Formation ABAP IV

© Said el Bouhali

BRAKE FRANCE SERVICES

Sommaire

Introduction

- ABAP IV est un langage de programmation de 4ème génération
- Un programme ABAP peut lire :
  - des bases de données SAP (séquentiellement ou directement)
  - des fichiers séquentiels UNIX
- Il peut créer :
  - des éditions On-Line ou Batch
  - des tables SAP
  - des fichiers séquentiels UNIX ou NT
  - des dossiers de mise à jour des bases SAP (**Batch Input**)
  - des états (pour impression ou affichage)
- Remarque : SAP est séparer en plusieurs mandant, une propriété principale d'un programme ABAP est qu'il est inter mandant.

Ce cours est destiner à former les gens au développement de rapport d'édition avec ABAP.
J'ai suivi certaines règles de notations :

- Les mots en gras sont des **mots clés d'ABAP** à recopier tel qu'elle dans le programme.
- Les mots entre [ ] sont des options.
- Les mots en italiques sont des *identifiant* qui doivent être significatif de ce qu'il désigne

- **Commentaires** :
  - Soit, le premier caractère d'une ligne est ' * ' => toute la ligne est un commentaire.
  - Soit, On trouve, en milieu de ligne, le caractère ' " ' => tout le reste de la ligne est un commentaire.
- Chaque instruction se termine par un ' **.** '
- Pour ne pas répéter une instruction, utiliser le ' : ' suivi d'une ' ; '.

- **Environnement de développement** :

Pour créer un programme ABAP on suit le chemin suivant :
OUTILS / ABAP Workbanche
Ou la transaction SE38.
Taper le nom du programme qui doit commencer avec un Z+7 caractères puis cliquer sur créer
Dans la zone titre donner une description court du programme.
Et dans la zone type saisir 1 => Progamme online
Et dans la zone Application saisir * => Inter-applications
Cliquer sur sauvegarder et la dans la fenêtre qui apparaît cliquer sur objet local.
Le formateur ne savait pas pourquoi ces paramètres là.

Le programme est créé et on tombe sur l'éditeur ABAP
On a deux choix soit charger un programme qui a été saisie dans un autre éditeur, soit travailler avec cet éditeur là.
Si vous avez pris le deuxième choix sachez que c'est F7 pour insérer une ligne et c'est d pour la supprimer. Pour avoir de l'aide positionnez le curseur sur le mot et taper F1.

Pour contrôler la syntaxe d'un programme faire : **Ctrl+F2**. Pour le compiler en sauvegardant :**Ctrl F3**. Pour l'exécuter :**F8** (il sera compiler sans sauvegarde avant de s'exécuter).
Pour télécharger à partir d'un autre editeur faire : UTILITAIRES / TELECHARGEMENT.

Déclarations

- **Déclaration d'un programme d'édition** : un rapport ABAP commence avec le mot clé :

REPORT *nom_rapport*
[**NO STANDARD PAGE HEADING**] " ne pas afficher le nom du rapport en entête
[**LINE-SIZE** *n1*] "
[**LINE-COUNT** *n2*(*n3*)] "
[**MESSAGE-ID** *xx*] "xx identifie une classe de message prédéfini pour avoir le liste des classes de message prédéfini utiliser la transaction

Exemple : **REPORT** *ztestsaid* **NO STANDARD PAGE HEADING MESSAGE-ID** *zz* **.**

- **d'objets (TABLE DB , TABLE INTERNES, STRUCTURES)** : au début de chaque programme, on déclare toutes les tables de la base de données SAP ou les tables internes ou les structures, qui seront utiliser dans ce programme .

TABLES : *nom_table* **,** *nom_struct* **,** *i_table* **,** … **.**

Exemple : **TABLES :** *KNA1 , ITCPO , USR03* **.**

- **de type** : Il existe un bon nombre de type prédéfinit

Exemple :
I pour les entiers,
P pour les décimales,
C pour le texte,
N numeric character,
D date,
T time,
F virgule flottante et
X hexadécimale.

Pour définir un nouveau type de données On procède comme suit :

TYPES [**:**] **BEGIN OF** *nom_type,*
…
END OF *nom_type* **.**

Exemple :

- **de variables** :

DATA [**:**] *nom_var***[(***longueur***)] TYPE** *nom_type* [**VALUE ‘** *valeur* **’ "** pour les char] [**VALUE** *valeur* **"** pour les entiers] **.**

DATA [**:**] *nom_var* **LIKE** *nom_var_déjà_déclarée* **.**

Exemple :
DATA : *somme(10)* **TYPE** *P* **DECIMALS** *2* **VALUE** *0* **. "** l'option DECIMALS n'est valide que pour les types P
DATA *code_client* **LIKE** *KNA1-KUNNR* **.**

- **de_constante** : attention le mot clé VALUE n'est plus optionnel, il devient obligatoire :

CONSTANTS [**:**] *nom_const*[(*longueur*)] **TYPE** *nom_type* **VALUE** *valeur* **.**

CONSTANTS [**:**] *nom_const* **LIKE** *nom_var_déjà_déclarée* **VALUE** *valeur* **.**

- **de_structure** : une structure est, en quelque sorte, une variable composé:

DATA [**:**] **BEGIN OF** *nom_struct* **,**
liste de déclaration de variables **,**
END OF *nom_struct* **.**

Ou bien encore :

DATA [**:**] **BEGIN OF** *nom_struct* **.**
[*liste de déclaration de variables* **.**] **"** pour déclaré des variables ici il faut changer tout les points par des virgules, sauf le dernier, comme sur l 'exemple .
INCLUDE STRUCTURE *nom_struct_déjà_déclarée* **.**
DATA [**:**] **END OF** *nom_struct* **.**

Exemple :
DATA [**:**] **BEGIN OF** *T_VBAK* **,**
code_client **TYPE** *I* **,**
NAME1 **LIKE** *KNA1-NAME1* **.**
INCLUDE STRUCTURE *VBAK* **.**
END OF *T_VBAK* **.**

- **d 'un intervalle de données** : c 'est une manière de définir des sous types.

RANGES *nom* **FOR** *zone* [**OCCURS** *n*]**.**

- **de_symbole de champ** :

FIELD-SYMBOLS **<***nom_field***>** [**TYPE** *nom_type*] **.**

Tables internes

Présentation des tables internes :

- Une table interne est un ensemble d'enregistrements qui ont la même description.
- **Sa structure et ses données ne vivent que pendant la durée du programme.**
- Elle est composée d'une zone de travail dite 'en-tête' et d'enregistrements

L'en-tête d'une table interne est une sort de banderole qui a la même structure que la table et qui, à un moment donnée, contient **UN** enregistrement de la table ou ne contient rien. Il est fondamental de comprendre cette notion.
Les données sont en mémoire, pas sur le disk.

