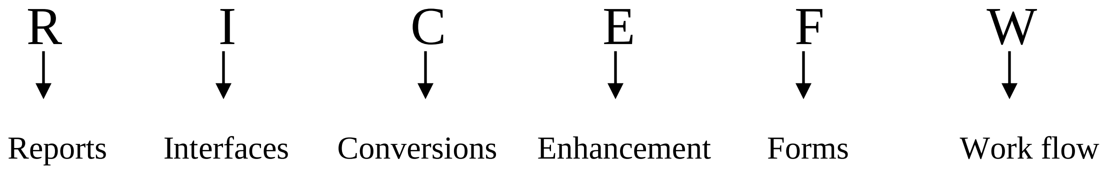
SAP: - Systems, Applications, Products in Data Processing

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.

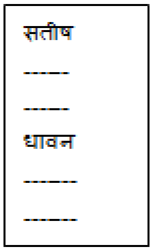
Report: - Based on the given input we'll fetch [read or get] data from data base and display in a predefined format.

Types of Reports:-

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

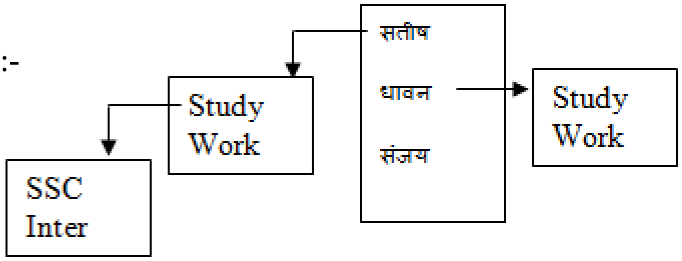
Classical Reports: - It's nothing but to display the entire information in a single list.

EX:-



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

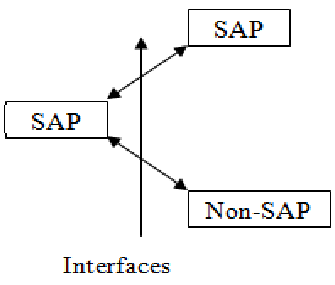
EX:-



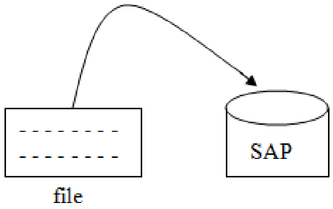
ALV Reports: - This is used to display the data with additional futures i.e. Borders, colors, shades, lines etc.

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

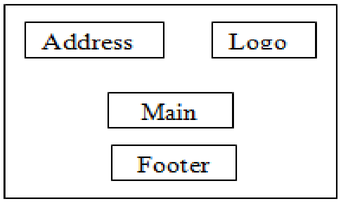


Conversion: - Conversion programs are used to transverse the data from file to SAP system.
Conversion programs are BDC,LSMW

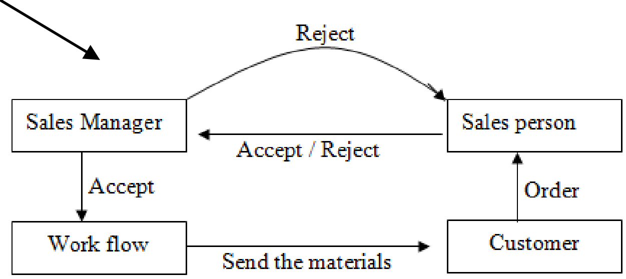


Enhancements: - Enhancements are used to adding some additional functionality to the standard functionality with out disturbing the standard functionality.
Enhancements are BADI, user exists, customer exits, enhancement frame work, enhancement spot

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.

<u>Phases of implementation project</u>:-

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

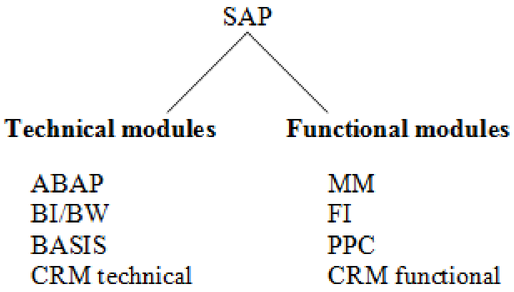
In the implementation project the ABAPer must start from Realization phase.

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.



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 CATAGERY.

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

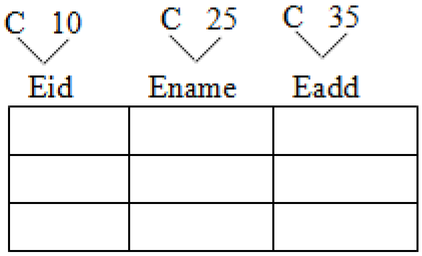
Technical data is dependent data.

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.



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.

Field	Key	Data type	Length	Short Description
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).

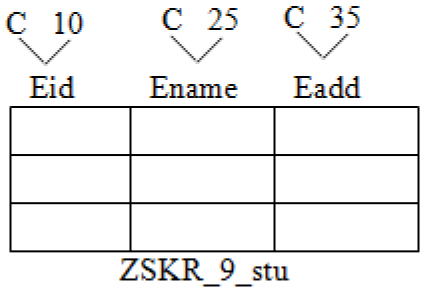
Steps to provide the data to the table directly: -

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: -

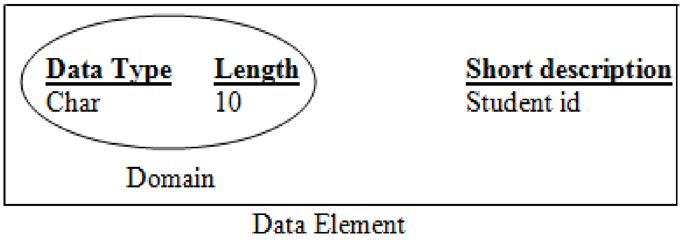
In the menu bar click on utilities → Table contents display. Click on execute.

→ Create the following table by using direct method



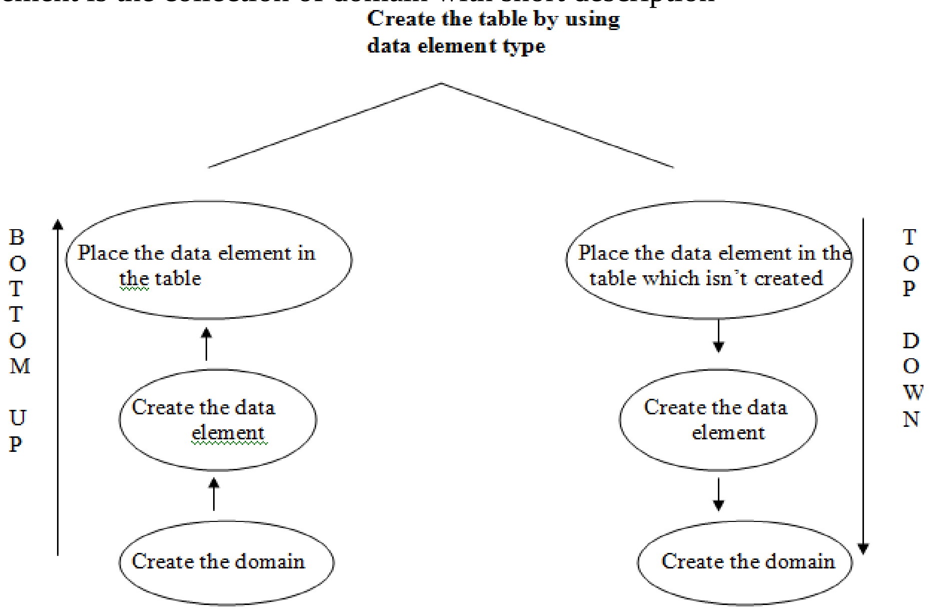
Note: – In the real time, we never create the data base table by using direct method or predefined type. Because, if you want to establish the relation between any two tables, then you must maintain the same domain name in both the tables. In direct method there is no domain concept. So we go for data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

Field	Key	Init	Data Element
Sid	✓	✓	

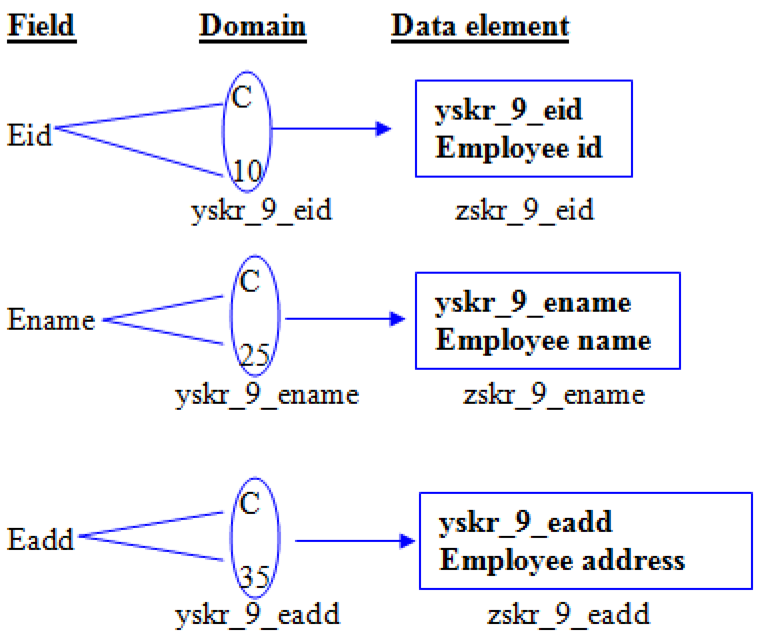


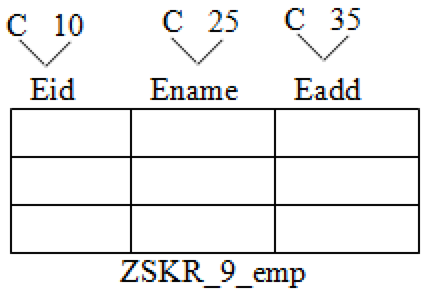
Domain: - Domain is the collection of data types & lengths.

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


Create the table by using data element type
BOTTOM UP
Place the data element in the table
Create the data element
Create the domain
TOP DOWN
Place the data element in the table which isn't created
Create the data element
Create the domain

➔ 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.

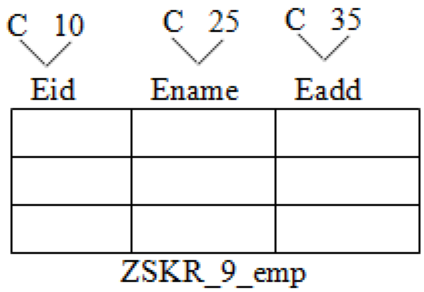
Steps to create the data element: - Execute the **SE11**. Select the radio button data type. Provide the data element name. Click on create. Click on enter. Provide short description. Provide domain which is already created. Save the data element. Check the data element. Activate the data element. Repeat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

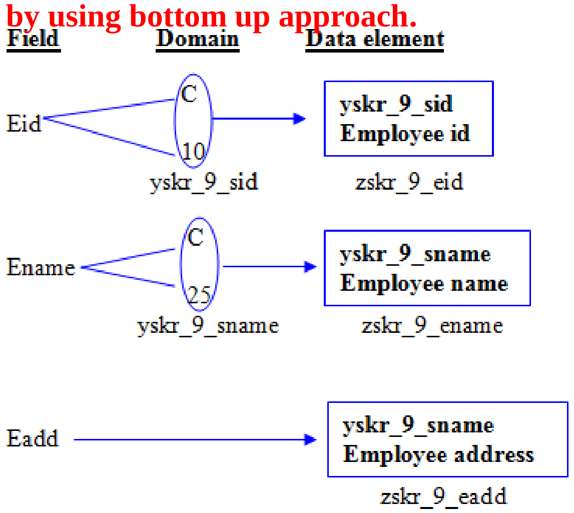
Steps to create the table by using data element type (bottom up approach): - Execute **SE11.** Select the radio button database table. Provide the table name. Click on create. Provide short description. Provide delivery class is '**A**'. Select **Display / Maintenance allowed**. Click on filed tab. Provide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

Field	**Key**	**Data Element**
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 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.

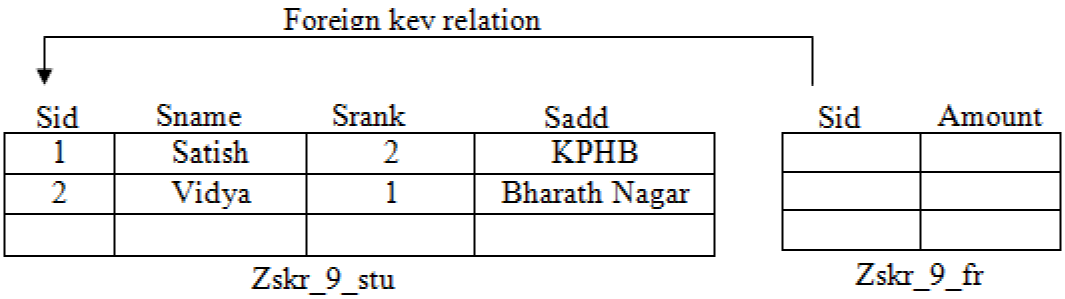
Foreign key: -

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.



Foreign kev relation

Sid	Sname	Srank	Sadd
1	Satish	2	KPHB
2	Vidya	1	Bharath Nagar

Zskr_9_stu

Sid	Amount

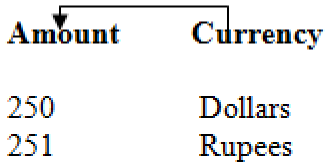
Zskr_9_fr

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.

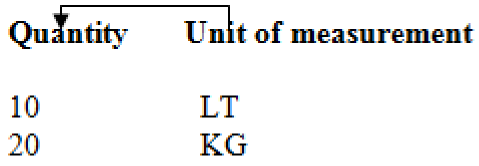
Working with reference fields: -

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.



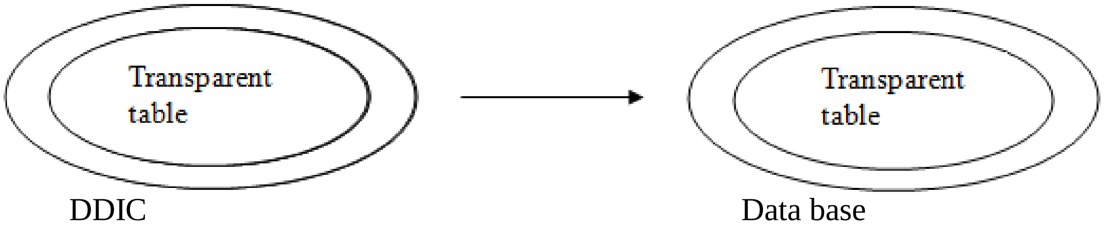
Quantity	**Unit of measurement**
10	LT
20	KG

Field	**Data type**
Amount	CURR
Currency	CUKY
Quantity	QUAN
Units	UNIT

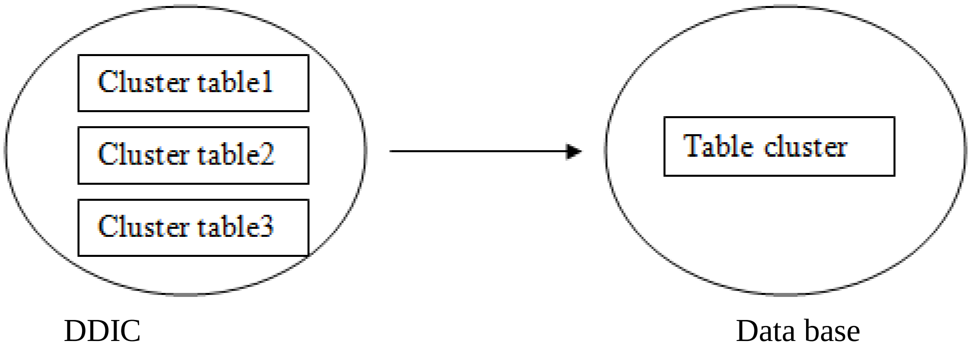
Types of tables: -

1. Transaparent tables
2. Cluster tables
3. Pool tables

Transaparent tables: - Transaparent tables are one to one relationship. That is if you create one transaparent 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.

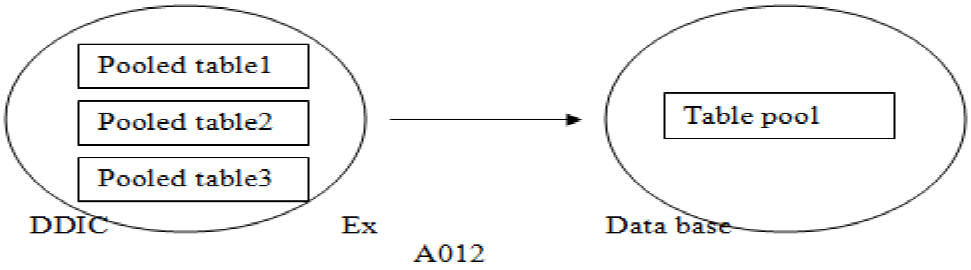


Buffering isn't possible for clustered table. From this reason only fetching the data from clustered table take more time.

Clustered table is suitable when you fetch the fewer amounts 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 Clustered & 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.

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.

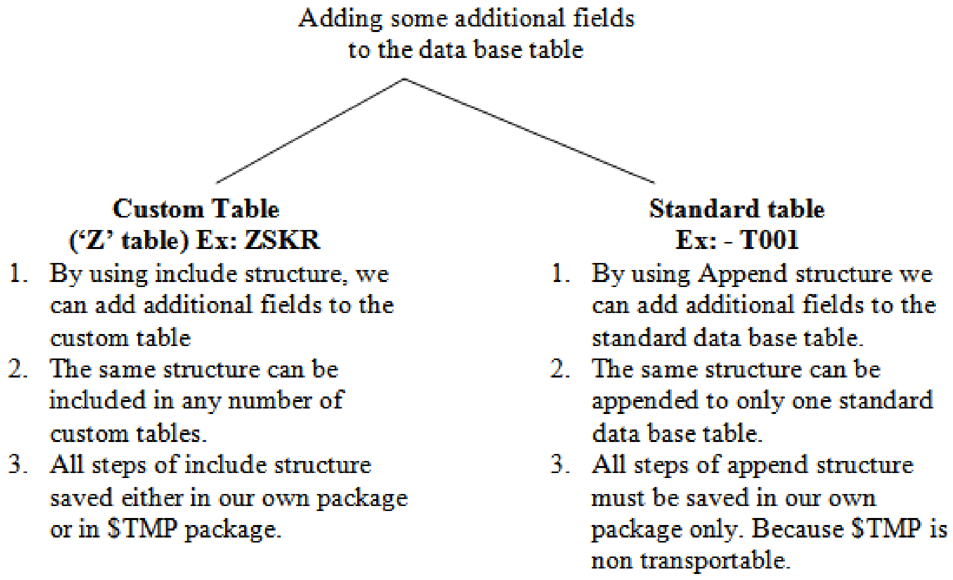
Steps to create secondary index: -

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.

Some of filed 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



Adding some additional fields to the data base table

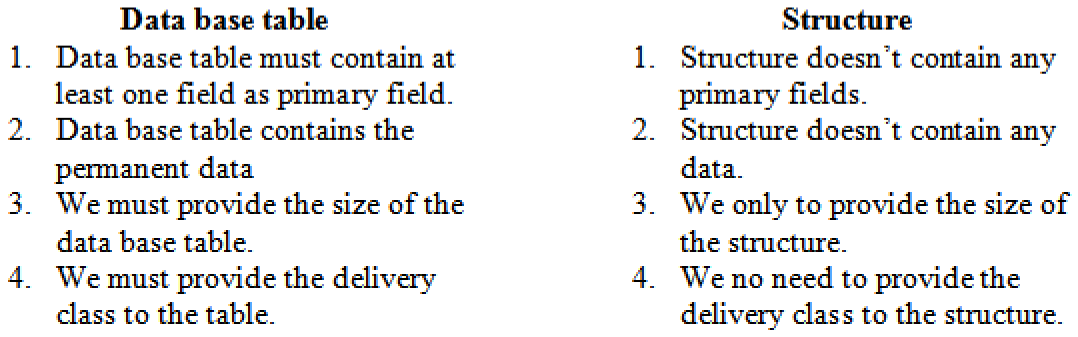
Custom Table
('Z' table) Ex: ZSKR

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.

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.

Structure

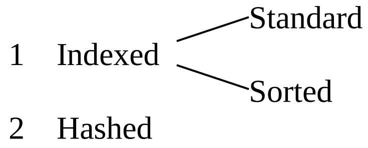
1. Structure doesn't contain any primary fields.
2. Structure doesn't contain any data.
3. We only to provide the size of the structure.
4. We no need to provide the delivery class to the structure.

Ex: -
Select BUKRS BUTXT LAND1 from T001 into table IT_T001 where BUKRS = P_BUKRS.

Differences between parameter & select-options: -

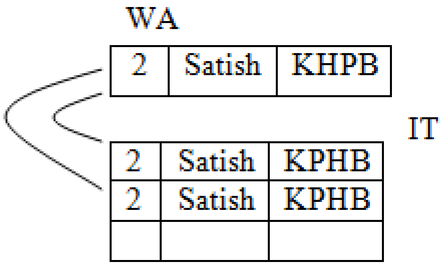
Parameter	Select-options
1. Parameter is the keyword which accepts the single value at run time.	1. Select-options is the keyword which accept the single value, multiple single values, single range & multiple ranges.
2. Without provide input parameter we can't get data.	2. Without providing input select-options we can get the entire data from database.

Types of Internal Table: -



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.



Append WA to IT
Append WA to IT

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 filed 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: -

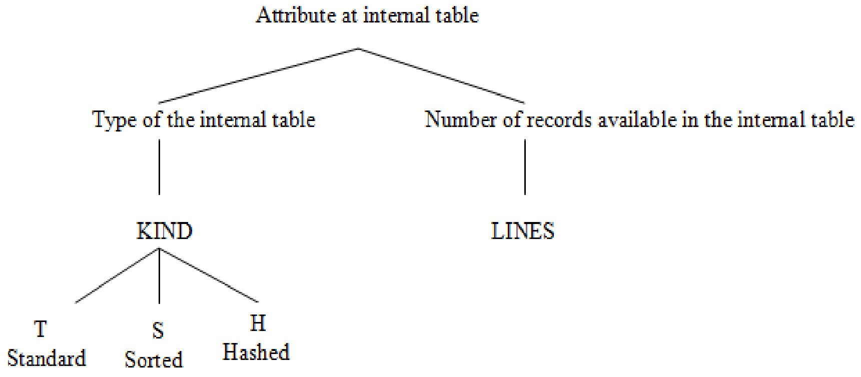
Data <Internal Table> like sorted table of <work area> with unique / non-unique key <filed> ---- .
Ex: - data IT like sorted table of WA with unique key Eid.

