\_PROGRESS\_INDICATOR’

Exporting

Percentage = ‘100’

Text = ‘Display the output’.

Call function ‘REUSE\_ALV\_GRID\_DISPLAY’

Exporting

I\_structure\_name = ‘BSEG’

Tables

T\_outtab = IT\_BSEG.

### **Types of project:-**

- a. Implementation project
- b. Upgradation project
- c. Maintenance / Support project
- d. Roll out project

**Implementation project:** - When ever we want to develop the SAP from fundamental on wards then we go for implementation project.

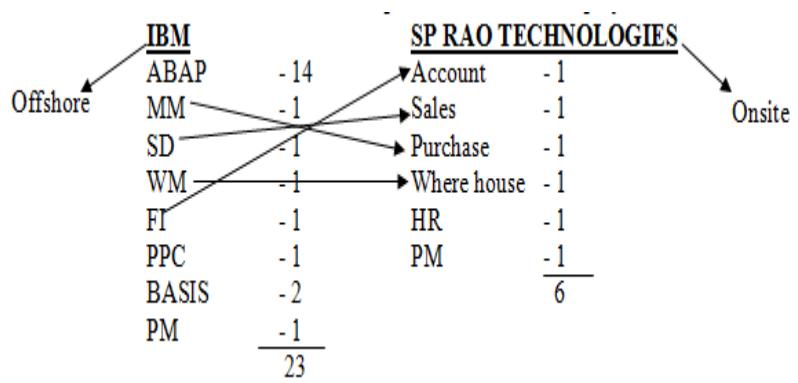
### **Phases of Implementation Project:-**

- a. Project preparation phase
- b. Business Blue Print phase
- c. Realization phase
- d. Final preparation phase
- e. Go-live & support phase

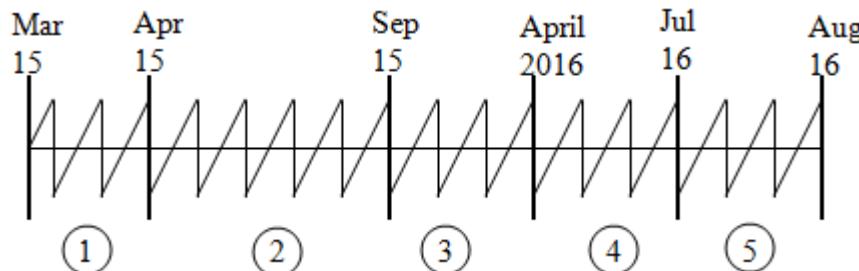
### **Project preparation phase:** -

During this phase the following activities will happened or performed.

- The offshore as well as onsite companies identified their project team members.



- Provide the necessary training to the employees.
- Design or prepare the project pipeline path to reach the destination.



- Kickoff (Project actual start date)

#### **Business Blue Print phase:** -

During this phase the functional people interact with process leads & collect the requirements & prepares the business blueprints & later offshore & onsite people sign off in the business blue prints.

#### **Realization phase:** -

During this phase functional people prepares the FD documents based on the business blue prints. These FS documents will send to technical team leader via E-mail or client website. After the technical team leader receives the FS (Functional Specification) documents, they convert the FS documents to detailed technical specifications (TS). These TS documents are assigned to ABAP consultant. Based on the TS documents the ABAPer develop the object in the development screen & prepares the unit test plan (UTP) & transport the objects to quality server.

All the Business Blue Prints will convert into FS. All the FS are converted into TS. All the TS are developed as objects. All the objects are moved into development server / quality server.

#### **Final Preparation phase:** -

During this phase the functional people test the each & every object in the quality server & prepare the test scripts (they prepare a document to execute the program) & provide the trainings to the end users.

#### **Go live & support phase:** -

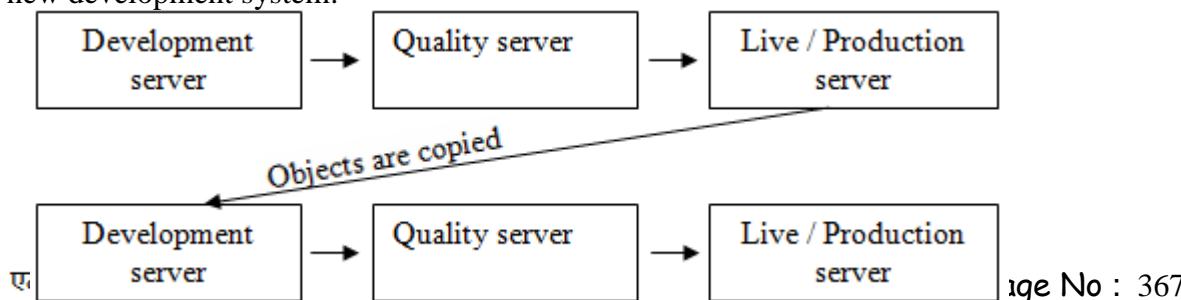
Go-live means all the objects are moved from quality to live server. After Go-live the company provides the support in day to today activities of client until a period.

#### **Upgradation project:** -

When ever the version is changed if you want to implement the new technologies into our existing systems then we go for up gradation project. There are two types of upgradations.

1. Technical Upgradation
2. Functional Upgradation

In the technical upgradation more ABAPers & BASIS people are involved & very few functional people are involved. It can take 30 days to 3 months time. In the technical upgradation first the basis people install the new SAP version later they copy the objects from existing production system to new development system.



### **Phases on up gradation project:-**

1. SPDD Phase
2. SPAU Phase
3. DBACOCKPIT
4. UCCHECK
5. Replace the function module

**Note:** - The ABAPer perform the above phases in the development of new server.

#### **SPDD phase: -**

During this phase, it compares the dictionary objects (tables, domains, etc) from old version to new version. After executing SPDD transaction some of the objects are displayed in green colour some of the objects are displayed in pink colour. Open the each & every pink colour object & activate the either old one or new one with the help of functional people.

#### **SPAU phase: -**

During this phase we compare the work bench objects (function module, programs) from old version to new version. After executing SPAU transaction some of the objects are displayed in green colour some of the objects are displayed in pink colour. Open the each & every pink colour object activate the either old one or new one.

#### **DBACOCKPIT: -**

During this phase we will identify all the data base tables which secondary index is in active. After executing the DBACOCKPIT transaction if displays the data base tables which secondary index is inactive.

Open the each & every table in 'SE11'. Click on indexes in the application tool bar. Activate the secondary index.

#### **UCCHECK phase: -**

In the old, each & ever character is identified by 32 bit. Now a day it identifies through 64 bit. After executing the UCCHECK transaction it displays all the programs which unique code is inactive. Then open the each & every program in SE38 in change mode. In the menu bar click on go to → attributes. Select the check box unique code. Check activated.

#### **Replace the function modules: -**

When ever the version is changed, some times some function modules are absolute (don't use). Instead of this function module we use new function module (Remove the old function module code & replace the new function module).

#### **Steps to identify the programs which contains absolute function module: -**

Execute SE37. Provide the function module (upload). Click on where used list (Ctrl + Shift + F3) in the application tool bar. Select the check box only programs. Enter. Select the program which contains the function module code. Remove this code & replace the new code in all the programs.

#### **Maintaince / Support project: -**

After Go-live & support phase in the implementation project each & every company requires 24/7 support. Then they go for support project. In the support project each & every company uses ticketing tools (remedy tools). If any issues are occurs or rose in the onsite then the process leads login into ticketing tools and raise the ticket & send to Offsite Company.

After receiving the ticket the offshore front office person forward that ticket to their respective functional people based on the ticketing category. After receiving the ticket the functional people discuss with core team members / process leads and prepares the FS document & sends the FS document to technical team leader.

After receiving the FS document then technical team leader converts the FS to detail technical spec and provide the estimated time based on the complexity of the object & assign the TS document to ABAP consultant.

After receiving the TS document the ABAP consultant develop the object in development server and prepare the UTP (Unit Test Plant) & send to quality server. In quality server the functional people test the object & prepare the UAT (Unit Acceptance Test) & transport the object to live server.

In the live server the process leads test the object if it's ok and close the ticket.  
The entire ticketing tool against the ticketing number the progress details are updated by function people.

### **Roll out project:** -

When ever we want to implement the new model in existing SAP system or when ever we purchase a new company if we want to extend the same SAP to that company then we go for roll out project. Roll out projects are also called one type of implementation of project.

**Note:** - In the real time at the time of implementation project we use ASAP methodology.

### **SAP notes:** -

Whenever the company implements the new technologies (Support packs, SAP patches) in the existing system sometimes damages anywhere in the SAP. If you want to overcome those damages we identify the right note & implement.

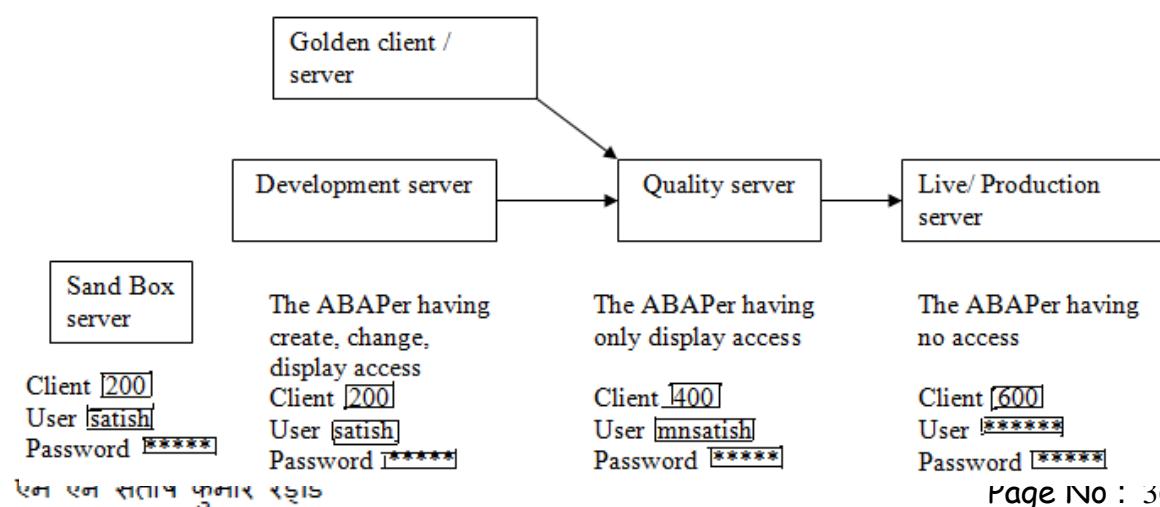
### **EX:** -

In my company at the time of implementing the HR support packs it damages the all other layout lines here we identify the SAP note based on the documentation which provided by SAP. We download the SAP note from service through SAP.Com website with the help of basis people. Later we implement the note in the development note through 'SNOTE' transaction. (We manually also apply the note & create the request number & transported the request number from development server to quality & live server).

SAP NOTE contains which programs are changed & which lines are comments and which new lines or code added in the program.

At the time of implementing the note manually then we need to open the program in change mode then it will ask access key take this screen shot and send a mail to BASIS people then the BASIS people provide the access key.

### **Real time system landscape:** -



In the development server the ABAPer having create, change, display access of ABAP transaction codes (SE11, SE12, SE16, SE38, SE93, SE91, SE80, SE41, SE51, SE09, SE10, - -) & only display access of transactional t.codes (XK03, XD03, MM03, ME23N, - -). Whenever we develop any conversion programs, enhancements if we need create access of any transactional t.codes then we put a mail to security people (Part of BASIS) then the security people take conversion from technical team leader & provide access.

In the real time after we receive the logon details of development server we login in the development client then navigated to change password. Then change the password. After change the password we create a test program in ABAP editor then it'll ask developer key at the first time. Then take a screen shot & sends to BASIS people. Then the BASIS people develop the developer key. Based on this developer key we logon into the system & implement the program. From next time onwards it won't ask any developer key. Whenever we execute any ABAP related t.code if you get the error message i.e. you aren't authorized for the transaction t.code (xyz). Then we execute '/OSU53' transaction. Take a print screen & put a mail to BASIS people then they provide access based on the screen shot.

#### **Sand box server:**

Sand box contains configuration settings as well as master & transactional data. We can develop any object in the Sand box for testing purpose.

#### **Golden server / golden client:**

Golden server is the sensitive client only. It contains configuration steps. It won't contain any master & transactional data. Golden server is directly connected to quality server. Only functional people have the access in golden server.

Based on the technical specification (TS), the ABAP consultants develop the object in development server based on the coding standards & naming standards & save in our own package & create a new request number. In the real time packages are already created by BASIS people based on the object category or module.

After completion of object the ABAP consultant prepares the Unit Test Plant (UTP).

Sno	Scenario	input	excepted output	actual output
1	Verification	Xyz	Invalid company	Invalid company
2	Display	1000-3000	3 company details are printed	3 company details are printed
3	Output		Company code with red colour, name with hotspot	Company code – red colour, Name - Hotspot

If the ABAP consultant gets the expected output & actual output is same then they informed to BASIS people to transport the request number of the object from development server to quality / test server.

The BASIS people release the request & transfer the request from development server to quality server & inform to ABAP consultant. (In some companies ABAPer release the request through SE09 / SE10 & BASIS people transport the request from development server to quality server through 'STMS' transaction). After receive a mail the ABAP consultant forward the mail to functional people & technical team leader to test the object in quality server. After receive a mail the functional people test the object in the quality server along with process leads & prepares the user acceptance test (UAT).

UAT							
	Sno	Scenario	input	excepted output	actual output	Sign & date	Remarks
एम	1	Verification	Xyz	Invalid company	Invalid company	महाराष्ट्र राज्य 11.02.2014	

If the functional people get the expected output & actual output is same then they informed to BASIS people to transport the same request from quality to live server. Otherwise they informed to ABAP consultant to provide the changes in the code.

In the live server the end users use the object if any changes are needed they informed to functional & technical people. If any changes are coming then we do the changes in development server create a new request & transported to quality & live server.

**Note:** - The object is copied from development server to quality server & copied from quality to live server (the object is available in all servers).

**Note:** - We can create any number of request numbers to the object.

After completion of the object in the development server we perform the following activities.

1. Extended program check (SLIN)
2. SQL trace (ST05)
3. Runtime analysis (SE30)

#### **Extended program check:** -

This is used to identify the unnecessary declarations of the program. The transaction code for extended program check is 'SLIN'.

#### **SQL trace:** -

This is used to identify the execution time of a particular select query. The transaction code for SQL trace is (ST05).

#### **Steps to perform the SQL trace:** -

Open the program in 'SE38'. Place the cursor on select query which select query execution time we want. Click on stop button in the application tool bar. Execute the program. Provide the input, execute. Now the cursor is stops at select query. Execute the ST05 transaction in separate session. Click on activate trace. Now go to debuggers screen. Click on F5 button. (Select query execution completed). Now go to SQL trace screen. Click on deactivate the trace in right side. Click on display trace. Enter. Identify the total execution time of the select query in micro seconds.

#### **Runtime analysis:** -

This is used to identify the total execution time of the function module. The transaction code for runtime analysis is SE30.

#### **Steps to work with runtime analysis:** -

Execute SE30. Select the radio button program. Provide the program name & provide the variant if it's available. Click on execute. Provide the input. Execute. Click on back to SE30 screen. Click on evaluate in the bottom. Then we identify the total execution time in micro seconds.

**Note:** - Runtime analysis also contains tip & tricks to identify which statements we use which we don't use.

**Note:** - After completion of the program we click on pretty printer in the application toolbar to align the code.

**Note:** - In the real time the maximum program execution time in the foreground is 600 seconds or 10 minutes. If the program exceeds 10 minutes then it goes to dump. If you want to know the total execution time of these types of programs then we run the program in background & identify the total execution time 'SM37' transaction.

**Note:** - ‘ST22’ is the transaction code to identify the all dumps.

**Note:** - There is no time limit for the background scheduling jobs.

Whenever we execute the program in background then we get the output in a spool request (SP01) transaction.

**Steps to execute the program in background:** -

Execute ‘SE38’. Provide the program name. Execute. Provide the input. In the menu bar click on program → execute in background. Click on continue. If you want to run the program in a particular date & time then click on date & time button, & provide the date & time. Save. If you want execute the program immediate then click on immediate button. Save it. If you want run the program period wise (daily, monthly, weekly, yearly) then click on period values in the bottom. Select the required filed. Click on save.

**Steps to identify the status of the program which is running background:** -

Execute ‘SM37’. Execute. Identify the status & also identify the total execution time.

Whenever we execute any program in background then it’s called as a job. There are different types of job status.

1. Schedule
2. Released
3. Ready
4. Active
5. Finished

If the job status is finished then only we get the output in spoon. If the job status is active, then the program is currently executed.

**Note:** - ‘SM36’ is the transaction code to schedule the program in background

**Steps to schedule the program in background by using SM36:** -

Execute ‘SM36’. Provide the job name (SJF). Select the job class is ‘C’ (low priority). Click on ‘STEP’ in the application tool bar. Provide the program name (ZSP\_ALV6). Click on save. Click on back. Click on start condition in the application tool bar. Click on immediate or provide the date / time. Click on save. Save. If you want to check the status of job & execution time then we go to ‘SM37’.

**Steps to check / open the output:** -

Execute SP01. Execute. Click on type & identify the output.

‘SE16’ → Data browser

‘SE12’ → Display ABAP dictionary

‘SE38’ → execute the program

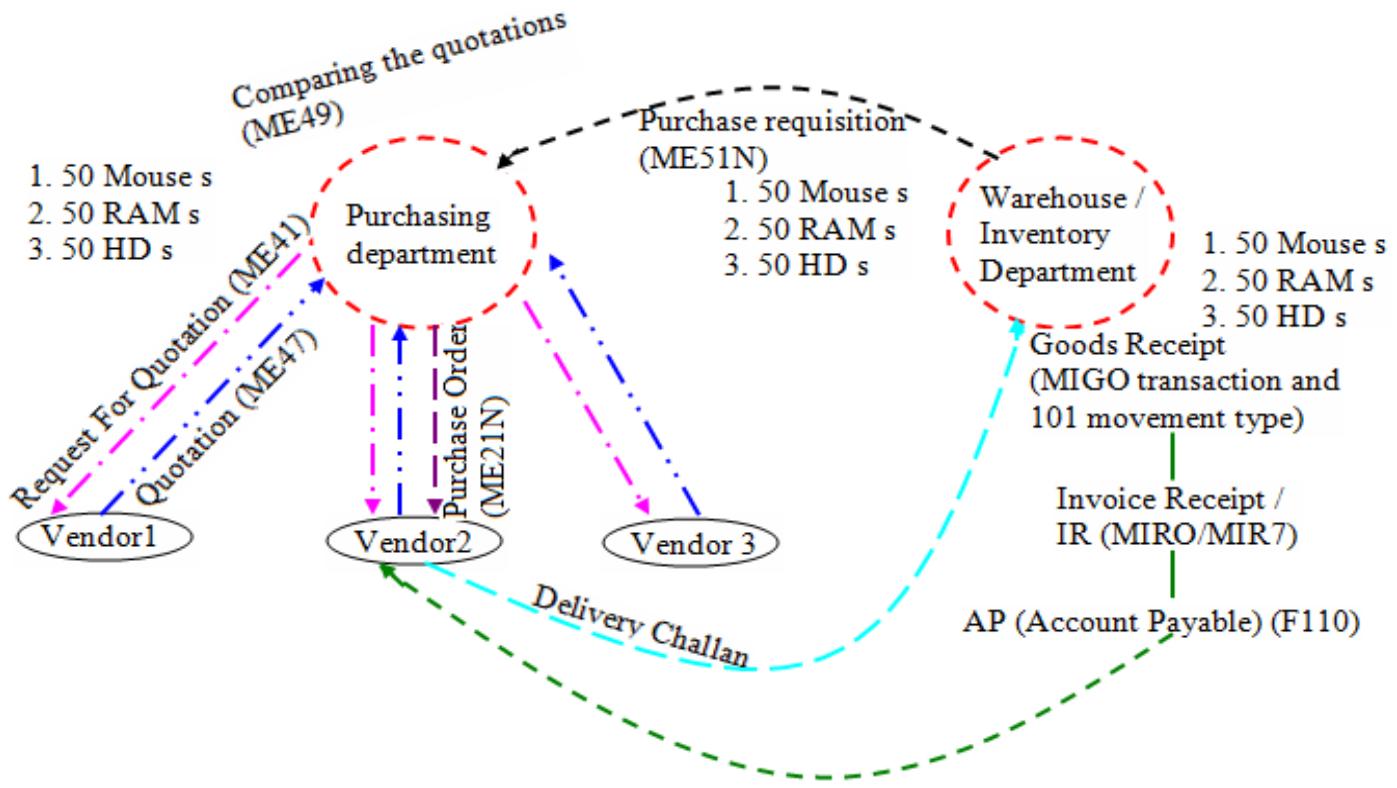
**Note:** - ‘SYST’ is the structure which contains all the system variables.

‘TSTC’ is the data base table which contains all the transaction codes & program names.

‘DD02L’ is standard data base table which contains the all standard data base tables.

‘DD03L’ is the standard data base table which contains all SAP table fields.

# MM Life-cycle



## Purchase Requisition (PR): -

The warehouse or inventory department creates the purchase Requisition with the required materials & sends to purchasing department.

## Request for quotation (RFQ): -

The purchasing department shall ask the vendors to provide the quotation for the materials which are requested by warehouse.

## Quotation: -

The vendor sends the quotation to purchasing department.

## Purchase order: -

After receiving the quotations from the vendor the purchasing department will compare those quotations based on the quality and pricing parameters & identify the best vendor & provide the purchase order to that vendor.

## Delivery Challan: -

The vendor people supply the goods to warehouse with delivery Challan.