- **Déclaration d'une table interne** : de la même façon qu 'une structure.

Ou bien encore:

DATA [**:**] **BEGIN OF** *nom_T_interne* **OCCURS** *nombre* **.** *" d 'enregistrement prévu dans cette tables on y met toujours 0*
[*liste de déclaration de variables* **.**]
INCLUDE STRUCTURE *nom_T_interne _déjà_déclarée* **.**
DATA [**:**] **END OF** *nom_T_interne* **.**

Lecture séquentielle de table interne

- **LOOP AT** *i_tab* **[WHERE** *cond1* **] [INTO** *...***]**
…
ENDLOOP .

A tout moment de la boucle, la variable system **SY-TABIX** contient le N° de l'enregistrement en cours de lecture.

Instructions liées :

- **AT FIRST**.
- **AT NEW** *chp1*.
- **AT END OF** *chp1*.
- **AT LAST**.
- **SUM**.

Exemple :

```
DATA [:] BEGIN OF T_VBAK OCCURS 0,
            code_client TYPE I ,
            NAME1 LIKE KNA1-NAME1 .
             INCLUDE STRUCTURE VBAK .
               END OF T_VBAK .

  LOOP AT T_VBAK WHERE code_client = '0000042587'
            IF SY-TABIX = 1.
                  WRITE : /1 ' c'est le 1er enregistrement ', (80) T_VBAK-NAME1 .
            ENDIF.
            AT LAST
                  WRITE : /1 'voici le nombre totale d'enregistrements de cette table',
                              (80) SY-TABIX.
    ENDLOOP .
```

Le résultat de l'exécution de ce programme sera :

c'est le 1er enregistrement *AVENANCE FRANCE*
voici le nombre totale d'enregistrements de cette table *1255*

Lecture directe de table interne

Lorsqu'on fait n'importe quelle opération de lecture dans une table interne ce que l'on lit c'est son en-tête, c'est pour ca qu'il très important de savoir a tout moment ce qu'il y a dedans, par exemple : si on fait une affectation, on veut mettre dans une variable le contenu d'une zone de la table, supposons le nom du client n° *0000042587,* il faut d'abord mettre cette enregistrement dans l'en-tête et pour cela on utilise les instructions suivantes :

- **READ TABLE** *i_tab* [**INTO** *g_str*] **INDEX** *ind*.
- **READ TABLE** *i_tab* [**INTO** *g_str*] **WITH KEY** *clé* [**BINARY SEARCH**].
- **READ TABLE** *i_tab* [**INTO** *g_str*] **WITH KEY** *zone1 = val1 ... zonen = valn*.

Exemple :
READ TABLE *T_VBAK* **WITH KEY** *code_client = '0000042587'***.**
MOVE *T_VBAK-NAME1* **TO** *var1* **.**

L'instruction **LOOP AT** contient un read .

Mises à jour

- APPEND : c'est comme un commit en SQL sur des tables normales.
- INSERT
- MODIFY : c'est comme un UPDATE, attention il faut faire un 'modify' après chaque changement
- DELETE
- COLLECT

Initialisations dans les table internes

- CLEAR : ne vide que l'en-tête de la table.
- REFRESH : ne vide que la table l'en-tête reste intacte.
- FREE : vide la table et l'en-tête.

Tri et informations

- DESCRIBE TABLE : donne la structure de la table que l'on peut aussi voir avec la transaction SE16.
- SORT : tri une table.

Les Écrans de sélection (Les entrées)

Les écrans de sélection est le moyen le plus utilisé pour fournir des données à un programme ABAP.

- **Mise en forme de l'écran de sélection :**

  - **SELECTION-SCREEN ULINE(***longueur***) .** » dessiner une ligne
  - **SELECTION-SCREEN SKIP** *nombre* **.** " un nombre de ligne à sauter
  - **SELECTION-SCREEN COMMENT** *nbr_colonnes***(***longueur***)** *TEXT-xxx* **.**
  - **SELECTION-SCREEN BEGIN OF BLOCK** *nom_bloc* **[WITH FRAME TITLE** *TEXT-xxx***].**
    Paramètres et options de sélection **.**
    SELECTION-SCREEN END OF BLOCK *nom_bloc* **.**

" cette dernière instruction permet de rassembler un ensemble de champs de sélection dans un rectangle avec un titre.

" *TEXT-xxx étant des éléments de texte on verra plus tard comment les définir, c 'est des sortes de variables qui contiennent du texte pour une langue données . On ne saisie jamais des chaîne de caractère dans l 'ABAP directement.*

- **Déclaration de paramètres :**

  - **PARMETERS** nom_pram**(***longueur***) TYPE** *nom_type* **.**
  - **PARMETERS** nom_pram **LIKE** *nom_zone* **.**
  - **PARMETERS** nom_pram **AS CHECKBOX .**
  - **PARMETERS** nom_pram **RADIOBUTTON GROUP** *nom_group* **.**

" le nom_param ne doit pas dépasser 8 caractères.

Options :

- **DEFAULT** '*valeur*' **.**
- **MATCHCODE OBJECT** *xxxx* **.** " pour utiliser un matchcode existant, le matchcode étant une sorte liste de sélection.
- **OBLIGATORY .** " rend la saisie dans ce champs obligatoire.

- **Déclaration d'options de sélection :**

  - **SELECT-OPTIONS** S_SELOP **FOR** zone.

Options :

- **NO-EXTENSION** : pour ne pas avoir droit à la flèche qui permet des critères plus approfondis.
- **NO INTERVALS :** pour n'avoir qu'une seul zones à la place de deux (min, max).
- **OBLIGATORY**

Exemple :

```
SELECTION-SCREEN BEGIN OF BLOCK B1  WITH FRAME TITLE TEXT-001 .
  SELECTION-SCREEN SKIP 1 .
  SELECTION-SCREEN COMMENT 1(79) TEXT-002 .          "  longueur maximale 79
  SELECTION-SCREEN SKIP 1 .
  SELECT-OPTIONS : REF  FOR ZFEBX-ZLIBL MATCHCODE OBJECT ZREF .
  SELECTION-SCREEN SKIP 1 .
  SELECTION-SCREEN COMMENT 1(79) TEXT-003 .
  SELECTION-SCREEN SKIP 1 .
  PARAMETERS : IMPR LIKE TSP03-PADEST DEFAULT 'BFJ2' .
SELECTION-SCREEN SKIP 1 .
```

SELECTION-SCREEN END OF BLOCK *B1* **.**

" pour afficher à l ’écran de sélection autre chose que REF et IMPR, à partir de l ’éditeur ABAP, on sélectionne SAUT / ELEMENTS DE TEXTE puis on coche ‘ Textes de sélection ’ et on clique sur ‘ modifier ’ et là dans la zone texte en face de l’objet on met le texte à afficher .