Hashed internal table: -

1. It won't accept the duplicate records
2. Here we must specify at least one filed 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.


Attribute at internal table
Type of the internal table
Number of records available in the internal table
KIND
LINES
T
Standard
S
Sorted
H
Hashed

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> $\underbrace{\text{KIND <variable1>}}_{\text{Optional}}$ $\underbrace{\text{lines <variable2>}}_{\text{Optional}}$

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:

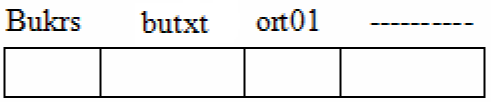
```
Types: begin of ty_t001,
       Bukrs type t001-bukrs,
       Ort01 type t001-ort01,
       End of ty_t001.
Data it_t001 type table of ty_t001 with header line.
```

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.



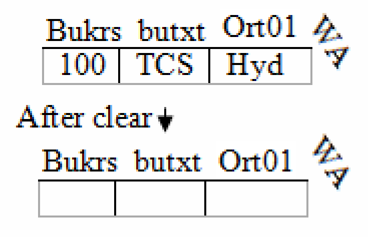
1 Append t001 to IT_t001.

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>

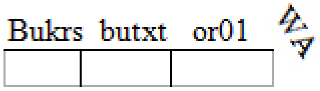
Clear <internal table>

Ex:-

Clear WA_t001.

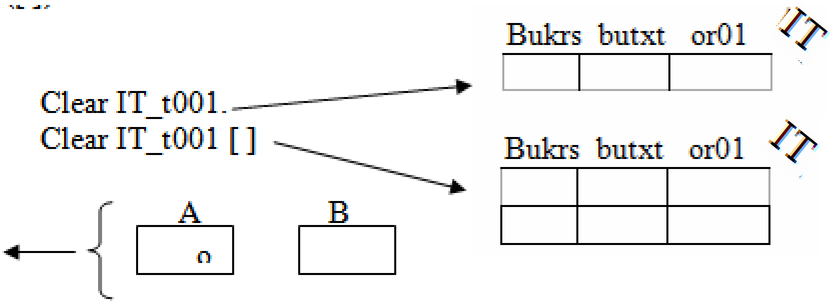
Clear IT_t001.

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.



Ex:-

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



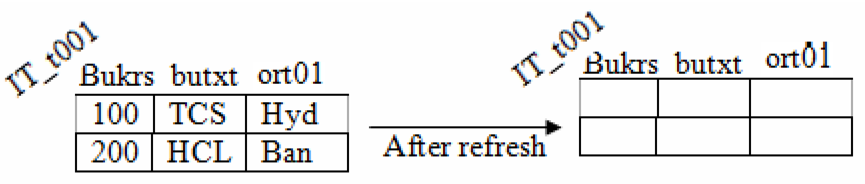
Refresh:-

Refresh always clears the contents at the internal table only.

Syntax: -

Refresh <internal table>.

Ex: - Refresh IT_t001.



IT_t001

Bukrs	butxt	ort01
100	TCS	Hyd
200	HCL	Ban

After refresh

IT_t001

Bukrs	butxt	ort01

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

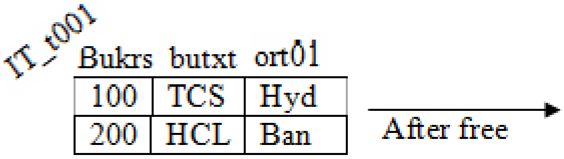
Free: -

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.



IT_t001

Bukrs	butxt	ort01
100	TCS	Hyd
200	HCL	Ban

After free

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.

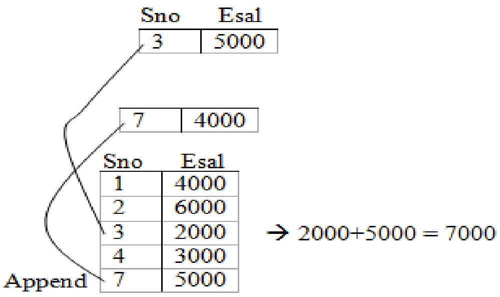
Note: - If we use insert keyword to standard internal table then it acts like append.

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

7	4000

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 1st 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.

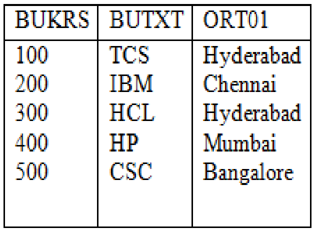
Ex: -
Read table IT into WA with key BUKRS = ‘2000’.

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 $\underbrace{\text{<index no>}}_{\text{optional}}$.

Endloop.



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

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
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

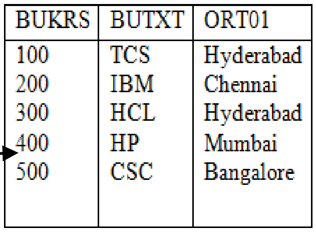
Syntax: -

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

Ex: -

```
Sort IT_t001 by BUKRS.
```

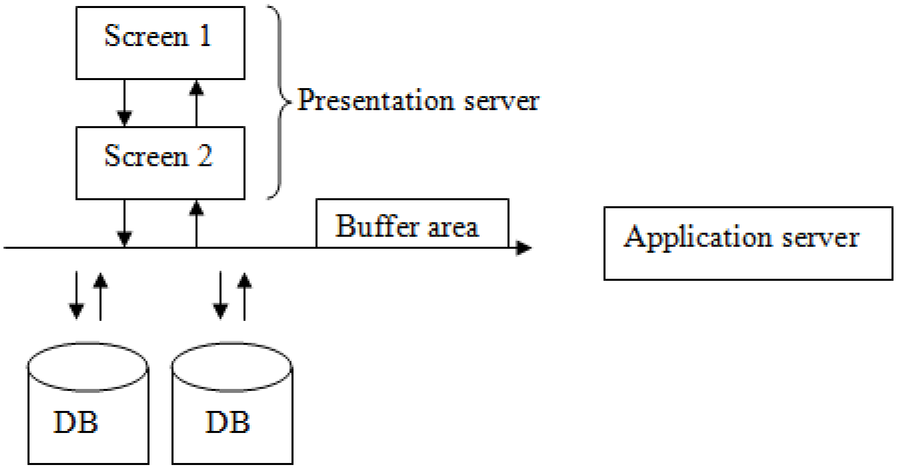
OP: -



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

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): -

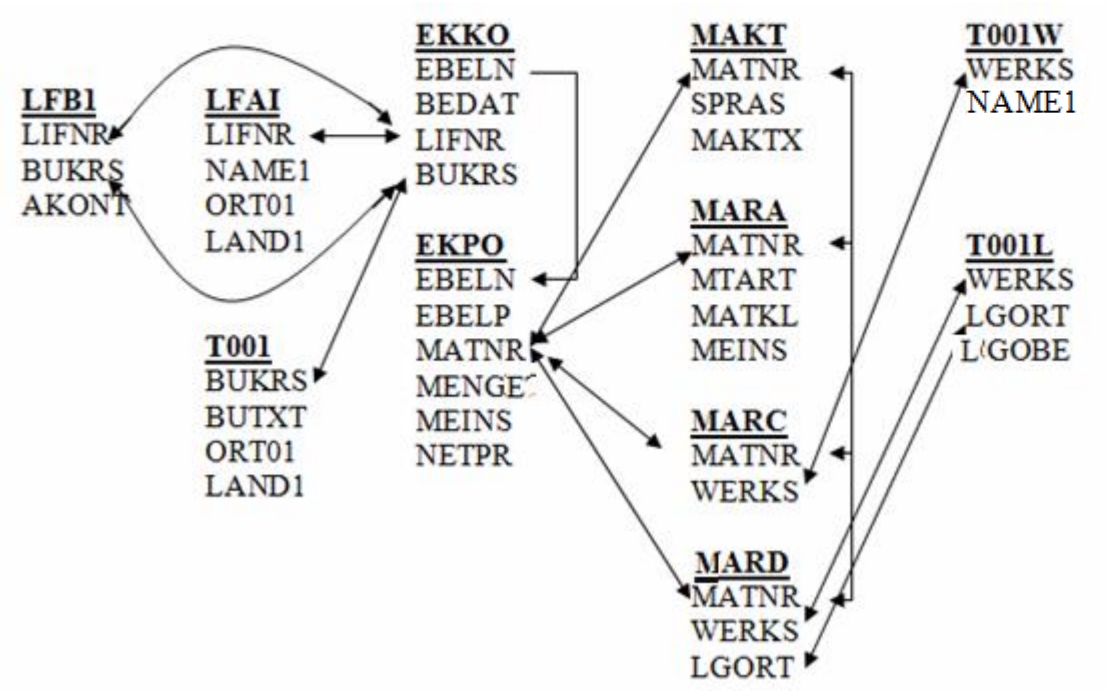
MATNR → Material umber
WERKS → Plant number

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 Maintaince Generator)

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

Steps to create Table Maintaince 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 Maintaince Generator. Select the authorization group which is provided by BASIS people. Provide the function group as table name. Select the Maintaince 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**
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 Maintaince. One step and two step.

One Step Maintaince: -

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

Two step Maintaince: -

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

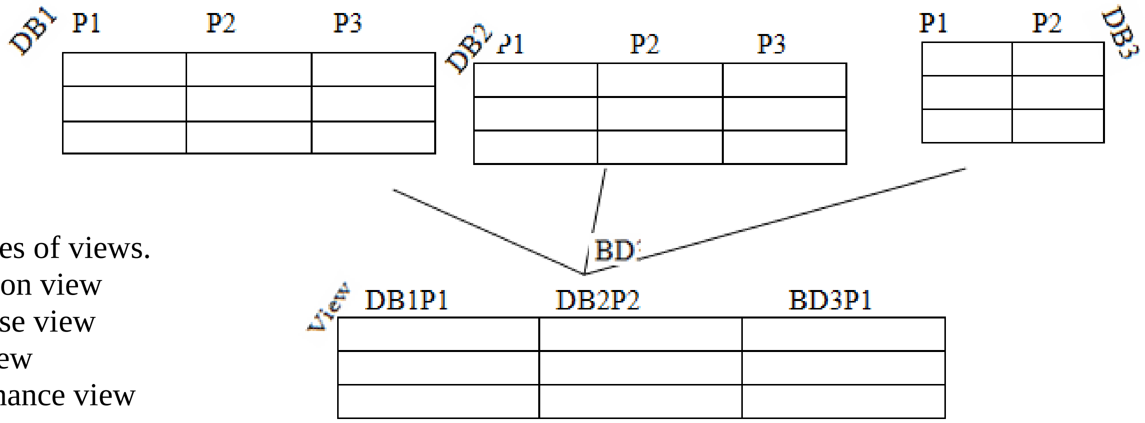
Steps to implement the events in TMG: -

Execute SE11. Open the table in change mode. In the menu bar click on utilities → Table Maintaince 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.

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 table. Database view is always involved with more than one table.

➔ Create the database view with BUKRS BUTXT LIFNR

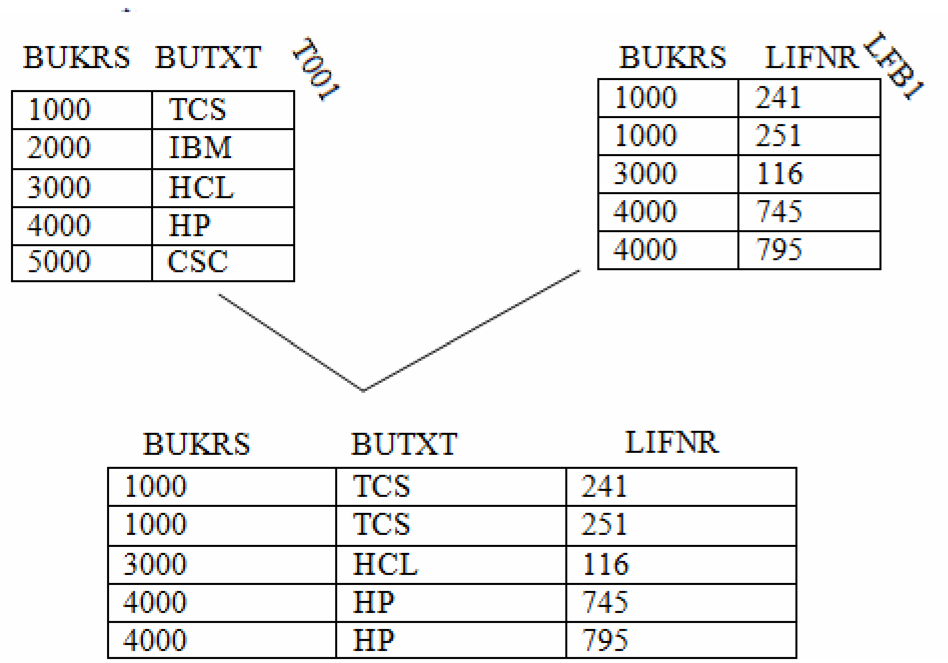
Steps to create database view: -

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.



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

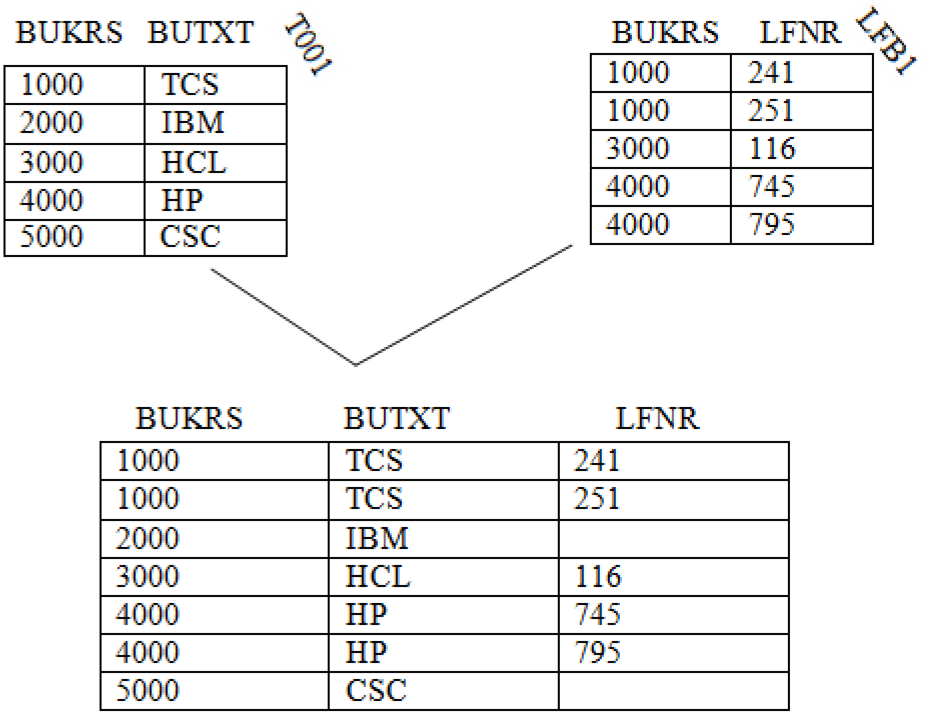
BUKRS	LIFNR
1000	241
1000	251
3000	116
4000	745
4000	795

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.



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

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

BUKRS	BUTXT	LFNR
1000	TCS	241
1000	TCS	251
2000	IBM	
3000	HCL	116
4000	HP	745
4000	HP	795
5000	CSC	

Note: – Data base view & help views are used in selection method of elementary search.

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: It can be create for one or more tables, if more, only related tables. This view is created will all key fields of table automatically and joins are auto generated by SAP. We can’t change it. This Object can’t be used as Dictionary Object for SE38 prorams, with help table maintenance generator we should convert Maintaince view as repository object. Unlike Database views, we can modify data from multiple tables also.

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.

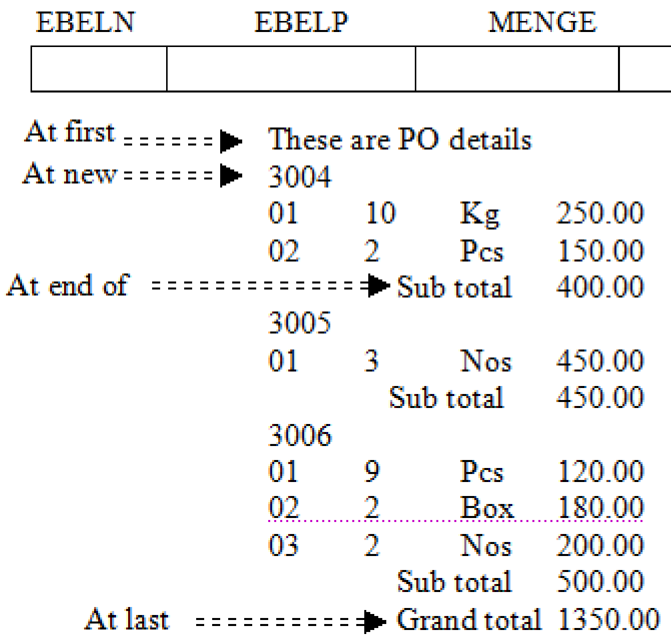
AT LAST: -

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
```

AT NEW	**ON CHANGE OF**
1. AT NEW work within the loop of internal table	1. ON CHANGE OF work with in the any loop.
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.	2. The right side field values of mention fields never change.
3. In the AT NEW we can't use the logical operations (and, or, not)	3. We can use logical operations.
4. This is used in ABAP objects.	4. This isn't 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.

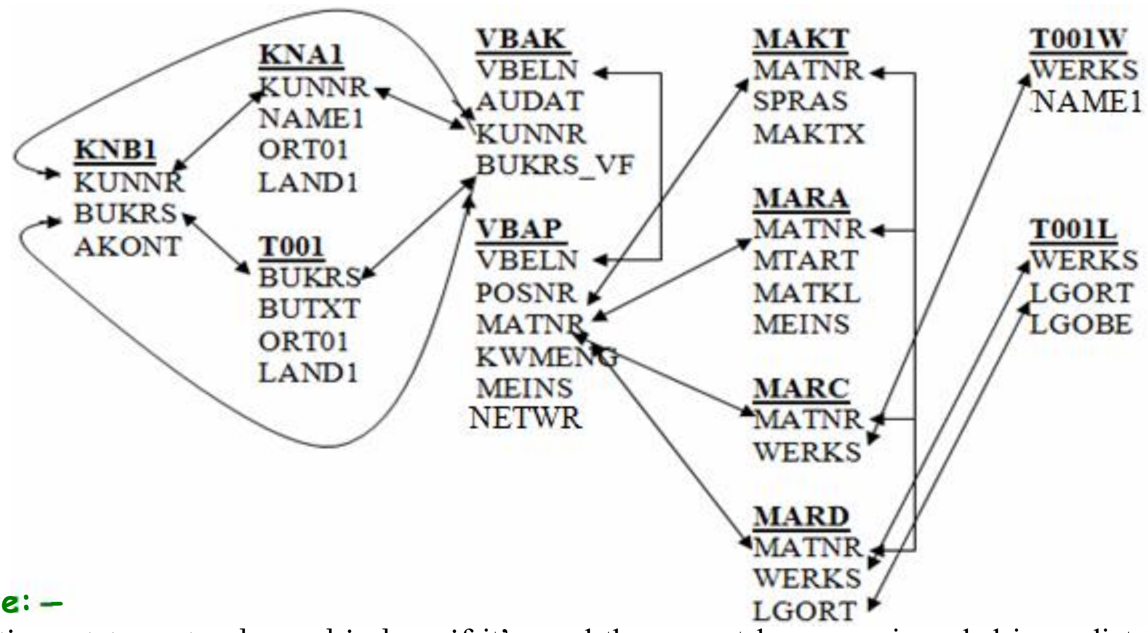
```
DATA V1 TYPE T001-BUKRS.
SELECT-OPTIONS S_BUKRS FOR V1.
TYPES: BEGIN OF TY_T001,
       BUKRS TYPE T001-BUKRS,
       BUTXT TYPE T001-BUTXT,
       ORT01 TYPE T001-ORT01,
       END OF TY_T001.
DATA WA_T001 TYPE TY_T001.
DATA IT_T001 TYPE TABLE OF TY_T001.
SELECT BUKRS BUTXT ORT01 FROM T001 INTO 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

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 continues.

➔ **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 in within a loop. It leaves the loop by ending the current loop process.