## Goods Receipt (GR): -

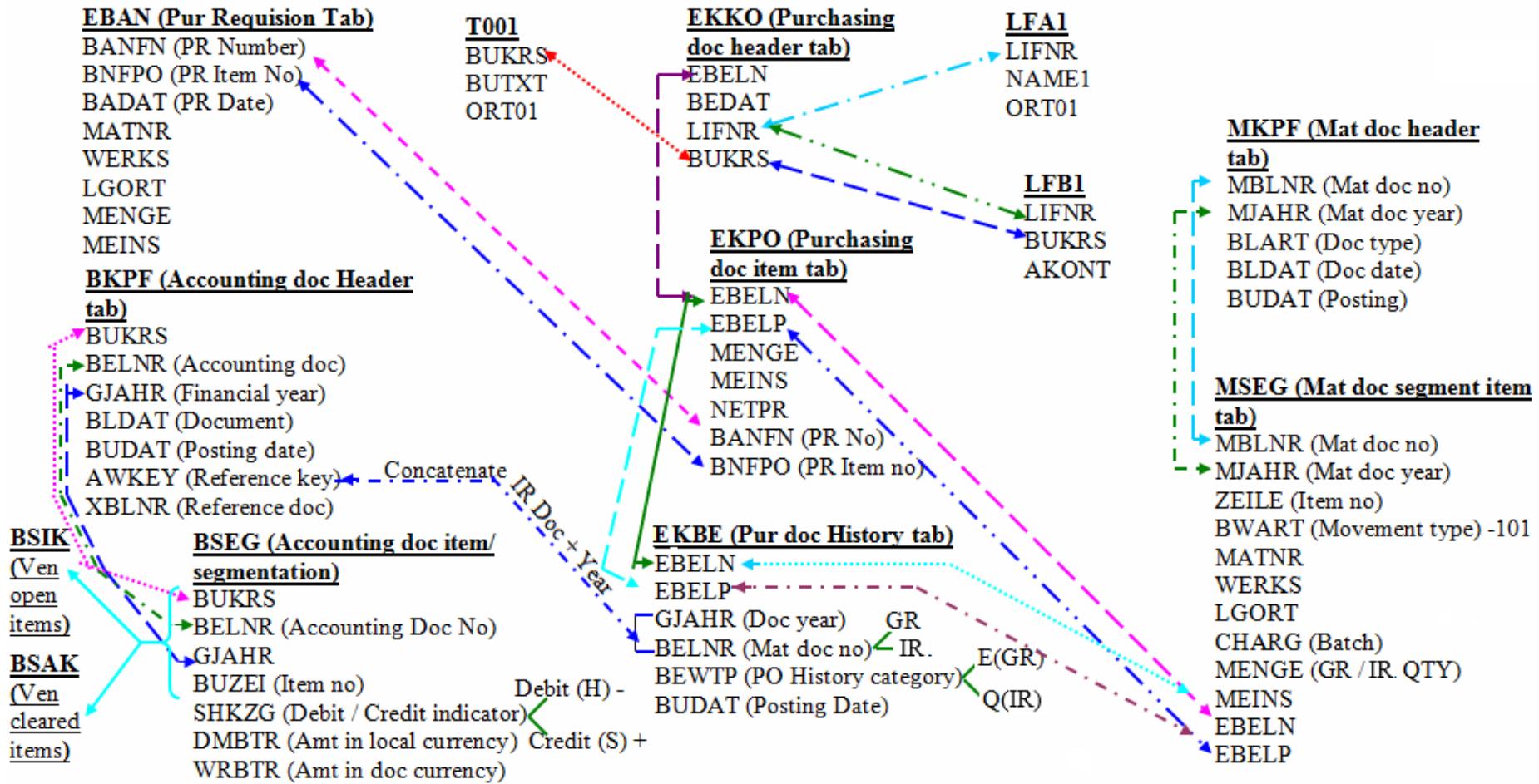
Against the purchase order the warehouse people create the goods receipt with 'MIGO' transaction & 101 movement time.

## Invoice receipt (IR): -

Against the goods receipt the plant finance people physically verify the stock & prepares the invoice receipt.

## Payment document: -

Against the invoice receipt the account payable department the amount to the vendor.



## Some of the standard code related to MM: -

1. Create purchase Requisition → ME51N
2. Create Request of quotation → ME41N
3. Create Quotation -- → ME47
4. Comparing the quotation -- → ME49
5. Create purchaser order (PO) → ME21N
6. Create Automatic purchase order → ME59
7. Release the purchase order → ME28 / ME29
8. Release the purchase Requisition → ME54N
9. Create the goods receipt -- → MIGO
10. Create the invoice receipt -- → MIR7 / MIRO
11. Create the payment doc -- → F110

### Steps to get the GR number based on PO number: -

**Method1** – Pass the PO number & item number to the ‘MSEG’ table & get the MBLNR as GR number.

**Method2** – Pass the PO number & item number to EKBE table & also pass BEWTP ‘E’ then we get the BELNR as GR document number.

### Steps to get the IR document number based on the PO number: -

Pass the PO number & item number to the ‘EKBE’ table & also pass BEWTP = ‘Q’ then we get BELNR as IR document number.

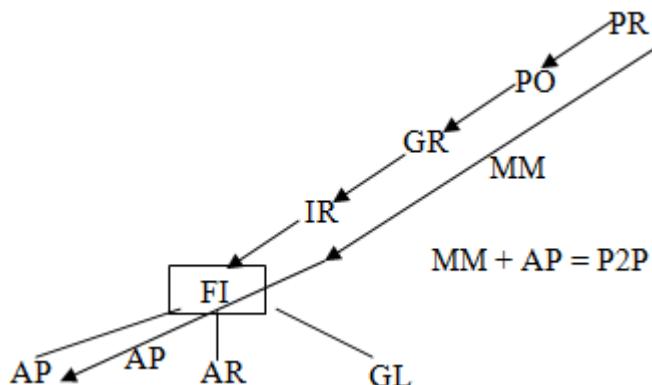
### Steps to get payment document number based on PO number: -

Pass the PO number & item number to the EKBE table & also pass ‘BEWTP’ = ‘Q’ then we get the BELNR (IR document) & GJAHR (IR year).

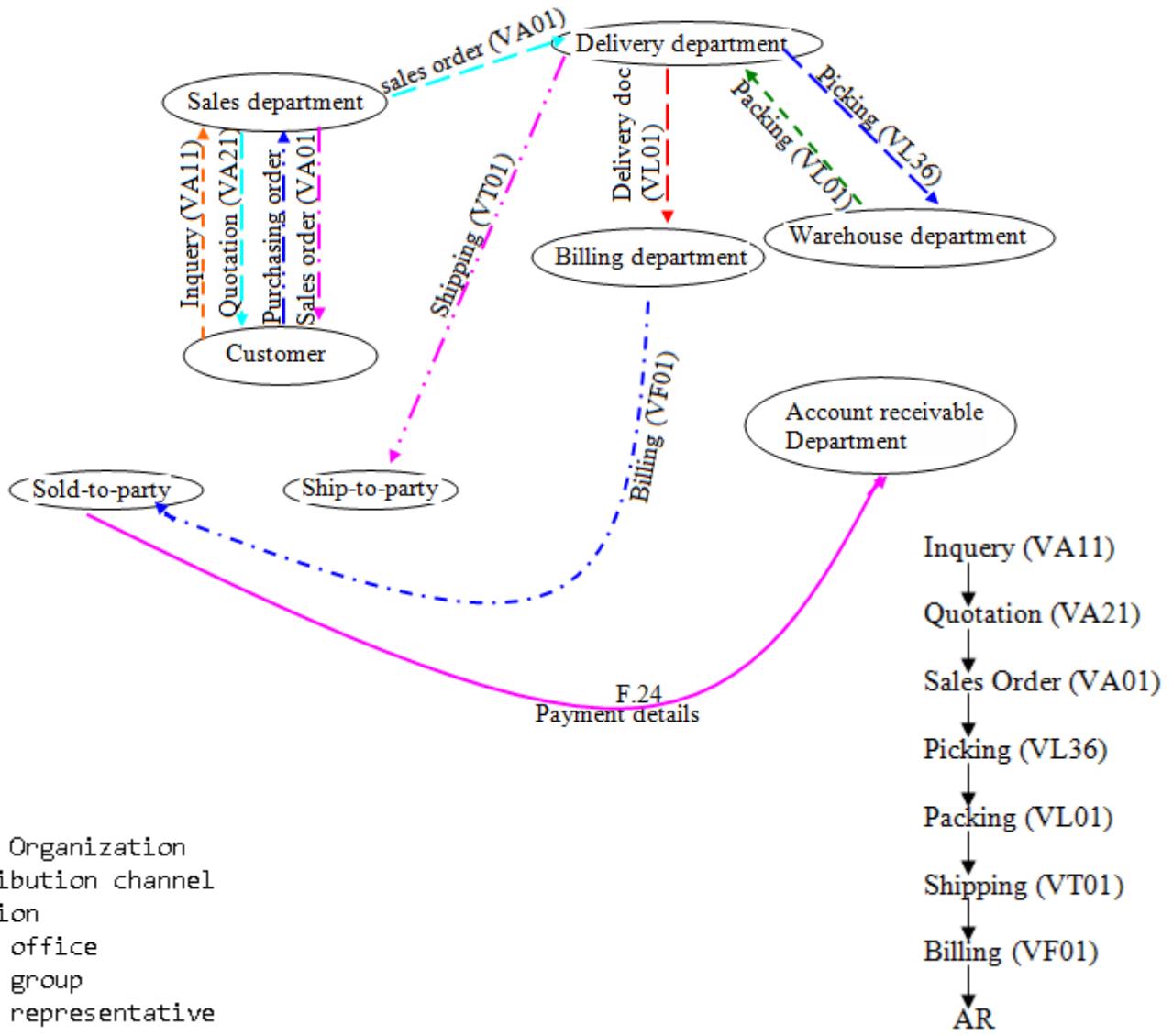
Concatenate the IR document with year & pass this value to ‘AWKEY’ filed in the BKPF table then we get the BELNR as payment document number.

PO number : 3000000008  
GR number : 5000002445  
IR document : 5105604171  
Payment doc : 5100000026  
Vendor : 3910

### **Payment cleared (BSAK)**



# SD Life-cycle



1. Sales Organization
2. Distribution channel
3. Division
4. Sales office
5. Sales group
6. Sales representative

**Inquiry:** - Customer enquiry about the product & their services.

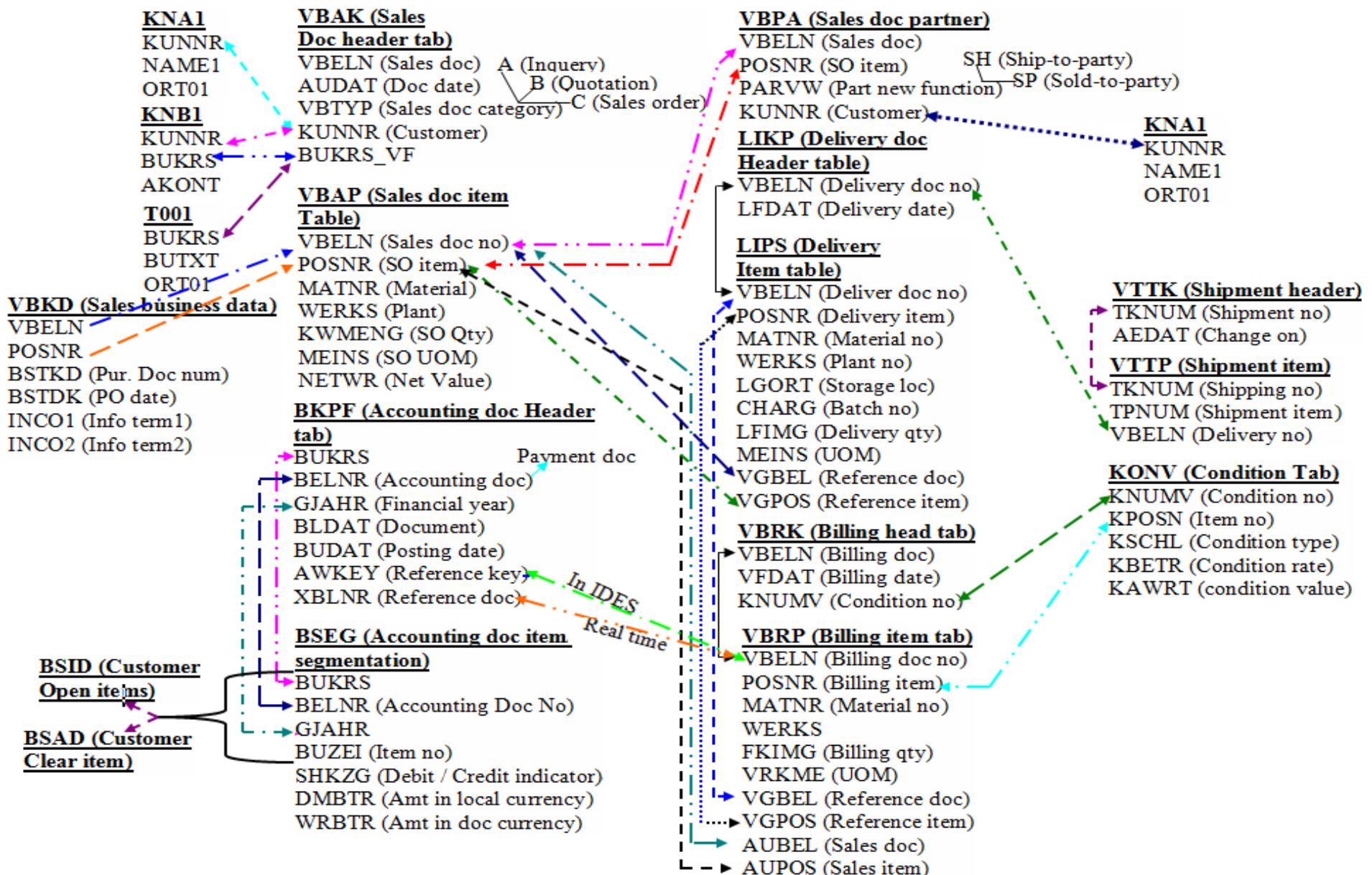
**Quotation:** - Against the customer query the sales people provide the quotation to the customer.

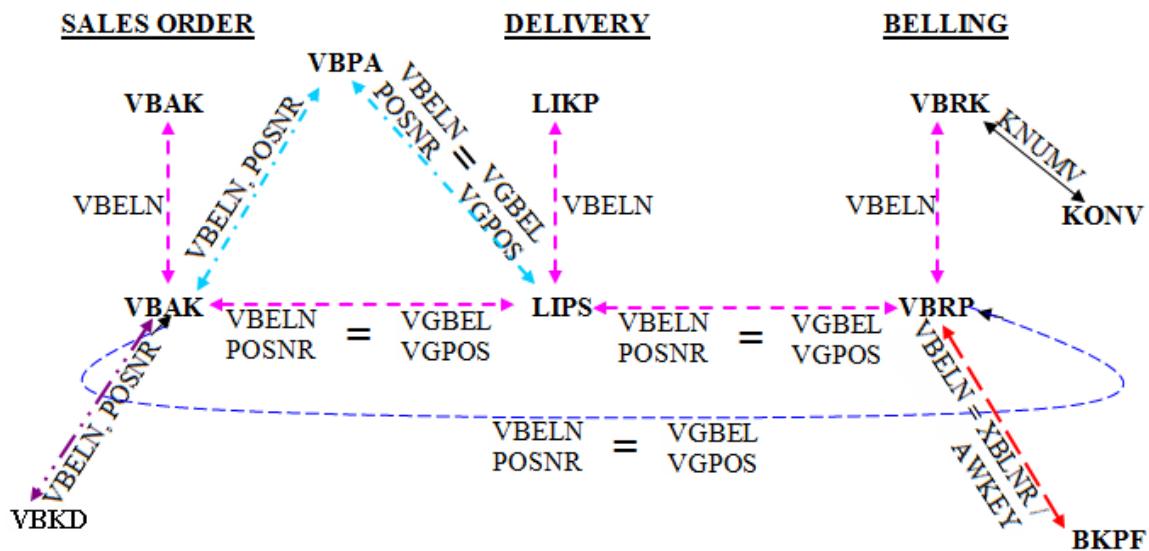
**Sales order:** - Against customer purchase order the sales people generate the sales order & given to customer.

**Delivery:** - Against the sales order the delivery people pick the goods from warehouse & packing with handling unit (Boxes, Bags, drums) & ship to customer ship to party address.

**Billing:** - Against the delivery document the billing people generate the bill or invoice & sends to customer sold to party address.

**Payment document:** - Against the billing document the account receivable department collects the money from the customer.

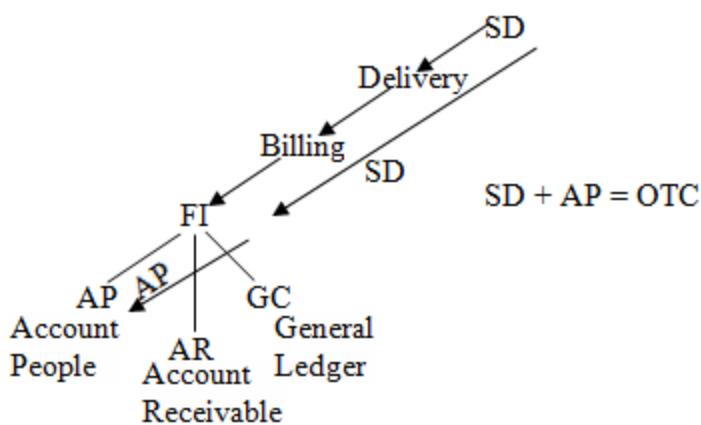




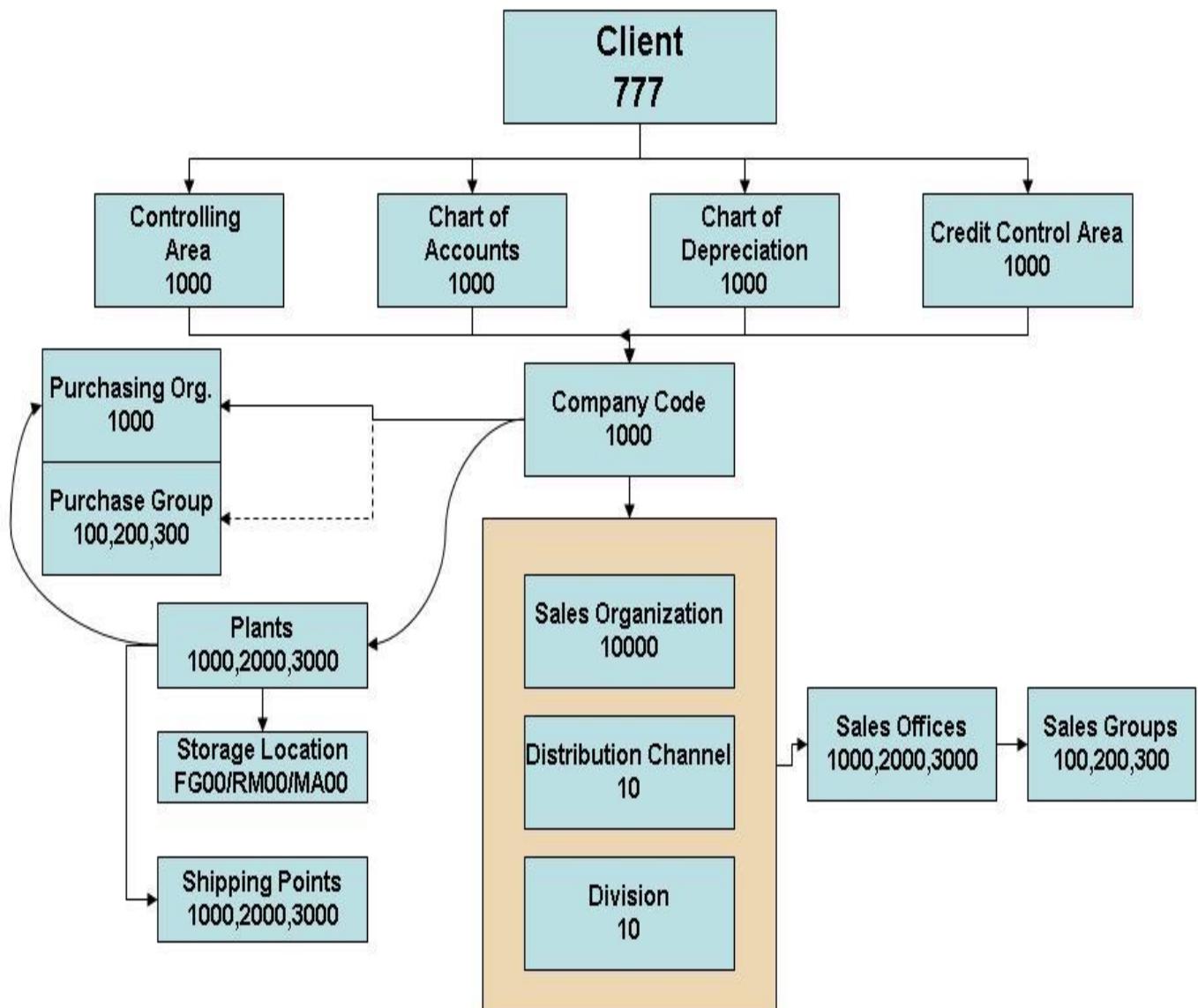
**Note:** - VBFA (sales flow table)

#### Sample document in IDES

SO number : 5323  
 Delivery no : 80003721  
 Billing doc / Invoice : 0090005594  
 Accounting doc : 0100007824  
 Customer : 2004  
 Company : 1000



# Organisation Structure



# PP Life-cycle

The PPC department creates the requirement plan in a particular period based on the requirement plan. They create the plant order. Then the plant order (system) check the requirement plant & warehouse stock & generate some of the process orders & some of the purchase Requisition. Purchase Requisition later converted into purchase orders and goods receipts.

The production department or PPC department generate the process order with the required materials & sends to warehouse department. Whenever the process order is created then automatically one reservation number is generated & reserved the stock. Against the process

order the warehouse people issue the goods to the production with MIGO transaction with 261 movement (and also issue the goods against cost center with MIGO transaction & 201 movement to calculate the cost).