Editions (Les sorties)

Ecriture (WRITE) : écrit sur la sortie standard (écran par exemple)

- **WRITE** [**:**] [/] [[**AT**] [*position*][**(***long***)**]] *zone* [*option*] **.** " '/ ' est un retour chariot
- **WRITE** *zone* **AS CHECKBOX .**
- **WRITE** *symbole* **AS SYMBOL .**
- **WRITE** *icône* **AS ICON .**

Exemple : **WRITE : /1 '** *BONJOUR* **' ,** *KNA1-KUNNR* **.**

Formats (FORMAT)

- **FORMAT** *opt1* [**ON** | **OFF**] *opt2* [**ON** | **OFF**] … .
- **FORMAT** *opt1* = [*var1*] *opt2* = [*var2*] … .
- **FORMAT COLOR** *num*.
- **FORMAT RESET**.

Sauts (NEW-PAGE, SKIP, POSITION, BACK)

Exemple : **SKIP** *2* **.**

Autres instructions (ULINE, RESERVE, SET)

Opérations

Affectations :

- *B* = *A* .
- **MOVE** *A* **TO** *B* .

- **MOVE-CORRESPONDING** *str1* **TO** *str2* . " Il faut que les deux structures aient les mêmes noms de champs.

- **CLEAR** *A* .

- **WRITE** [:] [/] *A* **TO** *B* [*option*].

Les options :
NO-ZERO.
NO SIGN.
DD/MM/YY.
DECIMALS *n* **.**
NO-GAP. Supprime les blancs.
/ . Retour à la ligne.
LEFT-JUSTIFIED | CENTRED | RIGHT-JUSTIFIED.

- **WRITE** *A+n(P)* **TO** *B+m(L)* . " (m et n) étant le debut et (P et L) la longueur

Exemple :
DATA : *VAR* **TYPE P DECIMALS** *2* **VALUE ‘** *12493.97* **’.**
DATA : *A(4)* **TYPE C VALUE ‘***said***’.**
DATA : *B(10)* **TYPE C VALUE ‘***EL BOUHALI***’.**
WRITE : / *VAR* **ROUND** *–2,*
/ *VAR* **ROUND** *0,*
/ *VAR* **ROUND** *2.*
WRITE : / *A+.3(2)* **TO** *B+5(1).*
WRITE : / ‘A’,’B’,’C’. " Il met un espace par défaut.
WRITE : / ‘A’ NO-GAP, ‘B’ NO-GAP, ‘C’.

Le résultat : " , est un séparateur de millier et . est un séparateur de décimales

1,249,397.00
12,493.97
124.93
EL BidHALI
A B C
ABC

Opérations numériques :

- **COMPUTE**
- **ADD** v_1 **THEN** v_2 **UNTIL** v_k **GIVING** S_1 **.**

ADD v_1 **THEN** v_2 **UNTIL** v_k **TO** S_2 **.**
Ou v_1 v_2 ….. v_k font partie d’une structure
GIVING S_1 contient $v_1 + v_2 + \ldots + v_k$
TO S_2 contient $v_1 + v_2 + \ldots + v_k + S_2$

- **SUBTRACT** v_1 **FROM** v_2
- **MULTIPLY** v_1 **BY** v_2
- **DIVIDE** v_1 **BY** v_2

- **A = B DIV C**
- **A = B MOD** C
- **A = B ** C** " exponentielle

Exemple :
COMPUTE *page* **= (*var1* DIV *var2*) MOD *9* .** "attention il faut mettre des espaces partout

<u>Opérations sur textes</u> :
- **CONCATENATE** *A B* **INTO** *C* **.**

CONCATENATE *'co' 'CO'* **INTO** *C* **.**

- **SHIFT** *A* **LEFT .** " enlève le 1er caractère et ajoute un blanc à la fin de la chaîne .

SHIFT *A* **CIRCULAR .** " déplace le 1er caractère à la fin .

- **TRANSLATE** *A* **TO UPPER CASE .**

TRANSLATE *A* **TO LOWER CASE .**
TRANSLATE *A* **USING** *B* **.**

- **STRLEN(** *A* **).** " Retourne la longeur

- **REPLACE '.' WITH ',' INTO (***var***) .**

- **CONDENSE** *C1* **.** " Laisse un seul espace à la place de chaque ensemble d'espace contiguë.

CONDENSE *C1* **NO-GAP .** " ne laisse aucun espace dans la chaîne de caractères C1 .

- **SEARCH** *C1* **FOR** *C2* **.** " Recherche la chaîne C1 dans la chaîne C2.

Les branchements

Appels (FORM, PERFORM, CALL) :

- Il existe différentes formes d'appel :
  - **PERFORM** *nom_sous_proc* **[USING** *nom_var1* **] [CHANGING** *nom_var2* **] [TABLES] [STRUCTURE] .**

```
" appel de sous-procédure ;
" Using pour les variables en entrée et Changing pour les variables en sortie
" pour  une variables en etrée-sortie, on fait Using variable Changing variable
" un mot clé par variable et les Using avant les Changing
```

  - **FORM** *nom_sous_proc* **.** " sous-procédure appelée ;

```
  …
ENDFORM .
" même option .
" Changing Value .
```

  - **CALL FUNCTION '** *nom_fonction* **' [Exporting] [Importing] [Tables] [Changing][Exceptions] .** " appel de fonction ou de transaction

  - **CALL TRANSACTION '** *nom_transaction* **' [AND SKIP FIRST SCREEN] [USING** *i_tab* **] [MODE** *mode* **] [UPDATE** *maj* **] [MESSAGES INTO** *i_mes* **] .**

```
" pour utiliser des fonction prédéfinit, faire Traiter / Modèle instruct….
" pour avoir la liste des fonctions prédéfinit OUTILS / ABAP / BIBLIO FONCTION
" ou bien transaction SE37
```

- Exemple :

```
REPORT test .
PERFORM C_FORM  USING var1 CHANGING var1 CHANGING var2 .
*---------------------------------------------
FORM C_FORM  USING var1 CHANGING var1 CHANGING var2 .
 CALL FUNCTION  'CLOSE_FORM '
  IMPORTING
    RESULT  =  var2
*   TABLES
*     OTFDATA  =
    EXCEPTIONS
      UNOPENED = 1
      OTHERS  = 2 .
var1 = var1 + var2 .
ENDFORM .                    " C_FORM
```

Boucles (DO, WHILE, LOOP, SELECT) :

  - **DO [VARYING** *zone1* **FROM** *zone2* **NEXT** *zone3* **] [** *n* **TIMES] .**

```
…
ENDDO .
```

" appel est une boucle simple, n est le nombre de fois que vous voulez passer dans la boucle.

