DDIC

- SQL is divided into two parts.
  DDL →Create/Alter/Drop
  DML→ Insert/Select/Update/Delete
- **DDL** part of SQL is handled by ABAP Dictionary in SAP.

Transparent Table

- It is one-to-one relation table i.e., when one transparent table is created in ABAP then exactly same table structure will be created within the database.

Delivery Class

- It determines the table type.
  A→ Application Table (Master & Transaction Data)
  C→ Customer Data (Maintained by Customer)
  L→ Table for storing temporary data.
- It is used for controlling data transport of the table during installation, upgrades and transporting between customer systems.

Data Class

- It determines the data type of the table.
  APPL0→ MASTER DATA → Rarely Changed
  APPL1→ TRANSACTION DATA → Frequently Changed
  APPL2 → ORG. DATA → Data defined during the system installation.

Data Element → Semantic Definition of the field.

Data Domain → Technical Info like datatype, length, fixed values.

Value Table

- It is a table containing all the possible value for a field and used for providing F4 Help for a field.
- The main advantage of this is to automate the system for Foreign Key relationship i.e., system will automatically display the check table name to generate the Foreign Key Proposal.

Check Table → Table which contains all valid entries of a particular foreign key table field.

Search help

1. **Elementary SH**→ Used to display SH from Single Table & provides single search path for a field.
2. **Collective SH**→ It is used to display multiple search path for a field from multiple tables.
   It consists of more than one elementary search help.

- Import Parameters: Input parameters that are passed to the search help when it is called.
- Export Parameters: These are the output parameters that are returned by the search help after it has been executed. They contain the selected values that are used to populate the input field.
- SPos (Selection Position): Position of field in search help restriction screen. When you press the F4 button, the selection screen will be displayed with fields to which Search-help is attached. At that time, the order of the field in the selection screen was determined by the SPOS.
- LPos (List Position): Position of field in search help results report/table. when our field data is displayed in the F4 help that time it will be based on the LPos position.
- **Proposal Search for input fields**: Option to display possible values based on input when user starts typing.

Reference Field

- It is a special field which provides the context of another field's data.
- It is important for field storing Currencies and Quantities.
- Imagine a field storing the value "100". Without a reference field, it's unclear if it's 100 dollars or 100 kilograms. The reference field clarifies the meaning.

View

- It is a virtual/logical table which is created by grouping columns of one or more database tables.

1. **Database View**
    - It is a type of view in which we have only read operation access.
2. **Maintenance View**
    - It is a type of view in which we have both read and write operations access.
3. **Help View**
    - Used in search help to display search path for a field by combining multiple tables.
4. **Projection View**
    - It is a type of view which used to display only few fields of a single table.

Difference B/w Structure and Workarea

- Both are user defined datatype which represents single row of an internal table.
- Structure is global datatype which can be used across all the SAP Programs. Whereas Workarea is a local datatype which we create within a program.

TMG → It is a tool in SAP to maintain the data in the table.

Lock Object

- It is a mechanism offered by SAP to synchronize access to the data and prevent inconsistencies when multiple programs try to modify the same data in a database table.

1. **Read/Shared Lock (S)**→ This lock allows read-only access to the object shared.
2. **Write/Exclusive Lock (E)**→ This lock allows neither read nor write access to shared objects by other transactions or users.

Initial Values marker IN SAP TABLE

From <https://techcommunity.microsoft.com/t5/running-sap-applications-on-the/confusion-around-initial-values-marker-in-sap-table-definition/ba-p/367492>

Select-Option

Select-option is an **internal table with header line**. It consists of four parameters.

1. SIGN → I / E (I = Include E = Exclude)
2. OPTION → Relational Operators (EQ, BT, LT, GT)
3. LOW → Low Value
4. HIGH → High Value

MODULARIZATION

- It enhances the readability and understanding of the codes.
- It makes easier to maintain the codes and promotes reusability of the code.

1. Macros

- If we want to reuse the same set of the statements more than once in a program, then we need to put those statements inside the macro.
- Macro definition should occur before the macro is used in the program.

2. Subroutine

- It is a processing block which we define in the program.
- Parameters are optional in subroutine.

Internal Subroutine→ Defined and Used in the same program.

PERFORM <subroutine_name> **TABLES** <itab_name> **USING** <parm1> <parm2> **CHANGING** <parm1>

External Subroutine → Created in one program and used in another program.

PERFORM <subroutine_name> **IN PROGRAM <**another_program> **USING** <parm1> <parm2> **CHANGING** <parm1> **IF FOUND.**

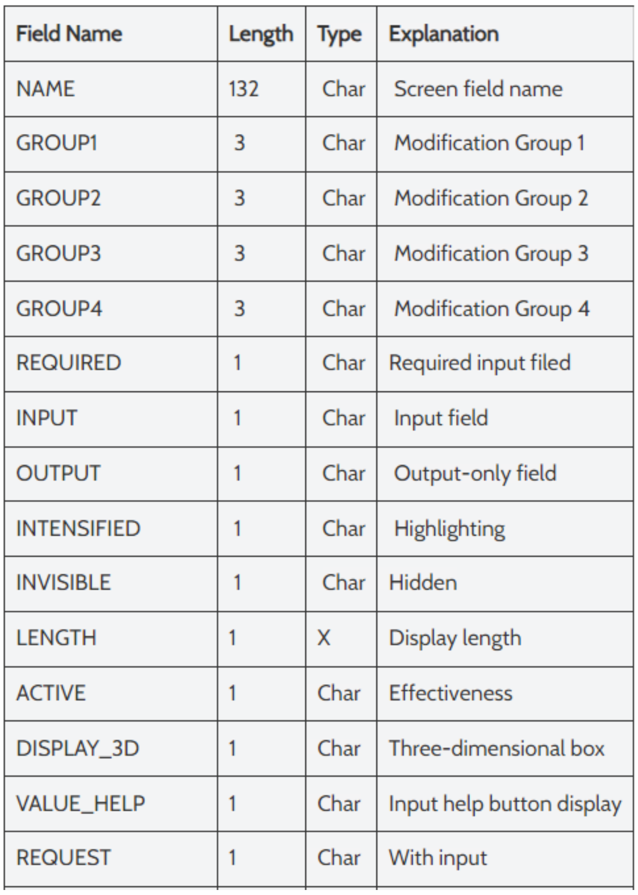
3. Function Module

4. Include Program

SELECTION SCREEN

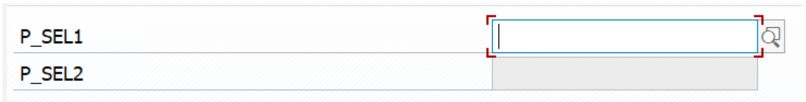
INITIALIZATION BEFORE THE SELECTION-SCREEN IS NOT BUILT

If we want to dynamically change the selection screen before it is displayed, we have to describe it in the initialization event.