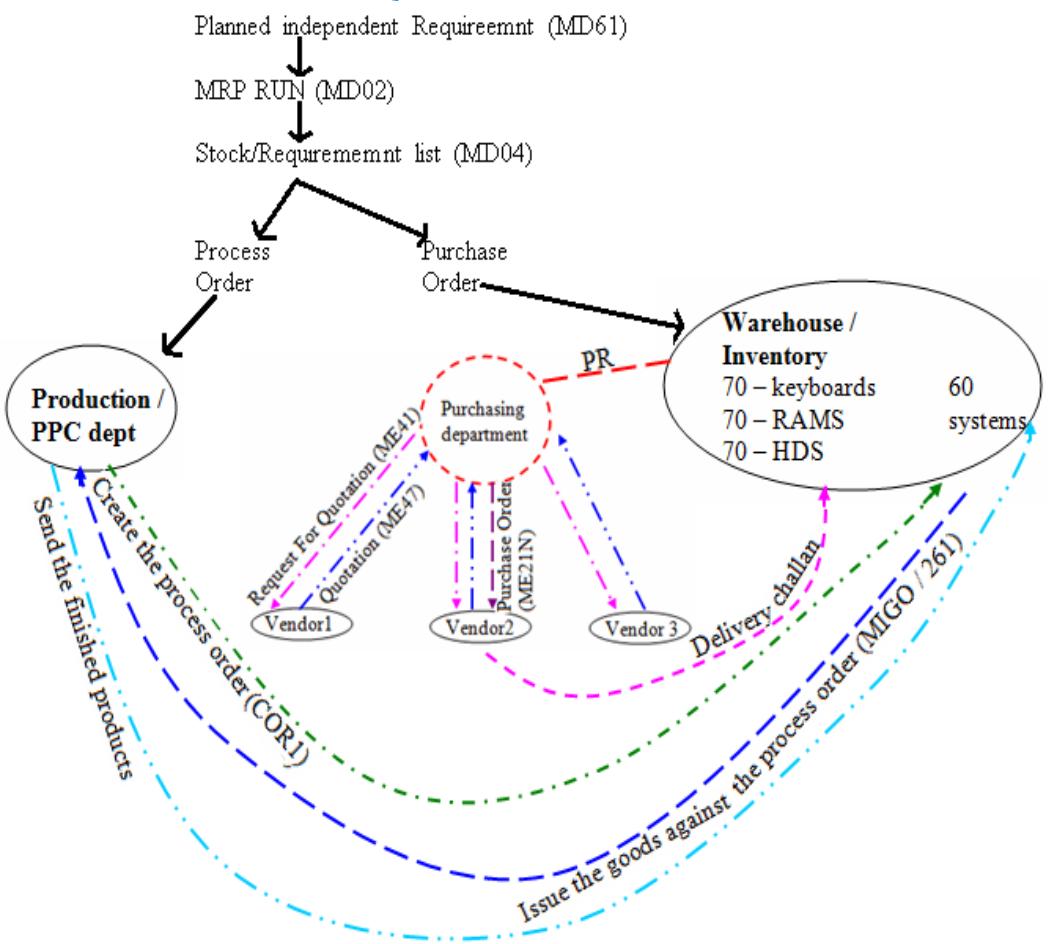
After receiving the materials the production people prepares the finished product based on the recipe (procedure) & sends to warehouse department. The warehouse department creates the goods receipt against process order with MIGO & 101 movement type.

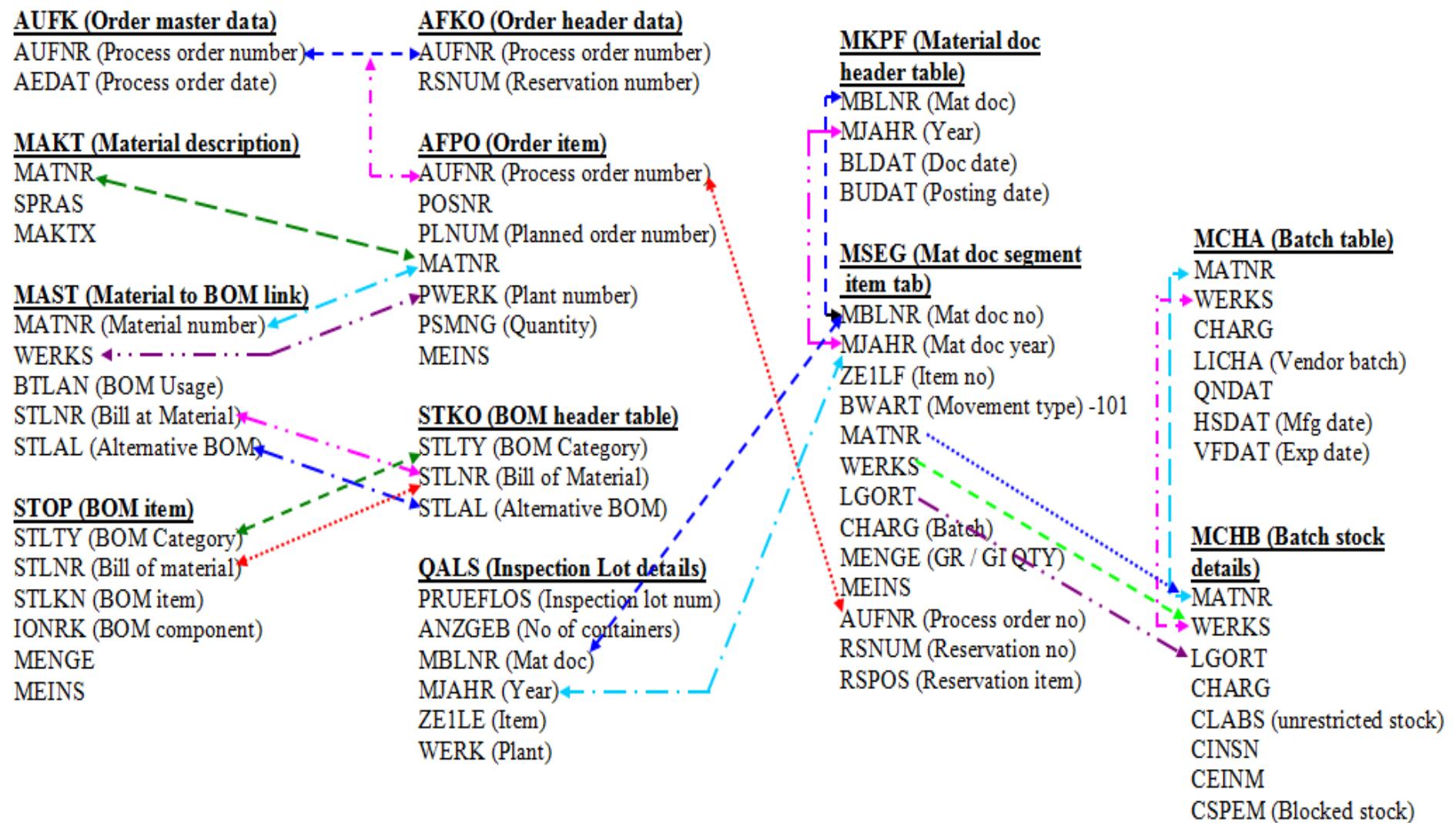
Whenever the warehouse people create the GR against process order then automatically one inspection lot number is generated.

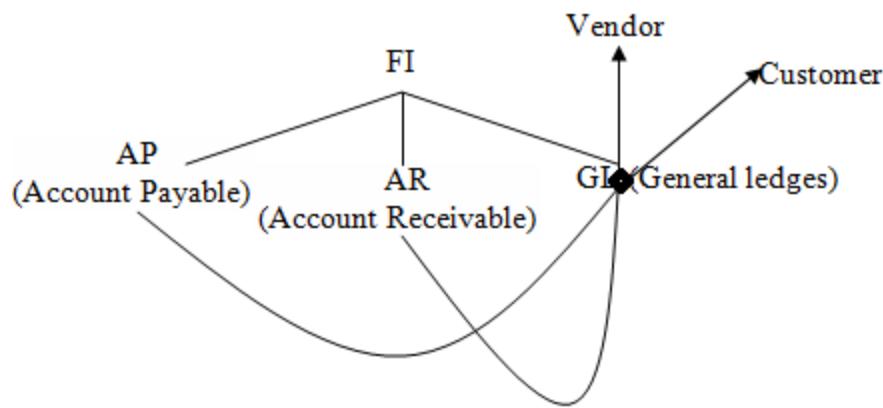
Based on the inspection lot number the quality control people came to warehouse & take some samples & conduct the tests based on QA people instruction & prepares the sampling check list & sends to warehouse. Based on the check list the warehouse people maintained the finished products either in accepted area or rejected area.

## Some of the Transaction codes in PPC

1. Create process order → C0R1
2. Create BOM (Bill of Material) → CS01
3. Create Batch → MSC1N
4. Create Recipe → C201
5. Create goods issue against process order → MIGO / 261
6. Create goods issue against cost center → MIGO / 201  
Create goods receipt → MIGO / 101







1. Create the vendor / employee → FK01
2. Advance payments to the vendor → F\_48
3. Credit note to the party (vendor) → FS60
4. Debit note to the party (vendor) → FB65
5. Partial payments to the vendor → F-53
6. Insert calculation over the items → F-24
7. Automatic payment to the vendor → F110
8. Bill of exchange (comp → bank → vendor) → F-40
9. TDS (Tax Deduction at Source) → MIRO

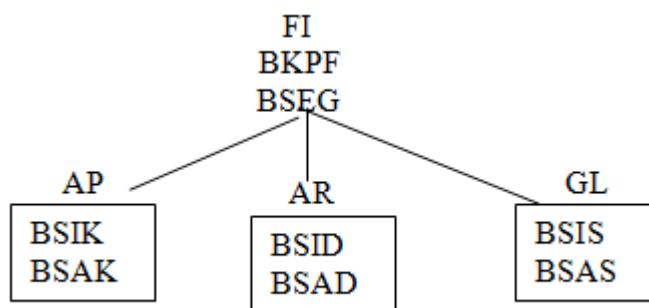
#### Amount Receivable (AR)

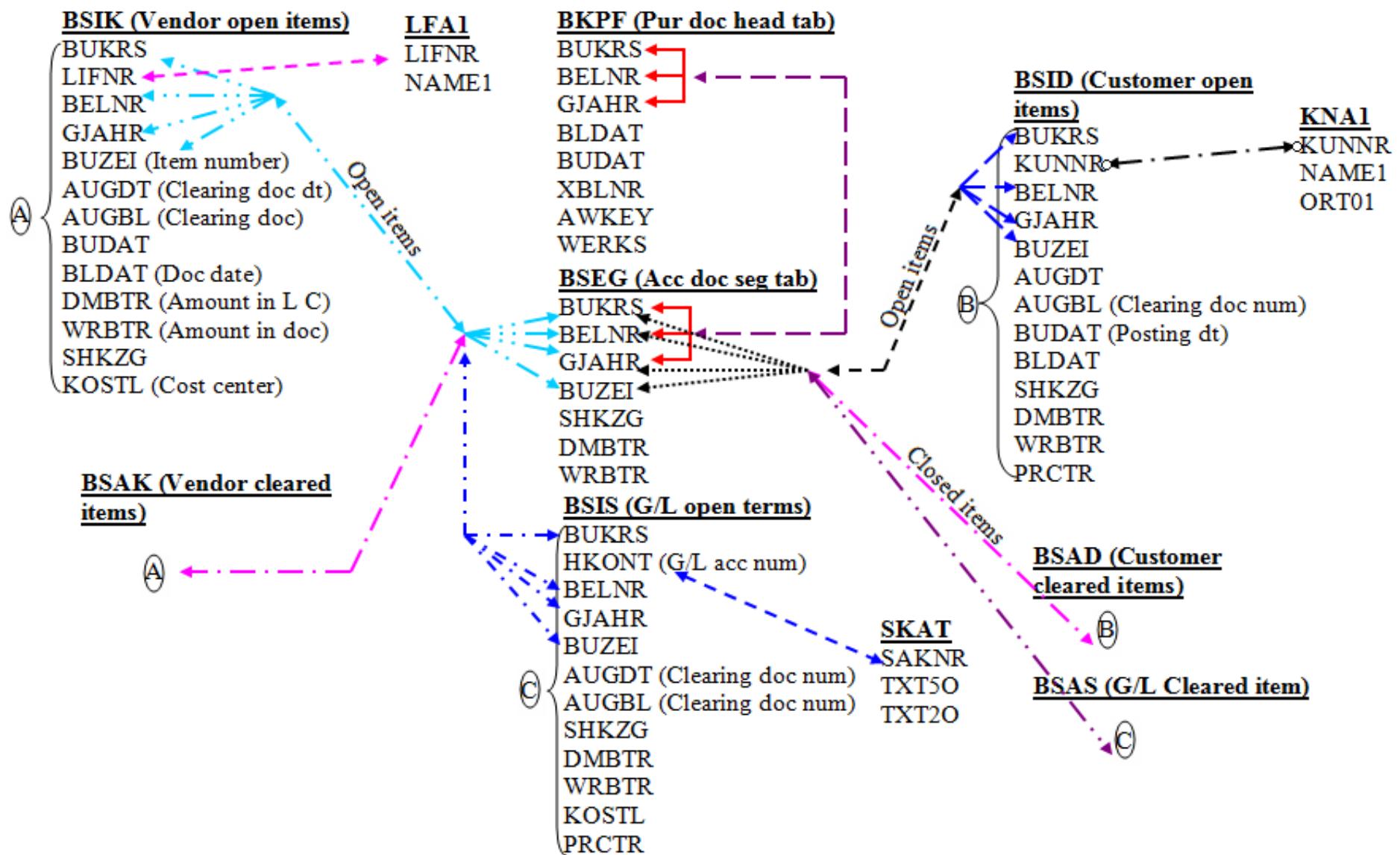
1. Create the customer master → FD01 / XD01
2. Security / deposits → F-49
3. Advance payments from customer → F-29
4. Adjustment of advance payments from customer → F-39
5. Credit note the party (customer) → FB75
6. Debit note to the party (customer) → FB70
7. Payment from the customer → F-28
8. Bill of exchange (customer → Bank → company) → F-36, F-33

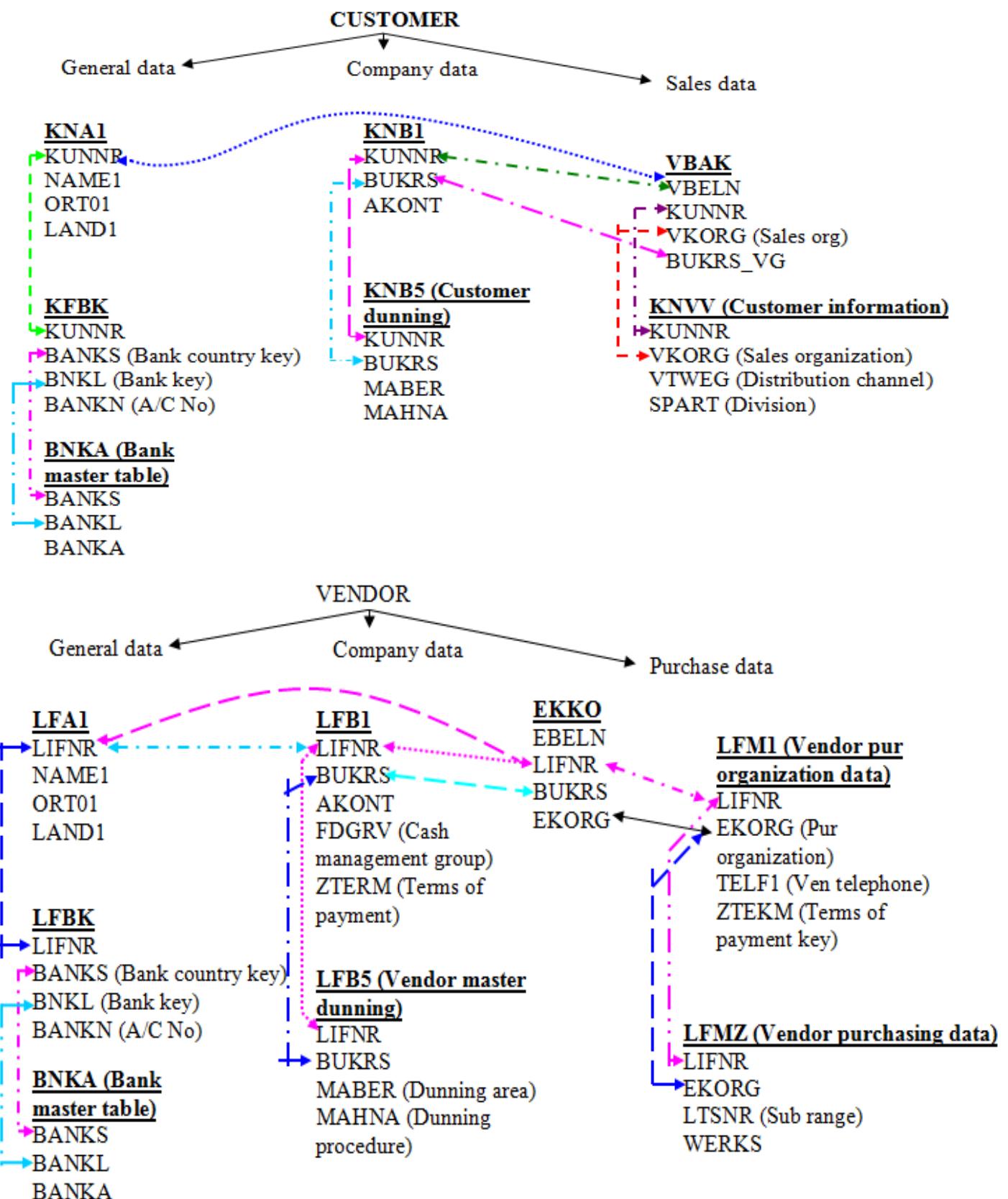
#### General ledger (GL)

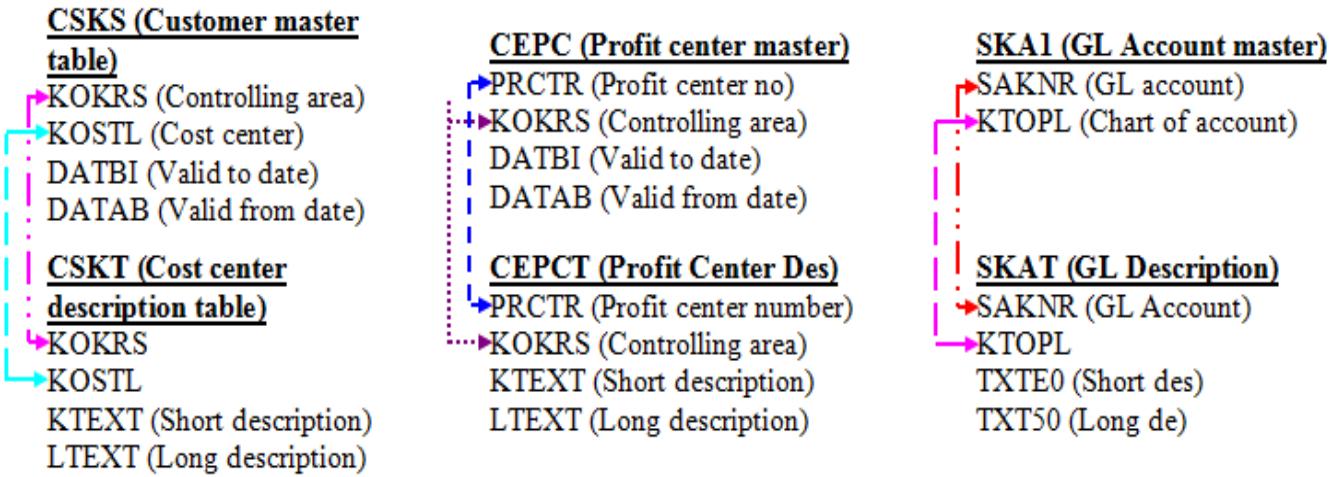
1. Cash payments → F-02
2. Cash receipts → F-03
3. Bank payments → F-04
4. Bank receipt → F-05

K → Vendor  
D → Customer  
S → G/L  
T → Open  
A → Cleared







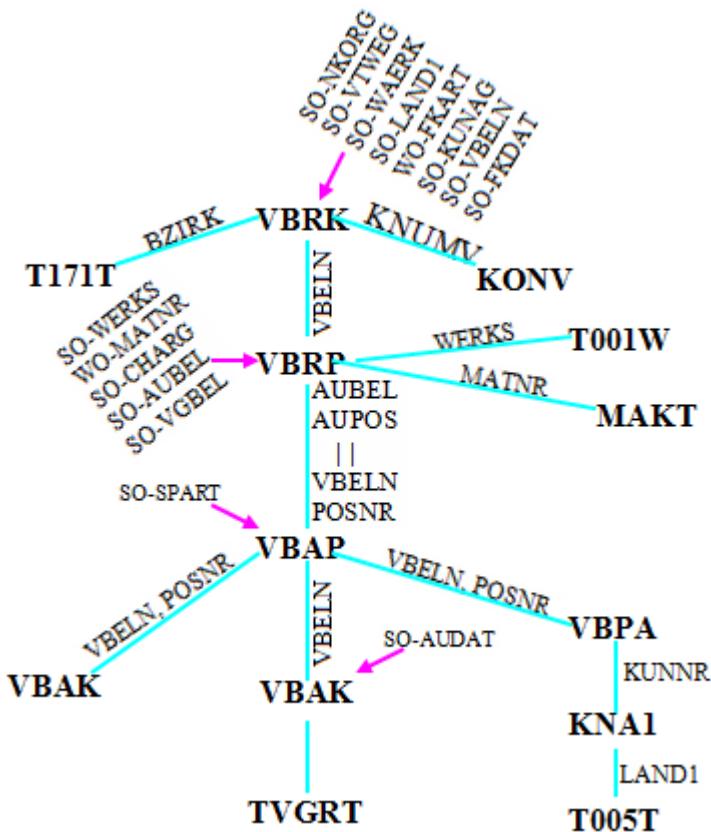


#### → DISPLAY SALES TAXES & INSURANCES Report

The following things are maintained in TS.

- Maintain the selection screen output fields, description & technical information
- Maintain the output fields or displayed fields, short descriptions & technical names.
- Maintain the pseudo code of the object.
- Prepare the Ven diagrams if it's needed.
- Provide the input screen shorts & output screen shorts after developed the object.
- Maintain all the parameters fields selection fields & mandatory fields.