Exemple :

```
DO .
  WRITE :  /  'SY-INDEX - Begin:',  (3)  SY-INDEX.
  IF SY-INDEX = 10.
    EXIT.
```

ENDIF.
WRITE: '*End :***', (**3**)** *SY-INDEX***.**
ENDDO .

- **WHILE** *cond1* **[VARY** *zone1* **FROM** *zone2* **NEXT** *zone3* **] .**
  …
  ENDWHILE .

Exemple :
WHILE *LETTER2* **<> '!'**
VARY *LETTER1* **FROM** *WORD-ONE* **NEXT** *WORD-THREE*
VARY *LETTER2* **FROM** *WORD-TWO* **NEXT** *WORD-FOUR* **.**
WRITE : *LETTER1* **,** *LETTER2* **.**
ENDWHILE.

- **LOOP AT** *nom_tab* **[WHERE** *cond1* **] [INTO** *wa* **] [FROM** *n1* **TO** *n2* **]**
  …
  ENDLOOP . " est une boucle de lecture de table interne

- **SELECT [SINGLE]| * |** *nom_table~nom_colonne* **FROM** *nom_table* **[WHERE** *condition* **] [INTO** *TABLE* **] [APPENDING] [CORRESPENDING** *itab* | *dbtab* | *struct* **] [ORDER BY** *nom_colonne* **]**
  …
  ENDSELECT . " est une boucle de lecture de table SAP
" pour utilisé **[CORRESPENDING** *itab* | *dbtab* | *struct* **]** il faut que l'objet destination ait les même noms de zones que le résultat du select .

Conditions (IF, CASE) :

- Il existe 2 types de branchements conditionnels :
  - **IF** *condition* **.**
    …
    ELSE [*condition* **] .**
    …
    ENDIF.

  - **CASE [** *var1* **] .**
    WHEN [*valeur1* **] .**
    …
    WHEN [*valeur2* **] .**
    …
    WHEN OTHERS .
    …
    ENDCASE .

Sorties (CHECK, EXIT, STOP, CONTINUE) :

- Il existe plusieurs instructions pour sortir d'une boucle ou d'un programme :
  - **CHECK** *condition* **. "** Si le condition est vrai on continue, si elle est fausse on arrête le programme, en générale la condition est (CHECK 1 = 2) au milieu d 'un IF, c 'est à dire qu 'on sort du programme si on arrive à cette instruction.
  - **EXIT . "** sort de n'import quelle boucle (DO, WHILE, LOOP, SELECT) .
  - **STOP . "** appelle le END-OF-SELECTION .
  - **CONTINUE . "** revient au début de la boucle en incrémentant l'indice de la boucle .

Données système

- Les données système principales sont des infos système stockées dans la structure SYST.

Cette structure n'a pas besoin d'être déclarée comme dans le **TABLES .**
A la place de mettre **SYST-NOM_DE_ZONE**, on met juste **SY-NOM_DE_ZONE .**
Voici les principales zone de cette structure, la description totale de cette structure est en annexe :

- SUBRC = Code retour (<> de 0 en cas d 'erreurs);
- INDEX = Indice courant d'une boucle ;
- TABIX = Indice courant d'une table interne ;
- TFILL = Nombre d'entrées dans une table interne ;
- DATUM = Date système ;
- UZEIT = Heure système ;
- REPID = Programme courant ;
- PAGNO = Numéro de page courante (liste) ;
- LINNO = Numéro de ligne courante (liste) ;
- UNAME = Code utilisateur courant ;
- LISEL = ligne sélectionnée (liste interactive) ;
- VLINE = caractère de cadre vertical (" | ") ;
- ULINE = caractère de cadre horizontal (" - ") …
- UCOMM = N° de la transaction courante .
- LILI = N° de ligne courante à l'écran .

Exemple :

var1 **= SY-INDEX .**
WRITE SY-ULINE(90) . " tire un trait.

Evénements

Les évènements sont des mots clé chacun d'eux se déclenche après ou avant une action bien connu. Ils exécute toutes les instruction qui existe après ce mot jusqu'au END ou bien jusqu'au prochain évènement.
Un évènement ne peut jamais être déclaré à l'intérieur d'un autre.

Événements de programmes

- **Initialization .**

Le corps de cette évènement est composé de variables et des valeurs à leurs affecter ;Ils 's'exécute le premier soit dés l'exécution du mot clé REPORT soit dés l'appelle de la sous procédure dans laquelle il existe.

- **Start-of-selection .**

Se déclenche après la validation de l'écran de sélection.

- **End-of-selection .**

- **Top-of page .**

Se déclenche à chaque passage à une nouvelle sélection, c'est là ou l'on définit l'en-tête des états.

- **End-of-page .**

Événements des écrans de sélections

- **AT SELECTION-SCREEN**

Se déclenche après la validation de l'écran de sélection et avant le START-OF-SELECTION, c'est là ou l'on définit des messages d'erreurs pour les saisies erronées ou hors des intervalles acceptés.

  - ON *param*
  - ON BLOCK *bloc*
  - ON VALUE-REQUEST FOR *param*
  - OUTPUT

L'option **AT SELECTION-SCREEN OUTPUT .**
Se déclenche avant l'affichage de l'écran de sélection, c'est là ou l'on peut interdire ou supprimer certains paramètres de sélections si on utilise un écran de sélections prédéfinit.

Événements de listes : AT ...

- **AT LINE-SELECTION**.