Field Name	Length	Type	Explanation
NAME	132	Char	Screen field name
GROUP1	3	Char	Modification Group 1
GROUP2	3	Char	Modification Group 2
GROUP3	3	Char	Modification Group 3
GROUP4	3	Char	Modification Group 4
REQUIRED	1	Char	Required input filed
INPUT	1	Char	Input field
OUTPUT	1	Char	Output-only field
INTENSIFIED	1	Char	Highlighting
INVISIBLE	1	Char	Hidden
LENGTH	1	X	Display length
ACTIVE	1	Char	Effectiveness
DISPLAY_3D	1	Char	Three-dimensional box
VALUE_HELP	1	Char	Input help button display
REQUEST	1	Char	With input

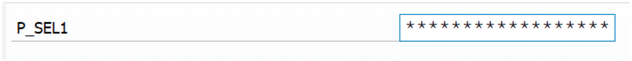
- The name defined by Parameter and Select-Options is stored in **SCREEN-NAME**
- **MODIFY SCREEN** is keyword which is used to apply screen modification.
- **SCREEN-REQUIRED** - By setting SCREEN-REQUIRED to ‘1’ as described above, the selection item becomes a mandatory input, and the same effect as when the OBLIGATORY option is present.
- **SCREEN-INPUT** - setting SCREEN-INPUT to ‘1’ enables input of the selection item (default) and setting SCREEN-INPUT to ‘0’ disables input.



P_SEL1

P_SEL2

- **SCREEN-INTENSIFIED** - When SCREEN-INTENSIFIED is set to ‘1’ and a value is entered from the screen, it is displayed in red.
- **SCREEN-INVISIBLE** - When SCREEN-INVISIBLE is set to ‘1’, the input field becomes *.



P_SEL1 ******************

- **SCREEN-LENGTH -** The number of displayed digits can be set in SCREEN-LENGTH.
- **SCREEN-ACTIVE -** When SCREEN-ACTIVE is set to ‘0’, the input field disappears. However, note that the selected text is displayed, so the line itself does not disappear. If you want to hide everything, specify the MODIF ID and set.

- **MODIF ID:** MODIF ID is a three-character id, we can process a screen elements group using this MODIF ID, this will be stored in SCREEN-GROUP1.

overview of processing events that apply to selection screens

```
INITIALIZATION.

AT SELECTION-SCREEN OUTPUT.

AT SELECTION-SCREEN ON HELP-REQUEST.

AT SELECTION-SCREEN ON VALUE-REQUEST.

AT SELECTION-SCREEN.

START-OF-SELECTION.

END-OF-SELECTION.
```

- **AT SELECTION-SCREEN OUTPUT** – Used for modifying selection screen like hiding or disabling specific fields based on user selections or program logic.
- **AT SELECTION-SCREEN ON <field>** → To Validate a Single Input Field and If we enter any wrong data, then only this input field will be highlighted, and remaining fields will be disabled.
- **AT SELECTION-SCREEN→** This event will trigger on user command & to Validate multiple Input Fields. Error field will be highlighted, and remaining fields will be enabled.

Diff. B/w Classical Report & ALV Report

- **Classical reports** are the most basic type of report in SAP ABAP. The output of a classical report is typically static and cannot be manipulated by the user. Classical reports are developed using ABAP statements like `WRITE` to control the output.

- **ALV reports** provide features like Sorting, Filtering, Exporting data to different formats like Excel. It is allowing users to interact with the data.

ALV – ABAP LIST VIEWER

Features of ALV

- We can sort the data in ascending or descending order.
- We can export the data.
- We have the filter data.
- We have inbuilt PF-Status bar.

Steps to Create ALV

1. Create a FieldCatalog
   FieldCatalog is a table type which contains information on the fields to be displayed on the ALV output.
   (a) Use SAP FM **REUSE_ALV_FIELDCATALOG_MERGE** to create fieldcatalog.
   (b) Manually create fieldcatalog.

2. Bind data with FieldCatalog – We have to push data to the fieldcatalog.
   (a) REUSE_ALV_LIST_DISPLAY
   (b) REUSE_ALV_GRID_DISPLAY

Difference b/w REUSE_ALV_LIST_DISPLAY & REUSE_ALV_GRID_DISPLAY

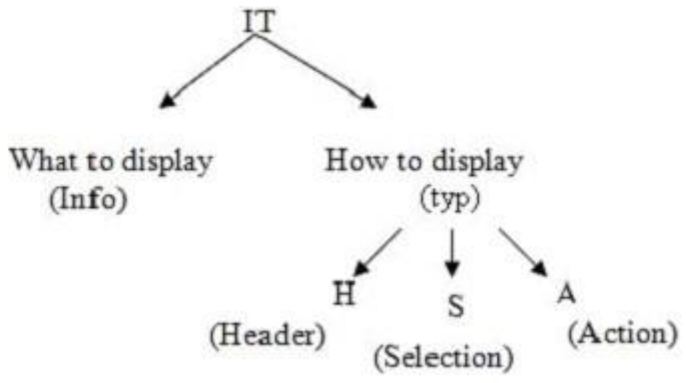
- Output List of LIST_DISPLAY is static, however output list of GRID_DISPLAY is dynamic which means that we can expand the width of each column and we can drag & drop the columns from their respective position.

REUSE_ALV_COMMENTARY_WRITE

It is the FM which is used to display the text in top or bottom events in ALV Report.

Various options available in this FM.