➔ **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.
```

Check: -

Check statement is used to terminate the sub routine conditional.

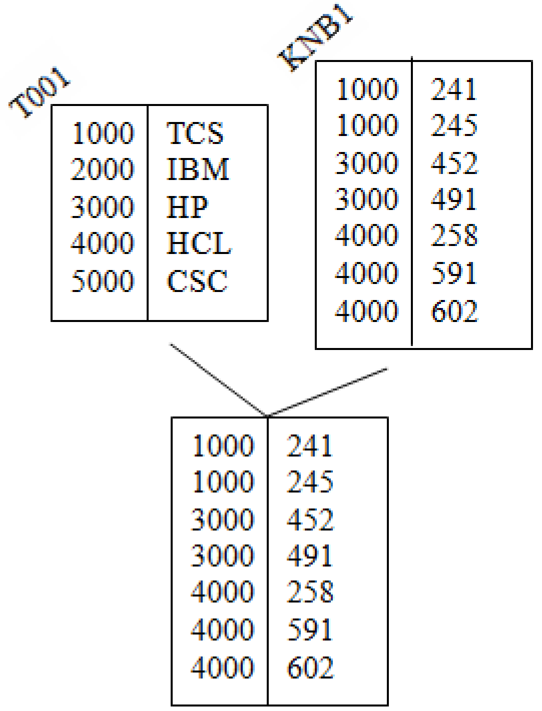
Joins

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.



T001

1000	TCS
2000	IBM
3000	HP
4000	HCL
5000	CSC

KNB1

1000	241
1000	245
3000	452
3000	491
4000	258
4000	591
4000	602

1000	241
1000	245
3000	452
3000	491
4000	258
4000	591
4000	602

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>.

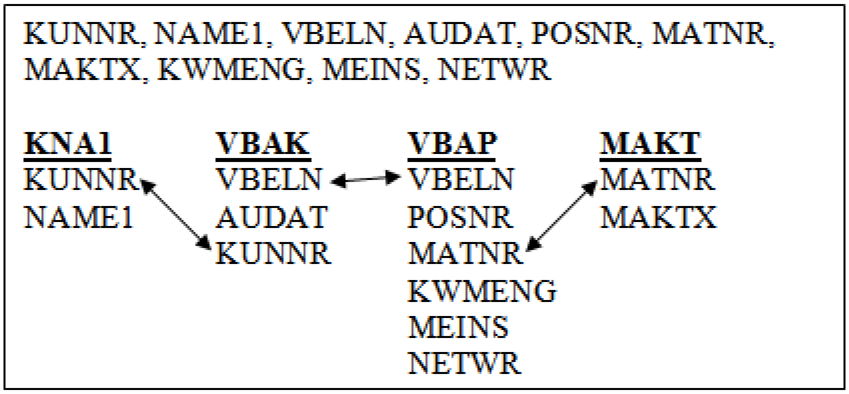
Note: – The link field must be primary fields in at least one data base table.

➔ 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.
```

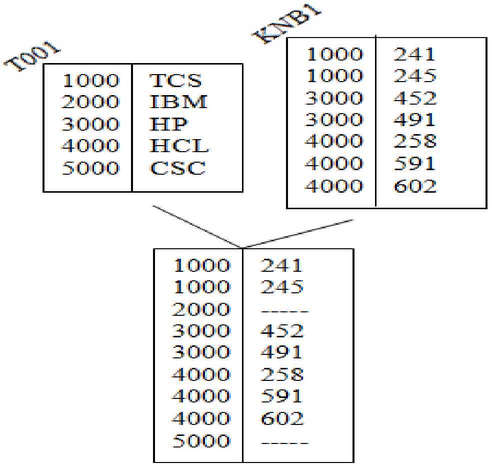
<u>**T001**</u>		<u>**LFB1**</u>
BUKRS	↔	BUKRS
BUTXT		LIFNR

```
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.
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.



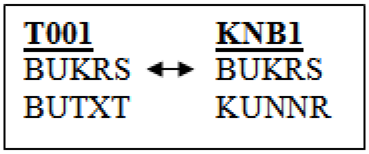
1000	TCS
2000	IBM
3000	HP
4000	HCL
5000	CSC

1000	241
1000	245
3000	452
3000	491
4000	258
4000	591
4000	602

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.

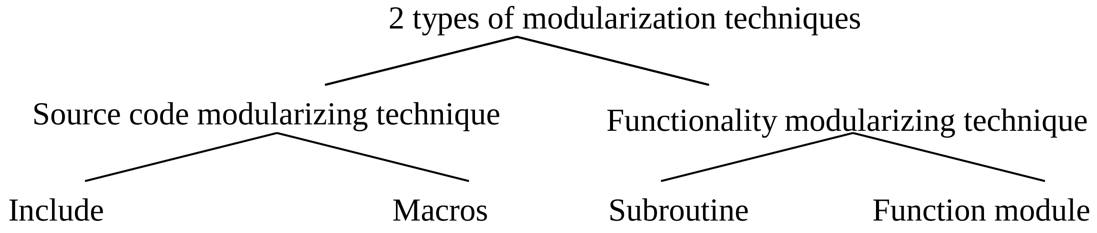
➔ Based on the given company codes display the company codes, company names & customer number based on left outer join.



```
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.
```

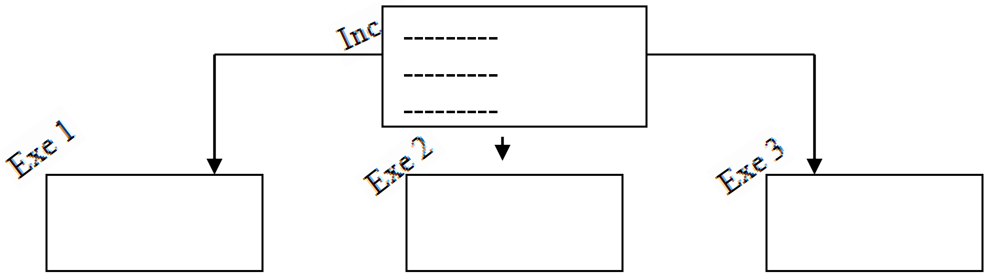
Modularization techniques:

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


2 types of modularization techniques
Source code modularizing technique
Functionality modularizing technique
Include
Macros
Subroutine
Function module

Include: -

We can't execute an include independently where as the same include program can be include 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> ---- calling <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: -

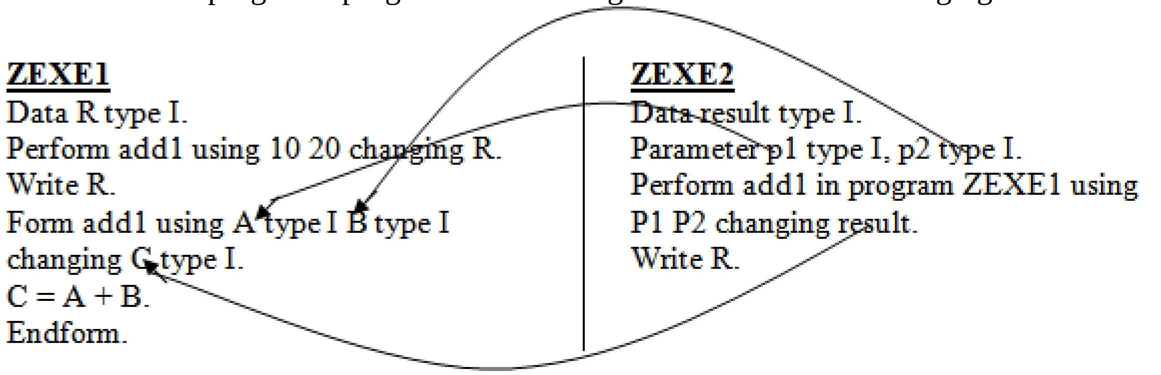
It's nothing but the definition of the subroutine as well as calling of the subroutine must be in the same program.

External subroutine: -

It's nothing but the definition of the subroutines taken from one program & the calling of the subroutine is 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> ---



ZEXE1

```
Data R type I.
Perform add1 using 10 20 changing R.
Write R.
Form add1 using A type I B type I
changing C type I.
C = A + B.
Endform.
```

ZEXE2

```
Data result type I.
Parameter p1 type I, p2 type I.
Perform add1 in program ZEXE1 using
P1 P2 changing result.
Write R.
```

Differences between Macros & subroutines: -

Macros	Subroutines
1. In macros definition & calling in the same program	1. In subroutines definition & calling may / mayn't in the same program.
2. In macros definition should be the first & calling should be next.	2. In subroutines calling should be first & definition should be next.
3. Macros contain up to 9 inputs.	3. Subroutine contain any number of inputs.
4. After the definition of macro we can place any executable statement.	4. After the definition of subroutine we can't place any executable statement.
5. Macros are used in HR-ABAP.	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 'SPRAO Technologies'.
  Write / 'SR Nagar'.
  Exit.
  Write / 'HYD'.
Endform.
```

```
OP: -   SPRAO 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.
```

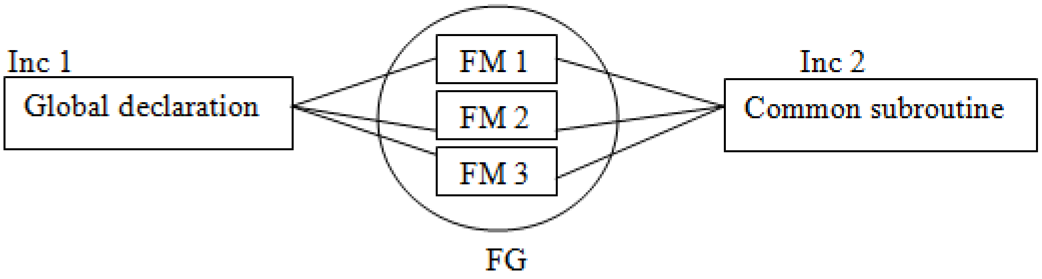
Working with central data:-

```
Data A type i.
A = 10.
Perform hai.
Write / A.
Form hai.
A = 20.
Write / A.     20
Endform.
```

```
Data A type i.
A = 10.
Perform hello.
Write A.
Form hello.
Local A.
A = 20.
Write A.
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.

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**. 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.

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: -

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.

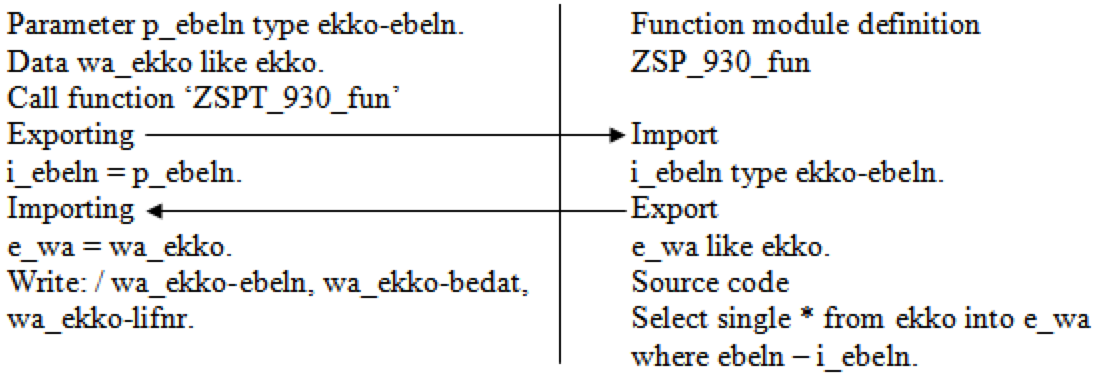
Note: - Function module writes single value & multiple values. So we need to maintain any display statement (write) in the function module definition.

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

Function module definition	**Function module calling**
Import ---	Export
Export ---	Import
Changing --------------------------------------	Changing
Tables --	Tables

Steps to call the function module: -

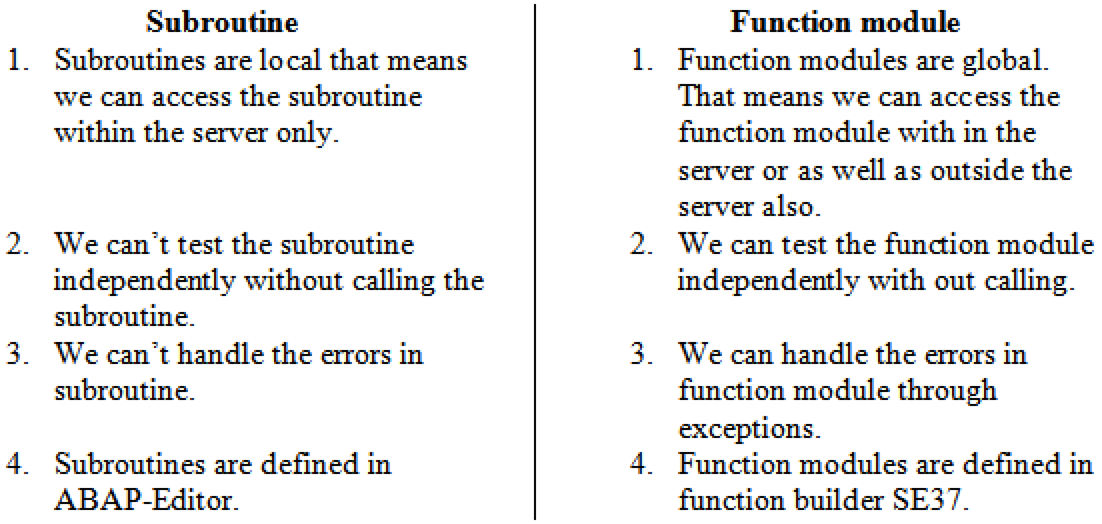
Place the cursor where you want to call the function module. Click on pattern in the application tool bar. Provide the function module name.



```
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	Function module
1. Subroutines are local that means we can access the subroutine within the server only.	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't test the subroutine independently without calling the subroutine.	2. We can test the function module independently with out calling.
3. We can't handle the errors in subroutine.	3. We can handle the errors in function module through exceptions.
4. Subroutines are defined in ABAP-Editor.	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: -

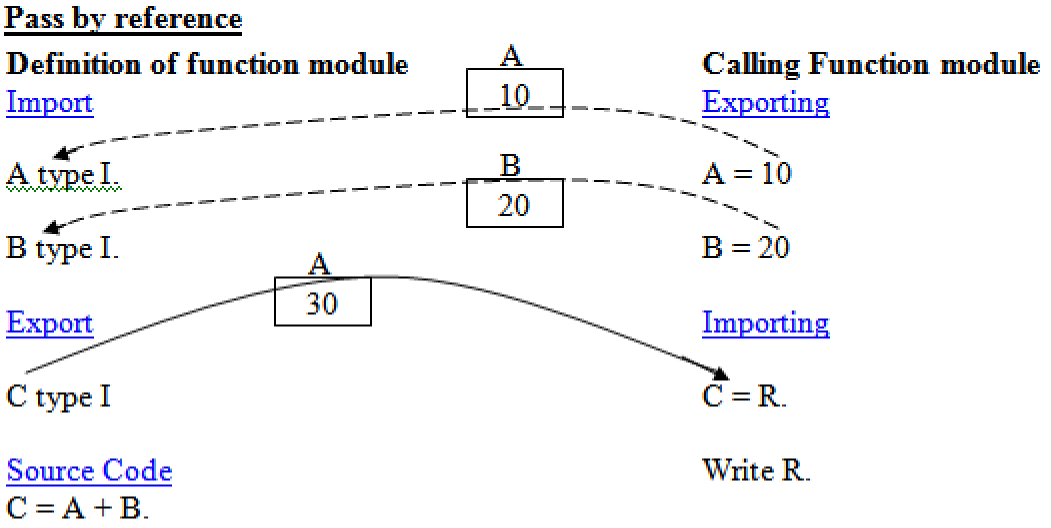
Execute SE11. Select the radio button data type. Provide the structure name. Click on create. Select the radio button structure. Enter. Provide the short description. Provide component, component type.

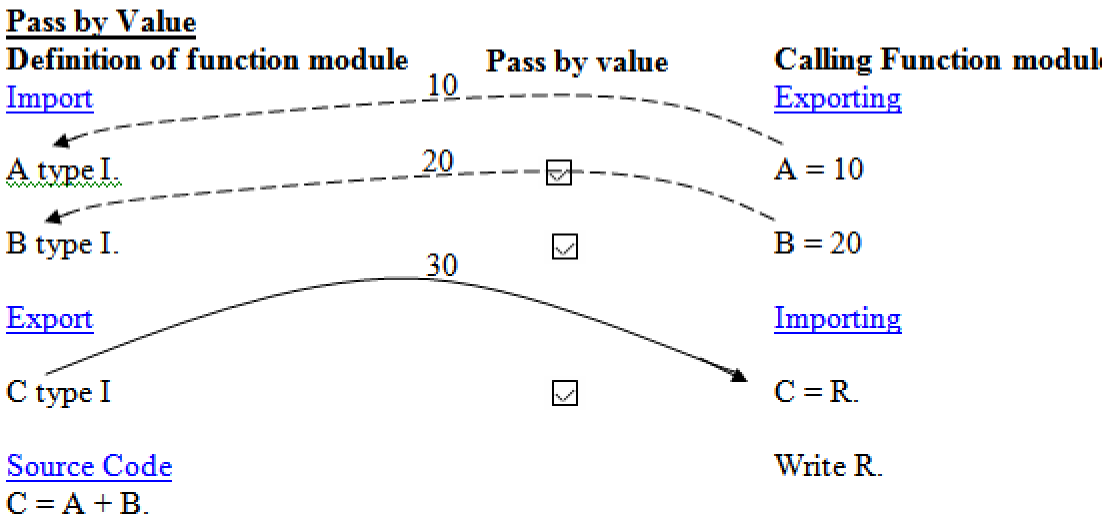
Bukrs	bukrs
Butxt	butxt
Ort01	ort01

```
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 transfering. Pass by value means from calling to definition & definition to calling parameter values only passing.





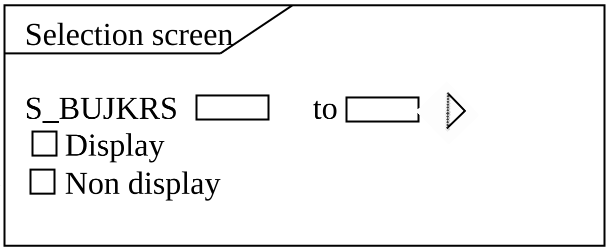
REPORTS

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> $\underbrace{\text{with frame}}_{\text{Optional}}$ $\underbrace{\text{title text-<no>}}_{\text{Optional}}$.

Selection-screen end of block <block name>.



Selection screen

S_BUJKRS ▭ to ▭ ▷

□ Display

□ Non display

Syntax of check box: -

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

Ex: -

Parameter P_DIS as checkbox.

⇩

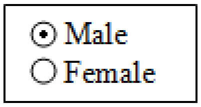
□ P_DIS

Syntax of radio button: -

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

Ex: -

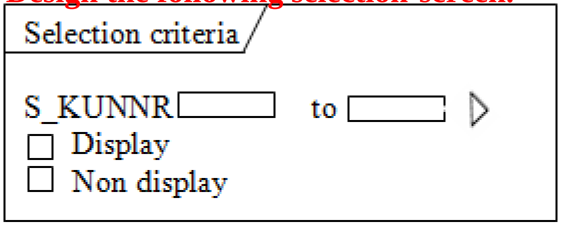
Parameter: Male radiobutton group g,
Female radiobutton group g.



● Male

○ Female

➔ Design the following selection-screen.



Selection criteria

S_KUNNR ▭ to ▭ ▷

□ Display

□ Non display

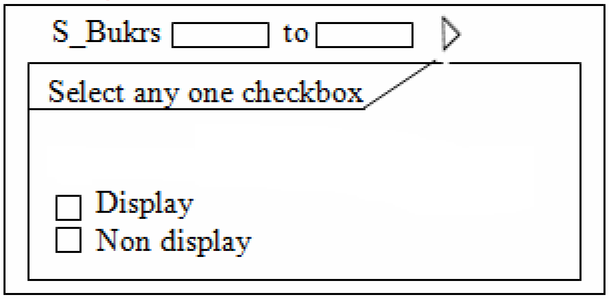
```
data v1 type kna1-kunnr.
selection-screen begin of block a with frame title text-001.
select-options s_kunr for v1.
parameter p_dis as checkbox.
parameter p_nondis as checkbox.
selection-screen end of block a.
```

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.**



S_Bukrs ▭ to ▭ ▷

Select any one checkbox

□ Display
□ Non display

```
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.
```

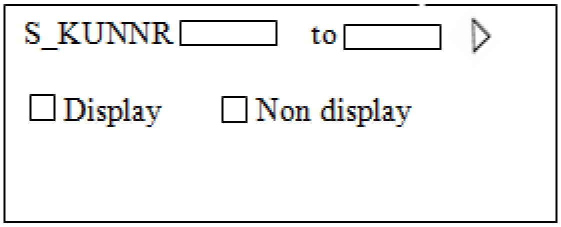
When ever we are working with begin of line & end of line then the name of the parameter is disable. So we must provide comment before, after block of a checkbox or radiobutton.

Syntax: -

Selection-screen comment x(y) text-<no>. → 3 digit number
Starting position

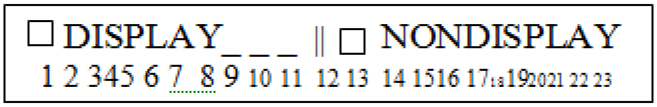
➔ **Design the selection-screen as shown in the figure.**

```
data v1 type kna1-kunnr.
selection-screen begin of block a
with frame.
select-options s_kunnr 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.
```



S_KUNNR ▭ to ▭ ▷

□ Display □ Non display

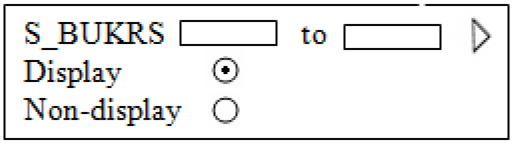


□ DISPLAY_ _ _ || □ NONDISPLAY

➔ **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.
```



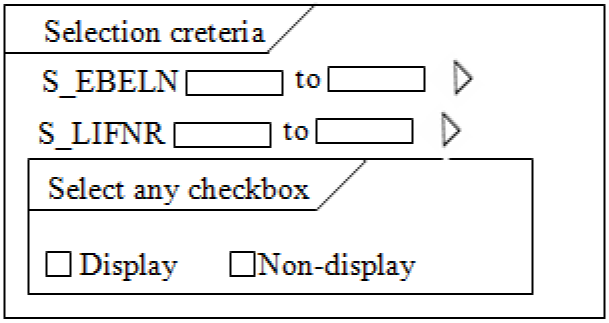
S_BUKRS ▭ to ▭ ▷

Display ⊙

Non-display ○

```
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: -

<Vendor details>
↳ <Vendor purchasing header details>
↳ <Purchasing item details>
↳ <Material details>

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 provide 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 display the output.

ADV: - This is used to fetch the data from data base & 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 of the top of the each page.

ADV: - It's used to display the header information.

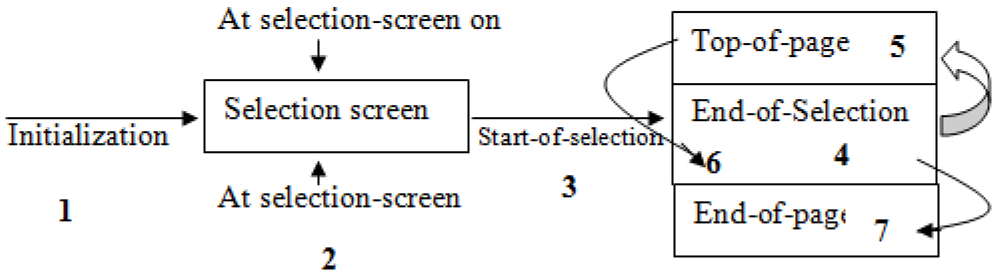
End-of-page: -

It's an event which is triggered of 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.

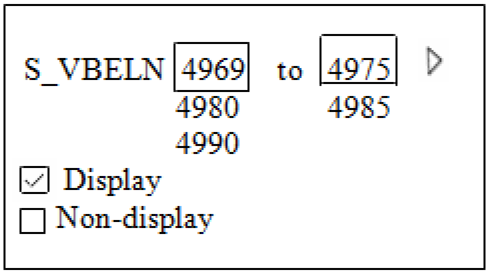
At selection-screen on help-request: -

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.

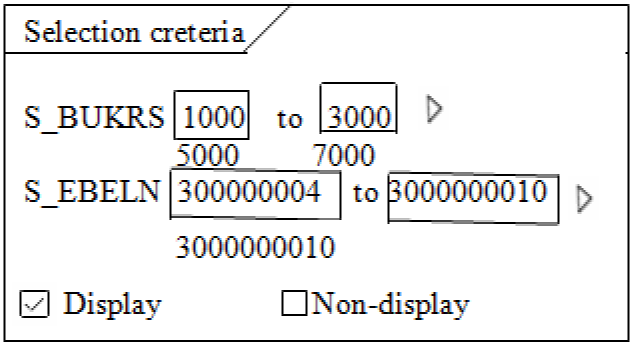
```
Data v1 type vbak-vbeln.
Selection-screen begin of block A with frame.
Select-options s_vbeln for v1.
Parameter p_dis as checkbox.
Parameter p_nondis as checkbox.
Selection-screen end of block A.