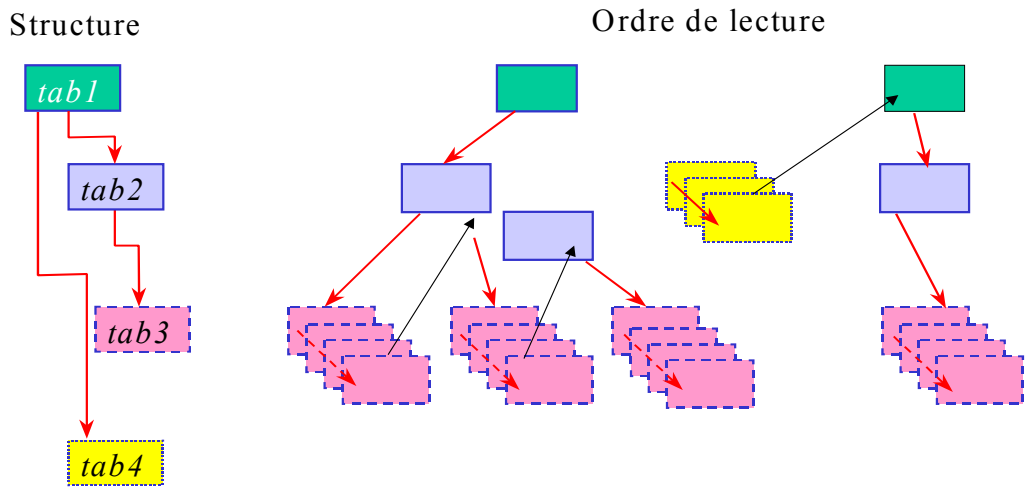
Se déclenche après la sélection d'un enregistrement à l'écran par 'double clique de la souris' ou par 'entrer'

- **AT USER-COMMAND**.
- **AT PFnn**.

Bases de données logiques

Les BD logiques sont des écrans de sélections prédéfinis sur un ensemble de zones existantes dans différentes tables.
L'intérêt est :

1. ces BD's sont optimiser pour l'affichage et pour l'extraction des données (jointures index et tous ça…)
2. il suffit de déclaré dans **TABLES** les feuilles de l'arbre



Pour trouver la liste des BD logique prédéfinis il faut suivre ce chemin sur SAP R/3 Version 3.1 H
OUTILS / ABAP / DEVELOPPEMENT /ENVIRONEMENT DEV. / BD LOGIQUE ou utiliser la transaction SE36

Exemple :
GET *tab3* **.**

Divers

- Fichiers du serveur
  - Ouverture (OPEN DATASET)
  - Lecture (READ DATASET)
  - Ecriture (TRANSFER)
  - Fermeture (CLOSE DATASET)

- Fichiers locaux
  - Télédéchargement (WS_DOWNLOAD)
  - Téléchargement (WS_UPLOAD)

Il existe deux fonctions prédéfinis qui font la même chose que ces deux instructions.

- Includes (INCLUDE)

Permet d'appeler un autre programme

- Messages (MESSAGE)

Après avoir déclarer dans la 1ere ligne la classe de message à utiliser,
MESSAGE [S | I | W | E | A | X]*nnn* [**WITH** *TEXT-nnn*]**.**
nnn est un numéro de message prédéfinit dans la classe des messages utilisé.
On utilise le **WITH** *TEXT-nnn* quand on veut définir le corps du message nous même et dans ces cas on met un nombre bidon dans nnn.
[S | I | W | E | A | X] c'est le type du message
I - Information : il faut appuyer sur ENTER pour continuer
W – Warning : Correction possible
E - Erreur : Correction obligatoire
A - Abend : Transaction terminer
X - Exit : Transaction terminer avec un 'dump' c'est à dire plantage
S - Success : Message sur l'écran suivant

- Pour dessiné des boutons dans des écrans ABAP on crée un écran qui va contenir les boutons avec :

OUTILS / ABAP / OBJECT BROWSER ou bien transaction SE80
Sélectionner PROGRAMME *nom_prog* et LISTE DES …

Et dans le programme ABAP on fait
SET PF-STATUS '*nom_ecran* **'.**

SAP-script

On appelle SAP-script une partie de SAP qui permet la création des formulaires.
Pour créer un formulaire on fait OUTILS / TRAITEMENT TEXTE / FORMULAIRE sinon transaction SE71.
On saisie le nom et on clique sur créer.
Les formulaires ne sont pas inter mandant.
UTILITAIRES / COMPARAISON MANDANT : trouve les différences entre deux versions d'un même formulaire.
UTILITAIRES / COPIE MANDANT : copie un formulaire d'un mandant à un autre.
UTILITAIRES / ACTIVER DEBOGUEUR : pour le désactiver il faut faire terminer sur le débogueur.
Pour charger et décharger un formulaire il faut exécuter le programme 'RSTXSCRP'

Le principe est :
On subdivise la feuille (le rapport ou le **FORMULAIRE** que l'on veut faire) en rectangles de différentes dimensions, ces rectangles sont appelés des **FENETRES**, la création d'une fenêtre consiste en : lui donner un nom puis déterminer ces coordonnées sur le formulaire (position X, position Y, hauteur et largeur)**.**
Après avoir créer les fenêtres on détermine les données qu'il y'aura dans chaque fenêtre à l'aide des **Eléments de texte**. Puis On détermine les fenêtres qu'il y'aura dans chaque **PAGE**, sachant que le nombre minimal exigé de pages pour un formulaire est de 2 : **FIRST** et **NEXT**, et dans chaque page il est obligatoire de définir une fenêtre **MAIN.**
Enfin : Il faut savoir que SAP-script gère automatiquement le passage à la page suivante et le positionnement au bon endroit quand les données de la fenêtre MAIN ne tiennent pas sur la page courante.
On peut voir à quoi ressemble notre formulaire à tout moment en faisant UTILITAIRE / IMPRESSION TEST
PARAGRAPHES : sert à définir polices tabulations et autres paramètres associés, on définit un paragraphes par défaut pour le formulaire et on peut associé d'autres paragraphes aux fenêtres.
On peut définir une **ENTETE** de formulaire et insérer images et logos dedans.

Erreur ! Liaison incorrecte.

Eléments de texte

C'est à partir des elements de texte que les programmes ABAP et les formulaires SAP script communiquent,
Dans les éléments de texte d'une fenêtre on définit des paquets séparer par des :
/E *nom_élément*
ainsi en appelant un élément d'une fenêtre on affiche seulement le contenu de cette élément.
tout ce qui ne fait pas partie d'un element est afficher quelque soit l'element de la fenetre appellé.

/ bla bla **,, &***nom_var***&** *bla bla*
le '/' permet d'afficher tout ce qui suit tel quel. On peut afficher le contenu des variables du programme ABAP appellant simplement en les mettant entre deux &. ,,est remplacer par des tabulations.

/* *commentaire*

nom_paragraphe *bla bla* **&***nom-var***&**
affiche ce qui suit mais en utilisant la police et tout les trucs du paragraphe.