- **H (HEADER)** → contents of info will display in bold, followed by empty line & key is ignored.
- **S (SELECTION)** → info will be bold, key will be normal, followed by empty line.
- **A (ACTION**) → info will display in normal font & key is ignored.



OOALV

- We can only create ALV Grid using Object Oriented concepts.

1. Create A FieldCatalog
   - Using SAP FM **LVC_FIELDCATALOG_MERGE**
   - Manually create a fieldcatalog
2. Create a **Custom Control** (Layout Element) on a **screen**.
3. Create an object of the Container Class (`cl_gui_custom_container`) and pass the name of **custom control** in the container_name parameter.
4. Create an object of the ALV Grid Class (`cl_gui_alv_grid`) and pass the object of the container class in the parent parameter.
5. Use method **SET_TABLE_FOR_FIRST_DISPLAY** of the ALV Grid Class (`cl_gui_alv_grid`) to bind the data to the fieldcatalog.

- **GET_SELECTED_ROWS** → Method of Grid Class Return Indexes of Selected Rows.
- We have a **DOUBLE_CLICK** & **HOTSPOT_CLICK** Event in the ALV Grid Class `cl_gui_alv_grid` to handle double click & hotspot click for which we have to implement Event Handler Method and register it.
- **TOOLBAR** event for assigning custom button on ALV Grid Toolbar and **USER_COMMAND** event to provide response on user interaction with ALV Grid.

ALV BY CL_SALV_TABLE (Imp from Interview Perspective)

- **Factory Method**.
- It is a part of ALV Object Model.
- **ALV Object Model** is an encapsulation of the pre-existing ALV Tools which means CL_SALV_TABLE class **combines both CL_GUI_ALV_GRID** class for container implementation and **REUSE_ALV_GRID_DISPLAY & REUSE_ALV_LIST_DISPLAY** FM for full screen display.
- **Disadvantage:** *We cannot go for **Edit ALV** functionality using ALV Object Model***.**
- No need to create FieldCatalog**.**

1. Call the **Factory Method** of CL_SALV_TABLE to get the instance of ALV Table Object.
2. Call the **Display Method** of CL_SALV_TABLE to display the ALV.

- **GET_FUNCTIONS**→ Returns Object of Functions Class which contains methods to enable Standard Function Buttons of Full Screen Display of ALV Table Output.
  SET_ALL→ Method of Functions Class which activates All Generic ALV Functions of Application Toolbar of ALV.
- **SET_SCREEN_STATUS** → Method of class CL_SALV_TABLE to assign the Custom GUI Status.

- **GET_EVENT** → Returns object of Event Class which contains event double-click, user-command.
- FM **GET_GLOBALS_FROM_SLVC_FULLSCR** → Returns the Object of the ALV Grid Class.

OOABAP

https://sapabapdatatempt.blogspot.com/p/sap-ooabap.html

- **CLASS** → Collection of Objects (methods, attributes, events, interfaces).
- **OBJECT** → Instance of class / user-defined datatype of type class. It holds the data and code to manipulate the data.

TYPES OF CLASS

1. **Global Class** → Created using SE24 Class Builder
   a) Usual Class – To write business logic (Like FM)
   b) Exception Class – To raise and handle the exception
   c) Persistence Class – To perform database operations
   d) Unit Test Class – To write Unit Test Cases

2. **Local Class** → Created within the program.
   Class Definition – Declaration
   Class Implementation – Implementation

- **METHOD – PROCESSING BLOCK OF CLASS WHERE WE WRITE THE LOGICS.**
  a) **INSTANCE METHOD** – We need to create the object to call this method.
     lo_object**->**display
     - If a method is declared like **METHODS** method1 in local class, then it is instance method
  b) **STATIC METHOD** – No need to create object. Call by Class name itself. **Cannot Redefine**.
     z31_cl1**=>**display
     - If a method is declared like **CLASS-METHODS** method1 in local class, then it is static method.

ACCESS TYPE

- **PUBLIC –** Accessed within the Class / Sub-class / Outside
- **PRIVATE –** Accessed within the Class
- **PROTECTED –** Accessed within the Class / Sub-class

Inheritance

- Creation of a new class from existing class which have all the properties of the parent class, without changing the properties of the parent class and may add new features to its own.
- Need to Redefine the Methods of parent class.
- **INHERITING FROM parent_class**

Final Class → It is a class which can't be inherited which means that cannot have sub-class.

Abstract Class → It is a special type of class which has at least one **abstract method**.

- **Abstract Method** is a method which has only definition, there will not be any implementation. It is created just by writing **ABSTRACT** next to method_name. **REDEFINITION** keyword will be used next to method_name in sub-class during redefinition.
- We cannot create object of Abstract Class.
- We can create object of sub-class of abstract class.
- **Never create Abstract Class as Final.** Otherwise we will not be able to create sub-classes.

- We have to redefine the method with **redefine button available** to in the sub-class in order to write the logic in method.
- **USEAGE** - We can define some common functionalities in Abstract class and those can be used in derived classes.

POLYMORPHISM → Ability to take **multiple forms**.

METHOD OVERRIDING → In this, method must have same signature in both the Parent Class & Sub-class but both have different method implementation.

- **METHOD OVERLOADING IS NOT POSSIBLE IN THE OOABAP.**

INTERFACE → **It is similar to the class having no control over visibility and can't code the logic.**

- In this, all the methods are public by default. (**No Visibility**)
- In this, all methods are abstract method. (**No Implementation**)
- We can achieve multiple inheritance with the help of interface.

INTERFACE interface_name (Definition)

- Interface are addressed in the class by **INTERFACES *interface_name*** during class definition.

Class_object->Interface~method_name *(calling interface attributes)*

Diff b/w Abstract Class and Interface

- In abs class, at least one method is abstract method. Whereas in Interface, all methods are abstract method.
- In abs class, we have control over method visibility. Whereas in Interface, all methods are public by default.
- In abs class, multiple inheritance is not possible.

EVENT

- Events are mechanism in OOABAP which can be leveraged to implement custom logic on how to react on user input.
- With the help of this, method of one class can call method of another class.
- **Triggering Method** – Class method which is raising the event. Only have exporting parameter.
- **Event Handler Method** – Class method which handles the event.
- Register the event handler method using **SET HANLDER** statement → When event will raise then the system will understand we have to execute the logic of event handler method.
- At a time, trigger method can call a single event handler method.