**Note:** - If the selection screen contains at least one header field then we starts the select query of header level.



Internal table contains input fields + link fields + output fields

<b>VBRK</b>		<b>T171T</b>	<b>MAKT</b>	<b>TVGRT</b>
VKORG		BZIRK	MATNR	VKGGRP
VTWEG		ZTXT	MAKTX	BEZEI
WAERK				
LAND1		<b>VBRP</b>	<b>T001W</b>	<b>VBPA</b>
FKART		MATNR	WERKS	VBELN
FKDAT		WERKS	NAME1	POSNR
KUNAG		CHARG	<b>VBPA</b>	KUNNR
VBELN		AUBEL	SPART	PARVW
+		VGBEL	VBELN	
BZIRK		AUPOS	POSNR	<b>KNA1</b>
VBELN		FKIMG	<b>VBKD</b>	NAME1
KNUMV		VRKME	VBELN	LAND1
+		KZWI3	POSNR	
VKORG		KZWI4	EMPST	<b>T005T</b>
VTWEG		KZWI6	BSTKD	LAND1
BZIRK		<b>KONV</b>	BSTDK	LANDX
KUNAG		KNUMV	INCO1	
VBELN		KAWRT	INCO2	
FKDAT		KBETR	<b>VBAK</b>	
WAERK		KSCHL	AUDAT	VBELN
KURRF (Exchange rate)				VKGGRP

REPORT ZPROJECT.

```
*****
* PROGRAM      : ZVR001-SPT_SALES_REGISTER          *
* AUTHOR       : SATISH                            *
* PURPOSE      : TO DISPLAY SALES TAXEX & INSURENCES   *
* START DATE   : 28.02.2015                         *
* FINISH DATE  : 06.03.2015                         *
* SUPPLIER     : SJF TECHNOLOGIES                  *
* PACKAGE      :                                     *
* REQUEST NUM: SRYK900117                          *
*****
```

DATA w\_cops TYPE I.

```
TYPES: BEGIN OF TY_FINAL,
  VKORG TYPE VBRK-VKORG,
  VTWEG TYPE VBRK-VTWEG,
  VKGRP TYPE VBAK-VKGRP,
  BEZEI TYPE TVGRT-BEZEI,
  WERKS TYPE VBRP-WERKS,
  NAME1 TYPE T001W-NAME1,
```

```

BZIRK TYPE VBRK-BZIRK,
BZTXT TYPE T171T-BZTXT,
SPC TYPE T005T-LANDX,
SHC TYPE T005T-LANDX,
KUNAG TYPE VBRK-KUNAG,
SPN TYPE KNA1-NAME1,
SHN TYPE KNA1-NAME1,
EMPST TYPE VBKD-EMPST,
BSTKD TYPE VBKD-BSTKD,
BSTDK TYPE VBKD-BSTDK,
AUBEL TYPE VBRP-AUBEL,
POSNR TYPE VBAP-POSNR,
AUDAT TYPE VBAK-AUDAT,
VBELN TYPE VBRK-VBELN,
FKDAT TYPE VBRK-FKDAT,
MATNR TYPE VBRP-MATNR,
MAKTX TYPE MAKT-MAKTX,
CHARG TYPE VBRP-CHARG,
FKIMG TYPE VBRP-FKIMG,
VRKME TYPE VBRP-VRKME,
WAERK TYPE VBRK-WAERK,
BP TYPE KONV-KBETR,
BED TYPE KONV-KBETR,
ECESS TYPE KONV-KBETR,
SHCESS TYPE KONV-KBETR,
VAT TYPE KONV-KBETR,
KZWI3 TYPE VBRP-KZWI3,
KZWI4 TYPE VBRP-KZWI4,
KZWI6 TYPE VBRP-KZWI6,
INCO1 TYPE VBKD-INCO1,
INCO2 TYPE VBKD-INCO2,
END OF TY_FINAL.

DATA: WA_FINAL TYPE TY_FINAL,
      IT_FINAL TYPE TABLE OF TY_FINAL.

* DECLARE WA_VBRK AND IT_VBRK
TYPES : BEGIN OF TY_VBRK,
         VBELN TYPE VBRK-VBELN,
         FKART TYPE VBRK-FKART,
         WAERK TYPE VBRK-WAERK,
         VKORG TYPE VBRK-VKORG,
         VTWEG TYPE VBRK-VTWEG,
         KURRF TYPE VBRK-KURRF,
         KNUMV TYPE VBRK-KNUMV,
         FKDAT TYPE VBRK-FKDAT,
         BZIRK TYPE VBRK-BZIRK,
         LAND1 TYPE VBRK-LAND1,
         KUNAG TYPE VBRK-KUNAG,
         END OF TY_VBRK.

DATA: WA_VBRK TYPE TY_VBRK,
      IT_VBRK TYPE TABLE OF TY_VBRK.

```

```

* DECLARE WA_T171T, IT_T171T
TYPES : BEGIN OF TY_T171T,
         BZIRK TYPE T171T-BZIRK,
         BZTXT TYPE T171T-BZTXT,
         END OF TY_T171T.
DATA: WA_T171T TYPE TY_T171T,
      IT_T171T TYPE TABLE OF TY_T171T.

* DECLARE WA_VBRP, IT_VBRP
TYPES : BEGIN OF TY_VBRP,
         VBELN TYPE VBRP-VBELN,
         POSNR TYPE VBRP-POSNR,
         FKIMG TYPE VBRP-FKIMG,
         VRKME TYPE VBRP-VRKME,
         VGBEL TYPE VBRP-VGBEL,
         AUBEL TYPE VBRP-AUBEL,
         AUPOS TYPE VBRP-AUPOS,
         MATNR TYPE VBRP-MATNR,
         CHARG TYPE VBRP-CHARG,
         WERKS TYPE VBRP-WERKS,
         KZWI3 TYPE VBRP-KZWI3,
         KZWI4 TYPE VBRP-KZWI4,
         KZWI6 TYPE VBRP-KZWI6,
         END OF TY_VBRP.
DATA: WA_VBRP TYPE TY_VBRP,
      IT_VBRP TYPE TABLE OF TY_VBRP.

* DECLARE WA_KONV, IT_KONV
TYPES : BEGIN OF TY_KONV,
         KNUMV TYPE KONV-KNUMV,
         KPOSN TYPE KONV-KPOSN,
         KSCHL TYPE KONV-KSCHL,
         KAWRT TYPE KONV-KAWRT,
         KBETR TYPE KONV-KBETR,
         END OF TY_KONV.
DATA: WA_KONV TYPE TY_KONV,
      IT_KONV TYPE TABLE OF TY_KONV.

* DECLARE THE WA_MAKT AND IT_MAKT
TYPES : BEGIN OF TY_MAKT,
         MATNR TYPE MAKT-MATNR,
         MAKTX TYPE MAKT-MAKTX,
         END OF TY_MAKT.
DATA: WA_MAKT TYPE TY_MAKT,
      IT_MAKT TYPE TABLE OF TY_MAKT.

TYPES : BEGIN OF TY_T001W,
         WERKS TYPE T001W-WERKS,
         NAME1 TYPE T001W-NAME1,
         END OF TY_T001W.

```

```

DATA: WA_T001W TYPE TY_T001W,
      IT_T001W TYPE TABLE OF TY_T001W.

TYPES : BEGIN OF TY_VBAP,
        VBELN TYPE VBAP-VBELN,
        POSNR TYPE VBAP-POSNR,
        SPART TYPE VBAP-SPART,
        END OF TY_VBAP.

DATA: WA_VBAP TYPE TY_VBAP,
      IT_VBAP TYPE TABLE OF TY_VBAP.

TYPES : BEGIN OF TY_VBKD,
        VBELN TYPE VBKD-VBELN,
        POSNR TYPE VBKD-POSNR,
        INCO1 TYPE VBKD-INCO1,
        INCO2 TYPE VBKD-INCO2,
        EMPST TYPE VBKD-EMPST,
        BSTKD TYPE VBKD-BSTKD,
        BSTDK TYPE VBKD-BSTDK,
        END OF TY_VBKD.

DATA: WA_VBKD TYPE TY_VBKD,
      IT_VBKD TYPE TABLE OF TY_VBKD.

TYPES : BEGIN OF TY_VBAK,
        VBELN TYPE VBAK-VBELN,
        AUDAT TYPE VBAK-AUDAT,
        VKGRP TYPE VBAK-VKGRP,
        END OF TY_VBAK.

DATA: WA_VBAK TYPE TY_VBAK,
      IT_VBAK TYPE TABLE OF TY_VBAK.

TYPES : BEGIN OF TY_TVGRT,
        VKGRP TYPE TVGRT-VKGRP,
        BEZEI TYPE TVGRT-BEZEI,
        END OF TY_TVGRT.

DATA: WA_TVGRT TYPE TY_TVGRT,
      IT_TVGRT TYPE TABLE OF TY_TVGRT.

TYPES : BEGIN OF TY_VBPA,
        VBELN TYPE VBPA-VBELN,
        POSNR TYPE VBPA-POSNR,
        PARVW TYPE VBPA-PARVW,
        KUNNR TYPE VBPA-KUNNR,
        END OF TY_VBPA.

DATA: WA_VBPA TYPE TY_VBPA,
      IT_VBPA TYPE TABLE OF TY_VBPA.

TYPES : BEGIN OF TY_KNA1,
        KUNNR TYPE KNA1-KUNNR,
        LAND1 TYPE KNA1-LAND1,
        NAME1 TYPE KNA1-NAME1,

```

```

        END OF TY_KNA1.
DATA: WA_KNA1 TYPE TY_KNA1,
      IT_KNA1 TYPE TABLE OF TY_KNA1.

TYPES : BEGIN OF TY_T005T,
         LAND1 TYPE T005T-LAND1,
         LANDX TYPE T005T-LANDX,
         END OF TY_T005T.
DATA: WA_T005T TYPE TY_T005T,
      IT_T005T TYPE TABLE OF TY_T005T.

DATA: IT_FCAT TYPE SLIS_T_FIELDCAT_ALV,
      WA_FCAT LIKE LINE OF IT_FCAT.

TABLES: VBRK, VBRP, VBAP, VBAK.
SELECTION-SCREEN BEGIN OF BLOCK A WITH FRAME TITLE TEXT-001.
SELECT-OPTIONS: SO_VKORG FOR VBRK-VKORG,
                 SO_VTWEG FOR VBRK-VTWEG,
                 SO_WERKS FOR VBRP-WERKS,
                 SO_WAERK FOR VBRK-WAERK,
                 SO_LAND1 FOR VBRK-LAND1,
                 SO_FKART FOR VBRK-FKART,
                 SO_KUNAG FOR VBRK-KUNAG,
                 SO_VBELN FOR VBRK-VBELN,
                 SO_FKDAT FOR VBRK-FKDAT,
                 SO_MATNR FOR VBRP-MATNR,
                 SO_CHARG FOR VBRP-CHARG,
                 SO_AUBEL FOR VBRP-AUBEL,
                 SO_VGBEL FOR VBRP-VGBEL,
                 SO_SPART FOR VBAP-SPART,
                 SO_AUDAT FOR VBAK-AUDAT.
SELECTION-SCREEN END OF BLOCK A.

***** ALL PERFORMS *****
PERFORM GET_DATA.
PERFORM POPULATE_FINAL_TABLE.
PERFORM FILL_FCAT.
PERFORM DISPLAY.
***** FORM GET_DATA *****

***** FORM GET_DATA *****
FORM GET_DATA.
  SELECT VBELN FKART WAERK VKORG VTWEG KURRF KNUMV FKDAT BZIRK LAND1
  KUNAG FROM VBRK INTO TABLE IT_VBRK WHERE VKORG IN SO_VKORG
  AND VTWEG IN SO_VTWEG AND WAERK IN SO_WAERK AND LAND1 IN SO_LAND1 AND
  FKART IN SO_FKART AND KUNAG IN SO_KUNAG AND VBELN IN SO_VBELN
  AND FKDAT IN SO_FKDAT.

  IF IT_VBRK IS NOT INITIAL.
```

```

SELECT BZIRK BZTXT FROM T171T INTO TABLE IT_T171T FOR ALL ENTRIES IN
IT_VBRK WHERE BZIRK = IT_VBRK-BZIRK AND SPRAS = SY-LANGU.
SELECT KNUMV KPOSN KSCHL KAWRT KBETR FROM KONV INTO TABLE IT_KONV FOR
ALL ENTRIES IN IT_VBRK WHERE KNUMV = IT_VBRK-KNUMV.
SELECT VBELN POSNR FKIMG VRKME VGBEL AUBEL AUPOS MATNR CHARG WERKS
KZWI3 KZWI4 KZWI6 FROM VBRP INTO TABLE IT_VBRP FOR ALL
ENTRIES IN IT_VBRK WHERE VBELN = IT_VBRK-VBELN AND WERKS IN SO_WERKS
AND MATNR IN SO_MATNR AND CHARG IN SO_CHARG AND
AUBEL IN SO_AUBEL AND VGBEL IN SO_VGBEL.
ENDIF.

IF VBRP IS NOT INITIAL.
  SELECT MATNR MAKTX FROM MAKT INTO TABLE IT_MAKT FOR ALL ENTRIES IN
  IT_VBRP WHERE MATNR = IT_VBRP-MATNR AND SPRAS = SY-LANGU.
  SELECT WERKS NAME1 FROM T001W INTO TABLE IT_T001W FOR ALL ENTRIES IN
  IT_VBRP WHERE WERKS = IT_VBRP-WERKS.
  SELECT VBELN POSNR SPART FROM VBAP INTO TABLE IT_VBAP FOR ALL ENTRIES
  IN IT_VBRP WHERE VBELN = IT_VBRP-VBELN AND SPART IN SO_SPART.
ENDIF.

IF VBAP IS NOT INITIAL.
  SELECT VBELN POSNR INCO1 INCO2 EMPST BSTKD BSTDK FROM VBKD INTO TABLE
  IT_VBKD FOR ALL ENTRIES IN IT_VBAP WHERE
    VBELN = IT_VBAP-VBELN AND POSNR = IT_VBAP-POSNR.
  SELECT VBELN AUDAT VKGRP FROM VBAK INTO TABLE IT_VBAK FOR ALL ENTRIES
  IN IT_VBAP WHERE VBELN = IT_VBAP-VBELN AND AUDAT IN SO_AUDAT.
  SELECT VBELN POSNR PARVW KUNNR FROM VBPA INTO TABLE IT_VBPA FOR ALL
  ENTRIES IN IT_VBAP WHERE VBELN = IT_VBAP-VBELN.
ENDIF.

IF IT_VBAK IS NOT INITIAL.
  SELECT VKGRP BEZEI FROM TVGRT INTO TABLE IT_TVGRT FOR ALL ENTRIES IN
  IT_VBAK WHERE VKGRP = IT_VBAK-VKGRP AND SPRAS = SY-LANGU.
ENDIF.

IF IT_VBPA IS NOT INITIAL.
  SELECT KUNNR LAND1 NAME1 FROM KNA1 INTO TABLE IT_KNA1 FOR ALL ENTRIES
  IN IT_VBPA WHERE KUNNR = IT_VBPA-KUNNR.
ENDIF.

IF IT_KNA1 IS NOT INITIAL.
  SELECT LAND1 LANDX FROM T005T INTO TABLE IT_T005T FOR ALL ENTRIES IN
  IT_KNA1 WHERE LAND1 = IT_KNA1-LAND1.
ENDIF.
ENDFORM.
***** FORM POPULATE_FINAL_TABLE *****
FORM POPULATE_FINAL_TABLE.
  LOOP AT IT_VBRP INTO WA_VBRP.
    WA_FINAL-WERKS = WA_VBRP-WERKS.

```

```

WA_FINAL-AUBEL = WA_VBRP-AUBEL.
WA_FINAL-MATNR = WA_VBRP-MATNR.
WA_FINAL-CHARG = WA_VBRP-CHARG.
WA_FINAL-FKIMG = WA_VBRP-FKIMG.
WA_FINAL-VRKME = WA_VBRP-VRKME.
WA_FINAL-KZWI3 = WA_VBRP-KZWI3.
WA_FINAL-KZWI4 = WA_VBRP-KZWI4.
WA_FINAL-KZWI6 = WA_VBRP-KZWI6.

READ TABLE IT_MAKT INTO WA_MAKT WITH KEY MATNR = WA_VBRP-MATNR.
IF SY-SUBRC = 0.
  WA_FINAL-MAKTX = WA_MAKT-MAKTX.
ENDIF.

READ TABLE IT_T001W INTO WA_T001W WITH KEY WERKS = WA_VBRP-WERKS.
IF SY-SUBRC = 0.
  WA_FINAL-NAME1 = WA_T001W-NAME1.
ENDIF.

READ TABLE IT_VBAP INTO WA_VBAP WITH KEY VBELN = WA_VBRP-AUBEL POSNR
= WA_VBRP-AUPOS.
IF SY-SUBRC = 0.
  WA_FINAL-POSNR = WA_VBAP-POSNR.

READ TABLE IT_VBKD INTO WA_VBKD WITH KEY VBELN = WA_VBAP-VBELN
POSNR = WA_VBAP-POSNR.
IF SY-SUBRC = 0.
  WA_FINAL-EMPST = WA_VBKD-EMPST.
  WA_FINAL-BSTKD = WA_VBKD-BSTKD.
  WA_FINAL-BSTDK = WA_VBKD-BSTDK.
  WA_FINAL-INCO1 = WA_VBKD-INCO1.
  WA_FINAL-INCO2 = WA_VBKD-INCO2.
ENDIF.

READ TABLE IT_VBAK INTO WA_VBAK WITH KEY VBELN = WA_VBAP-VBELN.
IF SY-SUBRC = 0.
  WA_FINAL-VKGRP = WA_VBAK-VKGRP.
  WA_FINAL-AUDAT = WA_VBAK-AUDAT.

READ TABLE IT_TVGRT INTO WA_TVGRT WITH KEY VKGRP = WA_VBAK-VKGRP.
IF SY-SUBRC = 0.
  WA_FINAL-BEZEI = WA_TVGRT-BEZEI.
ENDIF.
ENDIF.

READ TABLE IT_VBPA INTO WA_VBPA WITH KEY VBELN = WA_VBAP-VBELN
POSNR = WA_VBAP-POSNR PARVW = 'SP'.
IF SY-SUBRC = 0.

  READ TABLE IT_KNA1 INTO WA_KNA1 WITH KEY KUNNR = WA_VBPA-KUNNR.
  IF SY-SUBRC = 0.

```

```

        WA_FINAL-SPN = WA_KNA1-NAME1.

        READ TABLE IT_T005T INTO WA_T005T WITH KEY LAND1 = WA_KNA1-
LAND1.
        IF SY-SUBRC = 0.
            WA_FINAL-SPC = WA_T005T-LANDX.
        ENDIF.
    ENDIF.
ENDIF.

        READ TABLE IT_VBPA INTO WA_VBPA WITH KEY VBELN = WA_VBAP-VBELN
POSNR = WA_VBAP-POSNR PARVW = 'SH'.
        IF SY-SUBRC = 0.

            READ TABLE IT_KNA1 INTO WA_KNA1 WITH KEY KUNNR = WA_VBPA-KUNNR.
            IF SY-SUBRC = 0.
                WA_FINAL-SHN = WA_KNA1-NAME1.

            READ TABLE IT_T005T INTO WA_T005T WITH KEY LAND1 = WA_KNA1-
LAND1.
                IF SY-SUBRC = 0.
                    WA_FINAL-SHC = WA_T005T-LANDX.
                ENDIF.
            ENDIF.
        ENDIF.
    ENDIF.

        READ TABLE IT_VBRK INTO WA_VBRK WITH KEY VBELN = WA_VBRP-VBELN.
        IF SY-SUBRC = 0.
            WA_FINAL-VKORG = WA_VBRK-VKORG.
            WA_FINAL-VTWEG = WA_VBRK-VTWEG.
            WA_FINAL-BZIRK = WA_VBRK-BZIRK.
            WA_FINAL-KUNAG = WA_VBRK-KUNAG.
            WA_FINAL-VBELN = WA_VBRK-VBELN.
            WA_FINAL-FKDAT = WA_VBRK-FKDAT.
            WA_FINAL-WAERK = WA_VBRK-WAERK.

        READ TABLE IT_T171T INTO WA_T171T WITH KEY BZIRK = WA_VBRK-BZIRK.
        IF SY-SUBRC = 0.
            WA_FINAL-BZTXT = WA_T171T-BZTXT.
        ENDIF.

        LOOP AT IT_KONV INTO WA_KONV WHERE KNUMV = WA_VBRK-KNUMV AND KPOSN
= WA_VBRP-POSNR.
        IF WA_KONV-KSCHL = 'PR00'.
            WA_FINAL-BP = WA_KONV-KAWRT / 10 * WA_KONV-KBETR * WA_VBRK-
KURRF.
        ELSEIF WA_KONV-KSCHL = 'JEXP' OR WA_KONV-KSCHL = 'JEXQ'.
            WA_FINAL-BED = WA_KONV-KBETR * WA_VBRK-KURRF.
        ELSEIF WA_KONV-KSCHL = 'JCEP' OR WA_KONV-KSCHL = 'JCEQ'.
            WA_FINAL-ECESS = WA_KONV-KBETR * WA_VBRK-KURRF.

```

```

ELSEIF WA_KONV-KSCHL = 'JCPE' OR WA_KONV-KSCHL = 'JCPQ'.
    WA_FINAL-SHCESS = WA_KONV-KBETR * WA_VBRK-KURRF.
ELSEIF WA_KONV-KSCHL = 'JLST' OR WA_KONV-KSCHL = 'JCST'.
    WA_FINAL-VAT = WA_KONV-KBETR * WA_VBRK-KURRF.
ENDIF.
ENDLOOP.
ENDIF.
APPEND WA_FINAL TO IT_FINAL.
CLEAR WA_FINAL.
ENDLOOP.
ENDFORM.
*****
***** FORM FILL_FCAT *****
FORM FILL_FCAT.
    w_cops = w_cops + 1.
    WA_FCAT-FIELDNAME = 'VKORG'.
    WA_FCAT-COL_POS = w_cops.
    WA_FCAT-SELTEXT_M = 'Sales Organization'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

    w_cops = w_cops + 1.
    WA_FCAT-FIELDNAME = 'VTWEG'.
    WA_FCAT-COL_POS = w_cops.
    WA_FCAT-SELTEXT_M = 'Distribution Channel'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

    w_cops = w_cops + 1.
    WA_FCAT-FIELDNAME = 'VKGRP'.
    WA_FCAT-COL_POS = w_cops.
    WA_FCAT-SELTEXT_M = 'Sales Group'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

    w_cops = w_cops + 1.
    WA_FCAT-FIELDNAME = 'BEZEI'.
    WA_FCAT-COL_POS = w_cops.
    WA_FCAT-SELTEXT_M = 'Description'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

    w_cops = w_cops + 1.
    WA_FCAT-FIELDNAME = 'WERKS'.
    WA_FCAT-COL_POS = w_cops.
    WA_FCAT-SELTEXT_M = 'Plant'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

    w_cops = w_cops + 1.
    WA_FCAT-FIELDNAME = 'NAME1'.