Il existe des variable prédéfinit SAP script comme **PAGE** qui renvoi le n° de la page courante et **SAPSCRIPT-FORMPAGES** qui renvoi le nombre totale de pages.

/ : *les commandes*
IF ELSEIF ENDIF CASE WHEN et **ENDCASE** même principe que ABAP sans point de fin.
Attention SAPSCRIPT ne gère pas bien les conditions compliqués

DEFINE &var1& = { 'VALEUR' / on s'en fou du type de données / &var2&

permet de définir des variable dans SAPSCRIPT

INCLUDE ('nom_texte') **OBJECT** (type_objet) **ID** (id_texte) **LANGUAGE** (langue)
Permet d'inclure du texte prédéfinit. Pour définir du texte utiliser la transaction SO10.

= considère que la ligne courante est la suite de la ligne précédente (car l'éditeur ne permet pas d'écrire indéfiniment sur la même ligne)

EXEMPLE :

```
/       BONJOUR
/E      ELEMENT1
/ :     DEFINE &var& = &SY-UNAME&
/*      pour dessiner un cadre
/ :     BOX XPOS '0.2' CM YPOS '0.4' CM WIDTH 19 CM HEIGHT 4 LN FRAME 10 TW
/       PARIS,  le  &SY-DATUM&
/E      ELEMENT2
/ :     IF &SOMME& > 0
/       voici le total :
=       &SOMME+1(8)&
/ :     ELSE
/       ERREUR
/ :     ENDIF
```

et la dernière commande de SAPSCRIPT est la plus compliquée
PERFORM appelle une fonction définit dans le programme appelant.

/ : PERFORM '(*nom_fonction***)' IN PROGRAM '***nom_prog***'**
/ : USING &var1& …
/ : CHANGING &var2& …
/ : ENDPERFORM

la fonction 'nom_fonction' doit exister dans le programme 'nom_prog'

FORM *nom_fonction* **TABLES INPUT_TABLE STRUCTURE ITCSY**
OUTPUT_TABLE STRUCTURE ITCSY .
* les tables INPUT_TABLE et OUTPUT_TABLE ont la même structure, elles sont composés de deux colonnes
* (NAME et VALUE) dans la première on trouve le nom de la variable en entrée ou en sortie et dans la seconde
* sa valeur

DATA *v1* **type '***type_var1***'.**
DATA *v2* **type '***type_var2***'.**

* récupération des données

```
LOOP AT INPUT_TABLE.
        CASE INPUT_TABLE-NAME .
                WHEN 'var1'.
                        MOVE INPUT_TABLE-VALUE TO V1.
        ENDCASE.
ENDLOOP.

LOOP AT OUTPUT_TABLE.
        CASE OUTPUT_TABLE-NAME .
                WHEN 'var2'.
                        MOVE V2 TO OUTPUT_TABLE-VALUE.
                        MODIFY OUTPUT_TABLE INDEX SY-TABIX.
        ENDCASE.
ENDLOOP.

ENDFORM.
```

ABAP => SAP-script

On peut piloter un formulaire avec un programme ABAP
Il y'a toujours un OPEN FORM et un CLOSE FORM.

Ouverture (OPEN_FORM)

- CALL FUNCTION '**OPEN_FORM**'
  - EXPORTING
    - DEVICE = 'PRINTER' ou 'TELEX' ou 'TELEFAX' ou 'ABAP' ou 'SCREEN'
    - DIALOG = 'X' ou ' '
    - FORM = ' '
    - LANGUAGE= SY-LANGU
    - OPTIONS = ' '
  - EXCEPTIONS
    - CANCELED = 1
    - DEVICE = 2
    - FORM = 3
    - OPTIONS = 4
    - UNCLOSED = 5
    - OTHERS = 6.

Exemple :

*** Remplissage des paramètres d'impression ***
*** ITCPO est aussi une structure système qui n'a pas besoin d'être déclare dans TABLES
*** structure complète en annexe.

```
ITCPO-TDPAGESLCT = SPACE .      "sélection des pages a imprimer"
ITCPO-TDNEWID = 'X' .           "créer un nouveau ordre spool"
ITCPO-TDCOPIES = 1 .            "Nombre de copie"
ITCPO-TDDEST = IMPRIME .        "imprimante
ITCPO-TDPREVIEW = 'X' .         "Aperçu avant impression"
ITCPO-TDIMMED = 'X' .           "impression immédiate"
ITCPO-TDDELETE = SPACE .        "supprimer le spool après impression"

  CALL FUNCTION 'OPEN_FORM'
     EXPORTING
*       APPLICATION             = 'TX'
*       ARCHIVE_INDEX           = ' '
*       ARCHIVE_PARAMS          = ' '
        DEVICE                  = 'PRINTER'
        DIALOG                  = SPACE
        FORM                    = 'ZTESTMI3 '
        LANGUAGE                = SY-LANGU
        OPTIONS                 = ITCPO
*    IMPORTING
*       LANGUAGE                =
*       NEW_ARCHIVE_PARAMS =
*       RESULT          =
      EXCEPTIONS
         CANCELED               = 1
         DEVICE                 = 2
```

FORM = 3
OPTIONS = 4
UNCLOSED = 5
OTHERS = 6.

Début (START_FORM)

- CALL FUNCTION **'START_FORM'**
  - EXPORTING
    - FORM = ' '
    - LANGUAGE= ' '
    - STARTPAGE = ' '
  - EXCEPTIONS
    - FORM = 1
    - FORMAT = 2
    - UNENDED = 3
    - UNOPENED = 4
    - UNUSED = 5
    - OTHERS = 6.

Exemple :
On s'en sert pas

Ecriture (WRITE_FORM)

- CALL FUNCTION '**WRITE_FORM**'
  - EXPORTING
    - ELEMENT = ' ' ou autre
    - WINDOW = 'MAIN' ou autre
    - FUNCTION = 'SET' ou 'APPEND' ou 'DELETE'
    - TYPE = 'BODY' ou 'TOP' ou 'BOTTOM'
  - EXCEPTIONS
    - ELEMENT = 1
    - FUNCTION = 2
    - TYPE = 3
    - UNOPENED = 4
    - UNSTARTED = 5
    - WINDOW = 6
    - OTHERS = 7.