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'.
```



S_VBELN 4969 to 4975 ▷
4980 4985
4990
☑ Display
□ Non-display

➔ 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 creteria

S_BUKRS 1000 to 3000 ▷
5000 7000
S_EBELN 300000004 to 3000000010 ▷
3000000010

☑ Display □ Non-display

```
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 →
   Error message (E)

   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.

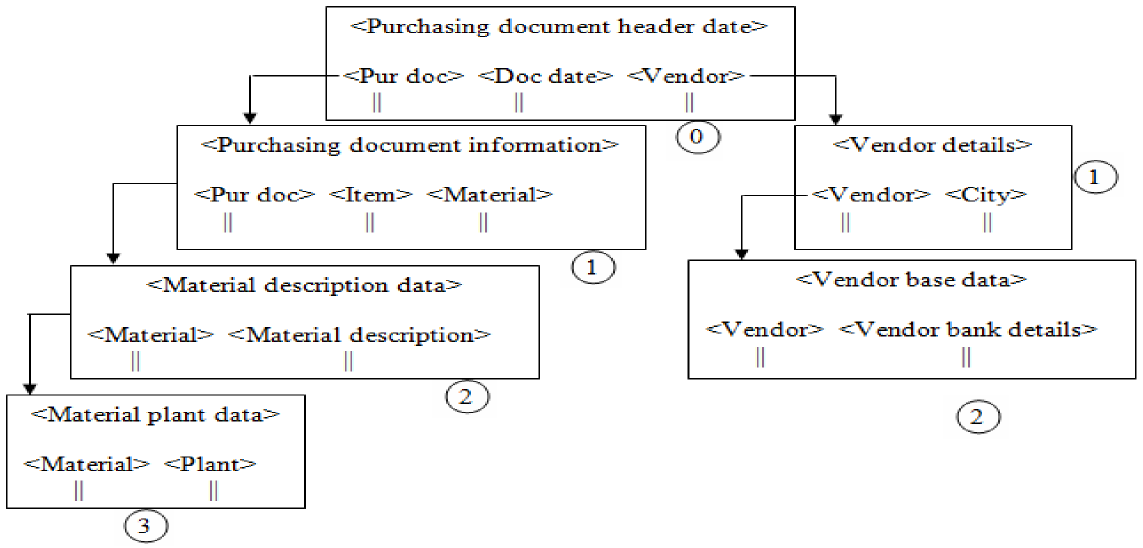
Ex: -

Parameter P type I.
If P < 10.
Message I000(zmessage1).
Elseif p > 10.
Message I001(zmessage1).
Else
Message I002(zmessage1).

```
               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: -

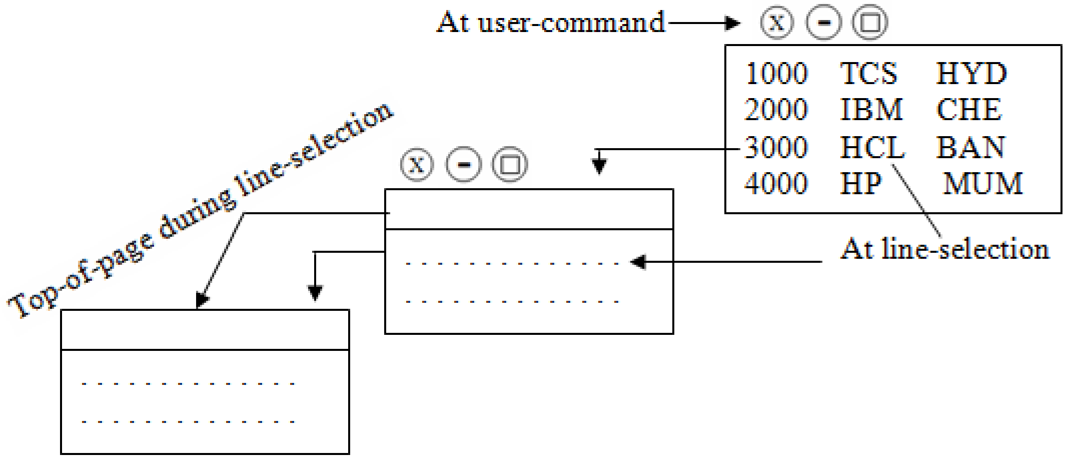
It's an event which is triggered at the top of each secondary list.

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.



```
1000   TCS   HYD
2000   IBM   CHE
3000   HCL   BAN
4000   HP    MUM
```

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.

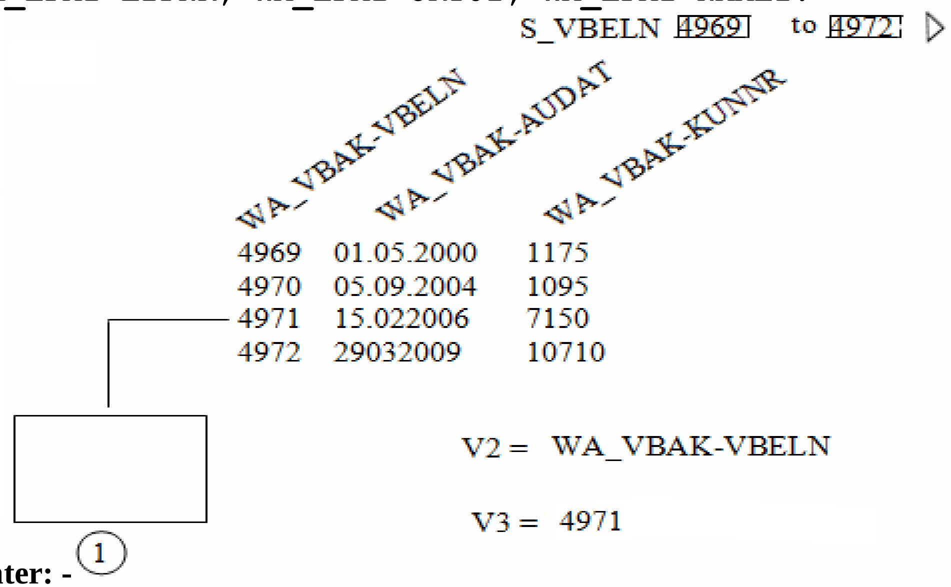
SY-LILLI: -

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.

```
      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.
```



```
4969   01.05.2000   1175
4970   05.09.2004   1095
4971   15.022006    7150
4972   29032009     10710
```

V2 = WA_VBAK-VBELN

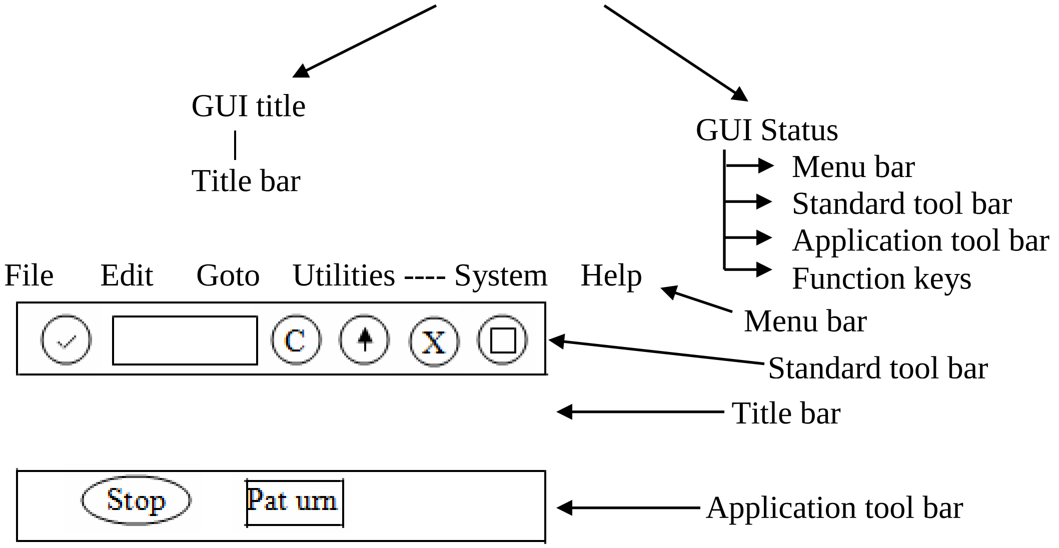
V3 = 4971

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: -

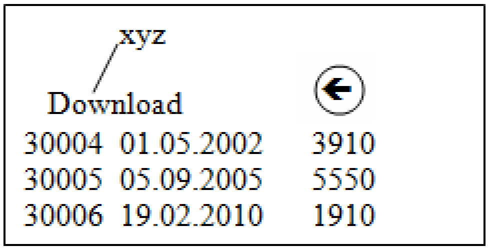
COMPONENTS OF **GUI**



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.



```
30004  01.05.2002    3910
30005  05.09.2005    5550
30006  19.02.2010    1910
```

<u>Steps to create the GUI to the program: -</u>
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.

<u>Syntax of attaching our own GUI to the program: -</u>
SET PF-STATUS ‘<status name>’.

<u>Ex: -</u> 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.
```

<u>Syntax of reading the displayed records:-</u>

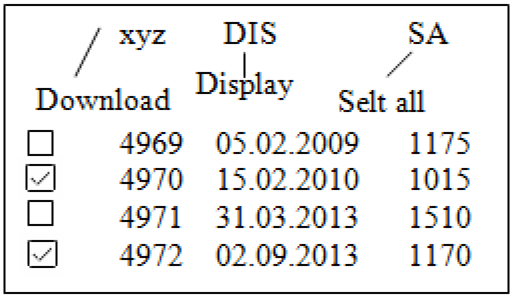
Read line <line number> field value <Displayed field name1> into <variable1>
<displayed field name2> into <variable2> ----.

<u>Ex: -</u>

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.



□	4969	05.02.2009	1175
☑	4970	15.02.2010	1015
□	4971	31.03.2013	1510
☑	4972	02.09.2013	1170

<u>Some of the standard transaction codes: -</u>

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

<u>Syntax of call transaction: -</u>

Call transaction '<Tcode>'.

<u>Ex: -</u>

Call transaction 'VA03'.

Before call the transaction we must set the value set parameter ID '<ID name>' field '<value>'.

<u>Steps to identify the parameter ID:-</u>

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>'.

<u>Ex: -</u>

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.

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. Filed name
   2. Column position
   3. Column Heading
   4. Co lour
   5. Hotspot

3. Call the 'REUSE_ALV_GRID_DISPLAY' function module.

(OR)

CALL THE 'RESUE_ALV_LIST_DISPLAY' function module.

Note: – REUSE_ALV_GIRD_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 on 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.

<u>Some of the fields in field catalog internal table: -</u>

1. Field name → Name of the field
2. Col_Pos → Column position
3. Seltext_S
Seltext_M } Column heading
Seltext_L
4. Emphasize → Co lour
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.

<u>Steps to create type-pools: -</u>

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.

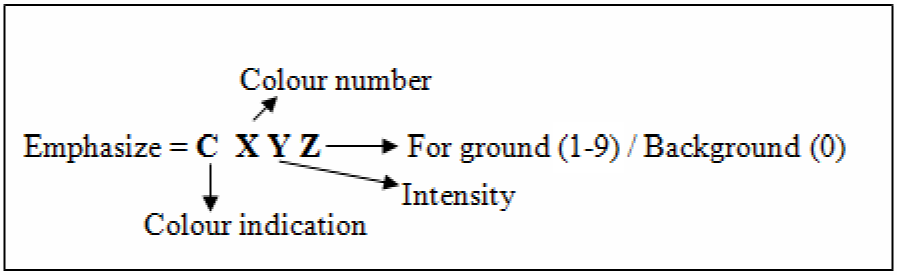
Note: - All the names under the group must be starts with type group name_ (underscore).

```
Type-pools ZTG.
Types: begin of ZTG_T001,
       Bukrs type t001-bukrs,
       Butxt type t001-butxt,
       Ort01 type t001-ort01,
       End of ZTG_T001.                  (IT)
Types ZTG_T_T001 type table of ZTG_T001.←—— (WA)
```

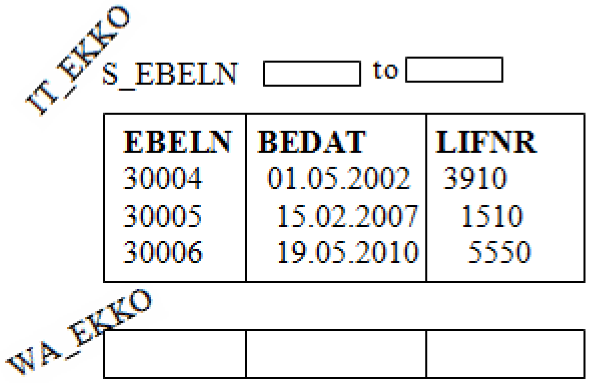
➔ Display the company codes, company names and cities.

```
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.
```



EBELN	BEDAT	LIFNR
30004	01.05.2002	3910
30005	15.02.2007	1510
30006	19.05.2010	5550

```
* 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'.
```

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.

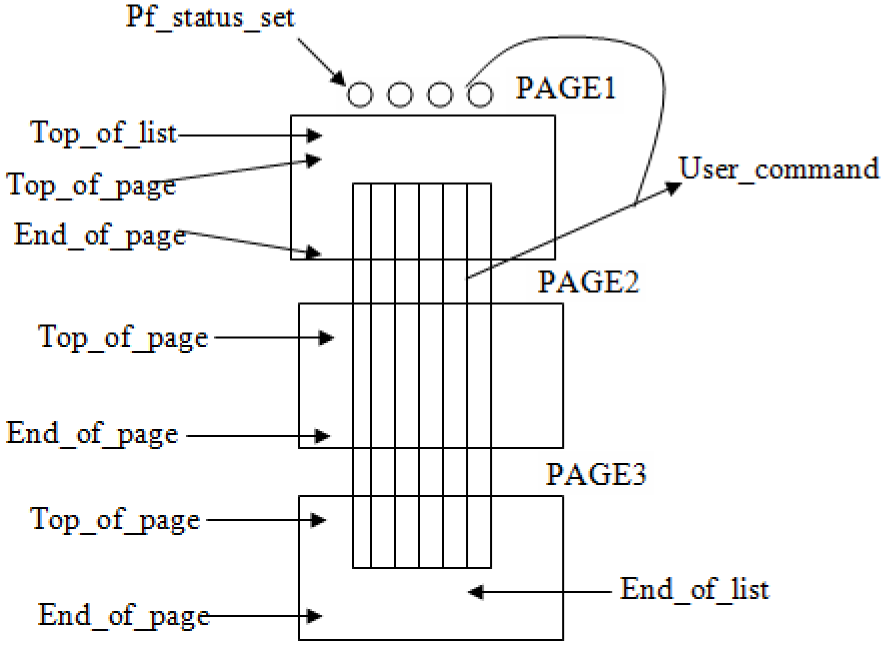
User_command: -

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.

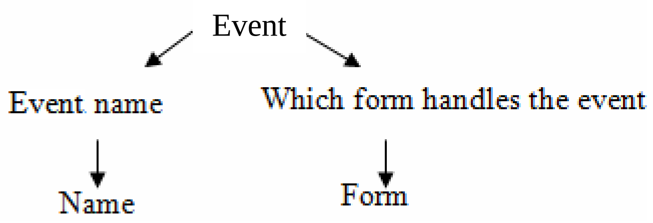
This event acts like both at line-selection & at user-command.

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.



```
* 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.

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
EKPO EBELN

HEADER 01 = 'EBELN'
ITEM01 = 'EBELN'

MARD WERKS
T001L LGORT

HEADER 01 = 'WERKS'
ITEM 01 = 'WERKS'
HEADER 02 = 'LGORT'
ITEM 02 = 'LGORT'.

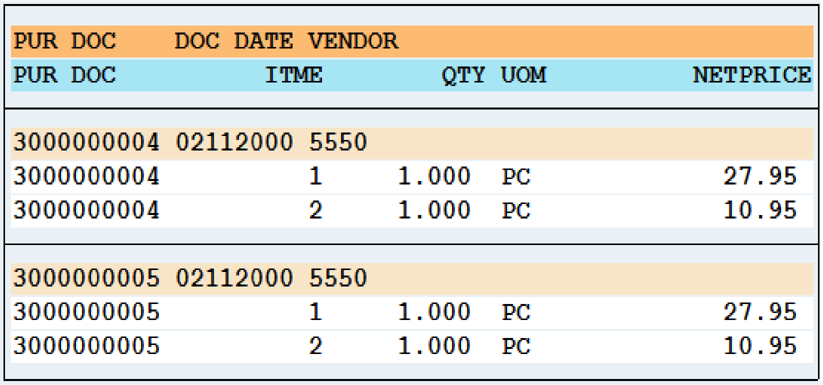
Note: – When ever we are working with hierarchical ALV then we must maintain link fields in between header and item internal table.

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.**



```
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
```

```
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'.
```

```
    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.
```

<u>Differences between GRID & LIST display: -</u>

Grid display	**List display**
1. Grid display supports OOP's ALV.	1. List display does not supports OOP's ALV.
2. Grid display slower	2. List display faster
3. Edit and logo is possible in grid display	3. These are not possible in list display.
4. Blocked and hierarchical ALV is not possible in grid display	4. These are possible in list display.
5. In the output ¡ symbol indicates F1 help.	5. In list display i indicates number of records is displayed.
6. By using 'commentary write' function module only we can print the text on top or bottom event.	6. Either by using write or commentary writes function module we can print the text on top or bottom event.
7. This is used to display the output in grid format.	7. This is used display the output in list format.

Note: - When ever we are working with ALV Repots 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'.

<u>Steps to identify the fields in field catalog:</u> -
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.

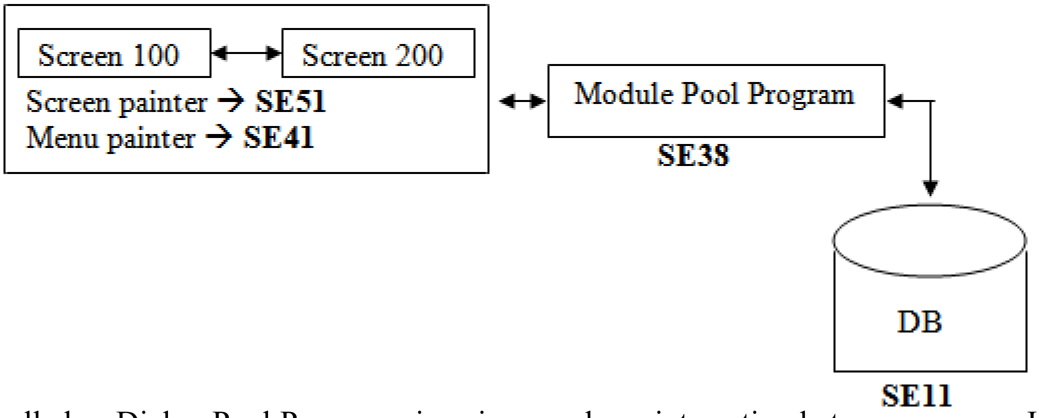
Note: - When ever we are filling the field catalog manually, if you want to refer the database table, field description to print as a heading then we use following fields in field catalog.

Ref_fieldname
Ref_tabname.

EX: -
Wa_fcat-fieldname = 'VBELN'.
Wa_fcat-col_pos = '1'.
Wa_fcat-ref_tabname = 'VBAK'.
Wa_fcat-ref_fieldname = 'VBELN'.
Append wa_fcat to it_fcat.
Clear wa_fcat.

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	**Module Pool Program**
1. We can execute the executable program either through transaction code or directly.	1. We execute the module pool program only through transaction.
2. Type of the executable program is '1'.	2. Type of the module pool program is 'M'.
3. All the standard reports are executable programs.	3. All the standard transactions are module pool programs.
Ex: - ME2L, ME2K, ME2M	**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: -

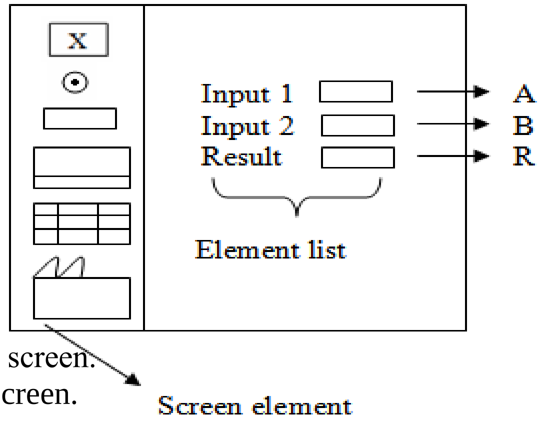
Layout is the collection of screen elements. I.e. check box, radio button, input output field, push button, table control, tab strip, etc.

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.

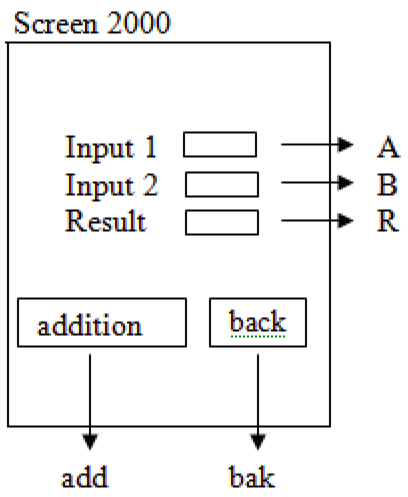
ADV: - This is used to provide the help document to the input variable.

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 on equality declaration in ABAP editor.

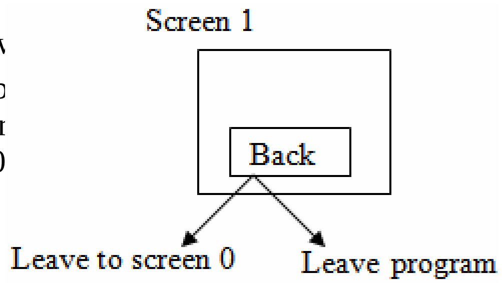
Steps to work with module pool program as per ABAPer point of view: -

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

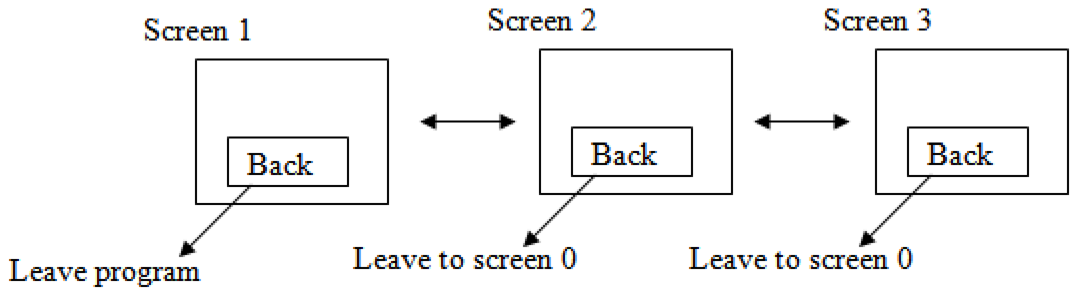
→ Design the screen as shown in the below.
After the user provide the input1 and input2 click on addition button then we display the output in the result field. If the user clicks on back button then we go to program.



Note: – If we are working v
only one screen then the b
button functionality of the scr
is either leave to screen 0
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 on wards 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 abaper 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: -

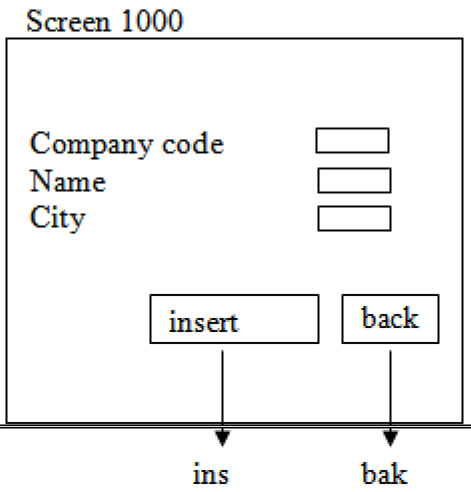
Select the program in the left panel. Right click, create → transaction. Provide the transaction code, short description. Enter. Provide the program name, screen number. Select the GUI checkbox. Save.

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'.
```