```

```

WA_FCAT-COL_POS      = w_cops.
WA_FCAT-SELTEXT_M   = 'Plant Desctiption'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME   = 'BZIRK'.
WA_FCAT-COL_POS     = w_cops.
WA_FCAT-SELTEXT_M   = 'Sales District'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME   = 'BZTXT'.
WA_FCAT-COL_POS     = w_cops.
WA_FCAT-SELTEXT_M   = 'District name'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME   = 'SPC'.
WA_FCAT-COL_POS     = w_cops.
WA_FCAT-SELTEXT_M   = 'SOLD TO COUNTRY'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME   = 'SHC'.
WA_FCAT-COL_POS     = w_cops.
WA_FCAT-SELTEXT_M   = 'DEST.COUNTRY'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME   = 'KUNAG'.
WA_FCAT-COL_POS     = w_cops.
WA_FCAT-SELTEXT_M   = 'Sold to party'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME   = 'SPN'.
WA_FCAT-COL_POS     = w_cops.
WA_FCAT-SELTEXT_M   = 'SOLD TO PARTY NAME'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME   = 'SHN'.
WA_FCAT-COL_POS     = w_cops.
WA_FCAT-SELTEXT_M   = 'SHIP TO PARTY NAME'.

```

```

APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'EMPST'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'Final Customer'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'BSTKD'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'PO Number'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'BSTDK'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'PO Date'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'AUBEL'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'Sales Order'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'POSNR'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'Item Number'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'AUDAT'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'SO Date'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'VBELN'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'Billing Document'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

```

```
w_cops = w_cops + 1.  
WA_FCAT-FIELDNAME = 'FKDAT'.  
WA_FCAT-COL_POS = w_cops.  
WA_FCAT-SELTEXT_M = 'Billing date'.  
APPEND WA_FCAT TO IT_FCAT.  
CLEAR wa_fcat.
```

```
w_cops = w_cops + 1.  
WA_FCAT-FIELDNAME = 'MATNR'.  
WA_FCAT-COL_POS = w_cops.  
WA_FCAT-SELTEXT_M = 'Material Number'.  
APPEND WA_FCAT TO IT_FCAT.  
CLEAR wa_fcat.
```

```
w_cops = w_cops + 1.  
WA_FCAT-FIELDNAME = 'MAKTX'.  
WA_FCAT-COL_POS = w_cops.  
WA_FCAT-SELTEXT_M = 'Material Description'.  
APPEND WA_FCAT TO IT_FCAT.  
CLEAR wa_fcat.
```

```
w_cops = w_cops + 1.  
WA_FCAT-FIELDNAME = 'CHARG'.  
WA_FCAT-COL_POS = w_cops.  
WA_FCAT-SELTEXT_M = 'Batch'.  
APPEND WA_FCAT TO IT_FCAT.  
CLEAR wa_fcat.
```

```
w_cops = w_cops + 1.  
WA_FCAT-FIELDNAME = 'FKIMG'.  
WA_FCAT-COL_POS = w_cops.  
WA_FCAT-SELTEXT_M = 'Billed Quantity'.  
APPEND WA_FCAT TO IT_FCAT.  
CLEAR wa_fcat.
```

```
w_cops = w_cops + 1.  
WA_FCAT-FIELDNAME = 'VRKME'.  
WA_FCAT-COL_POS = w_cops.  
WA_FCAT-SELTEXT_M = 'Sales Unit'.  
APPEND WA_FCAT TO IT_FCAT.  
CLEAR wa_fcat.
```

```
w_cops = w_cops + 1.  
WA_FCAT-FIELDNAME = 'WAERK'.  
WA_FCAT-COL_POS = w_cops.  
WA_FCAT-SELTEXT_M = 'Invoice Currency'.  
APPEND WA_FCAT TO IT_FCAT.  
CLEAR wa_fcat.
```

```
w_cops = w_cops + 1.
```

```

WA_FCAT-FIELDNAME = 'BP'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'BASE PRICE IN INR'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'BED'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'BED IN INR'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'ECESS'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'ECESS IN INR'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'SHCESS'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'SHCESS'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'VAT'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'VAT'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'KZWI3'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'Insurance'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'KZWI4'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'Freight'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'KZWI6'.
WA_FCAT-COL_POS = w_cops.

```

```

WA_FCAT-SELTEXT_M = 'Commission Amount'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'INCO1'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'INCO1'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

w_cops = w_cops + 1.
WA_FCAT-FIELDNAME = 'INCO2'.
WA_FCAT-COL_POS = w_cops.
WA_FCAT-SELTEXT_M = 'INCO2'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR wa_fcat.

ENDFORM.
*****
***** FORM DISPLAY *****
FORM DISPLAY.
  DATA WA_LAYOUT TYPE SLIS_LAYOUT_ALV.

    WA_LAYOUT-COLWIDTH_OPTIMIZE = 'X'.
    WA_LAYOUT-ZEBRA = 'X'.

    CALL FUNCTION 'REUSE_ALV_LIST_DISPLAY'
      EXPORTING
        I_CALLBACK_PROGRAM = SY-CPROG
        IS_LAYOUT          = WA_LAYOUT
        IT_FIELDCAT        = IT_FCAT
        I_SAVE             = 'X'
      TABLES
        T_OUTTAB           = IT_FINAL.
    ENDFORM.
*****

```

## → DISPLAY ACCOUNT RECEIVABLE AGING REPORT

The report will have the following displayed fields

**BSID**  
 BUDAT  
 KUNNR  
 BUKRS  
 HKONT  
 +  
 —KUNNR—  
 —BUKRS—  
 —HKONT—  
 +  
 UMSKZ  
 DMBTR  
 —BUKRS—  
 —KUNNR—  
 —HKONT—  
 SHKZG

Input fields

Link fields

Output fields

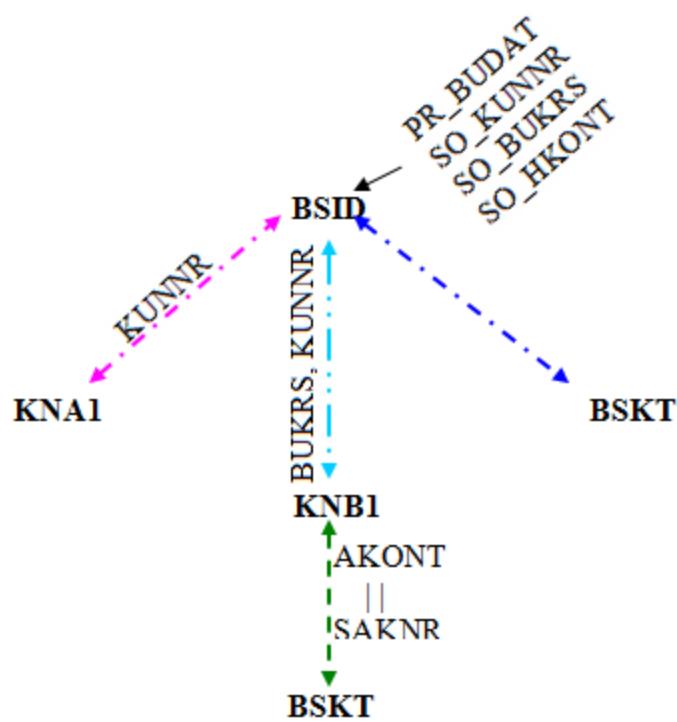
**KNA1**  
 KUNNR  
 NAME1  
 NAME2

**KNB1**  
 KUNNR  
 BUKRS  
 AKONT

**SKAT**  
 SAKNR  
 TXT50

Arrange the fields in order as in data base table

Create the selection criteria.



Selection Criteria	
PR_BUDAT	<input type="text"/>
SO_BUKRS	<input type="text"/> to <input type="text"/>
SO_KUNNR	<input type="text"/> to <input type="text"/>
SO_HKONT	<input type="text"/> to <input type="text"/>

IT\_BSID

BUKRS	KUNNR	UMSKZ	BELNR	BUDAT	SHKZG	DMBTR	HKONT
1000	241	-	500035	25.11.2014	S	140.00	101010
1000	241	-	500037	27.11.2014	S	750.00	101010
1000	241	-	500039	29.11.2014	H	120.00	101010
1000	241	-	500071	30.12.2014	S	1500.00	101010
1000	241	-	500079	15.01.2015	S	1300.00	101010
1000	241	-	500091	25.02.2015	S	120.00	101010
1000	295	-	500099	03.02.2015	S	1150.00	101010
1000	295	-	500111	04.03.2015	H	140.00	101010
1000	295	-	5000191	05.03.2015	S	1000.00	101010
2000	295	-	5000211	13.01.2015	S	900.00	101010
2000	295	-	5000215	15.02.2015	S	810.00	101010
2000	241	-	5000291	31.01.2015	S	750.00	101010
2000	241	-	5000315	20.03.2015	S	100.00	101010
2000	241	-	5000395	19.03.2015	S	75.00	101010

WA\_BSID

BUKRS	KUNNR	UMSKZ	BELNR	BUDAT	SHKZG	DMBTR	HKONT

WA\_FINAL

KUNNR	BUKRS	HKONT	NYD	AGE30	AGE60	AGE90	AGE120	AGE180	AGE365	AGE2Y	AGE3Y	AGE3YA

IT\_FINAL

KUNNR	BUKRS	HKONT	NYD	AGE30	AGE60	AGE90	AGE120	AGE180	AGE365	AGE2Y	AGE3Y	AGE3YA
241	1000	101010		195.00		2800.00	770.00					
295	1000	101010		860.00	1150.00							
295	2000	101010			810.00	900						
241	2000	101010	100.00		750.00							

#### REPORT ZPORGRAM2.

```
*****
* PROGRAM      : ZVR001-SPT_ACCOUNT RECEIVABLE AGING REPORT*
* AUTHOR       : SATISH                                     *
* PURPOSE      : ACCOUNT RECEIVABLE AGING REPORT          *
* START DATE   : 19.03.2015                                *
* FINISH DATE  : 22.03.2015                                *
* SUPPLIER     : SJF TECHNOLOGIES                         *
* PACKAGE      :                                         *
* REQUEST NUM: SRYK900117                                *
*****
```

```

TYPE-POOLS SLIS.

DATA W_CPOS TYPE I.
DATA V_DAYS(10) TYPE C.

TYPES : BEGIN OF TY_BSID,
        BUKRS TYPE BSID-BUKRS,
        KUNNR TYPE BSID-KUNNR,
        UMSKZ TYPE BSID-UMSKZ,
        BELNR TYPE BSID-BELNR,
        BUDAT TYPE BSID-BUDAT,
        SHKZG TYPE BSID-SHKZG,
        DMBTR TYPE BSID-DMBTR,
        HKONT TYPE BSID-HKONT,
        END OF TY_BSID.

DATA: WA_BSID TYPE TY_BSID,
      IT_BSID TYPE TABLE OF TY_BSID.

TYPES : BEGIN OF TY_KNA1,
        KUNNR TYPE KNA1-KUNNR,
        NAME1 TYPE KNA1-NAME1,
        NAME2 TYPE KNA1-NAME2,
        END OF TY_KNA1.

DATA: WA_KNA1 TYPE TY_KNA1,
      IT_KNA1 TYPE TABLE OF TY_KNA1.

TYPES : BEGIN OF TY_KNB1,
        KUNNR TYPE KNB1-KUNNR,
        BUKRS TYPE KNB1-BUKRS,
        AKONT TYPE KNB1-AKONT,
        END OF TY_KNB1.

DATA: WA_KNB1 TYPE TY_KNB1,
      IT_KNB1 TYPE TABLE OF TY_KNB1.

TYPES : BEGIN OF TY_SKAT,
        SAKNR TYPE SKAT-SAKNR,
        TXT50 TYPE SKAT-TXT50,
        END OF TY_SKAT.

DATA: WA_SKAT TYPE TY_SKAT,
      IT_SKAT TYPE TABLE OF TY_SKAT,
      WA_SKAT1 TYPE TY_SKAT,
      IT_SKAT1 TYPE TABLE OF TY_SKAT.

* DELCATE IT_FINAL1, WA_FINAL1
DATA: BEGIN OF WA_FINAL1,
      KUNNR TYPE KNA1-KUNNR,
      NAME1 TYPE KNA1-NAME1,
      NAME2 TYPE KNA1-NAME2,
      BUKRS TYPE KNB1-BUKRS,
      AKONT TYPE KNB1-AKONT,
      RDES TYPE SKAT-TXT50,
      HKONT TYPE BSID-HKONT,
      TXT50 TYPE SKAT-TXT50,

```

```

UMSKZ TYPE BSID-UMSKZ,
NYD TYPE BSID-DMBTR,
AGE30 TYPE BSID-DMBTR,
AGE60 TYPE BSID-DMBTR,
AGE90 TYPE BSID-DMBTR,
AGE120 TYPE BSID-DMBTR,
AGE180 TYPE BSID-DMBTR,
AGE365 TYPE BSID-DMBTR,
AGE2Y TYPE BSID-DMBTR,
AGE3Y TYPE BSID-DMBTR,
AGE3YA TYPE BSID-DMBTR,
TOTAL TYPE BSID-DMBTR,
END OF WA_FINAL1.

DATA IT_FINAL1 LIKE TABLE OF WA_FINAL1.
*****  

* SELECTION-SCREEN
TABLES BSID.
SELECTION-SCREEN BEGIN OF BLOCK A WITH FRAME.
PARAMETER PR_BUDAT TYPE BSID-BUDAT.
SELECT-OPTIONS: SO_BUKRS FOR BSID-BUKRS,
                 SO_KUNNR FOR BSID-KUNNR,
                 SO_HKONT FOR BSID-HKONT.
SELECTION-SCREEN END OF BLOCK A.

* Declare the field catalog.
DATA: IT_FCAT TYPE SLIS_T_FIELDCAT_ALV,
      WA_FCAT LIKE LINE OF IT_FCAT.

DATA WA_LAYOUT TYPE SLIS_LAYOUT_ALV.

* LOGIC
SELECT BUKRS KUNNR UMSKZ BELNR BUDAT SHKZG DMBTR HKONT FROM BSID INTO
TABLE IT_BSID WHERE BUKRS IN SO_BUKRS AND KUNNR IN SO_KUNNR
AND HKONT IN SO_HKONT AND BUDAT <= PR_BUDAT.

IF IT_BSID IS NOT INITIAL.
  SELECT KUNNR NAME1 NAME2 FROM KNA1 INTO TABLE IT_KNA1 FOR ALL ENTRIES
  IN IT_BSID WHERE KUNNR = IT_BSID-KUNNR.
  SELECT KUNNR BUKRS AKONT FROM KNB1 INTO TABLE IT_KNB1 FOR ALL ENTRIES
  IN IT_BSID WHERE KUNNR = IT_BSID-KUNNR AND BUKRS = IT_BSID-BUKRS.
  SELECT SAKNR TXT50 FROM SKAT INTO TABLE IT_SKAT FOR ALL ENTRIES IN
  IT_BSID WHERE SAKNR = IT_BSID-HKONT AND SPRAS = SY-LANGU.
ENDIF.

IF IT_KNB1 IS NOT INITIAL.
  SELECT SAKNR TXT50 FROM SKAT INTO TABLE IT_SKAT1 FOR ALL ENTRIES IN
  IT_KNB1 WHERE SAKNR = IT_KNB1-AKONT AND SPRAS = SY-LANGU.
ENDIF.
* End logic

```

```

DATA: BEGIN OF WA_FINAL,
      BUKRS TYPE BSID-BUKRS,
      KUNNR TYPE BSID-KUNNR,
      HKONT TYPE BSID-HKONT,
      NYD TYPE BSID-DMBTR,
      AGE30 TYPE BSID-DMBTR,
      AGE60 TYPE BSID-DMBTR,
      AGE90 TYPE BSID-DMBTR,
      AGE120 TYPE BSID-DMBTR,
      AGE180 TYPE BSID-DMBTR,
      AGE365 TYPE BSID-DMBTR,
      AGE2Y TYPE BSID-DMBTR,
      AGE3Y TYPE BSID-DMBTR,
      AGE3YA TYPE BSID-DMBTR,
      END OF WA_FINAL.

DATA IT_FINAL LIKE HASHED TABLE OF WA_FINAL WITH UNIQUE KEY BUKRS KUNNR
HKONT.

LOOP AT IT_BSID INTO WA_BSID.
  WA_FINAL-BUKRS = WA_BSID-BUKRS.
  WA_FINAL-KUNNR = WA_BSID-KUNNR.
  WA_FINAL-HKONT = WA_BSID-HKONT.
  IF BSID-SHKZG = 'H'.
    WA_BSID-DMBTR = 0 - WA_BSID-DMBTR.
  ENDIF.
  V_DAYS = PR_BUDAT - WA_BSID-BUDAT.
  IF V_DAYS = 0.
    WA_FINAL-NYD = WA_BSID-DMBTR.
  ELSEIF V_DAYS > 0 AND V_DAYS <= 30.
    WA_FINAL-AGE30 = WA_BSID-DMBTR.
  ELSEIF V_DAYS > 30 AND V_DAYS <= 60.
    WA_FINAL-AGE60 = WA_BSID-DMBTR.
  ELSEIF V_DAYS > 60 AND V_DAYS <= 90.
    WA_FINAL-AGE90 = WA_BSID-DMBTR.
  ELSEIF V_DAYS > 90 AND V_DAYS <= 120.
    WA_FINAL-AGE120 = WA_BSID-DMBTR.
  ELSEIF V_DAYS > 120 AND V_DAYS <= 180.
    WA_FINAL-AGE180 = WA_BSID-DMBTR.
  ELSEIF V_DAYS > 180 AND V_DAYS <= 365.
    WA_FINAL-AGE365 = WA_BSID-DMBTR.
  ELSEIF V_DAYS > 365 AND V_DAYS <= 730.
    WA_FINAL-AGE2Y = WA_BSID-DMBTR.
  ELSEIF V_DAYS > 730 AND V_DAYS <= 1095.
    WA_FINAL-AGE3Y = WA_BSID-DMBTR.
  ELSE.
    WA_FINAL-AGE3YA = WA_BSID-DMBTR.
  ENDIF.
  COLLECT WA_FINAL INTO IT_FINAL.
  CLEAR WA_FINAL.
ENDLOOP.

```

```

*FILLING IT_FINAL1
LOOP AT IT_FINAL INTO WA_FINAL.
  MOVE-CORRESPONDING WA_FINAL TO WA_FINAL1.
  READ TABLE IT_KNA1 INTO WA_KNA1 WITH KEY KUNNR = WA_FINAL-KUNNR.
  IF SY-SUBRC = 0.
    WA_FINAL1-NAME1 = WA_KNA1-NAME1.
    WA_FINAL1-NAME2 = WA_KNA1-NAME2.
  ENDIF.
  READ TABLE IT_KNB1 INTO WA_KNB1 WITH KEY KUNNR = WA_FINAL-KUNNR BUKRS =
WA_FINAL-BUKRS.
  IF SY-SUBRC = 0.
    WA_FINAL1-AKONT = WA_KNB1-AKONT.
    READ TABLE IT_SKAT1 INTO WA_SKAT1 WITH KEY WA_KNB1-AKONT.
    IF SY-SUBRC = 0.
      WA_FINAL1-RDES = WA_SKAT1-TXT50.
    ENDIF.
  ENDIF.
  READ TABLE IT_SKAT INTO WA_SKAT WITH KEY SAKNR = WA_FINAL-HKONT.
  IF SY-SUBRC = 0.
    WA_FINAL1-TXT50 = WA_SKAT-TXT50.
  ENDIF.
  READ TABLE IT_BSID INTO WA_BSID WITH KEY KUNNR = WA_FINAL-KUNNR BUKRS =
WA_FINAL-BUKRS HKONT = WA_FINAL-HKONT.
  IF SY-SUBRC = 0.
    WA_FINAL1-UMSKZ = WA_BSID-UMSKZ.
  ENDIF.
  WA_FINAL1-TOTAL = WA_FINAL-NYD + WA_FINAL-AGE30 + WA_FINAL-AGE60 +
WA_FINAL-AGE90 + WA_FINAL-AGE120 + WA_FINAL-AGE180 + WA_FINAL-AGE365
+ WA_FINAL-AGE2Y + WA_FINAL-AGE3Y + WA_FINAL-AGE3YA.
  APPEND WA_FINAL1 TO IT_FINAL1.
  CLEAR WA_FINAL1.
ENDLOOP.

*****
*PERFORM STATEMENTS.
PERFORM FCAT.
PERFORM DISPLAY.

*****
*Fillng the field catalog
FORM FCAT.
  W_CPOS = W_CPOS + 1.
  WA_FCAT-FIELDNAME = 'KUNNR'.
  WA_FCAT-COL_POS = W_CPOS.
  WA_FCAT-SELTEXT_M = 'CUSTOMER'.
  APPEND WA_FCAT TO IT_FCAT.
  CLEAR WA_FCAT.

  W_CPOS = W_CPOS + 1.
  WA_FCAT-FIELDNAME = 'NAME1'.

```

```

WA_FCAT-COL_POS      = W_CPOS.
WA_FCAT-SELTEXT_M   = 'NAME1'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME   = 'NAME2'.
WA_FCAT-COL_POS      = W_CPOS.
WA_FCAT-SELTEXT_M   = 'NAME2'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME   = 'BUKRS'.
WA_FCAT-COL_POS      = W_CPOS.
WA_FCAT-SELTEXT_M   = 'COMPANY'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME   = 'AKONT'.
WA_FCAT-COL_POS      = W_CPOS.
WA_FCAT-SELTEXT_M   = 'RECON'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME   = 'RDES'.
WA_FCAT-COL_POS      = W_CPOS.
WA_FCAT-SELTEXT_M   = 'RECON DESCRIPTION'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME   = 'HKONT'.
WA_FCAT-COL_POS      = W_CPOS.
WA_FCAT-SELTEXT_M   = 'G/L ACCOUNT'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME   = 'TXT50'.
WA_FCAT-COL_POS      = W_CPOS.
WA_FCAT-SELTEXT_M   = 'G/L LONG DES'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME   = 'UMSKZ'.
WA_FCAT-COL_POS      = W_CPOS.
WA_FCAT-SELTEXT_M   = 'SPECIAL G/L IND'.