ME Keyword

- If a method has a variable having same name as one of the variable of the class, to clearly distinguish the both variable **ME** keyword is used with class variable inside the method.
- **ME->lv_var**

<u>CONSTRUCTOR</u>

- It is special type of method which will execute automatically when object is created.
- Predefined Name
- Multiple CONSTRUCTOR & CLASS_CONSTRUCTOR is not allowed in a class.
- **TYPES**

  a) **CONSTRUCTOR** → Instance Constructor
  - Method Name – CONSTRUCTOR
  - Only Importing Parameters
  - Can access both instance and static attributes
  - Called every time when object is created

  b) **CLASS_CONSTRUCTOR** → Class Constructor
  - Method Name – CLASS_CONSTRUCTOR
  - Does have any parameters
  - Can access only static attributes
  - It is called First time when object is created

<u>MODULE POOL PROGRAMMING</u>

- It is also called as **Dialog Programming**.
- It is a special type of programming which is used to create SAP Custom Screens.
- *SE38 → Program Type Module Pool → Display Object List*
- PF-Status full form Personal Function (**SE41**) → Menu Painter
- Screen consist of 3 Tabs
  - **I. Attributes**
  - **II. Element List**
  - **III. Flow Logic**

<u>Module Pool Events</u>

1. **PBO** → This event called before displaying a particular screen.
2. **PBI** → This event called after performing some action on a particular screen.
3. **POV** → This event called when F4 pressed on a field of a Screen.
4. **POH** → This event called when F1 pressed on a field of a Screen.

- *It is a best practice to keep all PBO Modules in one include and all PBI Modules in one include.*
- <u>***T-code** is always needed to run Module Pool Program which differentiate this with Executable Program*</u>*.*
- ***SE51** T-CODE FOR SCREEN PAINTER**.***
- **Table Control** → It is an element of module screen which displays the data in the form of table.

NOTE: <u>Whenever we are going for Layout specific changes, always write the logic in PBO module</u>.

<u>SUBSCREEN</u>

1. Create subscreen area in the normal screen.
2. Call the SUBSCREEN in the subscreen area of the normal screen.
3. Calling always takes place in PBO.

SYNTAX→ **CALL SUBSCREEN subscreen_areaname INCLUDING sy-repid '0200'.**

<u>Modal Dialog Box Screen</u>

SYNTAX→ **CALL SCREEN 0200 STARTING AT 10 20 ENDING AT 60 50.**

- **LEAVE TO SCREEN 0** → Take us to previous screen.

<u>Tabstrip Control</u>

- The tab strip control contains one main screen and n number of sub screen. This n number denotes the number of tabs.
- For each tab, SAP generates a subscreen.
- By default, SAP generates subscreen-area inside the tabstrip.
- We have to write the logic inside user_command module of the normal screen inside which tabstrip is declared.

AT EXIT-COMMAND

- Generally if a screen contains any obligatory (Required) input field & we execute the MP program, the screen appears & if we want to come out of the program it will not allow us . We must fill the mandatory field on the screen and then we can come out of the program by clicking on the application tool bar buttons.
- To overcome this problem, generally we use AT EXIT-COMMAND at PAI of the screen to force exit from the screen that contains a required field without filling any value to it.
- The function type of the button must be 'E'.
- We have to create one module for this in PAI Block.
  MODULE module_name AT EXIT-COMMAND.

CHAIN-ENDCHAIN

- Generally, For Input Field Validation on input screen, when the user gives the input and there is no record present in the database, the user input is restricted by error message. This error message disables the input screen and now the user cannot provide correct values in the input field as it is disabled by the error message.
- To overcome this problem, **Chain** and **EndChain** Statement is used in the Flow Logic of the screen where Input fields are validated.
- We must create one module inside PAI block and add chain and endchain statement and inside it put all the fields that you want to enable again if they are disabled with validation error within the module.

  CHAIN.
  FIELD : screen_fieldname1, screen_fieldname2 **MODULE** module_name.
  ENDCHAIN.

FUNCTION MODULE "**VRM_SET_VALUES**"
It is used to bind the date to the screen field, mostly with drop-downlist.

SELECT-OPTION IN MODULE POOL

1. Firstly, we have to create sub-screen in the main program through coding in which we create select option.
2. Then, we declare subscreen-area in the main-screen.
3. Lastly, we have to call sub-screen in the subscreen-area in the PBO of the main-screen.

```
DATA LV_CARR TYPE S_CARR_ID.
SELECTION-SCREEN: BEGIN OF SCREEN 200 AS SUBSCREEN.
  SELECT-OPTIONS: S_CARR FOR LV_CARR.
SELECTION-SCREEN: END OF SCREEN 200.
```

PROCESS ON VALUE REQUEST (POV)

- This event called when F4 pressed on a field of a Screen.
- We have to create a module in POV block of the screen.

```
PROCESS ON VALUE-REQUEST.
FIELD screenfield_name MODULE module_name.
```

- Inside this module, firstly we have to fetch the possible values for the field and then we will bind the values to the screen-field with the help of standard FM.

```
CALL FUNCTION 'F4IF_INT_TABLE_VALUE_REQUEST'
  EXPORTING
    retfield              = 'CARRID'
   DYNPPROG              = SY-REPID
   DYNPNR                = SY-DYNNR
   DYNPROFIELD           = 'LV_CARR'
   VALUE_ORG             = 'S'
  tables
    value_tab             = IT_SCA.
```

Process On Help-Request (POH)

- This event called when F1 pressed on a field of a Screen. It will show technical info and documentation.
- Firstly we create documentation using tcode **SE61 → Documenation**
- Then we create a module in POH block of the screen in which we call the documentation using standard **FM HELP_OBJECT_SHOW**.

Difference b/w CALL SCREEN <screen no.> and LEAVE TO SCREEN <screen no.>

CALL SCREEN calls the specified screen by adding the specified screen to the stack.
Whereas **LEAVE TO SCREEN** calls the specified screen by replacing the last stacked screen.