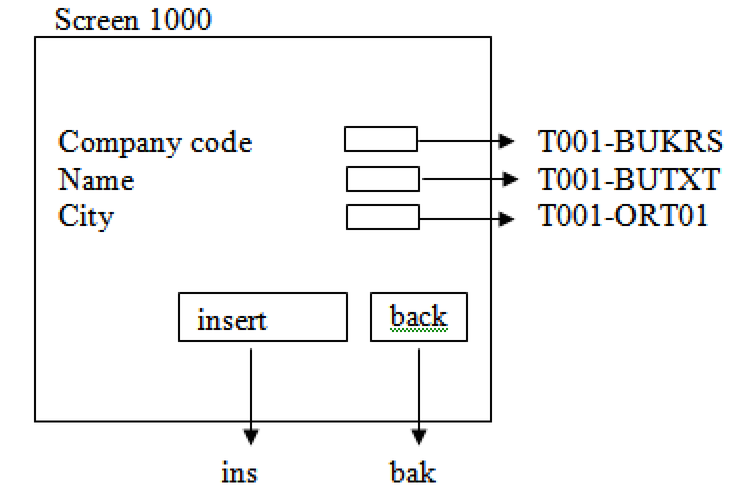


```
C = A + B.
WRITE C.
ELSEIF SY-UCOMM = 'BACK'.
LEAVE PROGRAM.
ENDIF.
ENDMODULE.
```

Flow logic of 2000 screen: -

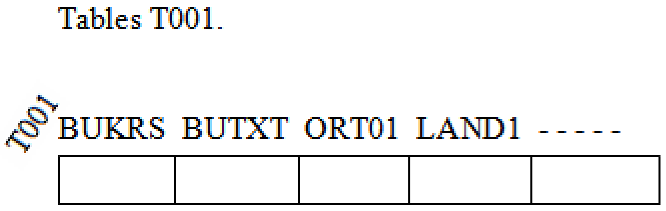
```
PROCESS BEFORE OUTPUT.
* MODULE STATUS_2000.
PROCESS AFTER INPUT.
 MODULE USER_COMMAND_2000.
```

➔ **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.

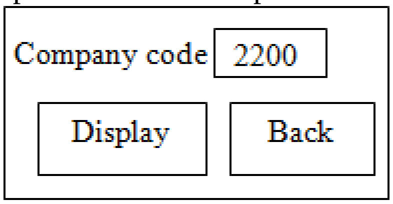


Programming: -

```
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.**

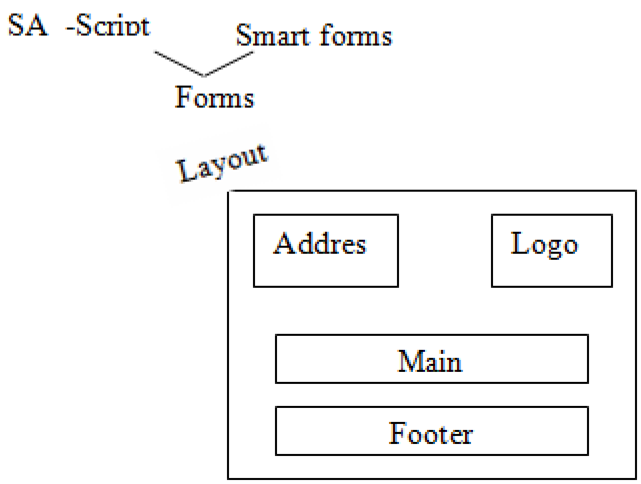
Note: – If you want to display the output like ordinary report then we must place leave to list processing before display the output.



```
DATA A TYPE KNB1-BUKRS.
TYPES: BEGIN OF TY_KNB1,
       BUKRS TYPE KNB1-BUKRS,
       KUNNR TYPE KNB1-KUNNR,
       AKONT TYPE KNB1-AKONT,
```

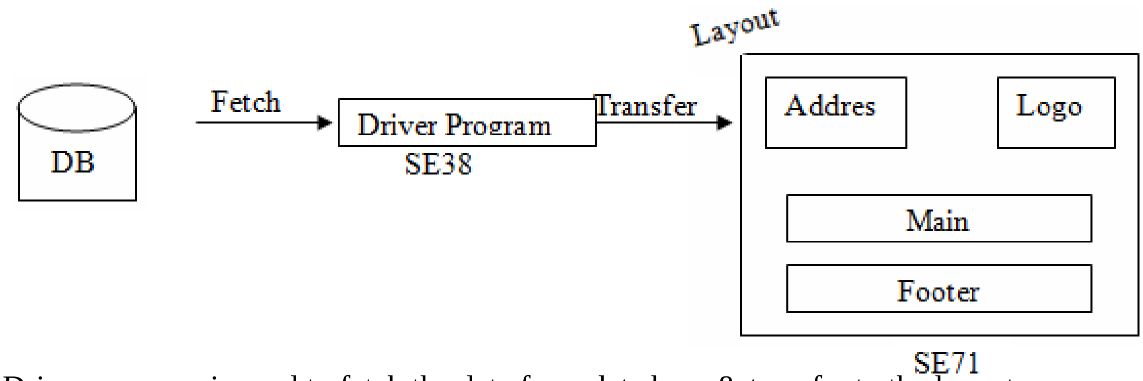
SAP-SCRIPT

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: -

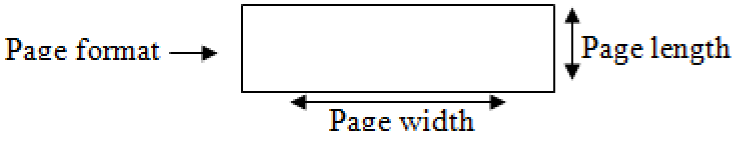
1. Header
2. Pages
3. Windows
4. Page window
5. Paragraph format
6. Character format
7. Documentation

Header: -

Header is used to maintain the administrative information. i.e. form name, language & page format.

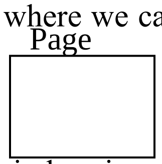
In the real time page formats are created by BASIS people through SPAD transaction.

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.



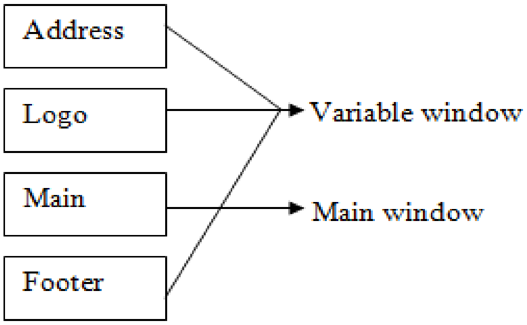
Windows: -

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

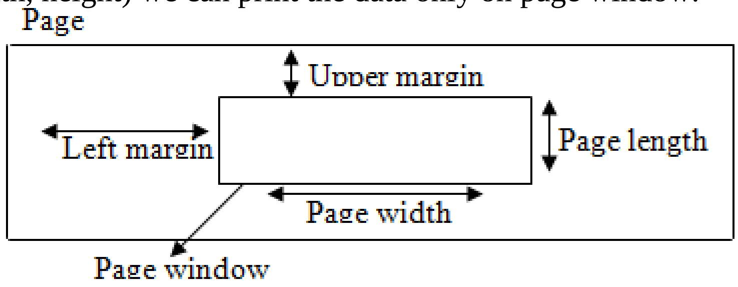
Note: – Main window is the default window in SAP script without a main window we can't design SAP script.

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: -

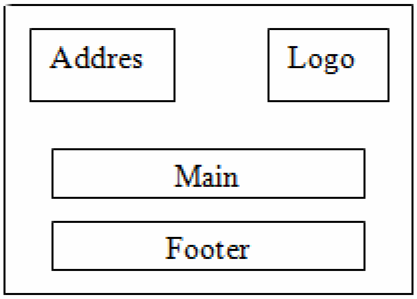
This is used to display the perticular text with the required font & style.

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.



<u>Steps to design the layout: -</u>

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,
```

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.
* SPRAO TECHNOLOGIES
/: endif.

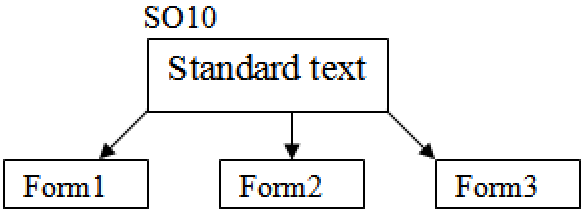
Control commands: -

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 maintain these information in each form its 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.

SPRAO 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
* SPRAO 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.

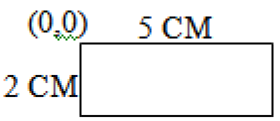
Box: -

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 = $^{1}/_{20}$ PT.



Ex: -

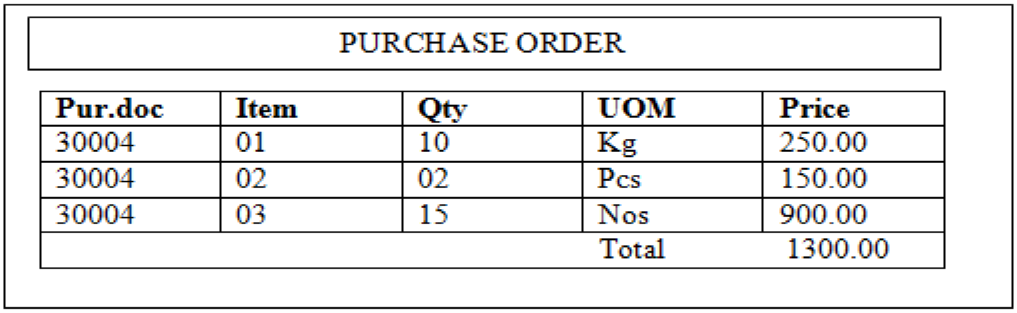
/: Box XPOS ‘0’ cm YPOS ‘0’ cm width ‘5’ cm height ‘2’ CM frame ‘20’ TW.

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.

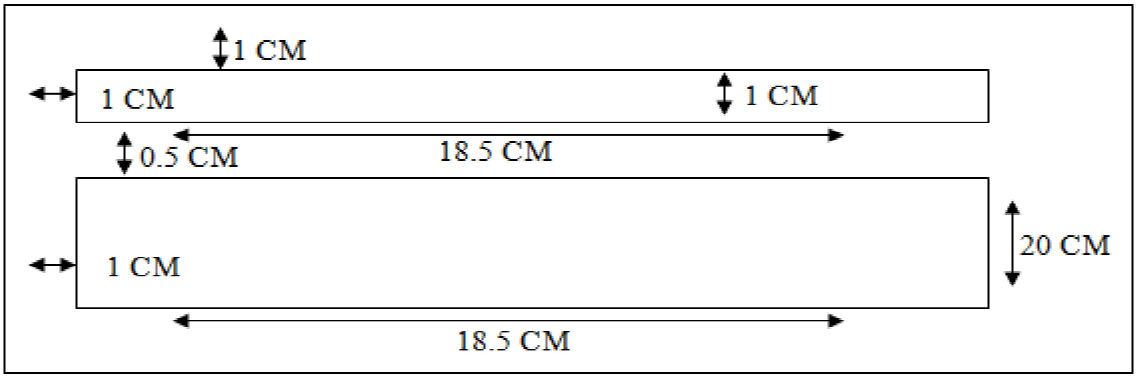
/: Box xpos ‘0’ cm ypos ‘0’ cm width ‘0’ cm height ‘3’ cm frame ‘20’ pw.

➔ **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: -

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.

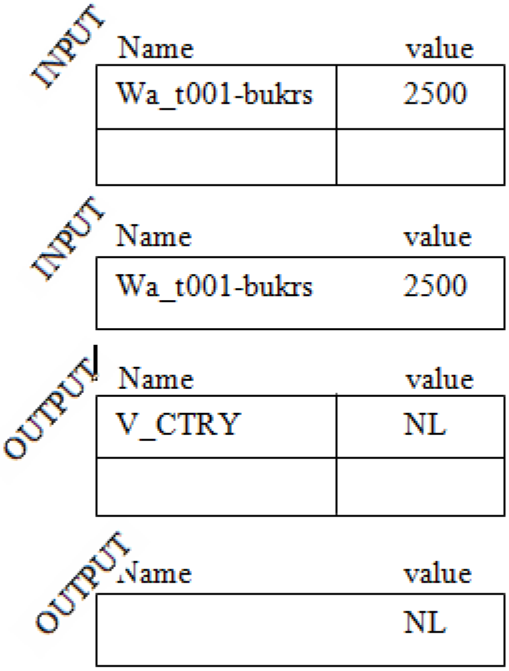
Note: – ITCSY is the structure which contains the two fields name & value.

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'.
```

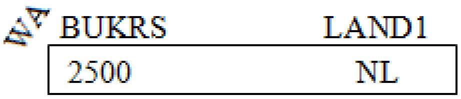


Name	value
Wa_t001-bukrs	2500

Name	value
Wa_t001-bukrs	2500

Name	value
V_CTRY	NL

Name	value
	NL

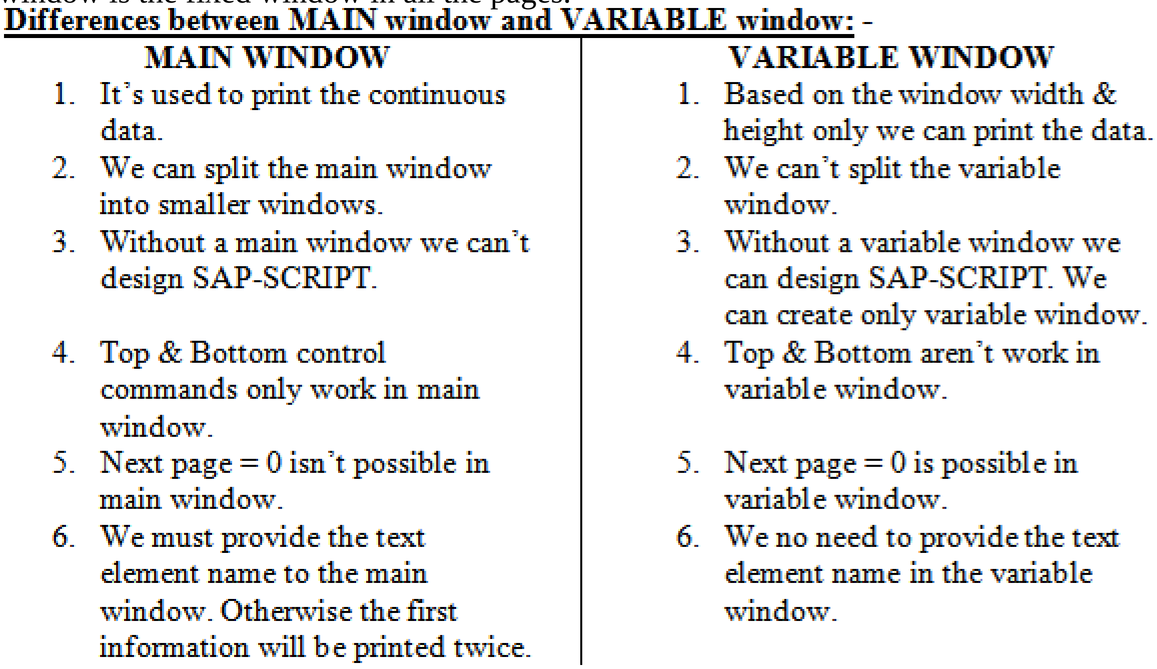


BUKRS	LAND1
2500	NL

Types of windows in the SAP-SCRIPT: -

1. Main window
2. Variable window
3. Constant window

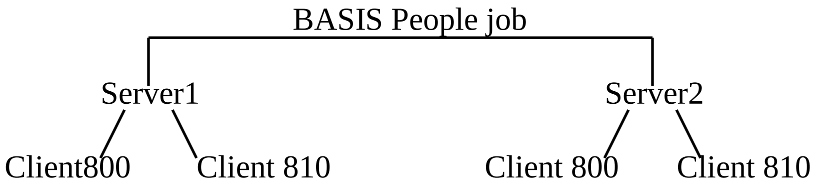
Control window is the fixed window in all the pages.



Differences between MAIN window and VARIABLE window: -

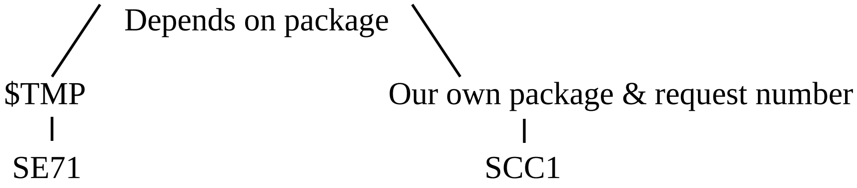
MAIN WINDOW	VARIABLE WINDOW
1. It's used to print the continuous data.	1. Based on the window width & height only we can print the data.
2. We can split the main window into smaller windows.	2. We can't split the variable window.
3. Without a main window we can't design SAP-SCRIPT.	3. Without a variable 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.



SAP script	✓	X
Driver program	✓	✓

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

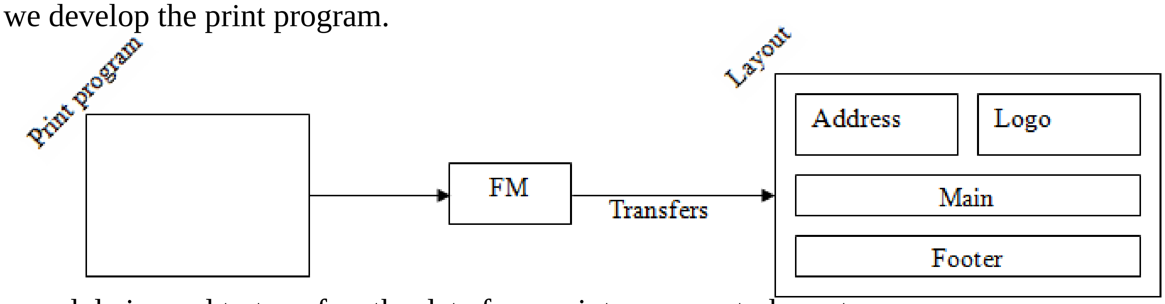
SMART FORMS: -

Smart forms are used to design the business documents such as invoices, purchase orders, sales orders. . . smart form is introduced from 4.6C version on wards.

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 are the collection of form attribute, form interface, global definitions.

Form attributes: -

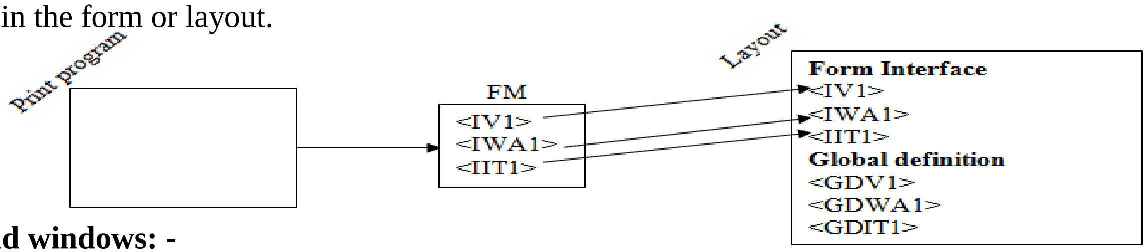
These are used to maintain the administrative information that is form name, language, page format, default style.

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.
```

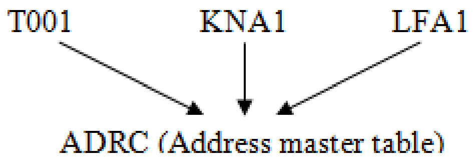
Expand the page in the left panel. Select the main window. Right click → create → text. Double click on text. Click on editor under general attributes tab. Provide symbols.