```

```

APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME = 'NYD'.
WA_FCAT-COL_POS = W_CPOS.
WA_FCAT-SELTEXT_M = 'NOT YET DUE'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME = 'AGE30'.
WA_FCAT-COL_POS = W_CPOS.
WA_FCAT-SELTEXT_M = '0-30 DAYS'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME = 'AGE60'.
WA_FCAT-COL_POS = W_CPOS.
WA_FCAT-SELTEXT_M = '30-60 DAYS'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME = 'AGE90'.
WA_FCAT-COL_POS = W_CPOS.
WA_FCAT-SELTEXT_M = '60-90 DAYS'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME = 'AGE120'.
WA_FCAT-COL_POS = W_CPOS.
WA_FCAT-SELTEXT_M = '90-120 DAYS'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME = 'AGE180'.
WA_FCAT-COL_POS = W_CPOS.
WA_FCAT-SELTEXT_M = '120-180 DAYS'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME = 'AGE365'.
WA_FCAT-COL_POS = W_CPOS.
WA_FCAT-SELTEXT_M = '180 DAYS-1 YEAR'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

```

```

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME = 'AGE2Y'.
WA_FCAT-COL_POS = W_CPOS.
WA_FCAT-SELTEXT_M = '1-2 YEARS'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME = 'AGE3Y'.
WA_FCAT-COL_POS = W_CPOS.
WA_FCAT-SELTEXT_M = '2-3 YEARS'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME = 'AGE3YA'.
WA_FCAT-COL_POS = W_CPOS.
WA_FCAT-SELTEXT_M = '3 YEARS ABOVE'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

W_CPOS = W_CPOS + 1.
WA_FCAT-FIELDNAME = 'TOTAL'.
WA_FCAT-COL_POS = W_CPOS.
WA_FCAT-SELTEXT_M = 'TOTAL'.
APPEND WA_FCAT TO IT_FCAT.
CLEAR WA_FCAT.

ENDFORM.

* DISPLAY THE FIELD CATALOG.

FORM DISPLAY.
  CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
    EXPORTING
      I_CALLBACK_PROGRAM = SY-CPROG
      IS_LAYOUT          = WA_LAYOUT
      IT_FIELDCAT        = IT_FCAT
      I_SAVE             = 'X'
    TABLES
      T_OUTTAB           = IT_FINAL1.
ENDFORM.

```

→ Develop a program to display the Raw material availability.

Input screen

Plant	F101
Scheduled start	01.07.2017
Scheduled start time	00:00:00

You see the output in the below screen.

Order Number	Material	Material Description	Plant	Schedule start date	Schedule start time	Quantity	Status
000085251381	000000000004000602	N G KNIFING PASTE FILLER	F101	31.08.2017	05:04:31	20.000	Ok
007000000005	000000000004000001	NEROLAC SYN CLR VARNISH	F101	31.08.2017	07:00:00	10.000	Not Ok
007000000006	000000000004000001	NEROLAC SYN CLR VARNISH	F101	31.08.2017	07:00:00	5.000	Not Ok
000085253504	000000000004000602	N G KNIFING PASTE FILLER	F101	01.09.2017	05:04:31	20.000	Not Ok
007000000020	000000000004000032	THINNER FOR HEAT REST MUFFLER PAINT	F101	31.10.2017	07:00:00	1,000.000	Ok
007000000021	000000000004000684	NEROLAC EXCEL IEX3 CCD BASE PAINT_C2	F101	31.10.2017	12:55:51	5,000.000	Not Ok

Here in the last column (Status) ok, not is coming. OK means materials are available for that order. Not ok means raw materials are not available. If you double click on not ok then you have to display how much quantity of raw materials shortage is there. You can see the below screen for shortage.

Material Num	Material des	Plant	Available Quantity	Required Quantity	Shortage Quantity	Alternate Material Status
000000000002300077	LONG OIL SOYA ALKYD (HIGH VISC.)	F101	0.000	7.899	7.899-	No alternate Material

```

REPORT z_piv_conf_new.
* ****
* Program Name : Z_PIV_CONF_NEW
* Technical Consultant : SATISH REDDY
* Functional Consultant : SUHEEL
* Description : Raw material availability check report*
* T.Code : ZPP_ORD_RM
* Date : 06.09.2017
* Client/Class : DEV 120/ ZPP
* ****

INCLUDE z_piv_conf_new_dic.
INCLUDE z_piv_conf_new_sel_screen.

START-OF-SELECTION.
  INCLUDE z_piv_conf_new_fetch_data.

END-OF-SELECTION.
  INCLUDE z_piv_conf_new_display.

AT SELECTION-SCREEN.
  IF s_gstrs-low+6(2) BETWEEN '05' AND '19'.
    if s_gstrs-high is not initial.
      IF s_gstrs-high+6(2) NOT BETWEEN '05' AND '19'.

```

```

MESSAGE 'Please check planing blocks' TYPE 'E'.
ENDIF.
endif.
ELSE.
if s_gstrs-high is not initial.
IF s_gstrs-high+6(2) BETWEEN '05' AND '19'.
MESSAGE 'Please check planing blocks' TYPE 'E'.
ENDIF.
endif.
ENDIF.
*-----*
*& Include          Z_PIV_CONF_NEW_DIC
*-----*
TABLES: mard, afko.
TYPE-POOLS slis.
***** types declaration.
TYPES: BEGIN OF ty_afko,
        aufnr  TYPE afko-aufnr,
        plnbez TYPE afko-plnbez,
        gamng  TYPE afko-gamng,
        rsnum   TYPE afko-rsnum,
        gstrs   TYPE afko-gstrs,
        gsuzs   TYPE afko-gsuzs,
    END OF ty_afko.

TYPES: BEGIN OF ty_mard,
        werks  TYPE mard-werks,
        matnr  TYPE mard-matnr,
    END  OF ty_mard.

TYPES: BEGIN OF ty_caufv,
        aufnr  TYPE caufv-aufnr,
        objnr  TYPE caufv-objnr,
        werks  TYPE caufv-werks,
    END OF ty_caufv.

TYPES: BEGIN OF ty_jest,
        objnr  TYPE jest-objnr,
        stat   TYPE jest-stat,
    END OF ty_jest.
TYPES: BEGIN OF ty_makt,
        matnr  TYPE makt-matnr,
        maktx  TYPE makt-maktx,
    END OF ty_makt.

TYPES: BEGIN OF ty_mard1,
        werks  TYPE mard-werks,
        aufnr  TYPE afko-aufnr,
        gamng  TYPE afko-gamng,
        rsnum   TYPE afko-rsnum,
        plnbez  TYPE afko-plnbez,
        gstrs   TYPE afko-gstrs,
        gsuzs   TYPE afko-gsuzs,
        objnr   TYPE jest-objnr,
        stat    TYPE jest-stat,
    END OF ty_mard1.

TYPES: BEGIN OF ty_final,
        aufnr   TYPE afko-aufnr,
        plnbez  TYPE afko-plnbez,
        maktx   TYPE makt-maktx,

```

```

werks      TYPE mard-werks,
gstrs      TYPE afko-gstrs,
rsnum      TYPE afko-rsnum,
gsuzs      TYPE afko-gsuzs,
gamng      TYPE afko-gamng,
status(10) TYPE c,
END OF ty_final.

TYPES: BEGIN OF ty_resb,
        rsnum  TYPE resb-rsnum,
        matnr  TYPE resb-matnr,
        werks  TYPE resb-werks,
        bdmng  TYPE resb-bdmng,
        lgort   TYPE resb-lgort,
        charg   TYPE resb-charg,
        rstrpos  TYPE resb-rstrpos,
        rsart   TYPE resb-rsart,
END OF ty_resb.

TYPES: BEGIN OF ty_resb1,
        matnr   TYPE resb-matnr,
        rsnum(10) TYPE c,
        werks   TYPE resb-werks,
        bdmng   TYPE resb-bdmng,
        lgort   TYPE resb-lgort,
        charg   TYPE resb-charg,
        aufnr   TYPE afko-aufnr,
        gamng   TYPE afko-gamng,
        plnbez  TYPE afko-plnbez,
        gstrs   TYPE afko-gstrs,
        gsuzs   TYPE afko-gsuzs,
        objnr   TYPE jest-objnr,
        stat    TYPE jest-stat,
        remain_qty type mard-labst,
        avail_stock type mard-labst,           "mard data
        required_qty type mard-labst,
        ALT_MAT  TYPE zrm_alt-rmat,
        alt_mat_stat(30) type c,
END OF ty_resb1.

TYPES: BEGIN OF ty_mchb,
        matnr  TYPE mchb-matnr,
        werks  TYPE mchb-werks,
        lgort   TYPE mchb-lgort,
        charg   TYPE mchb-charg,
        clabs   TYPE mchb-clabs,
END OF ty_mchb.

TYPES: BEGIN OF ty_res,
        rsnum(10) TYPE c,
        res     TYPE i,
END OF ty_res.

TYPES: BEGIN OF ty_mard_c,
        matnr  TYPE mara-matnr,
        werks  TYPE mard-werks,
        lgort   TYPE mard-lgort,
        labst   TYPE mard-labst,
END OF ty_mard_c.

TYPES: BEGIN OF ty_mard_list2,

```

```

matnr TYPE mara-matnr,
maktx TYPE makt-maktx,
werks TYPE marc-werks,
lgort TYPE mard-lgort,
labst TYPE labst,
bdmng TYPE bdmng,
avail_qty type bdmng,
diff TYPE bdmng,
alt_mat_stat(30) type c,
END OF ty_mard_list2.

```

```

TYPES: BEGIN OF ty_list1,
  matnr TYPE mara-matnr,
  werks TYPE marc-werks,
  lgort TYPE mard-lgort,
  labst TYPE labst,
  bdmng TYPE bdmng,
  avail_qty type bdmng,
  diff TYPE bdmng,
  alt_mat_stat(30) type c,
END OF ty_list1.

```

\*\*\*\*\* internal tables and work areas.

```

DATA: it_mard          TYPE STANDARD TABLE OF ty_mard,
      it_mard1        TYPE STANDARD TABLE OF ty_mard1,
      wa_mard          TYPE ty_mard,
      wa_mard1         TYPE ty_mard1,
      it_afko          TYPE STANDARD TABLE OF ty_afko,
      wa_afko          TYPE ty_afko,
      it_caufv         TYPE STANDARD TABLE OF ty_caufv,
      wa_caufv         TYPE ty_caufv,
      it_jest           TYPE STANDARD TABLE OF ty_jest,
      wa_jest           TYPE ty_jest,
      it_makt           TYPE STANDARD TABLE OF ty_makt,
      it_makt_not_ok   TYPE STANDARD TABLE OF ty_makt,
      it_makt1          TYPE STANDARD TABLE OF ty_makt,
      wa_makt           TYPE ty_makt,
      wa_makt_not_ok   TYPE ty_makt,
      wa_makt1          TYPE ty_makt,
      it_final          TYPE STANDARD TABLE OF ty_final,
      wa_final          TYPE ty_final,
      it_fcat           TYPE slis_t_fieldcat_alv,
      wa_fcat           LIKE LINE OF it_fcat,
      wa_layout          TYPE slis_layout_alv,
      it_resb            TYPE STANDARD TABLE OF ty_resb,
      it_resb_list       TYPE STANDARD TABLE OF ty_resb,
      it_resb_list_not_ok TYPE STANDARD TABLE OF ty_resb,
      it_resb1           TYPE STANDARD TABLE OF ty_resb1,
      it_resb_temp        TYPE STANDARD TABLE OF ty_resb1,
      it_resb1_list       TYPE STANDARD TABLE OF ty_resb1,
      it_resb1_list_not_ok TYPE STANDARD TABLE OF ty_resb1,
      wa_resb             TYPE ty_resb,
      wa_resb_list        TYPE ty_resb,
      wa_resb_list_not_ok TYPE ty_resb,
      wa_resb1            TYPE ty_resb1,
      wa_resb2            TYPE ty_resb1,
      wa_resb1_list       TYPE ty_resb1,
      wa_resb1_list_not_ok TYPE ty_resb1,
      wa_resb1_list1      TYPE ty_resb1,
      wa_resb_temp         TYPE ty_resb1,

```

```

it_mchb          TYPE STANDARD TABLE OF ty_mchb,
wa_mchb          TYPE ty_mchb,
it_res           TYPE STANDARD TABLE OF ty_res,
wa_res           TYPE ty_res,
it_mard_c        TYPE STANDARD TABLE OF ty_mard_c,
it_mard_list     TYPE STANDARD TABLE OF ty_mard_c,
it_mard_list_not_ok  TYPE STANDARD TABLE OF ty_mard_c,
it_mard_list1    TYPE STANDARD TABLE OF ty_mard_c,
wa_mard_c        TYPE ty_mard_c,
wa_mard_list     TYPE ty_mard_c,
wa_mard_list_not_ok  TYPE ty_mard_c,
wa_mard_list1    TYPE ty_mard_c,
it_mard_list2    TYPE STANDARD TABLE OF ty_mard_list2,
it_final2         TYPE STANDARD TABLE OF ty_mard_list2,
it_final_ok      TYPE STANDARD TABLE OF ty_mard_list2,
wa_mard_list2    TYPE ty_mard_list2,
wa_final2         TYPE ty_mard_list2,
wa_final_ok      TYPE ty_mard_list2,
it_mard_list2_fcat  TYPE slis_t_fieldcat_alv,
it_ok_fcat       TYPE slis_t_fieldcat_alv,
wa_mard_list2_fcat  LIKE LINE OF it_mard_list2_fcat,
wa_ok_fcat       LIKE LINE OF it_mard_list2_fcat,
it_list1_type    STANDARD TABLE OF ty_list1,
wa_list1_type    ty_list1,
it_ok_type       STANDARD TABLE OF ty_list1,
wa_ok_type       ty_list1.

```

\*\*\*\*\* variables used in this report

```

DATA: dup_matnr    TYPE matnr,
lv_clabs       TYPE mchb-clabs,
lv_clabs1      TYPE mchb-clabs,
v              TYPE i,
available_qty  TYPE resb-bdmng,
remain_qty     TYPE resb-bdmng,
remain_qty1    TYPE resb-bdmng,
TOT_STK        TYPE resb-bdmng,
old_stk(1)     type c,
alt(1)         type c,
no_alt(1)      type c,
mat_avail(10)   TYPE c,
tot_quan       TYPE labst,
temp_labst    TYPE mard-labst,
mat            TYPE mara-matnr.

```

```

TYPES: BEGIN OF ty_f1,
  aufnr  TYPE afko-aufnr,
  matnr  TYPE mara-matnr,
  werks  TYPE mard-werks,
  rsnum   TYPE afko-rsnum,
END OF ty_f1.

```

```

DATA: it_f1  TYPE STANDARD TABLE OF ty_f1,
wa_f1   TYPE ty_f1.

```

```

TYPES: BEGIN OF ty_xyz,
  matnr  TYPE mara-matnr,
  aufnr  TYPE afko-aufnr,
  rsnum   TYPE resb-rsnum,
  werks  TYPE mard-werks,
  bdmng  TYPE resb-bdmng,
  lgort   TYPE mard-lgort,

```

```

        labst TYPE mard-labst,
        END OF ty_xyz.
DATA: it_xyz TYPE STANDARD TABLE OF ty_xyz,
      wa_xyz TYPE ty_xyz.

types: begin of ty_zrm_alt,
      rm type zrm_alt-rm,
      rmat type zrm_alt-rmat,
      end of ty_zrm_alt.

data: it_zrm_alt type STANDARD TABLE OF TY_ZRM_ALT,
      WA_ZRM_ALT TYPE TY_ZRM_ALT.
*****->
*&-----*
*&   Include          Z_PIV_CONF_NEW_SEL_SCREEN
*&-----*
SELECTION-SCREEN begin of BLOCK a with frame.
  SELECT-OPTIONS: s_werks for mard-werks no INTERVALS,
                  s_GSTRS for afko-GSTRS,
                  s_GSUZS for afko-GSUZS.
  SELECTION-SCREEN end of block a.
*****->
*&-----*
*&   Include          Z_PIC_CONF_NEW_FETCH_DATA
*&-----*
SELECT aufnr
      plnbez
      gamng
      rsnum
      gstrs
      gsuzs
FROM afko INTO TABLE it_afko
WHERE gstrs IN s_gstrs
  AND gsuzs IN s_gsuzs.
* and plnbez like '4%'.

SORT it_afko BY aufnr plnbez gstrs gsuzs.

LOOP AT it_afko INTO wa_afko.
  dup_matnr = wa_afko-plnbez.
  SHIFT dup_matnr LEFT DELETING LEADING '0'.
  IF dup_matnr+0(1) <> '4'.
    DELETE it_afko.
  ENDIF.
ENDLOOP.

IF it_afko IS NOT INITIAL.
  SELECT aufnr
    objnr
    werks
  FROM caufv INTO TABLE it_caufv
  FOR ALL ENTRIES IN it_afko
  WHERE aufnr = it_afko-aufnr
    AND werks IN s_werks.
ENDIF.

IF it_caufv IS NOT INITIAL.
  SELECT objnr
    stat
  FROM jest INTO TABLE it_jest
  FOR ALL ENTRIES IN it_caufv
  WHERE objnr = it_caufv-objnr

```

```

        AND stat IN ('I0001','I0076')
        AND inact <> 'X'.
ENDIF.

SORT it_jest BY objnr stat.

LOOP AT it_jest INTO wa_jest.
  IF wa_jest-stat = 'I0076'.
    DELETE it_jest WHERE objnr = wa_jest-objnr.
*    DELETE it_jest WHERE stat = 'I0001'.
  ENDIF.
ENDLOOP.

SORT: it_jest BY objnr,
      it_caufv BY aufnr werks,
      it_afko BY aufnr plnbez.

LOOP AT it_jest INTO wa_jest.
  wa_mard1-objnr = wa_jest-objnr.
  wa_mard1-stat = wa_jest-stat.
  READ TABLE it_caufv INTO wa_caufv
  WITH KEY objnr = wa_mard1-objnr BINARY SEARCH.
  IF sy-subrc = 0.
    wa_mard1-aufnr = wa_caufv-aufnr.
    wa_mard1-werks = wa_caufv-werks.

    READ TABLE it_afko INTO wa_afko
    WITH KEY aufnr = wa_mard1-aufnr BINARY SEARCH.
    IF sy-subrc = 0.
      wa_mard1-plnbez = wa_afko-plnbez.
      wa_mard1-gamng = wa_afko-gamng.
      wa_mard1-rsnum = wa_afko-rsnum.
      wa_mard1-gstrs = wa_afko-gstrs.
      wa_mard1-gsuzs = wa_afko-gsuzs.
      APPEND wa_mard1 TO it_mard1.
      CLEAR: wa_mard1, wa_caufv, wa_jest, wa_afko.
    ENDIF.
  ENDIF.
ENDLOOP.

IF it_mard1 IS NOT INITIAL.
  SELECT rsnum
        matnr
        werks
        bdmng
        lgort
        charg
        rstrpos
        rsart
  FROM resb INTO TABLE it_resb
  FOR ALL ENTRIES IN it_mard1
  WHERE rsnum = it_mard1-rsnum
  AND werks = it_mard1-werks.

  SORT it_resb BY rsnum matnr werks lgort charg bdmng.

  LOOP AT it_resb INTO wa_resb.
    wa_resb_temp-rsnum = wa_resb-rsnum.
    wa_resb_temp-matnr = wa_resb-matnr.
    wa_resb_temp-werks = wa_resb-werks.
    wa_resb_temp-lgort = wa_resb-lgort.

```

```

*      wa_resb_temp-charg = wa_resb-charg.
wa_resb_temp-bdmng = wa_resb-bdmng.
COLLECT wa_resb_temp INTO it_resb_temp.
CLEAR wa_resb_temp.
ENDLOOP.

SELECT matnr
      makte
      FROM makt INTO TABLE it_makt
      FOR ALL ENTRIES IN it_mard1
      WHERE matnr = it_mard1-plnbez
      AND spras = sy-langu.
ENDIF.

IF it_resb_temp IS NOT INITIAL.
  SELECT rm
        rmat
        FROM zrm_alt INTO TABLE it_zrm_alt
        FOR ALL ENTRIES IN it_resb_temp
        WHERE rm = it_resb_temp-matnr.
ENDIF.

SORT it_resb_temp BY rsnum matnr.
SORT it_zrm_alt BY rmat.
SORT it_mard1 BY rsnum.
LOOP AT it_resb_temp INTO wa_resb_temp.
  wa_resb1-rsnum = wa_resb_temp-rsnum.
  wa_resb1-matnr = wa_resb_temp-matnr.                                "Raw Material
  wa_resb1-werks = wa_resb_temp-werks.
  wa_resb1-lgort = wa_resb_temp-lgort.
  wa_resb1-bdmng = wa_resb_temp-bdmng.                                "Required qty
  READ TABLE it_mard1 INTO wa_mard1 WITH KEY rsnum = wa_resb_temp-
rsnum BINARY SEARCH.
  IF sy-subrc = 0.
    wa_resb1-aufnr = wa_mard1-aufnr .
    wa_resb1-gamng = wa_mard1-gamng.
    wa_resb1-plnbez = wa_mard1-plnbez.                                "Material
    wa_resb1-gstros = wa_mard1-gstros .
    wa_resb1-gsuzs = wa_mard1-gsuzs .
    wa_resb1-objnr = wa_mard1-objnr .
    wa_resb1-stat = wa_mard1-stat .
  ENDIF.
  READ TABLE it_zrm_alt INTO wa_zrm_alt WITH KEY rm = wa_resb_temp-
matnr BINARY SEARCH.
  IF sy-subrc = 0.
    wa_resb1-alt_mat = wa_zrm_alt-rmat.
  ENDIF.
  APPEND wa_resb1 TO it_resb1.
  CLEAR wa_resb1.
ENDLOOP.

IF it_resb1 IS NOT INITIAL.
  SELECT matnr
        werks
        lgort
        labst
        FROM mard INTO TABLE it_mard_c
        FOR ALL ENTRIES IN it_resb1
        WHERE matnr = it_resb1-matnr
        AND werks = it_resb1-werks.
ENDIF.