LEAVE TO LIST-PROCESSING : This statement is used when we are processing a screen and want to print our output in a list. To navigate from a screen to list LEAVE TO LIST-PROCESSING statement is used with some conditions.

SUPPRESS DIALOG

If this statement is specified during PBO processing, the current dynpro is processed without displaying the screen, while the screen of the previous dynpro remains visible. After PBO processing, the system triggers the event PAI in the same way as if a user had pressed Enter.

FILE HANDLING

- File handling means performing operations on the file like opening a file, reading a file, writing into a file, closing a file, or deleting a file.
- **AL11** → TCODE for SAP Directories

NOTE: IF WE COPY THE WHOLE DATA FROM EXCEL TO NOTEPAD THEN TAB SEPARATED DATA WILL AUTOMATICALLY GENEARTED IN NOTEPAD.

FILE HANDLING ON PRESENTATION SERVER

To work with the files on Presentation Server, SAP provides various standard FM.

- **GUI_UPLOAD** → Reading data from file on Presentation Server
- **FM F4_FILENAME :** This FM is called when value_request event for the input file triggered at selection screen. It gives the pop-up of the file explorer and returns the path of the chosen file.
- **GUI_DOWNLOAD** → Writing data into file on Presentation Server.

FILE HANDLING ON APPLICATION SERVER

To work with the files on Application Server, SAP provided the below statements.

- **OPEN DATASET** → Opens the specified File
- **CLOSE DATASET** → Closes the specified File
- **DELETE DATASET** → Deletes the specified File
- **READ DATASET** → Read a record from a File
- **TRANSFER** → Write a record into a File

WRITING DATA INTO A FILE ON SAP SERVER

OPEN DATASET <filepath> FOR OUTPUT IN TEXT MODE ENCODING DEFAULT.
TRANSFER <data> TO <filepath>.
CLOSE DATASET <filepath>.

READING DATA FROM A FILE ON SAP SERVER

OPEN DATASET <filepath> FOR INPUT IN TEXT MODE ENCODING DEFAULT.
READ DATASET <filepath> INTO <variable>.
CLOSE DATASET <filepath>.

ARCHIVING : It is a concept in which we move the processed file to archive directory & deleting it from existing directory.

1. **READING** FILE FROM EXISTING DIRECTORY ON SAP SERVER
2. **WRITING** TO FILE INTO ARCHIVE DIRECTORY ON SAP SERVER
3. **DELETING** THE FILE FROM EXISTING DIRECTORY ON SAP SERVER
   SYNTAX → DELETE DATASET lv_filepath**.**

Smartforms

- *Smartforms* is a tool provide by SAP to design and maintain the layout and logic of a form.
- TCODE → **smartforms**
- **Form Attributes** → It provides general information about the form like who created, at which date and time, etc.
- **Form Interface** → This acts as a mediator between a driver program and smartforms**.** In this, we define the importing & exporting parameters, tables and exceptions for the form which will be used by the driver program.
- **Global Definitions** → It will contain the **variables/workarea/internal** table to be used throughout the smartform. We write the business logic in the **initialization tab**.
- **Page** → It is used to design the layout of the smartform.
- **Window** → It is used to display information or text at a particular place on a page.
- **Template** → It is used for fixed numbers of rows and columns, for static data. Its dimensions must not exceed window dimensions. In this, we have to **provide FROM/TO value** in details tab. Inside Text, we have to provide row & column details in output structure.
- **Table** → It is used for unfixed number of rows and columns, for dynamic data. In this, we get three components – Header/Main Area/Footer. In each component, we must **create Table Line** and **select the line type** which will create no. of cells as per the no. of columns. Here we have to use Loop option from the data tab to pass the data to workarea.
- **Program Lines** → It is used for writing business logic for a particular window.
- **Alternative** → It consists of two components TRUE/FALSE. If you want to print something when condition is true then put the elements in **TRUE Node** and If you want to print something when condition is false then put the elements in **False Node.**
- **Address** → Used to display address by providing address number present in the **table ADRC**.

<u>**NOTE**</u>:

1. *Whenever we have to design the layout of the form, firstly we have to understand what the main section or parts of the forms are*.
2. *For each section, we must create a window and provide four dimensions to it.*
3. *We have to create text node to display information. (**Character width = 0.3cm height = 0.5cm**)*

- *We can call the smartform from driver program using the **FM generated during runtime** for the respective smartform.*
- `SSF_FUNCTION_MODULE_NAME` → returns the FM name of the smartform.
- It is very important whenever it is possible, please code the select query in the driver program.

- <u>*Whenever we did not get the output on running the driver program and also didn't get the error, always execute the smartform independently to check whether we get the output or error msg.*</u>
- Whenever we are passing the values of amount/quantity from program to smartform, if the reference-field is in the same structure then it is ok and if we are passing the reference-field of existing structure then we get an **error** "**unable to recognise the reference field**".
  To resolve this error, we make use of <u>*currency/quantity field tab*</u> of global definition.

__TYPES OF WINDOWS__

1. **MAIN WINDOW →** used for continuous output such as table data.
2. **SECONDARY WINDOW →** used for output with fixed length.
3. **COPIES WINDOW →** Special type of secondary window __for **marking pages** as **copy** or **original**__.
   SFSY-COPYCOUNT **–** system variable for copy count.
4. **FINAL WINDOW →** Special type of secondary window used for details which are needed to be processed only at the end of processing form.

__TYPES OF TEXTS__

1. **TEXT ELEMENT →** not a reusable text. Dedicated to only one smartform.
2. **TEXT MODULE →** It is a reusable text.
3. **INCLUDE TEXT →** Also a reusable text which can be reused by smartforms, programs, etc.
   TCODE- SO10   FM for displaying Include Text in Report: READ_TEXT
4. **DYNAMIC TEXT →** Used to display the text at runtime in smartforms.
   Predefined Table Type: **TSFTEXT**

NOTE: *We cannot put breakpoint in smartforms but we can put breakpoint in the FM generated.*

- Format of Smartform Upload/Download → **XML**
- **Always take backup of the original smartform before working on it.**
- **SAP** supports only **BMP** (**bitmap**) type images. We can also *dynamically display graphics* in forms
- **SE78 → TCode** to upload graphics in SAP.