Exemple :

```
*---------------------------------------------------------------------*
FORM W_MAIN.

  CLEAR SOMME.

  LOOP AT T_DONNEES WHERE KUNNR = T_CLIENT-KUNNR .
    CALL FUNCTION 'WRITE_FORM'
      EXPORTING
        ELEMENT         = 'E_DONNEES'
*       FUNCTION        = 'SET'
*       TYPE            = 'BODY'
```

```
          WINDOW               = 'MAIN'
*    IMPORTING
*         PENDING_LINES =
        EXCEPTIONS
            ELEMENT            = 1
            FUNCTION           = 2
            TYPE               = 3
            UNOPENED           = 4
            UNSTARTED          = 5
            WINDOW             = 6
            OTHERS             = 7.

    SOMME = SOMME + T_DONNEES-NETPR.
  ENDLOOP.

    CALL FUNCTION 'WRITE_FORM'
      EXPORTING
        ELEMENT                = 'TOTAL'
*       FUNCTION               = 'SET'
*       TYPE                   = 'BODY'
        WINDOW                 = 'MAIN'
*    IMPORTING
*         PENDING_LINES =
        EXCEPTIONS
            ELEMENT            = 1
            FUNCTION           = 2
            TYPE               = 3
            UNOPENED           = 4
            UNSTARTED          = 5
            WINDOW             = 6
            OTHERS             = 7.


ENDFORM.                       " W_MAIN
```

Fin (END_FORM)

- CALL FUNCTION **'END_FORM'**
  - EXCEPTIONS
    - UNOPENED = 1
    - OTHERS = 2.

Exemple :
On s'en sert pas

Fermeture (CLOSE_FORM)

- CALL FUNCTION '**CLOSE_FORM**'

```
*    IMPORTING
*         RESULT   =
*    TABLES
*         OTFDATA  =
```

- EXCEPTIONS
  - UNOPENED = 1
  - OTHERS = 2.