&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.
```

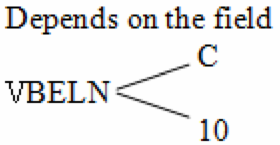
Select the page in the left panel. Right click → create → address. Double click on address. Provide the address number in the general attributes tab. Save, check, activate. In the menu bar click on environment → function module name. Based on this function module we develop the print layout.

```
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



Structure: ZVBAK1

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

Steps to design the smort 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_SEBELN   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.

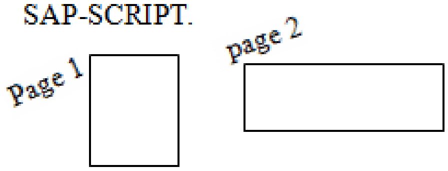
&wa_vbak-vbeln& &wa_vbak-audat& &wa_vbak-kunnr&

Click on back, save, check, activate.

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	**SMART FORM**
1. Multiple page formats aren't possible in SAP-SCRIPT.  	1. Multiple page formats are possible in SMARTFORMS.
2. Labels are possible in SAP-SCRIPT.	2. Labels aren't possible in smart form.
3. With out a Main window we can't create SAP-SCRIPT.	3. Without a Main window we can design smart forms.
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.	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 worked in SAP-SCRIPT (Protect, top, bottom)	5. Control commands are aren't possible in smart form.
6. Colors aren't possible in this.	6. Colors are possible in smart form.
7. Water mark / background picture isn't possible in script.	7. Watermark / background picture is possible in smart form.
8. Paragraph & character formats aren't reusable in script.	8. Paragraph & character formats are reusable in smart form.
9. By using 'RSTXDBUG' standard program we can debug the script.	9. By using static break points (break-point) we can debug the smart form.
10. We maintain the backup of script in .TXT file.	10. We maintain the backup of smart form in .XML file.
11. When ever we activate the script it won't generates function module.	11. When ever we activate the smart form it generates a function module.
12. We can't develop the code in script.	12. We can develop the code in smart form.
13. We can convert script to smart form.	13. We can't convert smart form to script.
14. Script s suitable for complex coding	14. Smart form is suitable for complex design.

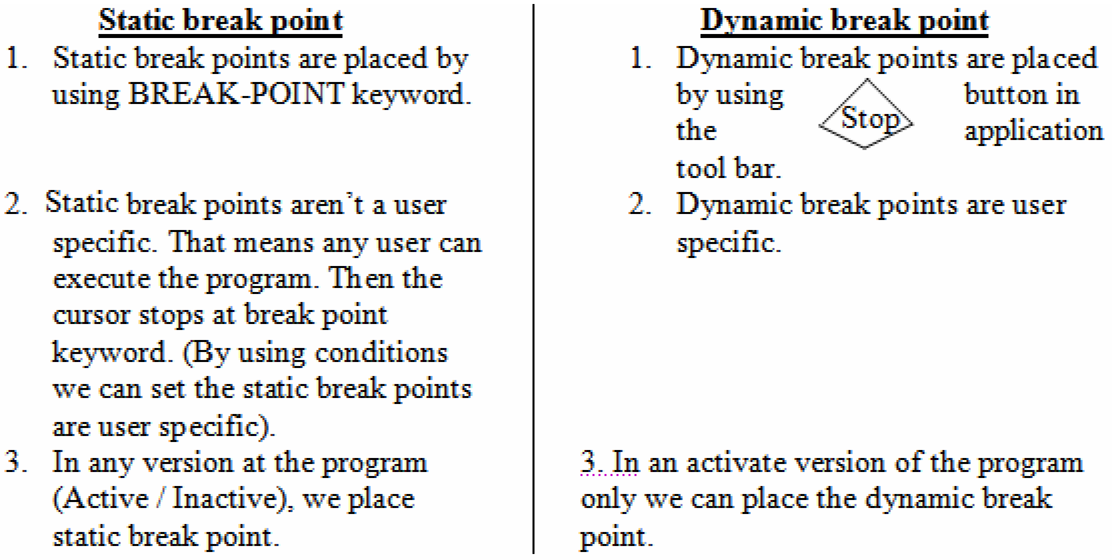
Note: – In the smart forms function module number we can't transported to quality & live server. Depends on server configuration function module number is generated. So we can't fix the function module number in the print program. We always generate the function module by using

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	Dynamic break point
1. Static break points are placed by using BREAK-POINT keyword.	1. Dynamic break points are placed by using the Stop button in application tool bar.
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).	2. Dynamic break points are user specific.
3. In any version at the program (Active / Inactive), we place static break point.	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.

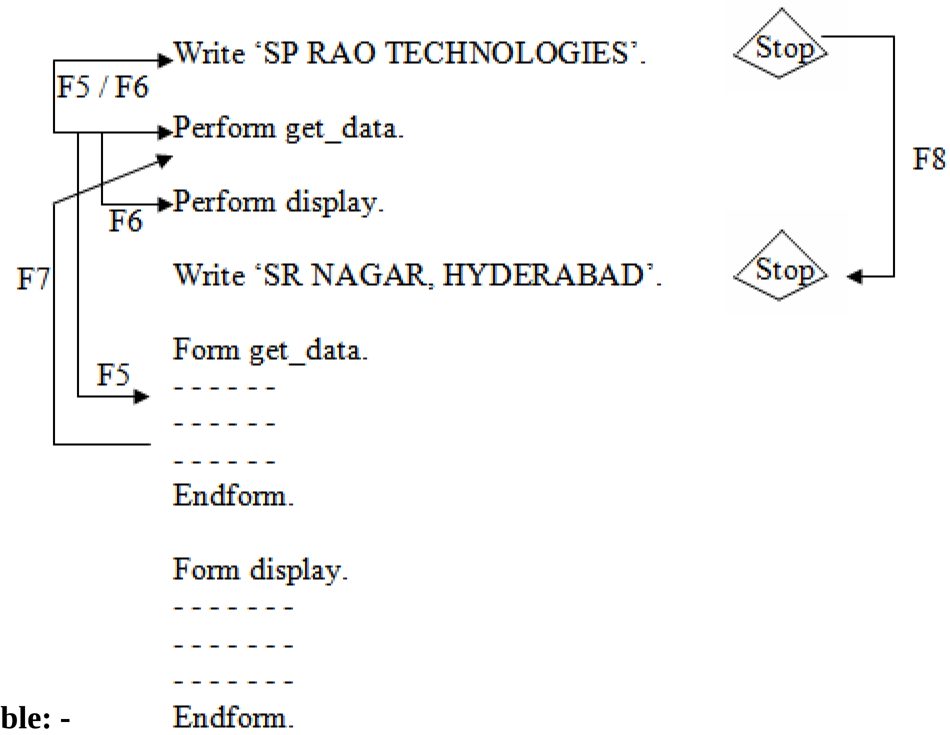
Note: - We can place up to 30 break points in the program. At the time of debugging mode F5 → line by line execution, F6 → At a time one block is executed (At a time subroutine & function module is executed), F7 → come of the block, F8 → first it'll check is there any other break points available or 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).



```
Write 'SP RAO TECHNOLOGIES'.

Perform get_data.

Perform display.

Write 'SR NAGAR, HYDERABAD'.

Form get_data.
- - - - - -
- - - - - -
- - - - - -
Endform.

Form display.
- - - - - - -
- - - - - - -
- - - - - - -
Endform.
```

Table: -

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.

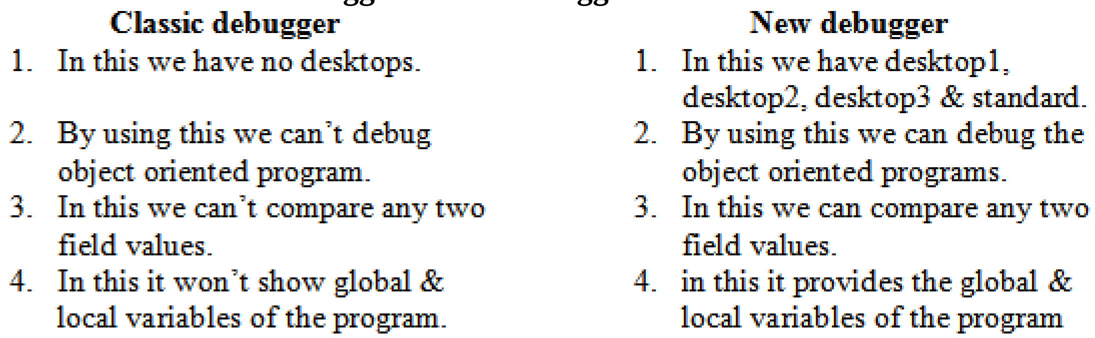
Break point / watch points: -

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	New debugger
1. In this we have no desktops.	1. In this we have desktop1, desktop2, desktop3 & standard.
2. By using this we can't debug object oriented program.	2. By using this we can debug the object oriented programs.
3. In this we can't compare any two field values.	3. In this we can compare any two field values.
4. In this it won't show global & local variables of the program.	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.

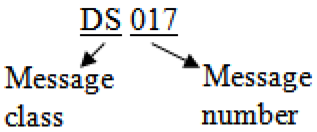
The following ways are used to identify the errors in standard program
1. By using where used list we identify the error location.
2. By using watch point.
3. By using break point.
4. By using source code scanner.
5. By using ABAP runtime analysis [SE30].
6. By using SQL trace [ST05].
When ever we execute 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.

<u>By using watch points:</u> -

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.

<u>By using break points:</u> -

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.

<u>By using source code:</u> -

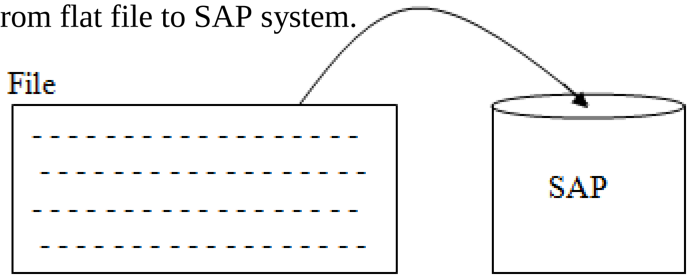
Execute SE93. Provide the transaction code. Click on display. Identify the package. Execute SE38. Provide the program name RS_ABAP_SOURCE_SCAN. Execute. Provide the package & provide string searched for (S017). Execute. it provide the all the locations where the message is available. Double click on 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.



Develop 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 the thousands of records, one way is execute the same transaction thousands of time. Another way is 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

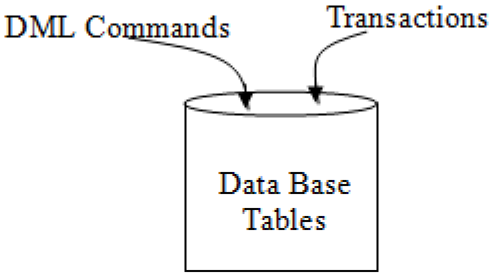
Note: – 1 → Create 2 → Change 3 → Display

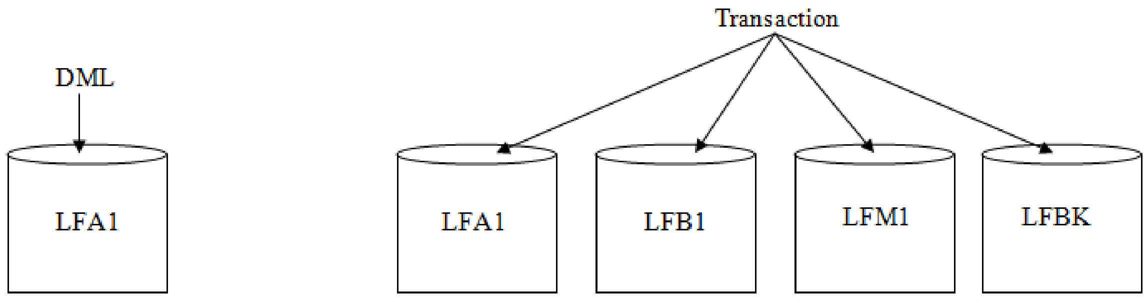
Steps to create a vendor : -

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, where as transaction code is used to update their relevant data base tables at a time.





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 filed 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 provide 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.

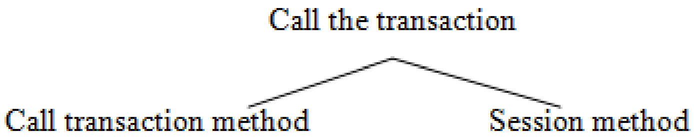
Step 4: -

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: -

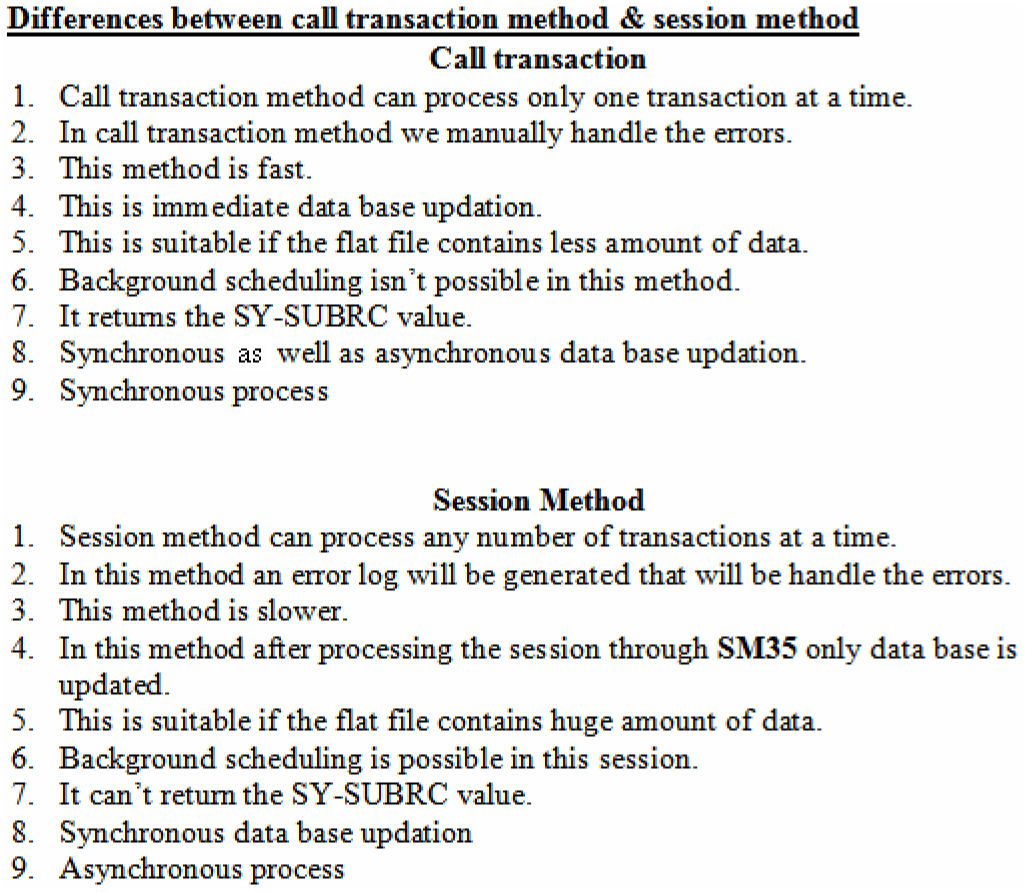
Call transaction '<Transaction code>' using <BDCDATA internal table> mode 'A/N/E'.
A → All screens
N → No screens
E → Error screens

→ 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.
  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 fie 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.
```

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 'SPRAO'.
Data B(20) type C value 'TECH'.
Data C(30) type C.
Concatenate A B into C separated by ' '.
Write C.
```

O/P → SPRAO TECH

Ex: -

```
Data A(2) type C.
Data R(20) type C.
A = 01.
Concatenate 'SPRAO(' A ')' into R.
Write R.
```

O/P → SPRAO(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> - - - - - -

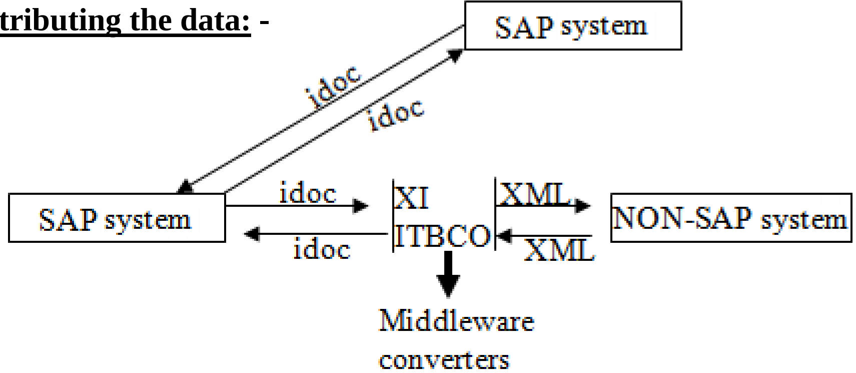
Ex: -

```
Data A(30) type C value 'SPRAO TECH'.
Data B(10) type C.
Data C(20) type C.
Split A at ' ' into B C.
Write:/ B, C.
```

CROSS APPLICATIONS

- Cross application is the concept to exchange the data among the systems.
- ALE is (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.

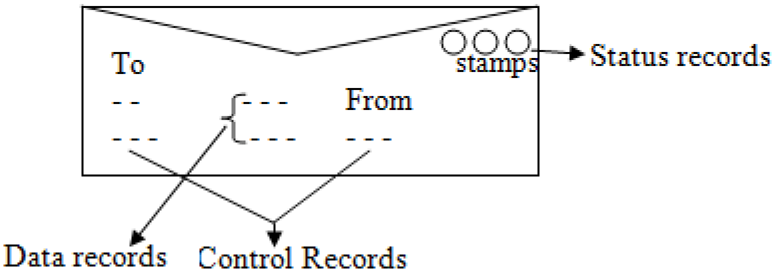
Distributing the data: -



- 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: -

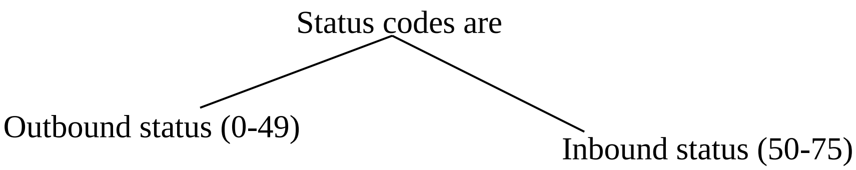
1. It generates a unique Idoc number, which is 16 digit.
2. It generates 3 types of records.
   i. Control records
   ii. Data records
   iii. Status records



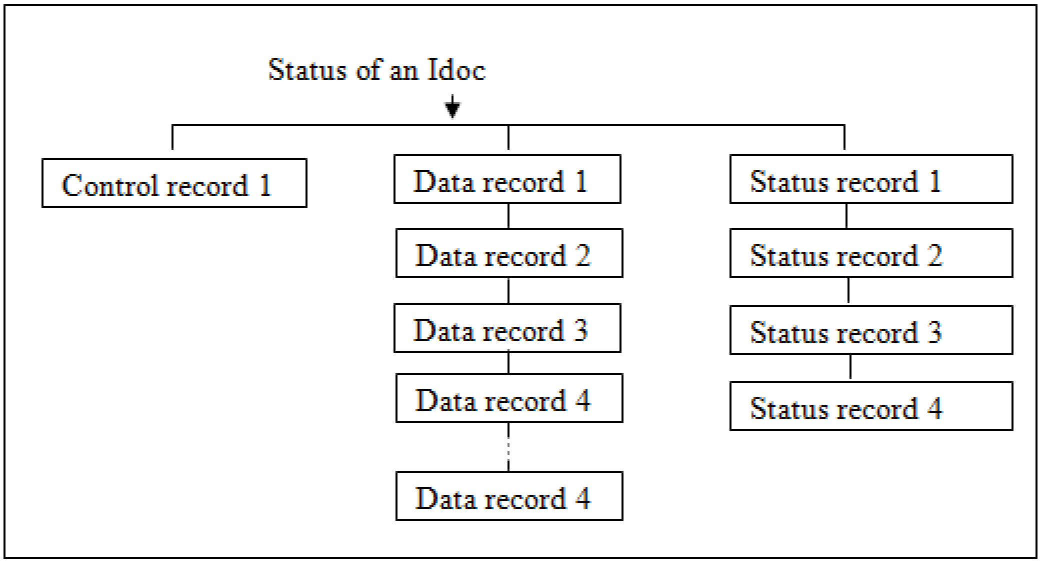
1. **Control record** : -
   i. Control record specifies the sender as well as receiver information.
   ii. It generates only one control record.
   iii. This information will be saved on **EDIDC** table

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 EDIC, EDIDD and EDIDS is the IDOC number.



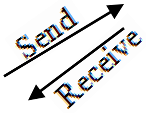
Types of IDOCS

1. BASIC type → Standard Idoc
   → Custom 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.

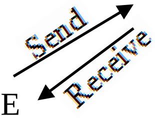
LFA1
LIFNR
NAME1
ORT01



Custom Idoc –

If you want to send as well as receives the custom table information then we go for custom idoc.

Zhai
EID
ENAME
ESAL



Extension Idoc –

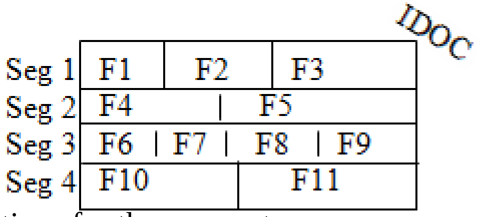
If you want to send as well as receives the additional fields information of standard data base table along with standard fields information then we go for extension Idoc.