__FOLDER__

It is used to ***achieve page protection*** in smartforms so that everything in the folder will be printed on the same page.

__PAGE__

Sometimes we have a requirement like we want to display only the line-items details on the next page then in such cases we create another page and *link it with first page using* **NEXT PAGE option**.

- ***SFSY-PAGE*** *→System Variable for* ***Current Page Number***
- ***SFSY-FORMPAGES*** *→System Variable for* ***Total Page Number***

__**SMARTSTYLES →**__ TCODE to create smart styles.

__BARCODE/QRCODE__ → TCode **SE73**

We can **assign the BARCODE/QR** to smartform using **character format of the smartsyle** into which barcode/qrcode is already mappped.

- To ***disable print dialog and set output device name & print preview*** of smartform, we have to pass `preview, no_dialog` values to the **control parameter** and `tddest` value to **output option paramete**r & set the flag **user_setting** to **False** to the FM of the smartform.

CONVERT SMARTFORM OUTPUT TO PDF

1. Get the OTF (Output Text Format) of Smartform.
Path: *Global Setting →Form Attributes →Output Options →Output Format*
Standard Output – It is text format for smartforms.
*Set **CONTROL_PARAMETERS-GETOTF = True** of FM of the smartform and it will return OTF in an **internal table OTFDATA** of exporting parameter **JOB_OUTPUT_INFO**.*

2. Convert OTF to PDF.
CONVERT_OTF – FM to convert OTF to PDF.
We pass the internal table otfdata to this FM and get the pdf in the table **lines**.

Sending Smartform PDF as an E-Mail Attachment

SAP have provided standard classes related to Business Communication Services (BCS).

- **CL_BCS** → This class is used for creating the send request, adding the mail recipient, sending the document, etc.
- **CL_DOCUMENT_BCS** → This class is used for creating the document (subject/mail body), adding the attachement, etc.
- **CL_SAPUSER_BCS** → This class is used to create SAP Users.
- **CL_CAM_ADDRESS_BCS** → This class is used to create the external recipient/internet address.

1. Get the **BIN_FILE** of the smartform**.**

2. Convert the **BIN_FILE** from **XSTRING** Format to **Binary** Format→*FM*: SCMS_XSTRING_TO_BINARY

3. Create the **Send Request**. → `cl_bcs=>create_persistent`

4. Create the **SAP User.** → `cl_sapuser_bcs=>create`

5. Create the **External Recipient.** → `cl_cam_address_bcs=>create_internet_address`

6. Add the `Recipient` → `lo_bcs->add_recipient`

7. Create the **Document**. → `cl_document_bcs=>create_document`

8. Add the **Attachment**. → `lo_document->add_attachment`

9. Sets the Document to be sent. → linking the **CL_DOCUMENT_BCS** to **CL_BCS**

10. Activate/Deactivate Immediate Sending. → `lo_bcs->set_send_immediately`

11. Send. → `lo_bcs->send`

12. Commit Work→ It is only required if we are sending the mail to external recipient.

SBWP → T-CODE To Check Mail Box

SOST → T-CODE to used to view the status of the sent mails from SAP.

Background Jobs

There are two types of Jobs.

1. **Foreground Jobs** → Interactive Jobs
2. **Background Jobs** →Non-interactive Jobs

SM36 → To Schedule the background job.
SM37 → To check the status of the background job.

JDBG → Debug Background Job
To debug a job, select the job in SM37 and put **JDBG** in command.

Status

- **Scheduled** → It means all the required condition to create a job are completed but start conditions are still not defined.
- **Released** → It means all the required condition to create a job are completed including start condition and frequency are finished.
- **Ready** → It means the job is ready for execution but put in the queue by the job scheduler waiting for the next available background work process.
- **Active** → It means the job is running.
- **Finished** → It means job is executed successfully.
- **Cancelled** → It means Administrator forcefully cancelled the job or it might be some error with the Job.

Capture/Debug an Active Job

1. Select the job and click JOB option in menu Bar.
2. Choose Capture: Active Job Option. This will automatically land us into debugger.

Enhancements & Modifications

Enhancement → We will enhance the SAP Functionality in Customer Namespace (Z or Y).

Modification → We will enhance the SAP Functionality in SAP Namespace (Name should not starts with Z or Y).

TYPES OF Enhancements & Modifications

1. Implicit & Explicit Enhancements **- Enhancement**
2. Customer Exit (FM Exit, Menu Exit, Screen Exit) **– Enhancement**
3. BADI **– Enhancement**
4. User Exit **– Modification**

1. Implicit & Explicit Enhancements - Enhancement

- **Implicit Enhancement**
  In this, **Implicit Point** is available at the start or at the last of the Program, FM, Subroutine, etc.
  We will create implicit Implementation using those points.
  Denoted with """"""""""""""""""""""
- **Explicit Enhancement**
  In this, **Explicit Point** & **Explicit Section** are available at any line.
  We will create explicit Implementation using those points and sections.

  Explicit Point
  - It will not give default implementation.
  - We can only add additional codes but cannot replace the existing code.

  Explicit Section
  - It will give default implementation.
  - We can change or replace the existing SAP code.
  - When we create implementation in explicit section, then all SAP Codes will automatically get written inside the implementation and from now onwards this implementation will execute, not the default implementation.

  Enhancement Spot → It is a container for Explicit Enhancement Options.

How can we decide which option is better Implicit or Explicit?

➢ If we are able to achieve the desire result by implicit or explicit, then implementing implicit or explicit will not make any difference, but yes, if it is an explicit section, then we have an advantage to replace the existing SAP code with our code.

2. Customer Exit

- **Function Module Exit**
  - It allows us to add our codes to the SAP Programs with the help of FMs.
  - Syntax→ **CALL CUSTOMER-FUNCTION 'three-digit number'**.

How to find FM Exit

- Put the breakpoint at the statement CALL CUSTOMER-FUNCTION**.**
- Put the package name of the program in SMOD transaction.
  Utilities → Find → Package Name
  Or
- Goto T-Code SE84 **Enhancements → Customer Exits → Enhancements.**

Steps to Implement a Function Module Exit

1. Goto T-Code **CMOD.**
2. Provide a Project Name and Create.
3. Click in Enhancement Assignment Tab and provide the available Customer Exit Name.
4. Click on Components Tab – It shows the various FM exits available.
5. Double Click on the required FM exit which navigates to FM and put the logic inside the Z include available there.
6. Activate the Project.