```

```

IF it_resbl IS NOT INITIAL.
  SELECT matnr
    werks
    lgort
    labst
      "available stock
  FROM mard APPENDING TABLE it_mard_c
  FOR ALL ENTRIES IN it_resbl
  WHERE matnr = it_resbl-alt_mat
  AND werks = it_resbl-werks.
ENDIF.

SORT: it_resbl BY matnr gstrs gsuzs rsnum werks lgort bdmng,
      it_mard_c BY matnr werks lgort.

LOOP AT it_resbl INTO wa_resbl.
  wa_resb2 = wa_resbl.

  AT NEW matnr.

    CLEAR: available_qty, lv_clabs, lv_clabs1, remain_qty, remain_qty1, old_stk,
tot_stk, alt, no_alt.
    IF wa_resb2-lgort IS NOT INITIAL.
      READ TABLE it_mard_c INTO wa_mard_c
      WITH KEY matnr = wa_resb2-matnr
      werks = wa_resb2-werks
      lgort = wa_resb2-lgort BINARY SEARCH.
      IF sy-subrc = 0.
        remain_qty = wa_mard_c-labst - wa_resb2-bdmng.

        IF remain_qty < 0.
          IF wa_resb2-alt_mat IS NOT INITIAL.
            LOOP AT it_mard_c INTO wa_mard_c
            WHERE matnr = wa_resb2-alt_mat
            AND werks = wa_resb2-werks.
              lv_clabs1 = lv_clabs1 + wa_mard_c-labst.           "avaialble qty
            ENDLOOP.
            remain_qty1 = lv_clabs1 - wa_resb2-bdmng.

            IF remain_qty1 > remain_qty.
              remain_qty = remain_qty1.
              lv_clabs = lv_clabs1.
              alt = 'X'.
              CLEAR: remain_qty1, lv_clabs1.
            ELSE.
              no_alt = 'X'.
            ENDIF.
          ENDIF.
        ENDIF.

        IF wa_resb2-alt_mat IS NOT INITIAL.
          tot_stk = lv_clabs.
        ELSE.
          tot_stk = wa_mard_c-labst.
        ENDIF.
        old_stk = 'X'.
      ENDIF.

      *      IF wa_resb2-bdmng <= wa_mard_c-labst.
      IF remain_qty > 0.
        wa_res-res = 0.           "it is ok
      ELSE.

```

```

        wa_res-res = 1.                      "it is not ok
ENDIF.

IF alt = 'X'.
    wa_resbl-alt_mat_stat = wa_resb2-alt_mat .
ELSEIF no_alt = 'X'.
    wa_resbl-alt_mat_stat = 'Alternate Mat is not satisfied'.
ENDIF.

wa_res-rsnum = wa_resb2-rsnum.

COLLECT wa_res INTO it_res.
CLEAR wa_res.

IF alt = 'X'.
    wa_resbl-avail_stock = lv_clabs.
ELSE.
    wa_resbl-avail_stock = wa_mard_c-labst.
ENDIF.
wa_resbl-remain_qty = remain_qty .           "remaining qty
wa_resbl-required_qty = wa_resb2-bdmng.
IF wa_resbl-alt_mat_stat+0(1) = '*'.
    CLEAR wa_resbl-alt_mat_stat.
ENDIF.
MODIFY it_resbl FROM wa_resbl TRANSPORTING avail_stock remain_qty required_qty alt_mat_stat WHERE rsnum = wa_resb2-rsnum AND matnr = wa_resb2-matnr.
ENDIF.
ELSE.
LOOP AT it_mard_c INTO wa_mard_c
    WHERE matnr = wa_resb2-matnr
    AND werks = wa_resb2-werks.
    lv_clabs = lv_clabs + wa_mard_c-labst.      "avaialble qty
ENDLOOP.
remain_qty = lv_clabs - wa_resb2-bdmng.

IF remain_qty < 0.
    IF wa_resb2-alt_mat IS NOT INITIAL.
        LOOP AT it_mard_c INTO wa_mard_c
            WHERE matnr = wa_resb2-alt_mat
            AND werks = wa_resb2-werks.
            lv_clabs1 = lv_clabs1 + wa_mard_c-labst.      "avaialble qty
        ENDLOOP.
        remain_qty1 = lv_clabs1 - wa_resb2-bdmng.

        IF remain_qty1 > remain_qty.
            remain_qty = remain_qty1.
            lv_clabs = lv_clabs1.
            alt = 'X'.
            CLEAR: remain_qty1, lv_clabs1.
        ELSE.
            no_alt = 'X'.
       ENDIF.
    ENDIF.
old_stk = 'X'.
tot_stk = lv_clabs.
ENDIF.

IF wa_resb2-bdmng <= lv_clabs.
    wa_res-res = 0.                      "it is ok

```

```

ELSE.
  wa_res-res = 1.           "it is not ok
ENDIF.

IF alt = 'X'.
  wa_resb1-alt_mat_stat = wa_resb2-alt_mat.
ELSEIF no_alt = 'X'.
  wa_resb1-alt_mat_stat = 'Alternate Mat is not satisfied'.

ENDIF.
wa_res-rsnum = wa_resb2-rsnum.
COLLECT wa_res INTO it_res.
CLEAR wa_res.

wa_resb1-avail_stock = lv_clabs .
wa_resb1-remain_qty = remain_qty .           "remain qty
wa_resb1-required_qty = wa_resb2-bdmng .
IF wa_resb1-alt_mat_stat+0(1) = '*'.
  CLEAR wa_resb1-alt_mat_stat.
ENDIF.
MODIFY it_resb1 FROM wa_resb1 TRANSPORTING avail_stock remain_qty required
_qty alt_mat_stat WHERE rsnum = wa_resb2-rsnum AND matnr = wa_resb2-matnr.
ENDIF.
v = 0.
ENDAT.

IF v <> 0.
  IF old_stk = 'X'.
    remain_qty = tot_stk.
  ENDIF.

  lv_clabs = remain_qty.           "available qty = last material remain
qty
  remain_qty = lv_clabs - wa_resb1-bdmng.

  IF remain_qty < 0.
    IF wa_resb2-alt_mat IS NOT INITIAL.
      LOOP AT it_mard_c INTO wa_mard_c
      WHERE matnr = wa_resb2-alt_mat
      AND werks = wa_resb2-werks.
        lv_clabs1 = lv_clabs1 + wa_mard_c-labst.           "avaialble qty
      ENDLOOP.
      remain_qty1 = lv_clabs1 - wa_resb2-bdmng.

      IF remain_qty1 > remain_qty.
        remain_qty = remain_qty1.
        lv_clabs = lv_clabs1.
        alt = 'X'.
        CLEAR: remain_qty1, lv_clabs1.
      ELSE.
        no_alt = 'X'.
      ENDIF.
    ENDIF.

    tot_stk = lv_clabs.
    old_stk = 'X'.
  ENDIF.

  IF wa_resb1-bdmng <= lv_clabs.
    wa_res-res = 0.           "it is ok
  ELSE.

```

```

        wa_res-res = 1.           "it is not ok
ENDIF.

IF alt = 'X'.
  wa_resbl-alt_mat_stat = wa_resb2-alt_mat.
ELSEIF no_alt = 'X'.
  wa_resbl-alt_mat_stat = 'Alternate Mat is not available'.
ENDIF.
wa_res-rsnum = wa_resbl-rsnum.

COLLECT wa_res INTO it_res.
CLEAR wa_res.

wa_resbl-avail_stock = lv_clabs .
wa_resbl-remain_qty = remain_qty .
wa_resbl-required_qty = wa_resb2-bdmng.
MODIFY it_resbl FROM wa_resbl TRANSPORTING avail_stock remain_qty required_qty alt_mat_stat WHERE rsnum = wa_resb2-rsnum AND matnr = wa_resb2-matnr..
ELSE.
  v = 1.
ENDIF.
CLEAR : wa_resb2, wa_resbl.
ENDLOOP.

SORT: it_mard1 BY aufnr plnbez werks rsnum,
      it_makt BY matnr,
      it_res BY rsnum.

LOOP AT it_mard1 INTO wa_mard1.
  wa_final-aufnr = wa_mard1-aufnr.
  wa_final-gamng = wa_mard1-gamng.
  wa_final-plnbez = wa_mard1-plnbez.
  wa_final-gstrs = wa_mard1-gstrs.
  wa_final-rsnum = wa_mard1-rsnum.
  wa_final-gsuzs = wa_mard1-gsuzs.
  wa_final-werks = wa_mard1-werks.

  READ TABLE it_makt INTO wa_makt WITH KEY matnr = wa_final-plnbez BINARY SEARCH.
  IF sy-subrc = 0.
    wa_final-maktx = wa_makt-maktx.
  ENDIF.

  READ TABLE it_res INTO wa_res WITH KEY rsnum = wa_final-rsnum BINARY SEARCH.
  IF sy-subrc = 0.
    IF wa_res-res = 0.
      wa_final-status = 'Ok'.
    ELSEIF wa_res-res >= 1.
      wa_final-status = 'Not Ok'.
    ENDIF.
  ENDIF.
  IF wa_final-status <> ' '.
    APPEND wa_final TO it_final.
    CLEAR wa_final.
  ENDIF.
ENDLOOP.

SORT it_final BY gstrs gsuzs aufnr plnbez werks rsnum.
SORT it_resbl.

wa_fcat-fieldname = 'AUFNR'.

```

```

wa_fcat-col_pos = '1'.
wa_fcat-seltext_m = 'Order Number'.
APPEND wa_fcat TO it_fcat.
CLEAR wa_fcat.

wa_fcat-fieldname = 'PLNBEZ'.
wa_fcat-col_pos = '2'.
wa_fcat-seltext_m = 'Material'.
APPEND wa_fcat TO it_fcat.
CLEAR wa_fcat.

wa_fcat-fieldname = 'MAKTX'.
wa_fcat-col_pos = '3'.
wa_fcat-seltext_m = 'Material Description'.
APPEND wa_fcat TO it_fcat.
CLEAR wa_fcat.

wa_fcat-fieldname = 'WERKS'.
wa_fcat-col_pos = '4'.
wa_fcat-seltext_m = 'Plant'.
APPEND wa_fcat TO it_fcat.
CLEAR wa_fcat.

wa_fcat-fieldname = 'GSTRS'.
wa_fcat-col_pos = '5'.
wa_fcat-seltext_m = 'Schedule start date'.
APPEND wa_fcat TO it_fcat.
CLEAR wa_fcat.

wa_fcat-fieldname = 'GSUZS'.
wa_fcat-col_pos = '6'.
wa_fcat-seltext_m = 'Schedule start time'.
APPEND wa_fcat TO it_fcat.
CLEAR wa_fcat.

wa_fcat-fieldname = 'GAMNG'.
wa_fcat-col_pos = '7'.
wa_fcat-seltext_m = 'Quantity'.
APPEND wa_fcat TO it_fcat.
CLEAR wa_fcat.

wa_fcat-fieldname = 'STATUS'.
wa_fcat-col_pos = '8'.
wa_fcat-seltext_m = 'Status'.
APPEND wa_fcat TO it_fcat.
CLEAR wa_fcat.

wa_layout-zebra = 'X'.
wa_layout-colwidth_optimize = 'X'.
*-----*
*& Include          Z_PIV_CONF_NEW_DISPLAY
*-----*
CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
  EXPORTING
    i_callback_program      = sy-cprog
    i_callback_user_command = 'UC'
    is_layout               = wa_layout
    it_fieldcat             = it_fcat  TABLES
    t_outtab                = it_final .

```

```

FORM uc USING a LIKE sy-ucomm b TYPE slis_selfield.
  REFRESH : it_mard_list2_fcat, it_ok_fcat, it_list1, it_ok, it_final2, it_final
  _ok, it_mard_list2, it_makt.

  IF b-fieldname = 'STATUS'.
    mat_avail = b-value.
  ENDIF.

  IF mat_avail = 'Not Ok'.
    READ TABLE it_final INTO wa_final INDEX b-tabindex.
    IF sy-subrc = 0.

      " code is added by satish on 20.11.17
      LOOP AT it_resb1 INTO wa_resb1 WHERE plnbez = wa_final-
plnbez AND aufnr = wa_final-aufnr.
        wa_list1-matnr = wa_resb1-matnr.
        wa_list1-werks = wa_resb1-werks.
        wa_list1-lgort = wa_resb1-lgort.
        wa_list1-labst = wa_resb1-avail_stock.
        wa_list1-alt_mat_stat = wa_resb1-alt_mat_stat.
        IF wa_list1-alt_mat_stat IS INITIAL.
          wa_list1-alt_mat_stat = 'No alternate Material'.
        ENDIF.
        IF wa_list1-labst < 0.
          wa_list1-labst = 0.
        ENDIF.

        wa_list1-bdmng = wa_resb1-required_qty.
        wa_list1-diff = wa_list1-labst - wa_list1-bdmng.

        IF wa_list1-diff < 0.
          APPEND wa_list1 TO it_list1.
        ENDIF.
        CLEAR wa_list1.
      ENDLOOP.

      LOOP AT it_resb1 INTO wa_resb1 WHERE plnbez = wa_final-
plnbez AND aufnr = wa_final-aufnr.
        wa_list1-matnr = wa_resb1-matnr.
        wa_list1-werks = wa_resb1-werks.
        wa_list1-lgort = wa_resb1-lgort.
        wa_list1-labst = wa_resb1-avail_stock.
        wa_list1-alt_mat_stat = wa_resb1-alt_mat_stat.
        IF wa_list1-alt_mat_stat IS INITIAL.
          wa_list1-alt_mat_stat = 'No alternate Material'.
        ENDIF.
        IF wa_list1-labst < 0.
          wa_list1-labst = 0.
        ENDIF.

        wa_list1-bdmng = wa_resb1-required_qty.
        wa_list1-diff = wa_list1-labst - wa_list1-bdmng.

        IF wa_list1-alt_mat_stat+0(1) = '0'.
          APPEND wa_list1 TO it_list1.
        ENDIF.
        CLEAR wa_list1.
      ENDLOOP.

      SELECT matnr

```

```

        maktx
FROM makt
INTO TABLE it_makt1
FOR ALL ENTRIES IN it_list1
WHERE matnr = it_list1-matnr
AND spras = sy-langu.

LOOP AT it_list1 INTO wa_list1.
wa_final2-matnr = wa_list1-matnr.
wa_final2-werks = wa_list1-werks.
wa_final2-labst = wa_list1-labst.      " available qty
wa_final2-bdmng = wa_list1-bdmng.      "REQ QTY
wa_final2-diff = wa_list1-diff.
wa_final2-alt_mat_stat = wa_list1-alt_mat_stat.

READ TABLE it_makt1 INTO wa_makt1 WITH KEY matnr = wa_final2-matnr.
IF sy-subrc = 0.
    wa_final2-maktx = wa_makt1-maktx.
ENDIF.

APPEND wa_final2 TO it_final2.
CLEAR wa_final2.
ENDLOOP.

wa_mard_list2_fcat-fieldname = 'MATNR'.
wa_mard_list2_fcat-col_pos = '1'.
wa_mard_list2_fcat-seltext_m = 'Material Num'.
APPEND wa_mard_list2_fcat TO it_mard_list2_fcat.
CLEAR wa_mard_list2_fcat.

wa_mard_list2_fcat-fieldname = 'MAKTX'.
wa_mard_list2_fcat-col_pos = '2'.
wa_mard_list2_fcat-seltext_m = 'Material des'.
APPEND wa_mard_list2_fcat TO it_mard_list2_fcat.
CLEAR wa_mard_list2_fcat.

wa_mard_list2_fcat-fieldname = 'WERKS'.
wa_mard_list2_fcat-col_pos = '3'.
wa_mard_list2_fcat-seltext_m = 'Plant'.
APPEND wa_mard_list2_fcat TO it_mard_list2_fcat.
CLEAR wa_mard_list2_fcat.

wa_mard_list2_fcat-fieldname = 'LABST'.
wa_mard_list2_fcat-col_pos = '4'.
wa_mard_list2_fcat-seltext_m = 'Available Quantity'.
APPEND wa_mard_list2_fcat TO it_mard_list2_fcat.
CLEAR wa_mard_list2_fcat.

wa_mard_list2_fcat-fieldname = 'BDMNG'.
wa_mard_list2_fcat-col_pos = '5'.
wa_mard_list2_fcat-seltext_m = 'Required Quantity'.
APPEND wa_mard_list2_fcat TO it_mard_list2_fcat.
CLEAR wa_mard_list2_fcat.

wa_mard_list2_fcat-fieldname = 'DIFF'.
wa_mard_list2_fcat-col_pos = '6'.
wa_mard_list2_fcat-seltext_m = 'Shortage Quantity'.
APPEND wa_mard_list2_fcat TO it_mard_list2_fcat.
CLEAR wa_mard_list2_fcat.

wa_mard_list2_fcat-fieldname = 'ALT_MAT_STAT'.

```

```

wa_mard_list2_fcat-col_pos = '6'.
wa_mard_list2_fcat-seltext_1 = 'Alternate Material Status'.
APPEND wa_mard_list2_fcat TO it_mard_list2_fcat.
CLEAR wa_mard_list2_fcat.

CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
  EXPORTING
    i_callback_program = sy-cprog
    is_layout          = wa_layout
    it_fieldcat        = it_mard_list2_fcat
  TABLES
    t_outtab           = it_final2      .
ENDIF.

ELSE.

READ TABLE it_final INTO wa_final INDEX b-tabindex.
IF sy-subrc = 0.

" code is added by satish on 20.11.17
LOOP AT it_resbl INTO wa_resbl WHERE plnbez = wa_final-
plnbez AND aufnr = wa_final-aufnr.
  wa_ok-matnr = wa_resbl-matnr.
  wa_ok-werks = wa_resbl-werks.
  wa_ok-lgort = wa_resbl-lgort.
  wa_ok-labst = wa_resbl-avail_stock.
  wa_ok-alt_mat_stat = wa_resbl-alt_mat_stat.
  IF wa_ok-labst < 0.
    wa_ok-labst = 0.
  ENDIF.
  IF wa_ok-alt_mat_stat IS INITIAL.
    wa_ok-alt_mat_stat = 'No alternate Material'.
  ENDIF.
  wa_ok-bdmng = wa_resbl-required_qty.
  wa_ok-diff = wa_list1-labst - wa_list1-bdmng.
  IF wa_ok-alt_mat_stat+0(1) = '0'.
    APPEND wa_ok TO it_ok.
  ENDIF.
  CLEAR wa_ok.
ENDLOOP.

SELECT matnr
      makte
  FROM makt
  INTO TABLE it_makt1
  FOR ALL ENTRIES IN it_ok
  WHERE matnr = it_ok-matnr
  AND spras = sy-langu.

LOOP AT it_ok INTO wa_ok.
  wa_final_ok-matnr = wa_ok-matnr.
  wa_final_ok-werks = wa_ok-werks.
  wa_final_ok-labst = wa_ok-labst.      " available qty
  wa_final_ok-bdmng = wa_ok-bdmng.      "REQ QTY
  wa_final_ok-diff = wa_ok-diff.
  wa_final_ok-alt_mat_stat = wa_ok-alt_mat_stat.

  READ TABLE it_makt1 INTO wa_makt1 WITH KEY matnr = wa_final2-matnr.
  IF sy-subrc = 0.
    wa_final_ok-makte = wa_makt1-makte.
  ENDIF.
ENDIF.

```

```

ENDIF.

APPEND wa_final_ok TO it_final_ok.
CLEAR wa_final_ok.
ENDLOOP.

wa_ok_fcat-fieldname = 'MATNR'.
wa_ok_fcat-col_pos = '1'.
wa_ok_fcat-selttext_m = 'Material Num'.
APPEND wa_ok_fcat TO it_ok_fcat.
CLEAR wa_ok_fcat.

wa_ok_fcat-fieldname = 'MAKTX'.
wa_ok_fcat-col_pos = '2'.
wa_ok_fcat-selttext_m = 'Material des'.
APPEND wa_ok_fcat TO it_ok_fcat.
CLEAR wa_ok_fcat.

wa_ok_fcat-fieldname = 'WERKS'.
wa_ok_fcat-col_pos = '3'.
wa_ok_fcat-selttext_m = 'Plant'.
APPEND wa_ok_fcat TO it_ok_fcat.
CLEAR wa_ok_fcat.

wa_ok_fcat-fieldname = 'LABST'.
wa_ok_fcat-col_pos = '4'.
wa_ok_fcat-selttext_m = 'Available Quantity'.
APPEND wa_ok_fcat TO it_ok_fcat.
CLEAR wa_ok_fcat.

wa_ok_fcat-fieldname = 'BDMNG'.
wa_ok_fcat-col_pos = '5'.
wa_ok_fcat-selttext_m = 'Required Quantity'.
APPEND wa_ok_fcat TO it_ok_fcat.
CLEAR wa_ok_fcat.

wa_ok_fcat-fieldname = 'ALT_MAT_STAT'.
wa_ok_fcat-col_pos = '5'.
wa_ok_fcat-selttext_l = 'Alternate Material Status'.
APPEND wa_ok_fcat TO it_ok_fcat.
CLEAR wa_ok_fcat.

IF it_final_ok IS NOT INITIAL.
  CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
    EXPORTING
      i_callback_program = sy-cprog
      is_layout          = wa_layout
      it_fieldcat       = it_ok_fcat
    TABLES
      t_outtab          = it_final_ok
    .
ENDIF.
ENDIF.
ENDIF.
ENDFORM.

```

**Note:** - Get parameter ID is used to get the current open document value. If you want to get the current purchase document which is opened.

Data V1 type EKKO-EBELN.

Get parameter ID 'BES' field V1.

Write V1.

**Note:** - After receiving the FS (Functional Specification), if we don't know any field name and their table name then ask the function people where they get data or feed the data. Then the functional people open the screen and show the values. Then we place cursor on the field. Click on F1. Click on technical information. Identify the table name and field name. Some times instead of table structure is there then we identify the data element. Based on the data element we get the table field.

#### **Steps to get the table name based on the data element:**

Execute SE11. Select the radio button data type. Provide the data element name. Click on where used list. Select the check box table fields. Enter. Then we get the all the table names, which are used this data element. Click on find. Provide the related short description. Enter. Identify our table.