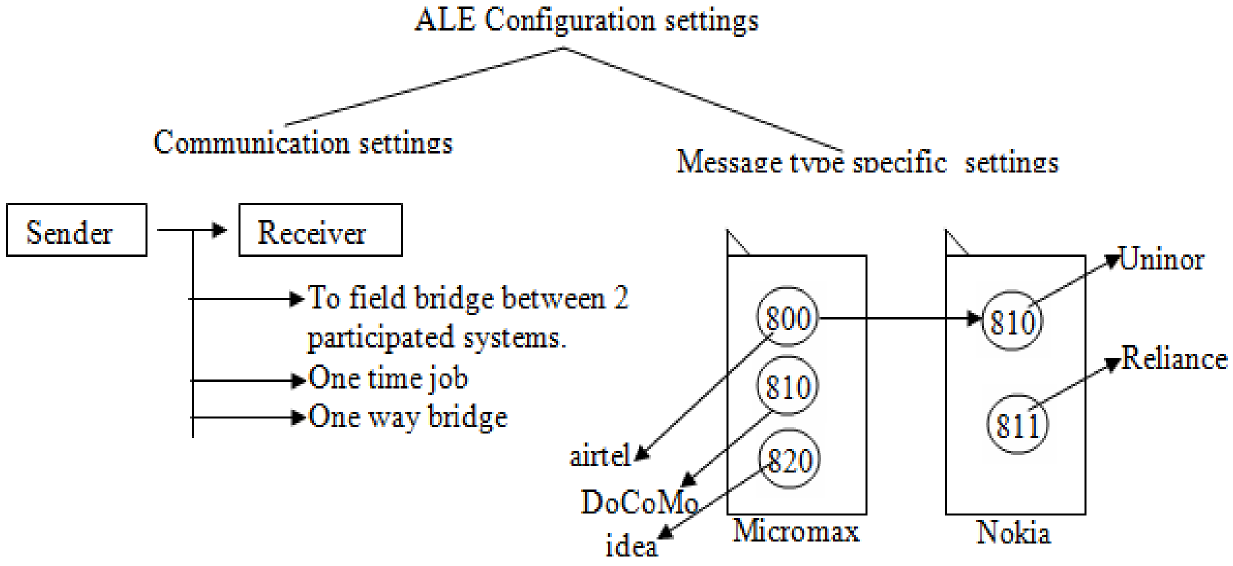
LFA1
LIFNR
NAME1
ORT01
||
+
ZHAI
EID
ENAME
ESAL

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 repeations for the segment.





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.

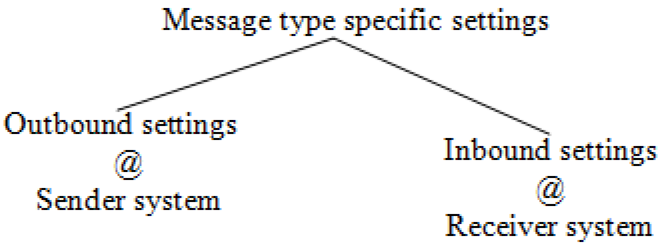
Steps to maintain RFC destination details: -

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.

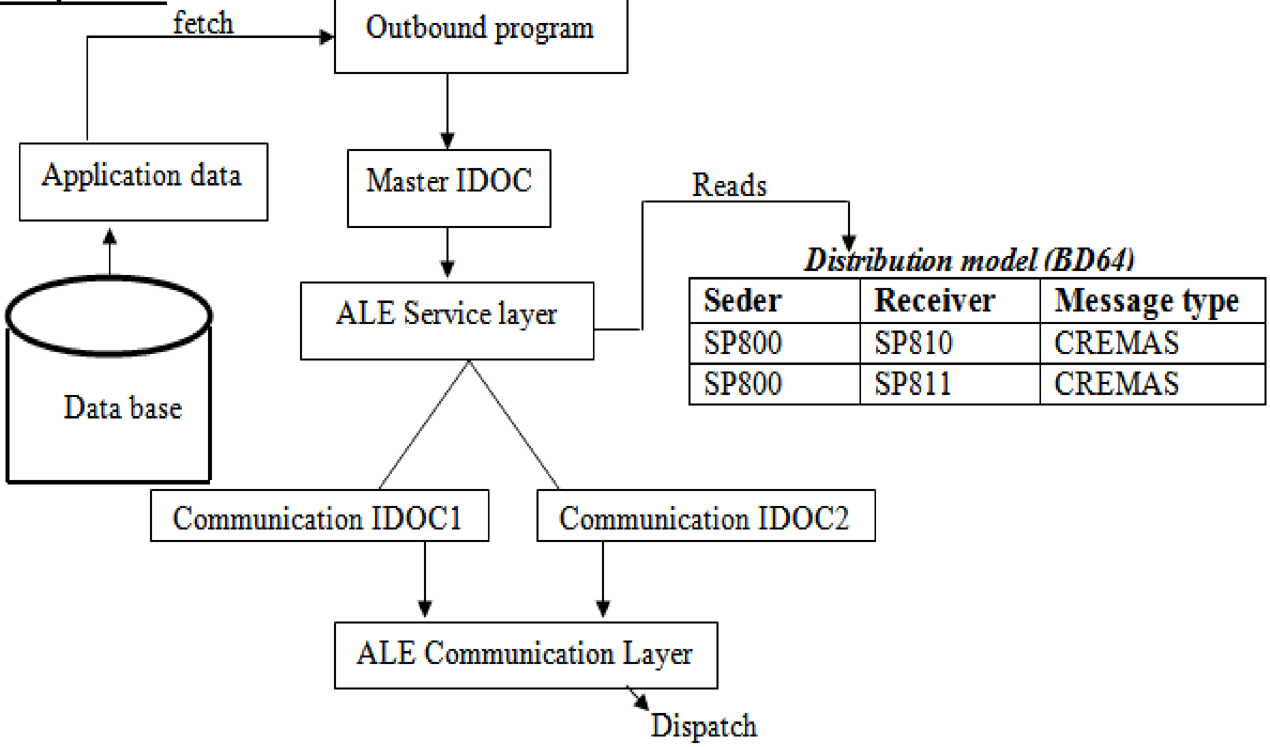
Client	:	810
User	:	SAPUSER
PW Status	:	is initial
Password	:	india123.

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.



Outbound process: -



Seder	Receiver	Message type
SP800	SP810	CREMAS
SP800	SP811	CREMAS

- ➢ 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.

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.

```
WA_CONT-MESTYP          =       ‘S930GRMAS’.
Append WA_CONT to IT_CONT.
```

Repeat the same steps for all the receivers.

<u>Step 4:</u> -

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.
```

<u>Steps to create TCODE:</u> -

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 (ZSPRAO_930AM_CUS_IDOC_OB). Select the GUI checkboxes. Save.

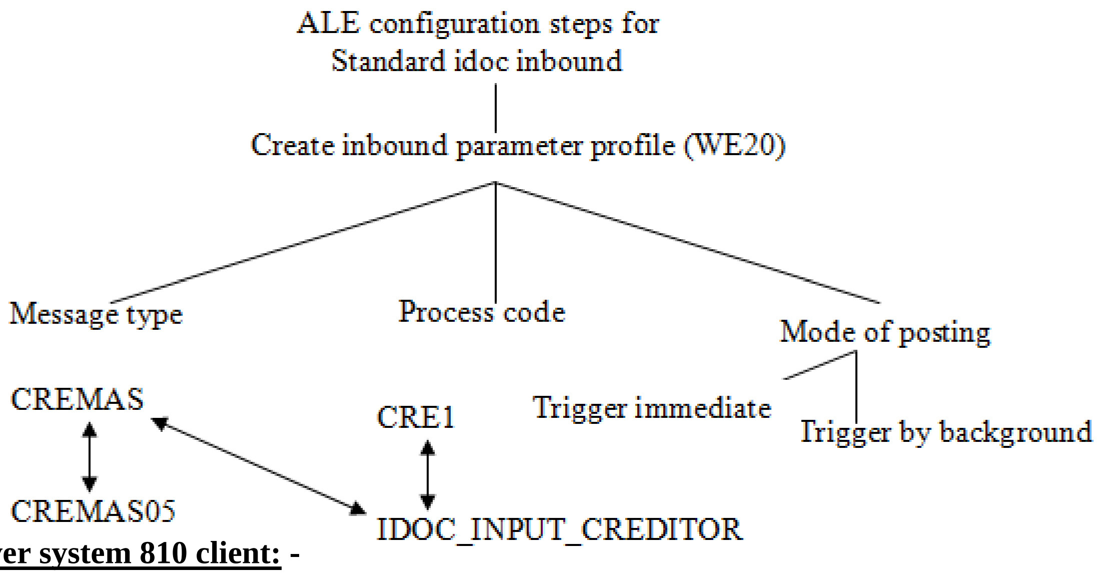
<u>ALE configuration steps for custom IDOC inbound:</u> -

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 programs 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).



<u>In receiver system 810 client:</u> -

<u>Steps to identify the existing function module:</u> - Execute 'WE42'. Click on position. Provide the known process code (CRE1). Double click on process code. Identify the identification as function module (IDOC_INPUT_CREDITOR).

<u>Steps to create function group:</u> -

Execute 'SE37'. In the menu bar click on go to → function groups → create group. Provide function group name (ZS930FGI). Provide short description (function group for idoc). Click on save.

<u>Steps to activate the function group:</u> -

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

<u>Steps to copy the function module:-</u> Execute 'SE37'. Click on copy in the application tool bar. Provide the from function module (IDOC_INPUT_CREDITOR), to function module (ZS930_IDOC_INPUT_GRTOR) & function group (ZS930FGI). Click on copy. Open our function module in change mode (ZS930_IDOC_INPUT_GRTOR). Click on source code tab. Remove the code until end function. Save, check, activate the function module.

<u>Steps to link the message type to function module & idoc:</u> -

Execute 'WE57'. Click on change mode, click on new entries in the application tool bar. Provide the function module name (ZS930_IDOC_INPUT_GRTOR). Select function type is (function module). Provide Basic type (ZS930GRI), message type is (S930GRMAS). Select the direction is inbound. Save.

<u>Steps to create mode of posting:</u> -

Execute 'BD51'. Click on new entries. Provide the function module name (ZS930_INPUT_GRTOR), select the input type 0. Click on save.

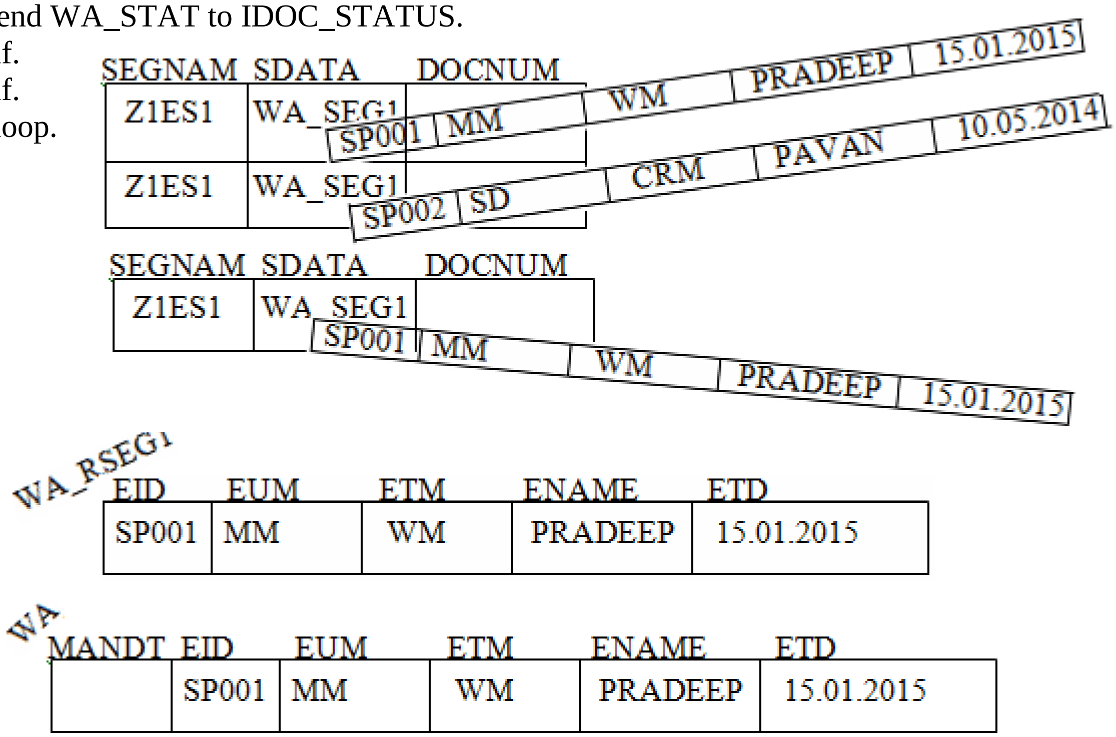
<u>Steps to create process code as well as link the provide code to function module:</u> -

Execute 'WE42'. Click on change mode. Click on new entries in the application tool bar. Provide the process code (ZGRP), short description (process code), provide identification as function module (ZS930_IDOC_INPUT_GRTOR). Select the radio button processing type as processing by function module. Click on save. Enter. Select the function module from the drop down (ZS930_IDOC_INPUT_GRTOR), save.

<u>Steps to create inbound partner profile:</u> -

Execute 'WE20'. Select the partner 'SP800'. Click on create inbound parameter. Provide the message type (S930GRMAS), process code (ZGRP), save. After the sender sends the idoc then it goes to receiver system & later it goes to inbound partner 'WE20'. In the inbound partner profile it check the 'S930GRMAS' message type is available or not. It's available. So it goes to process code 'ZGRP'.

```
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
Z1ES1	WA_SEG1	
Z1ES1	WA_SEG1	

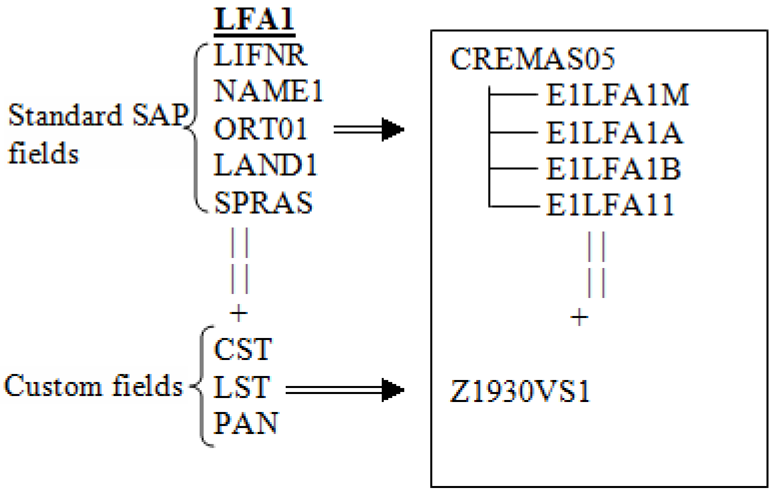
SEGNAM	SDATA	DOCNUM
Z1ES1	WA_SEG1	

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: -

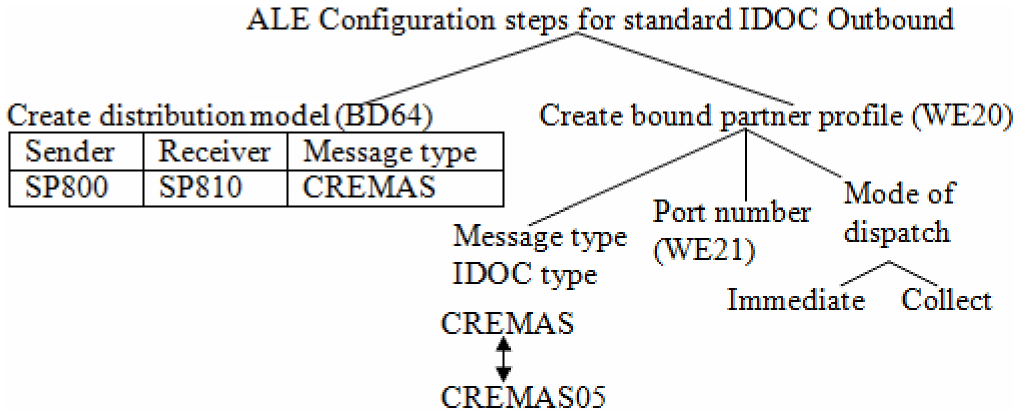
It's the collection of standard idoc + custom segment. As per client requirement if we add the CST number (Central Sales Tax), LST Number (Local Sales Tax), PAN Number to the LFA1table through append structure. If you want to send as well as receives the custom fields information along with standard 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.

Field	**Data element**	**Data type**	**Length**
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:-

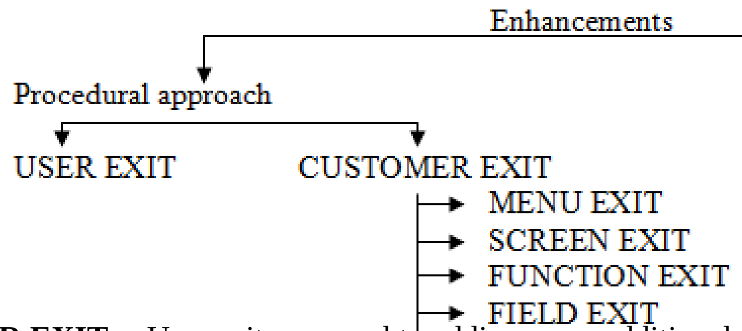
Execute WE20. Select the partner if the message type is already available then select message type. Click on detail. Provide the extension. Click on save.

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.



USER EXIT: - User exits are used to adding some additional functionality to the standard functionality is always through sub routines (form, end form).

CUSTOMER EXIT: - It's used to adding some additional functionality to the standard functionality is always through function modules.

Note: – Customer exit is either menu exit or screen exit or function exit or filed exit.

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 provide by SAP people).

Steps to work with screen exit: -

Identify the package of the transaction. Based on the package we identify the customer exits. 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. (RVEXAKK1).

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.

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.

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.

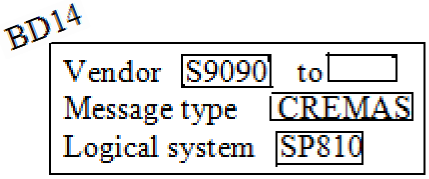
Steps to identify the right function exit based on the break point: -
Execute ‘CMOD’. Provide the project name (Z930VE). Click on create. Provide short description. Click on save. Click on enhancement assignments in the application tool bar. Provide the enhancement as customer exit names. Enter. Save. 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.

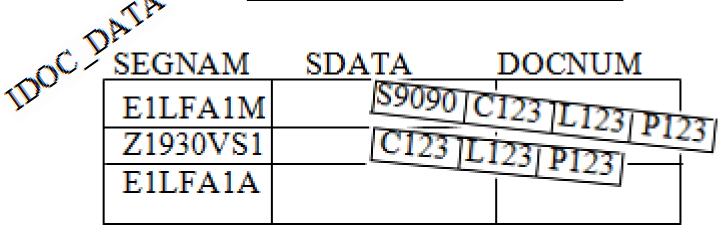
Syntax rules of an IDOC: -
1. The data for the segment must exist, if it specified as mandatory.
2. We shouldn’t exceed maximum number of repetitions for the segment.
3. The data for the segments must exist in the same physical sequence of the segments in the idoc.
4. The data for the child segment can’t 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.
```


BD14

Vendor	S9090	to	
Message type	CREMAS		
Logical system	SP810		


IDOC_DATA

SEGNAM	SDATA	DOCNUM
E1LFA1M	S9090 C123 L123 P123	
Z1930VS1	C123 L123 P123	
E1LFA1A		

```
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	

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: -

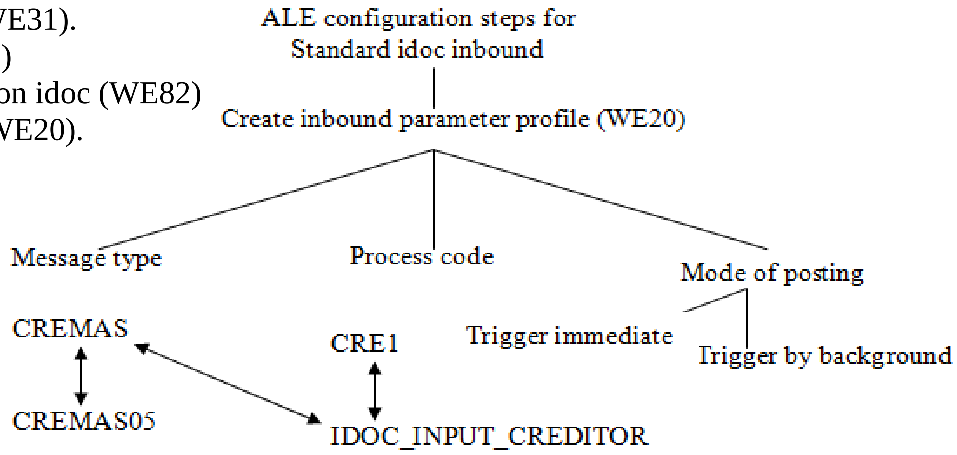
Execute BD14. Provide vendor number (S9090), message type (CREMAS), target system (SP810). Execute. Now it sends the standard segment information & custom segment information. Go to WE05/02 to absorb the all the segments.

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.



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 2nd 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.

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: -

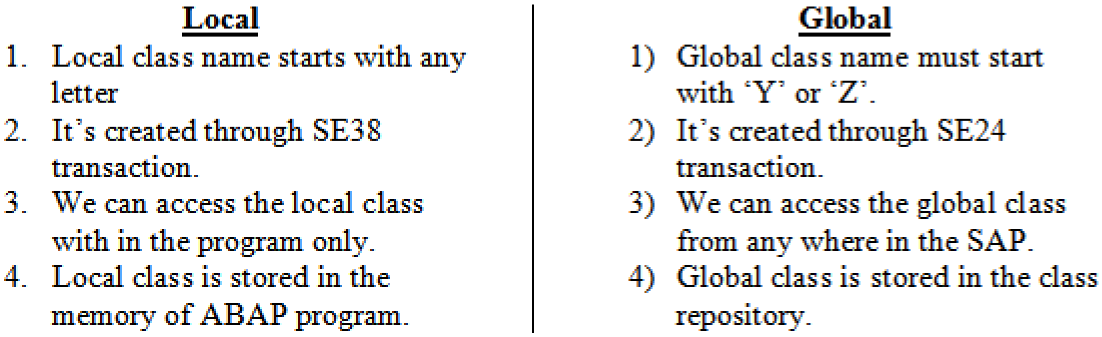
Class is the blueprint or template of an object. Object is the real one.