- **Menu Exit**
  - It allows us to add new menu items to the Menu of SAP Programs.
  - These menu items have a function code which begins with "**+**".
  - We can perform customer function on these menu items.

How to find Menu Exit

- Check for menu items starting with "+" sign in the menu bar.
  System → Status → GUI Status
  Or
- Put the package name of the program in SMOD transaction.
  Utilities → Find → Package Name
  Or
- Goto T-Code SE84 **Enhancements → Customer Exits → Enhancements.**

Steps to Implement a Menu Exit

1. Goto T-Code **CMOD.**
2. Provide a Project Name and Create.
3. Click in Enhancement Assignment Tab and provide the available Customer Exit Name.
4. Click on Components Tab – It shows the various Menu exits available.
5. Double Click on the required Menu exit.
6. Provide the details like function text, icon, icon text and information text.
7. To write the logic, we need to implement required FM Exit.
8. Double Click on the required FM exit which navigates to FM and put the logic inside the Z include available there.
9. Activate the Project.

- **Screen Exit**
    - It allows us to add new fields to the screen of the SAP program with the help of customer sub screen.
    - SAP provides the sub screen areas within a standard screen.
    - The customer sub-screen is called within the standard screen flow logic.
    - **Syntax to call customer screen into standard screen.**
      CALL CUSTOMER-SUBSCREEN 'subscreen-area' INCLUDING 'program-name' 'customer screen-number'.

How to find Screen Exit

- Check in the screen flow-logic for the statement CALL CUSTOMER-SUBSCREEN.

  Or
- Put the package name of the program in SMOD transaction.
  Utilities → Find → Package Name

  Or
- Goto T-Code SE84 **Enhancements → Customer Exits → Enhancements.**

Steps to Implement a Screen Exit

1. Goto T-Code **CMOD.**
2. Provide a Project Name and Create.
3. Click in Enhancement Assignment Tab and provide the available Customer Exit Name.
4. Click on Components Tab – It shows the various Screen exits available.
5. Double Click on the required Scree exit.
6. Create the sub-screen and design the respective layout of the screen.
7. To write the logic, we need to implement required FM Exit.
8. Double Click on the required FM exit which navigates to FM and put the logic inside the Z include available there.
9. Activate the Project.

Key Points / T-Codes / Tables

- One Customer Exit can only be assigned to one Project at a time.
- SMOD → T-Code to find a Customer Exit.
- CMOD → T-Code to implement a Customer Exit
- Table MODSAP → Table to find a Customer Exit
- Table MODACT → Table to find a Project for a Customer Exit.

3. BADI

- Business ADD In's.
- It is based upon OOPs concepts (Interfaces and Classes).
- SE18 → T-Code for BADI Definition
- SE19 → T-Code for BADI Implementation

<u>TYPES of BADI</u>

- **CLASSIC BADI**
  How to find Classic BADI
  - We can use SAP Class **CL_EXITHANDLER** in which we have a method **GET_INSTANCE**. Put the breakpoint inside this method.
    Or
  - Put the package name of the program in SE18.
    F4 Help of BADI Name → New Selection → Classic BADI Radio button.
    Or
  - Goto T-Code SE84 **Enhancements → Business ADD INs → Definitions.**
- **New BADI (Kernal BADI) →** It has enhancement spot.
  How to find New BADI
  - Put the breakpoint at statement – CALL BADI OR GET BADI.
    Or
  - Put the package name of the program in SE18.
    F4 Help of BADI Name → New Selection → Kernel BADI Radio button.
    Or
  - Goto T-Code SE84 **Enhancements → Business ADD INs → Definitions.**

<u>Steps to Implement New BADI</u>

1. Goto T-Code SE19.
2. Choose the New BADI radio button. Provide available Enh Spot and click on create.
3. Provide the Enhancement Implementation name and description.
4. Provide the name of BADI Implementation, Implementing Class and choose the respective BADI Definition.
5. Double Click on the Class and put the logic inside the respective method.
6. Activate the Class.

<u>Difference between Classic BADI and New BADI</u>

- In classic Badi, there is no enhancement spot. Whereas in new Badi, there is an enhancement spot.
- In classic Badi, there is no enhancement implementation. Whereas in new Badi, there is an enhancement implementation which is the container for new BADI implementation.
- In classic Badi, the name of implementing class automatically appears. Whereas in new Badi, we need to provide the name of implementing class.

<u>Filtering of BADI Implementation:</u> Filters act like dynamic conditions to control the BADI behaviour. **<u>When you have multiple implementations for a single BADI, filters allow you to choose the relevant one based on specific criteria.</u>** For example, imagine separate BADI implementations for handling customer data in different countries. Filters ensure the appropriate implementation executes based on the customer's country code.

<u>Fallback Class</u>

Fallback class is a **default implementation** class that is executed when no implementation is executed for a BADI.

4. User Exit

- We will enhance the SAP Functionality in SAP Namespace (Name should not starts with Z or Y).
- They are only available in **SAP SD Module**.
- They are available in the form of a Subroutines.
- Most Important User Exits relates to sales order are available in the program MV45AFZZ.

BTE (Enhancement)

- Business Transaction Events.
- We enhance the SAP Functionalities in Customer Namespace.
- They are only available in **SAP FI Module**.
- BTE is a custom FM attach to the standard Program to enhance the standard functionality.
- T-Code → **FIBF.**

TYPES OF BTE

I. **PUBLIC and SUBSCRIBE INTERFACE**
   - It is also called as **Informing Interface**. Because this is used for passing the information.
   - This **do not update any application data**. That's why custom FM attach to the BTE do not have any changing parameters.
   - It is used to add additional steps to the application.
     For example, email functionality, sending data to the external system, etc.
   - It does not influence the standard process anyway.
   - FM for Public & Subscribe Interface always end with **E**.