#### **Steps to identify the data base table based on the transaction code:**

Execute SE93. Provide the transaction code. Click on display. Click on display object list. Expand the package. Expand the dictionary objects. Expand the data base tables. Identify the tables.

If you want to identify the vendor tables then go to SE11. select the radio button data base table. Provide the table L\* & click o F4 button. Then we get the all vendor related tables. The same way as follows.

Vendor tables → L\*

Customer tables → K\*

Material tables → M\*

Company tables → T\*

Purchase tables → E\*

Sales tables → V\*

Finance tables → B\*

#### **Steps to identify the call the link tables based on any table:**

Execute SE11. select the radio button data base table. Provide the table name (LFA1). Click on where used list in the application tool bar (Ctrl + Shift + F3). Select only data base table check box. Enter. Then it provides the all the related tables.

#### **Steps to identify the tables based on idocs:**

Execute SE30. Provide related idoc name (CREMAS05). Click on display. In the each segment 3-6 characters are tables. Identify the all the tables.

**EX**

E I L F A 1 B  
  3 4 5 6  
  Table

## ABAP 7.4 Version Features

1. Declare a variable and pass one value in that one.

```
DATA(lv_text) = 'Sathish Reddy'.
```

2. Using Work area in select query with out declaration

```
SELECT SINGLE * FROM mara
INTO @DATA(wa_mara) WHERE matnr = '0001'.
WRITE:/ wa_mara-matnr, wa_mara-mtart.
```

3. Using Internal table in select query without declaration

```
SELECT matnr, mtart, meins
FROM mara
INTO TABLE @DATA(it_mara)
WHERE mtart = 'FERT'.

LOOP AT it_mara INTO DATA(wa_mara).
  WRITE:/ wa_mara-matnr, wa_mara-mtart, wa_mara-meins.
ENDLOOP.
```

4. Joins

```
DATA(lv_mtart) = 'FERT'.
DATA(lv_spras) = 'E'.

SELECT mara~matnr, mara~mtart, makt~maktx, makt~spras
FROM mara INNER JOIN makt ON mara~matnr = makt~matnr
WHERE mara~mtart = @lv_mtart
AND makt~spras = @lv_spras
INTO TABLE @DATA(it_materials).

LOOP AT it_materials INTO DATA(wa_material).
  WRITE : / wa_material-matnr, wa_material-mtart, wa_material-
maktx, wa_material-spras .
ENDLOOP.
```

5. Using for all entries

```
DATA(lv_mtart) = 'FERT'.

SELECT matnr
FROM mara
INTO TABLE @DATA(it_matnr)
WHERE mtart = @lv_mtart.
IF it_matnr IS NOT INITIAL.

  SELECT *
  FROM makt
  INTO TABLE @DATA(it_makt)
  FOR ALL ENTRIES IN @it_matnr
  WHERE matnr = @it_matnr-matnr.
ENDIF.

LOOP AT it_makt INTO DATA(wa_makt).
  WRITE : / wa_makt-matnr, wa_makt-maktx, wa_makt-spras.
ENDLOOP.
```

6. Using CASE in select query

```
DATA: lv_fert  TYPE text30 VALUE 'Finished Products',
      lv_halb  TYPE text30 VALUE 'Semifinished Products',
```

```

lv_other TYPE text30 VALUE 'Other Material Type'.

SELECT SINGLE matnr, mtart,
CASE
    WHEN mtart = 'FERT' THEN @lv_fert
    WHEN mtart = 'HALB' THEN @lv_halb
    ELSE @lv_other
END AS material_type
FROM mara
INTO @DATA(wa_mara)
WHERE matnr = '000001-002'.

WRITE:/ 'Material Type :', wa_mara-material_type.

```

#### 7. Change the internal table field names in select query

```

SELECT matnr AS material, mtart AS material_type
FROM mara
INTO TABLE @DATA(it_mara)
WHERE mtart = 'FERT'.

LOOP AT it_mara INTO DATA(wa_mara).
    WRITE:/ wa_mara-material, wa_mara-material_type.
ENDLOOP.

```

#### 8. Initialize values in work area

```

TYPES: BEGIN OF ty_mara,
        matnr TYPE mara-matnr,
        mtart TYPE mara-mtart,
    END OF ty_mara.

DATA: wa_mara TYPE ty_mara.
wa_mara = VALUE #( matnr = 0001 mtart = 'FERT' ) .

```

#### 9. Initialize Internal table values

```

TYPES: BEGIN OF ty_mara,
        matnr TYPE mara-matnr,
        mtart TYPE mara-mtart,
    END OF ty_mara.

DATA: it_mara TYPE TABLE OF ty_mara.

it_mara = VALUE #( ( matnr = 0001 mtart = 'FERT' )
                  ( matnr = 0001 mtart = 'HALB' )
                  ( matnr = 0003 mtart = 'HAWA' )
                  ( matnr = 0004 mtart = 'FERT' )
                  ( matnr = 0005 mtart = 'FERT' )
                  ( matnr = 0006 mtart = 'HALB' )
                  ( matnr = 0007 mtart = 'HAWA' ) ) .

```

#### 10. Read table based on index

```
DATA(wa_mara) = it_mara[ 2 ]. "Read index 2 from IT_MARA
```

Here SY-SUBRC won't work. So we have to take as below

```

TRY .
    DATA(wa_mara) = it_mara[ 51 ]. "Read 51 index will fail because only 5
0 records are there
    CATCH cx_sy_itab_line_not_found.
        WRITE:/ 'Error Reading Record'.
ENDTRY.
WRITE:/ wa_mara-matnr.

```

### 11. Read table based on condition

```
TRY .  
  DATA(wa_mara) = it_mara[ matnr = '0001' mtart = 'FERT' ].  
  CATCH cx_sy_itab_line_not_found.  
    WRITE:/ 'Error Reading Record'.  
  ENDTRY.
```

### 12. Check records in internal table

```
IF line_exists( it_mara[ matnr = '0001' mtart = 'FERT' ] ).  
  WRITE:/ 'Records Exists'.  
ENDIF.
```

## Interview questions

1. What is Check table, value table
2. Buffering types
3. Types of Control break statements
4. Difference b/w Data base view and Maintaince view
5. Difference b/w export and changing in FM.
6. In Interactive reports, we will display the data in primary list. If you double click on any record of any list, we will display the data which belongs to that record in secondary list. Is it possible to display the data primary list and secondary list in single page in ALV? If it's possible how is it possible?
7. Is it possible to give the list of possible values to the input in MPP? If it's possible, how is it and in how many ways is it possible in MPP?
8. How can we debug the SAP SCRIPT?
9. What is the difference b/w table and template in SF? How can we create Template in SF?
10. We are processing either call transaction or session method. Suddenly system is struck. Then what will happen?
11. How can we display the errors in call transaction method?
12. When will you choose call transaction & session method?
13. T.code for Application server? How can you upload the data into application server and download the data from application server?
14. T.code for creation of partner profile? Types of messages in IDOC?
15. What is ALE? How can you know idoc is reached to receiver or not?
16. Tell me few FM and methods which are used in OOPS ALV?
17. Difference b/w Abstract class and Interface?
18. Difference b/w Instance and static events?
19. What is constructor? Difference b/w instance and static constructor?
20. What is API? What is RFC? Why we use RFC?
21. In how many ways we upload the data from flat file to sap system?
22. Difference b/w BAPI & RFC?
23. Types of Badies?
24. What is enhancement spot, Enhancement frame work?
25. What is single implementation badi, multiple implementation badi?
26. Performance issues & tools?
27. What is field symbol?
28. How can we insert the logo in SAP SCRIPT?
29. Is it possible to write the code in SAP SCRIPT?
30. How can we insert the logo in ALV Report?
31. What is at user-command?
32. For MIGO transaction you can attach some fields. If you enter some values in that separate fields, where they can save?
33. Difference b/w type and like?
34. Difference b/w refresh and free?
35. What is search help & lock object?
36. What is search help exit?
37. What is the use of BAPI transaction?
38. Types of windows in SF?
39. How can we handle the double click event in ALV?
40. What is the use of chain & end chain in MPP?
41. What is the purpose of POV & POH in MPP?

42. Differences b/w badi & exits?
43. Scenarios of badi?
44. How can we know the badi implementations are active or inactive?
45. How can we insert the logo in SF & script? Process flow?
46. Differences b/w driver program & report program?
47. Differences b/w Main & Variable window?
48. In which internal table the messages will store in BDC?
49. Differences among APPEND, INSERT, COLLECT?
50. How we modify the standard db table structure?
51. How can we format the data before WRITE statements in reports?
52. What is EDI?
53. What is Test, Persistent, Final, Singleton class?
54. What is casting? Different types of casting?
55. What is event? Process for events?
56. How can we insert a table in Module Pool Programming?
57. What is sub screen?
58. Differences between normal screen & sub screen?
59. Pre requisitions for ‘FOR ALL ENTRIES’?
60. How can debug the ALE/IDOC?
61. How can debug the Smart form?
62. Events TMG? T.Code for TMG?
63. Can we insert any new records in TMG?
64. How can we identify the Customer Exit?
65. Process flow of ALV?
66. How many main windows we can create in Smart form?
67. What is the Polymorphism?
68. List, Grid display these two use in OOABAP?
69. Why we go for parallel cursor technique?
70. Events in Smart form?
71. How can we run the program in background?
72. How can we identify the package of transaction?
73. How can we display the barcodes instead of material number in Smart form?
74. How can we identify the standard badi?
75. How can we identify the standard bapi?
76. What are BAPI methods?
77. What is Delivery class, data class?
78. How can we identify the errors?
79. What are the contents in Technical settings?
80. Why SAP Script is client dependent, SF is independent?
81. How can we hide one field at screen in report and MPP?
82. What is the difference between structure and table in data dictionary in ABAP?
83. What is the difference between collect and sum?
84. How we format the data before write statement in report?
85. What is the difference between Table and Template?
86. When do we use End-of-selection?
87. In events start-of-selection is default event. When we have to use this event explicitly?  
Why?
88. What is the differences between ABAP and OOABAP? In which situation we use OOABAP?
89. What is table buffer? Which type of tables used this buffer?

- 90. What is the use of pretty printer?**
- 91. What is the difference between SAP memory and ABAP memory?**
- 92. What is the difference between Type and Like?**
- 93. What is Tcode SE16. For what is it used. Explain briefly?**
- 94. What is difference between dialog program and a report?**
- 95. What is Field symbol?**
- 96. Howmany Main windows we have in Smartforms**
- 97. Types of windows in smartforms**
- 98. How can you find the name of the function module in smartforms**
- 99. Difference between Global definitions and Form interface in smartforms?**
- 100. How can you convert smartform output to pdf**
- 101. How can you choose printer name as default in smartforms**
- 102. How can you display the total number of pages in smartforms**
- 103. What is the page protection in smartforms**
- 104. How can you debug the smartform**
- 105. How can you use barcode in smartform?**

## Important transactions

SALE → Define the logical system  
SCC4 → Assign the client to LS  
SM59 → Maintain RFC destination details  
BD64 → Distribution model  
WE21 → Port number  
WE20 → Create partner profile  
WE02 → Check Idoc  
WE05 → Check Idoc  
BD87 → Reprocess the IDOC.  
BD61 → Activate the change pointer technique  
BD50 → Activate message type  
BD21 → Generate as well as dispatch the idoc.  
BD56 → Segment filtering  
BD53 → Reduced Idoc  
WE31 → Create segment  
WE30 → Create Idoc.  
WE81 → Create message type  
WE82 → Link the message type to idoc  
WE57 → Link the message type to function module & idoc  
BD51 → Create the mode of posting  
WE42 → Create Process code & link the process code to function module  
WE19 → Test the custom inbound program  
XK01 / MK01 / FK01 → Create Vendor  
XD01 / VD01 / FD01 → Create Customer  
MM01 → Create Material  
ME51N → Create Purchase order  
ME21N → Create Purchase order  
MB01 → Create Material document  
VA01 → Create Sales Order  
VL01 → Create Delivery  
VF01 → Create Billing  
FI01 → Create Bank  
FB01 → Create account  
KS01 → Create Cost Center  
KE51N → Create Profit Center  
CS01 → Create BOM  
MSC1N → Create Batch  
COR1 → Create Process Order  
C201 → Create Recipe.  
SE38 → ABAP Editor  
SE11 → Data Base Tables, Create BAPI structure  
SE37 → Function Module  
SE41 → Menu Painter  
SE93 → Transaction Code  
SE91 → Message  
SE80 → MPP, Package  
SE51 → Screen Painter  
SE71 → SAP – SCRIPT

SM35 → Update Session Method  
SMARTFORMS → Smart forms  
SMARTSTYLES → Smart styles  
SDMO → All Message types, IDOCs  
SQVI → Link among the tables  
SM12 → Close the user  
AL11 → Application Server  
NACE → All applications & their layouts & their driver programs  
SE24 → Global class  
SE47 → Status codes & their short descriptions  
WE11 → Delete the IDOC  
SE18 → BADI Definition  
SE19 → BADI Implementation  
CMOD → Implement customer exit  
ST22 → Identify all the dumps  
SE30 → Runtime analysis  
SLIN → Extended program check  
ST05 → SQL Trace  
I18N → Text element to MS Word  
SE16 → Data browser  
SE12 → Display ABAP dictionary  
SE09/SE10 → Release the TR  
SM30 → TMG  
OAER → Logo in reports  
SPAD → Create Page format  
SE78 → .BMP image convert into graphical image  
SO10 → Create include which is used in Scripts  
SMOD → Know user exit name  
SHDB → Do the recording  
BAPI → BAPI Explorer  
SWO1 → Business Object builder  
/N → Terminate the session  
/NEX → Terminate all sessions  
/O → Open a new session  
/H → Debugging mode  
SP01/02 → Spool Request Number  
SCC1 → Client copy transport  
SM37 → Over view of background jobs

## Function modules

1. CONVERSION\_EXIT\_ALPHA\_INPUT
2. REUSE\_ALV\_LIST\_DISPLAY
3. REUSE\_ALV\_GRID\_DISPLAY
4. REUSE\_ALV\_FIELDCATALOG\_MERGE
5. REUSE\_ALV\_COMMENTARY\_WRITE
6. REUSE\_ALV\_BLOCK\_LIST\_INIT
7. REUSE\_ALV\_BLOCK\_LIST\_APPEND
8. REUSE\_ALV\_BLOCK\_LIST\_DISPLAY
9. REUSE\_ALV\_HIERSEQ\_LIST\_DISPLAY
10. OPEN\_FORM
11. WRITE\_FORM
12. CLOSE\_FORM
13. START\_FORM
14. END\_FORM
15. BDC\_OPEN\_GROUP
16. BDC\_INSERT
17. BDC\_CLOSE\_GROUP
18. MASTER\_IDOC\_DISTRIBUTE
19. DEQUEUE\_ALL
20. F4IF\_INT\_TABLE\_VALUE\_REQUEST
21. VRM\_SET\_VALUES
22. SSF\_FUNCTION\_MODULE\_NAME
23. GUI\_UPLOAD
24. GUI\_DOWNLOAD
25. DAY\_ATTRIBUTES\_GET
26. MONTHS\_BETWEEN\_TWO\_DATES
27. END\_OF\_MONTH\_DETERMINE\_2
28. HR\_JP\_MONTH\_BEGIN\_END\_DATE
29. FIRST\_AND\_LAST\_DAY\_IN\_YEAR\_GET
30. CURRENCY\_AMOUNT\_DISPLAY\_TO\_SAP
31. HR\_IN\_CHG\_INR\_WRDS
32. SAVE\_TEXT
33. SPELL\_AMOUNT
34. CS\_BOM\_EXPLOSION
35. F4\_FILENAME
36. READ\_TEXT

## **System Variables**

SY-UNAME → Current user name  
SY-CPROG → Current program  
SY-LINNO → Last line number  
SY-LSIND → Current list index number  
SY-LANGU → Current language  
SY-SUBRC → To know the condition is true or false  
SY-LISEL → Select Exact line number  
SY-LILLI → Select exact line number  
SY-UCOMM → Function code of selected menu  
SY-MSGID → Message ID  
SY-DATUM → Current date  
SY-INDEX → Current loop passing number  
SY-TABIX → Exact line number of the record is moving form IT to WA  
SY-SUBRC → Checking success or failure  
SY-DBCNT → To know number of records are successfully inserted  
SY-TCODE → To pass the current transaction code  
SY-DYNNR → Current screen Number  
SY-VLINE → Provide the vertical line  
SY-ULINE → Provide horizontal line

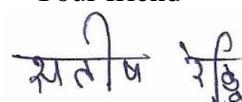
SNO	CONCEPT	PAGE NO
1	<b>Role of an ABAPer in Real time</b>	1
2	<b>Types of projects</b>	2
3	<b>Data types</b>	4
4	<b>Data dictionary</b>	7
5	<b>Foreign key</b>	11
6	<b>Reference fields</b>	12
7	<b>Types of tables</b>	13
8	<b>Index</b>	14
9	<b>Types of internal table</b>	21
10	<b>Types of declaring internal table</b>	22
11	<b>Internal table with header line</b>	24
12	<b>Initializing techniques</b>	25
13	<b>Operations on internal table</b>	26
14	<b>Adjacent duplicates</b>	29
15	<b>Search help</b>	33
16	<b>Lock objects</b>	34
17	<b>Buffering</b>	35
18	<b>TMG</b>	37
19	<b>Views</b>	39
20	<b>Control break statements</b>	41
21	<b>Joins</b>	50
22	<b>For all entries</b>	53
23	<b>Modularization techniques</b>	57
24	<b>Components of select-options</b>	65
25	<b>Reports</b>	70
26	<b>Classical report</b>	72
27	<b>Message</b>	76
28	<b>Interactive report</b>	82
29	<b>Working with menu painter</b>	94
30	<b>ALV Reports</b>	98
31	<b>Hierarchical ALV</b>	125
32	<b>MPP / Transaction / Dialog Pool Program</b>	137
33	<b>Ranges</b>	150
34	<b>SAP – SCRIPT</b>	154
35	<b>Control Commands</b>	159
36	<b>Format Options</b>	164
37	<b>Smart forms</b>	176
38	<b>Smart styles</b>	179
39	<b>Debugging</b>	198
40	<b>BDC</b>	202
41	<b>Handle the errors</b>	216
42	<b>Cross applications</b>	238
43	<b>Types of distributing data</b>	245
44	<b>Filtering techniques</b>	249
45	<b>Custom IDOC</b>	252

46	<b>Extend IDOC</b>	260
47	<b>Enhancement</b>	262
48	<b>OOABAP</b>	272
49	<b>Constructor</b>	278
50	<b>Inheritance</b>	284
51	<b>Method Over loading &amp; riding</b>	285
52	<b>Abstract</b>	288
53	<b>Interface</b>	289
54	<b>Polymorphism, Encapsulation</b>	290
55	<b>Exceptions</b>	291
56	<b>Casting</b>	291
57	<b>OOPS ALV</b>	292
58	<b>OOPS BDC</b>	329
59	<b>BAPI</b>	335
60	<b>RFC</b>	345
61	<b>BADI</b>	346
62	<b>Scenarios on BADI</b>	347
63	<b>Enhancement Frame work</b>	354
64	<b>LSMW</b>	355
65	<b>Performance Issues</b>	359
66	<b>MM life cycle</b>	370
67	<b>SD life cycle</b>	373
68	<b>PP life cycle</b>	377
69	<b>Sales tax &amp; Insurance Real time report</b>	382
70	<b>Account receivable aging real time report</b>	397
71	<b>Interview questions</b>	427
72	<b>ABAP 7.4 Syntaxes</b>	424
72	<b>Important standard transactions</b>	428
73	<b>Important function modules</b>	432
74	<b>Important system variables</b>	433

Hi friends..

This is Sathish Reddy. If you find any spelling mistakes or anything wrong in this material, please send me mail to [satishkumarreddy.mn@gmail.com](mailto:satishkumarreddy.mn@gmail.com) & forward this material to your friends who want to learn ABAP.

Your friend









Hi friends....

Please read the bellow message whenever you are free.

Today we have food, clothes, shelter. Because of we are employees. But lot of people don't have these much of facilities. I mean orphan children, handicapped children, destitute senior citizens are suffering with minimum facilities. Some children are begging in the bus stops, platforms... some people will leave their old age parents at bus stops also. Orphan homes or Old age home people take care of them. They are running their organizations without providing the food three times per day. They can provide when they have fund. In remaining days they may provide 1 time or 2 times per day.

Some people will think like, 'I want to help to the orphan children or handicapped children. But is it possible only with me... How can I provide that much of money for them??' They may have these type of doubts. So we started one foundation named as SHREE JANANI FOUNDATION. All employees will donate their 1% salary to this foundation every month (less than or equal to 1% of employee salary). For example, one employee will get 10K salary, 100 Rs is not a big amount to him to provide for orphan children. That's why this foundation has a rule to collect less than or equal to 1% from employee salary. Collected fund will spend for orphan children / handicapped children / old age people. These foundation will provide something whatever they don't have. But this foundation never give 1 rupee also for them, just provide their necessity.

Foundation will provide all the bank transactions to every donor twice per month. Every month on 10th, 30<sup>th</sup>(last day of month) foundation management members provide these details to your mails, whats app numbers, SMS. Event photos & videos send to your mail and whatsapp. So you can know 'how many people are donating the fund, how much fund is coming. How much fund is spending for orphan children? How the event conducted like these'. Whenever event will conduct then we will intimate you in whats app & sms & mail about the event details. If you are free, want to participate in that event you can also join with them.

You can also send messages for helping in your locations through what app or sms to 8332999399 or send mail to [helpdesk@shreejanani.org](mailto:helpdesk@shreejanani.org). You can help orphan children from **SHREE JANANI FOUNDATION**. You can search for event details or donation details or any other queries from [www.shreejanani.org](http://www.shreejanani.org) website.

If anybody want any blood group urgently, you can call to foundation contact numbers or send a mail to the foundation mail id. We will try to send the blood donor to your place. Students, Unemployees or any other person can also join with us to do help.

Thank you for reading this page with patience..

General Secretary  
M N Sathish Kumar Reddy  
Mobile: 9866079202  
Skype: satish.reddy.mn  
Mail: satishkumarreddy.mn@gmail.com