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	**Global**
1. Local class name starts with any letter	1) Global class name must start with 'Y' or 'Z'.
2. It's created through SE38 transaction.	2) It's created through SE24 transaction.
3. We can access the local class with in the program only.	3) We can access the global class from any where in the SAP.
4. Local class is stored in the memory of ABAP program.	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.

Events: - 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>

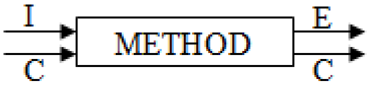
Syntax of creating the object base on reference: -

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

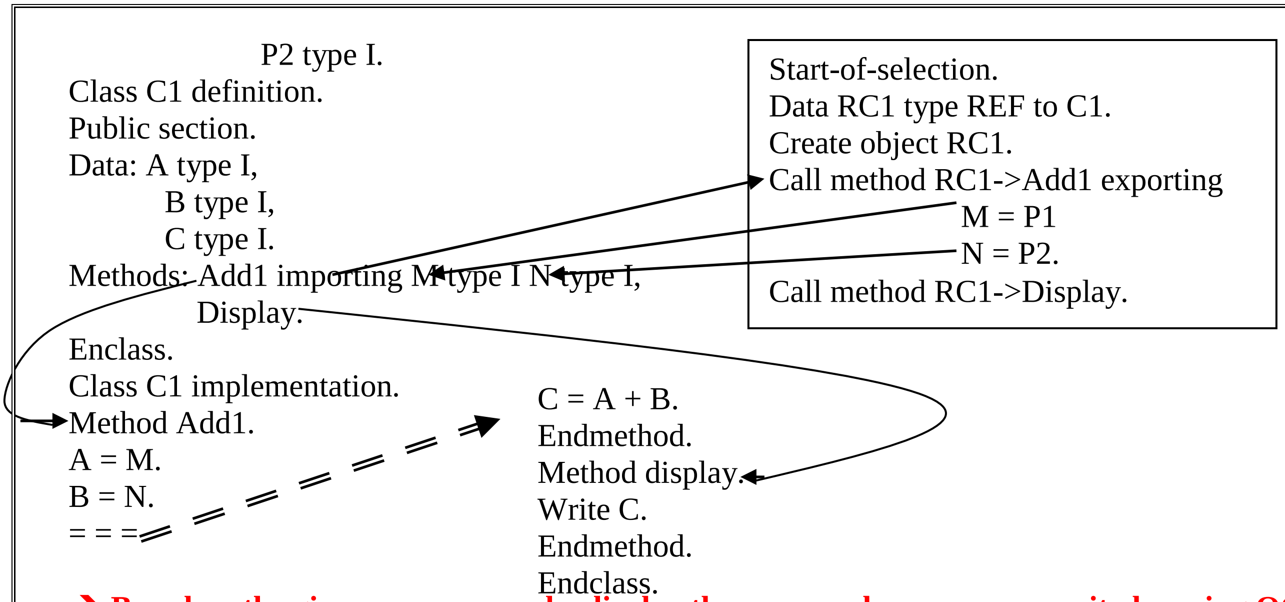
```
Method <interface>~<method>.
    - - - -
    - - - -        logic
    Endmethod.
```

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.
```

```
Start-of-selection.
Data RC1 type REF to C1.
Create object RC1.
Call method RC1->Add1 exporting
                 M = P1
                 N = P2.
Call method RC1->Display.
```

➔ **Based on the given company code, display the comp code, comp name, city by using OOABAP.**

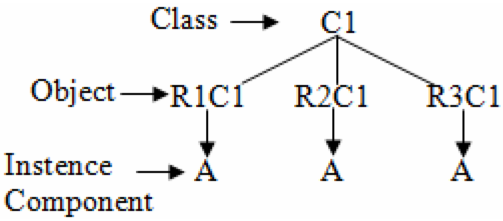
```
PARAMETER P_BUKRS TYPE T001-BUKRS.
DATA: BEGIN OF WA_T001,
      BUKRS TYPE T001-BUKRS,
      BUTXT TYPE T001-BUTXT,
      ORT01 TYPE T001-ORT01,
      END OF WA_T001.
CLASS C1 DEFINITION.
  PUBLIC SECTION.
    METHODS: GET_DATA IMPORTING
              I_BUKRS TYPE T001-BUKRS,
             DISPLAY.
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.



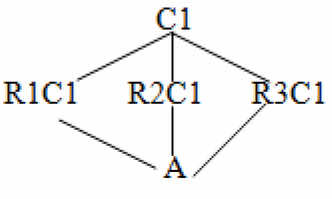
ii. Instance components are declared with

a) Data
b) Methods
c) Events
d) 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

a. Class-Data
b. Class-Methods
c. Class-Events
d. Class-Interfaces

iii. Static components always accessed through class object or class name.

```
CLASS C1 DEFINITION.
  PUBLIC SECTION.
    DATA A TYPE I VALUE '20'.
    DATA B TYPE I VALUE '30'.
ENDCLASS.
START-OF-SELECTION.
  DATA RC1 TYPE REF TO C1.
  CREATE OBJECT RC1.
  WRITE RC1->A.
  WRITE RC1->B.
  WRITE C1=>B.
```

➔ Manually filling the company code, company name & city internal tables by using OOABAP.

```
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.
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.
```

<u>Inheritance:</u> - Inheritance is used to create a new class based on existing class. 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 the all the components at super class which are defined under public or protection 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.

<u>Syntax of defining the subclass:</u> -

```
Class <sub class name> definition inheriting from <super class name>.
Public / protected / private section.
- - -  ⎫
- - -  ⎬   components.
- - -  ⎭
Endclass.
```

➔ **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.
```

➔ 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 (ZSPRAO_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.

Parameter	Type	Typing message	Associated type
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_EBLEN.
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 (ZSPRAO_10AM_GC1). Enter.

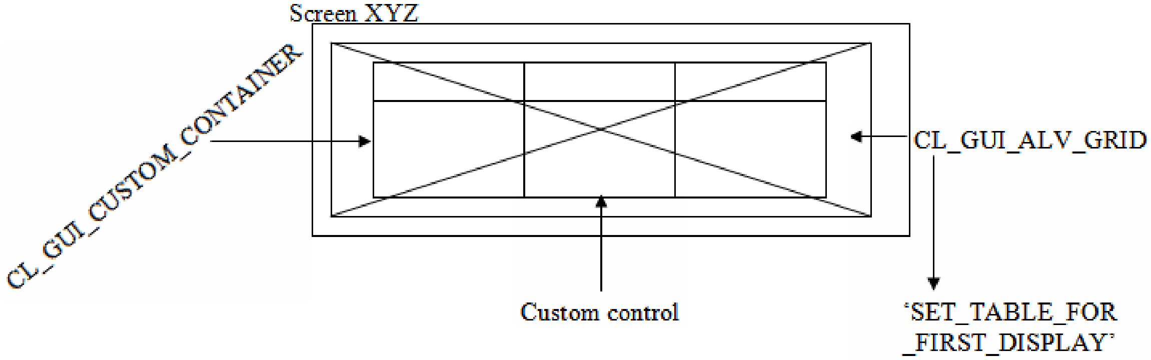
Note: - Constructer is the one special type of method in the class. We no need to call the constructer extensity by using call method. When ever we create the object for the class then automatically constructer will be triggered & asked the input output parameters.

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

OOPS ALV

When ever we are working with OPPS 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.



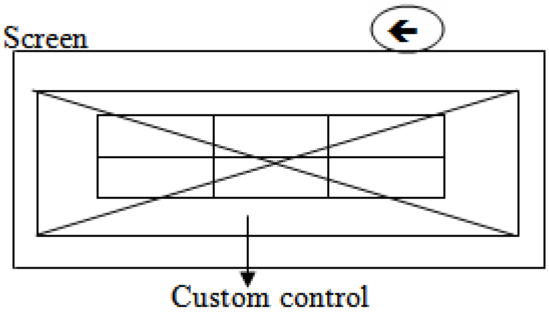
CL_GUI_CUSTOM_CONTAINER: - The Global class which refers the custom control screen element.
CL_GUI_ALV_GRID: - The global class which refers the ALV Grid.
SET_TABLE_FOR_FIRST_DISPLAY: - This is the Global method under grid class which is used to display the output in an OOPS ALV. The input for the above 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:-

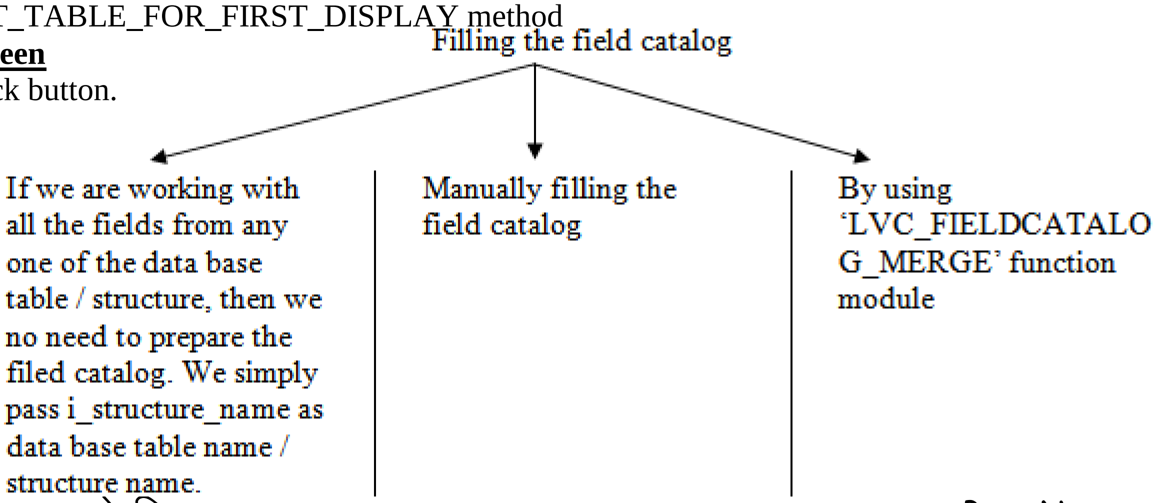
1. Create the selection-screen / input fields.
2. Declare the data internal table and field catalog internal table
3. Create the reference to the container and grid class start-of-selection.

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

<u>Some of the fields in field catalog:</u> -

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.
```

<u>Flow logic of 2000 screen</u>

```
PROCESS BEFORE OUTPUT.
 MODULE STATUS_2000.
```

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’
                                ├→<Field name>
                                ├→‘X’
                                └→<Data IT>
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.

```
        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

<u>Abstract class:</u> -

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.
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 'SPRAO 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.
```

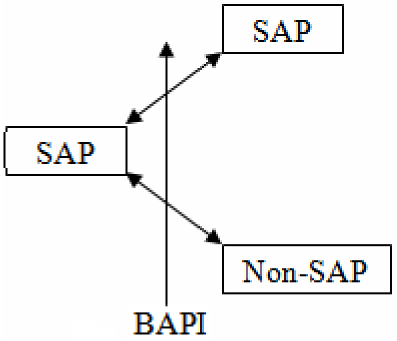
BAPI

(Business Application Programming Interface)

BAPI's are used to connecting from SAP to SAP as well as SAP to NON-SAP.

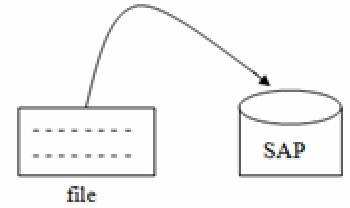
BAPI's are defined as an API method (Application Programming Interface) of SAP objects. These methods & their objects are stored as well maintained in BOR (Business Object Repository). BOR taking care about version change management (when ever the version is changed if any changes required in BAPI those changes is done by BOR).

BAPI's are also called as one of the remote enable function module.



BAPI

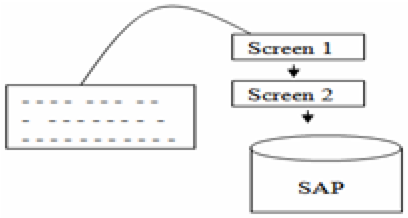
1. BAPI is used to upload the data from file to SAP system directly.



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.

<u>Some of the important points related to BAPI:</u> -

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 written those errors through written parameter.
6) BAPI must not contain dialogue programs.
7) BAPI mustn't contain Submit reports.

<u>Steps to create our own custom BAPI</u>

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	☑

Export

Parameter	**type**	**associated type**	**pass by value**
E_WA	TYPE	ZBAPI_10AM_STR	☑
RETURN	TYPE	BAPIRET2	☑

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 hierarchical tab. Expand the module. Expand the application (sales). Expand the object. Double click on method. Identify the BAPI in right side.

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 crate, 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.
```

<u>Steps to create the bank details manually:</u> -
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.

<u>Steps to update the bank details manually:</u> -
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.
```

```
    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 connected 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.
```

WA_BANK

MCTRY	BKEY	bank name	Reginon	Street	city
IN	54520	CITY Bank	1	KOTI	HYD

IT_BANK

CTRY	BKEY	bank name	Reginon	Street	city
IN	54520	CITY Bank	1	KOTI	HYD
IN	54522	AXIS Bank			BAN
IN	54523			D.NA	HYD

WA_ADD

Bank name	Reginon	Street	City
CITY Bank	1	KOTI	HYD

```
        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.
```

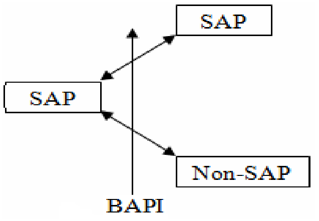
Enhanced BAPIs: -
Based on the client requirement if you add additional fields to the standard data base table. If you want to update these fields information by using BAPI then we go for BAPI enhanced.

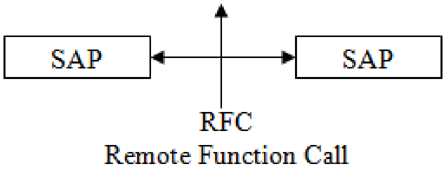
Steps to work with enhanced BAPI: -
Open the BAPI n SE37 & identify the extension in parameter & double click on their associated type & add our additional fields to the structure by using append structure.

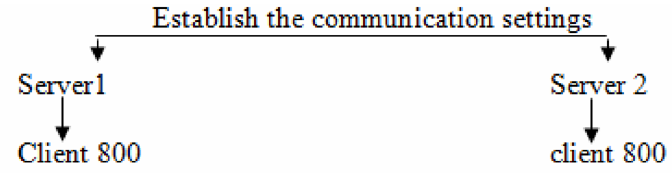
At the time of calling the BAPI we need to pass the additional fields information. Those information is automatically updated into standard data base table.

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: – BAPI is used to connecting from SAP to SAP as well as SAP to NON – SAP. Where as RFC is used to connecting from SAP-SAP only.







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 on wards. BADI contains 2 sections.

1. BADI Definition
2. BADI Implementation

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**.

<u>BADI Implementation:</u> - 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)	**BADI**
1. We can implement the customer exit only once.	1. We can implement the same BADI any number of times.
2. Customer exit is the procedural approach. So it takes some extra time to enhancing.	2. BADIs are object oriented approach. So it takes very less time to enhancing.
3. In the screen exit the screen number is fixed which is provide by SAP.	3. In the screen BADI we provide our own screen number.
4. By using CMOD transaction we can implement the customer exit.	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.

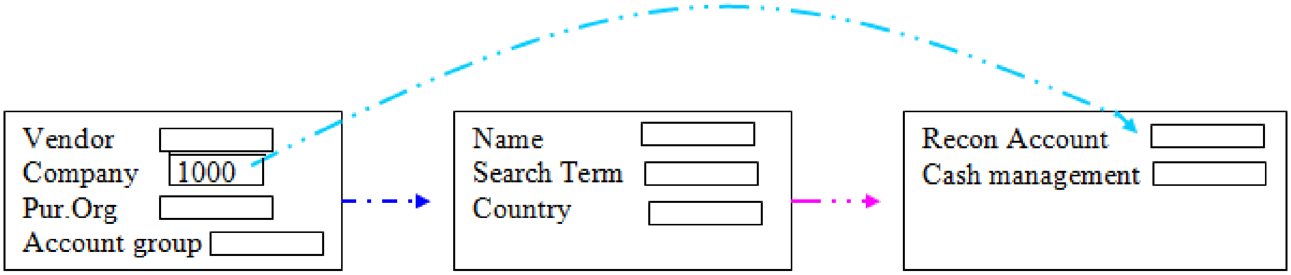
<u>Some of the scenario</u>: -

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.

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 **SPRO**. Click on SAP Reference IMG. Expand the Logics-General (If we want to customer or vendor BADIs). Expand the business partners. Expand the vendors. Expand the control. Expand the adoption of customers own master data fields. Identify the BADIes. OR click on find function key. Provide the search term as BADI. Enter. Identify the all BADIs.

→ 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 BADIs 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 provide 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.
```

<u>Steps to identify the all the implementations of BADI: -</u>

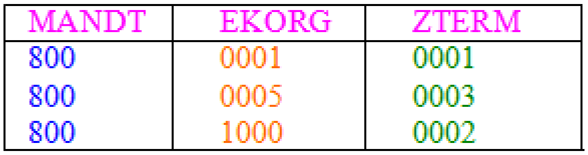
Execute '**SE18**'. Provide the BADI name (VENDOR_ADD_DATA). Click on display. In the menu bar click on implementation → Display. Identify the all the implementations of BADI. Yellow color Badis are active and Blue color Badis are inactive.

➔ 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)

<u>Table Name:</u> (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.
```

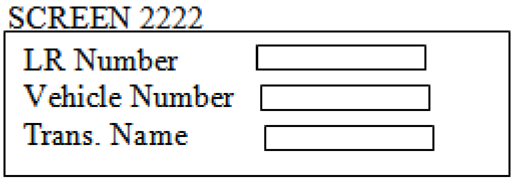
<u>Steps to work with Screen BADI: -</u>

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

Vehicle Number

Trans. Name

<u>Steps to create module pool program & screen: -</u>

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'.
```

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 'CUSTOMER IS NOT BELONGS TO THIS
PLANT'.
    ENDIF.
  ENDIF.
```

MB1A WITH 201 Movement type

BADI Name : MB_QUAN_CHECK_BADI

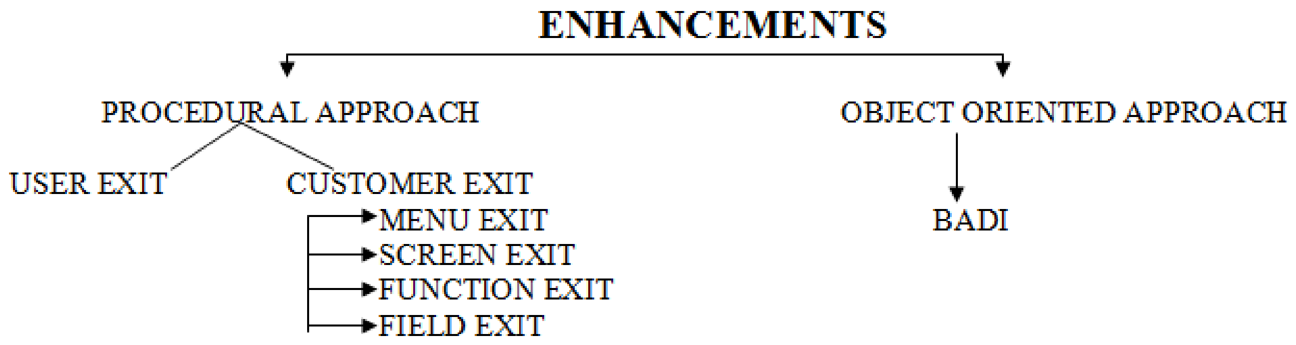
Method Name: CHECK_ITEM_DATA

```
  IF IS_MSEG-BWART = '201'.
    IF IS_MSEG-WERKS <> IS_MSEG-KOSTL+0(4).
      MESSAGE E000(ZMESSAGE1) WITH 'CUSTOMER 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.



USER EXIT: - User exit is nothing but adding some additional functionality to the standard functionality is always through sub routines (form, end form). In the real time most of the times we use user exits in SD

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: -

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 3: -

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 4: -

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 SPRO. Click on SAP reference IMG. Expand sales and distribution. Expand system modification. Expand user exits. Expand user exits in sales. Click on documentation of user exits in sales document processing.

→ 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: -

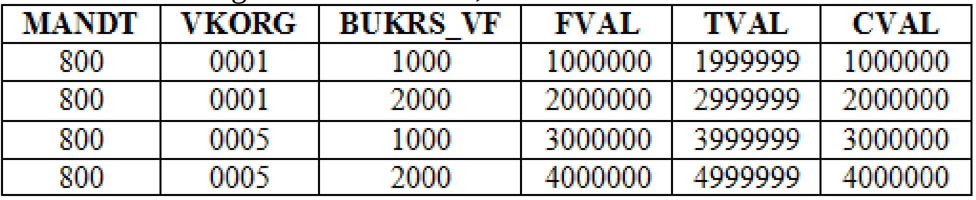
Execute SE38. Provide the program name MV45AZZ. Click on display. Click on find function key. Provide the user exit name (USEREXIT_DELETE Document). Enter. Double click on form or user exit. identify the place where we implement the logic. Click on enhance ( ) symbol in the application tool bar. In the 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 ranger 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.

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.

Customer exits: -

Customer exit is nothing but adding some additional functionality to the standard functionality is always through function module. Customer exit is either menu exit or screen exit or function exit or field exit.

Note: - Customer exits are always identified through package of transaction.

Steps to implement the customer exit: -

Identify the package at the transaction (SE93). Based on the package identify the customer exits (SMOD transaction). Identify the right customer exit based on the short description or break point. Implement the customer exit through project (CMOD transaction).

Menu exit: - Menu exits are used to adding some additional menus to the standard GUI. Menu exits are not possible for all transaction codes.

Some of the standard transaction codes which contains menu exit: -

VX11: - Create financial document
CO01: - Create production order
MC94: - Change flexible LIS planning

➔ **Implement the menu exit for the standard transaction VX11.**

Steps to identify the package: -

Execute SE93. Provide the transaction code VX11. Click on display. Identify the package (VEI).

Steps to identify the customer exits based on package: -

Execute SMOD. Click on find function key. Provide the package (VEI). Execute. Identify the all the customer exits.