II. **PROCESS INTERFACE**
   - This type of interface is used to **update the application data**.
     For example, passing default value to field while creating FI Document.
   - It intervenes the standard process.
   - FM for Process Interface always end with **P.**

How to find PUBLIC & SUBSCRIBE INTERFACE

- Put the breakpoint in the Function Module: **BF_FUNCTIONS_FIND** and execute the relevant tcode to trigger the breakpoint. Then go to screen stacks in debugger and look for the FM **OPEN_FI_PERFORM_<BTE NUMBER>_E.**

  Or
- Search for the statement OPEN_FI_PERFORM in the source code.
  For Public & Subscribe Interface, the naming convention for the function module will be **OPEN_FI_PERFORM_<BTE NUMBER>_E**.

  Or
- Goto T-Code FIBF. Menu Bar **Environment → Info System P/S → Execute.**

How to find Process Interface

- Put the breakpoint in the Function Module: **PC_FUNCTION_FIND** and execute the relevant tcode to trigger the breakpoint. Then go to screen stacks in debugger and look for the FM **OPEN_FI_PERFORM_<BTE NUMBER>_P.**

  Or

- Search for the statement OPEN_FI_PERFORM in the source code.
  For Public & Subscribe Interface, the naming convention for the function module will be **OPEN_FI_PERFORM_<BTE NUMBER>_P**.

  Or

- Goto T-Code FIBF. Menu Bar **Environment → Infosystem Processes → Execute.**

Steps to Implement Process Interface

1. Identify the required BTE.
2. Copy the sample interface FM into Z or Y FM and write the code inside it.
3. Create a Product and activate the product.
4. Link the FM and event using the product.

Difference B/w BTE and BADI

- BTE is applicable only for SAP FI Module. Whereas BADI is generalized to all SAP Modules.
- BTE uses FM to enhance SAP standard functionality. Whereas BADI is based upon OOPs concept.

ADOBE FORM

- Adobe Forms are interactive smartforms in which we can interact with the output.
- Its output always in PDF.
- **T-Code → SPF**
- In adobe form, *firstly we create an **interface*** which contains all the business logics. The same interface can be used by multiple adobe forms.
- It also *generates **FM** during runtime*.
- Adobe Lifecycle Designer (**ADLC**) is required to design the layout with the help of adobe forms.
- **Standard Layout** is used for PDF based print forms or for interactive forms used in an offline scenario (sent by email).
- **Context →** We drag and drop the data from interface that is to be displayed in the form at runtime is part of the form context.
- **DATE/TIME/UNAME/SUBRC →** These system variables by default available in context in AForm.

DIFFERENCE B/W SMARTFORMS AND ADOBE FORMS

In *smartfrom*, form interface, global definitions, pages, and windows are all together. Whereas in *adobeform*, form interface and global definition are the part of interface & we design the layout separately in the form.

Data Migration

- Migrating data from Legacy System to SAP System.
- Steps of Data Migration:
  1. Extract the data from the legacy system into a file.
  2. Converting the data into an appropriate format. This is called as **Conversion**.
  3. Importing the data into the SAP System.
  4. Verifying the data → Checking the accuracy of data in SAP System.

Data Migration Techniques

1. **BDC** (Batch Data Communication)
2. **LSMW** (Legacy System Migration Workbench)
3. **BAPI** (Business Application Programming Interface)

1. BDC (Batch Data Communication)

- Migrating data from Legacy System to SAP System.
- It works on the principle of **Screen Recording**.
- T-Code for recording → **SHDB**
- Precautions for recording: Never use F4 & F1 Help. Do not pass wrong values.
- Every step we performed while recoding will get stored in **BDC internal table** in Program Processing.

Methods Of BDC

a. Call Transaction Method

CALL TRANSACTION 'T-CODE' USING BDCTable MODE 'A/N/E' UPDATE 'A/S/L' MESSAGES INTO LT_MESSAGETAB.

Processing Modes

A→ It will show screen by screen processing.
N→ It will not show screen, we will directly get an output.
E→ If there is an error in BDC, then only it will show the screen where the error. Otherwise, not.

Update Modes

S (Synchronous) → the database is updated before the next transaction is taken for processing in a batch input.
A (Asynchronous) → the system doesn't wait for updating the database before the next transaction is taken for processing in a batch input. **It is faster**.

b. Session Method

- It uses three FM:
  BDC_OPEN_GROUP→ for opening the Batch Input Session.
  BDC_INSERT_GROUP→ for inserting the transactional data into SAP.
  BDC_CLOSE_GROUP→ for closing the Batch Input Session.
  Then go to T-Code **SM35** to process the session.

- Significance of **KEEP parameter** in BDC_OPEN_GROUP → flag to indicate whether we should keep the processed session or not in SM35. But still, we can see details of it in the Log.

Processing Modes

Foreground→ It will show screen by screen processing.
Background→ It will not show screen, we will directly get an output.
Display Error Only→ If there is an error in BDC, then only it will show the screen where error is.

- Call Transaction Method & Session Method are called as **Batch Input Method.**

c. **Direct Input Method**
  - These are SAP standard program to post the data into the SAP.

Difference B/w Call Transaction Method & Session Method
- Call Transaction Method is faster than Session Method.
- We do not have the Log in Call Transaction Method which is available in Session Method.

2. **LSMW** (Legacy System Migration Workbench)
- It is a ABAP Workbench Tool used for data migration.
- T-Code→ **LSMW**

Difference B/w LSMW and BDC
LSMW is for functional consultants which does not require much coding. While BDC is for technical consultant.

LSMW METHODS
1. Direct Input
2. **Batch Input Recording**
3. **BAPI**
4. IDoc

BAPI
- Business Application Programming Interface.
- It is a technique used to **migrate a large scale of data** from legacy system into the SAP system.
- BAPIs are implemented as FM in the system. **BAPI = FM + Business Object**
- BAPI is always preferred over BDC for data migration.
  BDC run through screen-by-screen flow processing to update the database due to which it is slow. Whereas BAPI directly updates the database. Hence it is faster than BDC.

- SAP provides a number of Standard BAPIs for all the modules to achieve the business process. But in very rare case we may need a custom BAPI our specific business needs.
- **EXTENSIONIN/EXTENSIONINX** → BAPI parameter which is used to pass the value of custom fields to standard BAPI.