Voici un programme complet :

```
REPORT ZTESTMI3 NO STANDARD PAGE HEADING MESSAGE-ID ZZ.

TABLES : KNA1, VBEP, VBAK, VBAP, SADR, T001, ITCPO, ZVBAK.

DATA : BEGIN OF T_DONNEES OCCURS 0,
             MANDT LIKE VBAK-MANDT,
             KUNNR LIKE VBAK-KUNNR,
             VBELN LIKE VBAK-VBELN,
             BUKRS_VF LIKE VBAK-BUKRS_VF,
             WAERK LIKE VBAK-WAERK,
             ERDAT LIKE VBAP-ERDAT,
             WERKS LIKE VBAP-WERKS,
             MATNR LIKE VBAP-MATNR,
             KWMENG LIKE VBAP-KWMENG,
             NETPR LIKE VBAP-NETPR,
             MEINS LIKE VBAP-MEINS,
             MVGR5 LIKE VBAP-MVGR5,
             WMENG LIKE VBEP-WMENG,
END OF T_DONNEES.


DATA : BEGIN OF T_CLIENT OCCURS 0 ,
             KUNNR LIKE KNA1-KUNNR,
             NAME1 LIKE KNA1-NAME1 ,
             NAME3 LIKE KNA1-NAME3 ,
             NAME4 LIKE KNA1-NAME4 ,
             PSTLZ LIKE KNA1-PSTLZ ,
             ORT01 LIKE KNA1-ORT01 .
DATA : END OF T_CLIENT.

DATA : BEGIN OF T_SOC OCCURS 0 ,
              BUTXT LIKE T001-BUTXT,
              ADRNR LIKE SADR-ADRNR ,
              NAME3 LIKE SADR-NAME3 .
DATA : END OF T_SOC.

** variable de cumul *****
DATA : SOMME(10) TYPE P DECIMALS 2 VALUE 0.

** parametttres de selection pour imprimante **
PARAMETERS : IMPRIME LIKE ITCPO-TDDEST.

AT SELECTION-SCREEN OUTPUT .
PERFORM AVANT_SEL .

AT SELECTION-SCREEN .
PERFORM APRES_SEL .

START-OF-SELECTION.
 SELECT * FROM ZVBAK.
  DELETE ZVBAK.
```

```
 ENDSELECT.
GET VBEP.

  MOVE          VBAK-KUNNR TO T_DONNEES-KUNNR.
  MOVE        VBAK-BUKRS_VF TO T_DONNEES-BUKRS_VF.
  MOVE          VBAK-WAERK TO  T_DONNEES-WAERK.
  MOVE          VBAK-VBELN TO  T_DONNEES-VBELN.
  MOVE          VBAP-ERDAT  TO T_DONNEES-ERDAT.
  MOVE          VBAP-WERKS  TO T_DONNEES-WERKS.
  MOVE          VBAP-MATNR TO  T_DONNEES-MATNR.
  MOVE          VBAP-KWMENG TO T_DONNEES-KWMENG.
  MOVE          VBAP-NETPR  TO T_DONNEES-NETPR.
  MOVE          VBAP-MEINS  TO T_DONNEES-MEINS.
  MOVE          VBAP-MVGR5  TO T_DONNEES-MVGR5.
  MOVE          VBEP-WMENG  TO T_DONNEES-WMENG.

  APPEND T_DONNEES.

END-OF-SELECTION.

  SORT T_DONNEES.


  LOOP AT T_DONNEES .
    INSERT ZVBAK FROM T_DONNEES.
  ENDLOOP.

  PERFORM GET_DATA_CLIENT .
  PERFORM GET_DATA_SOC .
  READ TABLE T_SOC .
  LOOP AT T_CLIENT .
    PERFORM O_FORM .
    PERFORM W_ADR1 .
    PERFORM W_ADR2 .
    PERFORM W_TXT  .
    PERFORM W_MAIN .
    PERFORM C_FORM .
  ENDLOOP.

*&---------------------------------------------------------------------*
*&      Form  WRITE_ADR
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*----------------------------------------------------------------------*
FORM W_ADR1.

  CALL FUNCTION 'WRITE_FORM'
       EXPORTING
            ELEMENT = ' '
            WINDOW  = 'ADRESSE'
*    IMPORTING
*         PENDING_LINES =
  EXCEPTIONS
       ELEMENT       = 1
       FUNCTION      = 2
       TYPE          = 3
```

```
       UNOPENED      = 4
       UNSTARTED     = 5
       WINDOW        = 6
       OTHERS        = 7.
ENDFORM.                               " W_ADR1

*&---------------------------------------------------------------------*
*&      Form  W_ADR2
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*----------------------------------------------------------------------*
FORM W_ADR2.

  CALL FUNCTION 'WRITE_FORM'
       EXPORTING
            ELEMENT = ' '
            WINDOW  = 'ADRESSE2'
*    IMPORTING
*         PENDING_LINES =
  EXCEPTIONS
       ELEMENT       = 1
       FUNCTION      = 2
       TYPE          = 3
       UNOPENED      = 4
       UNSTARTED     = 5
       WINDOW        = 6
       OTHERS        = 7.

ENDFORM.                               " W_ADR2
*&---------------------------------------------------------------------*
*&      Form  O_FORM
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*----------------------------------------------------------------------*
FORM O_FORM.
*** Remplissage des paremettres d'ipression  ***
ITCPO-TDPAGESLCT = SPACE . "selection des pages a imprimer"
ITCPO-TDNEWID = 'X' ."creer un nouveau ordre spool"
ITCPO-TDCOPIES = 1 . "Nombre de copie"
ITCPO-TDDEST = IMPRIME . "imprimante
ITCPO-TDPREVIEW = 'X' . "Appercu avant impression"
ITCPO-TDIMMED = 'X' ."impression imediate"
ITCPO-TDDELETE = SPACE . "supprimer le spool apres impression"

  CALL FUNCTION 'OPEN_FORM'
      EXPORTING
*         APPLICATION        = 'TX'
*         ARCHIVE_INDEX      = ' '
*         ARCHIVE_PARAMS     = ' '
           DEVICE             = 'PRINTER'
           DIALOG             = SPACE
           FORM               = 'ZTESTMI3 '
           LANGUAGE           = SY-LANGU
```

```
           OPTIONS            = ITCPO
*    IMPORTING
*         LANGUAGE           =
*         NEW_ARCHIVE_PARAMS =
*         RESULT             =
       EXCEPTIONS
            CANCELED           = 1
            DEVICE             = 2
            FORM               = 3
            OPTIONS            = 4
            UNCLOSED           = 5
            OTHERS             = 6.

ENDFORM.                                " O_FORM
*&---------------------------------------------------------------------*
*&      Form  W_MAIN
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*----------------------------------------------------------------------*
FORM W_MAIN.

  CLEAR SOMME.

  LOOP AT T_DONNEES WHERE KUNNR = T_CLIENT-KUNNR .
    CALL FUNCTION 'WRITE_FORM'
        EXPORTING
         ELEMENT       = 'E_DONNEES'
*         FUNCTION      = 'SET'
*         TYPE          = 'BODY'
            WINDOW        = 'MAIN'
*    IMPORTING
*         PENDING_LINES =
         EXCEPTIONS
              ELEMENT       = 1
              FUNCTION      = 2
              TYPE          = 3
              UNOPENED      = 4
              UNSTARTED     = 5
              WINDOW        = 6
              OTHERS        = 7.

     SOMME = SOMME + T_DONNEES-NETPR.
  ENDLOOP.

      CALL FUNCTION 'WRITE_FORM'
          EXPORTING
           ELEMENT       = 'TOTAL'
*         FUNCTION      = 'SET'
*         TYPE          = 'BODY'
           WINDOW        = 'MAIN'
*    IMPORTING
*         PENDING_LINES =
          EXCEPTIONS
               ELEMENT       = 1
               FUNCTION      = 2
               TYPE          = 3
```

```
               UNOPENED      = 4
               UNSTARTED     = 5
               WINDOW        = 6
               OTHERS        = 7.


ENDFORM.                               " W_MAIN
*&---------------------------------------------------------------------*
*&      Form  C_FORM
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*----------------------------------------------------------------------*
FORM C_FORM.
  CALL FUNCTION 'CLOSE_FORM'
*    IMPORTING
*         RESULT   =
*    TABLES
*         OTFDATA  =
       EXCEPTIONS
            UNOPENED = 1
            OTHERS   = 2.
ENDFORM.                               " C_FORM
*&---------------------------------------------------------------------*
*&      Form  GET_DATA_CLIENT
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*----------------------------------------------------------------------*
FORM GET_DATA_CLIENT.

  LOOP AT T_DONNEES.
    ON CHANGE OF T_DONNEES-KUNNR.
      SELECT KNA1~KUNNR KNA1~NAME1 KNA1~NAME3 KNA1~NAME4 KNA1~PSTLZ
     KNA1~ORT01 FROM  KNA1
      APPENDING CORRESPONDING FIELDS OF TABLE T_CLIENT
          WHERE KUNNR = T_DONNEES-KUNNR .
    ENDON.

  ENDLOOP.

ENDFORM.                               " GET_DATA_CLIENT
*&---------------------------------------------------------------------*
*&      Form  GET_DATA_SOC
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*----------------------------------------------------------------------*
FORM GET_DATA_SOC.

  LOOP AT T_DONNEES.
    ON CHANGE OF T_DONNEES-BUKRS_VF.
      SELECT T001~BUTXT
```

```
                        T001~ADRNR  SADR~NAME3
    FROM ( T001 INNER JOIN SADR ON  SADR~ADRNR = T001~ADRNR )
       APPENDING CORRESPONDING FIELDS OF TABLE T_SOC
         WHERE T001~BUKRS = T_DONNEES-BUKRS_VF .
    ENDON.

  ENDLOOP.

ENDFORM.                                " GET_DATA_SOC
*&---------------------------------------------------------------------*
*&      Form  AFFICHE
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*----------------------------------------------------------------------*
FORM AFFICHE TABLES INPUT_TABLE STRUCTURE ITCSY
                    OUTPUT_TABLE STRUCTURE ITCSY.

  TABLES : USR03.
  DATA : BEGIN OF V1,
           UNAME LIKE SY-UNAME,
         END OF V1.

  DATA : BEGIN OF V2,
          NAME1  LIKE USR03-NAME1,
          NAME2  LIKE USR03-NAME2,
        END OF V2.

* recuperation
  LOOP AT INPUT_TABLE.
    CASE INPUT_TABLE-NAME.
      WHEN 'VAR1'.
        MOVE INPUT_TABLE-VALUE TO V1-UNAME.
*      WRITE : /1 V1-UNAME.
    ENDCASE.
  ENDLOOP.

  SELECT SINGLE USR03~NAME1 USR03~NAME2 FROM USR03  INTO V2
      WHERE BNAME = V1-UNAME.
*     MOVE-CORRESPONDING TO V2.
*  ENDSELECT.

  LOOP AT OUTPUT_TABLE.
    CASE OUTPUT_TABLE-NAME.
      WHEN 'PRENOM'.
        MOVE V2-NAME1 TO OUTPUT_TABLE-VALUE.
        MODIFY OUTPUT_TABLE INDEX SY-TABIX .

      WHEN 'NOM'.
        MOVE V2-NAME2 TO OUTPUT_TABLE-VALUE.
        MODIFY OUTPUT_TABLE INDEX SY-TABIX .
    ENDCASE.

  ENDLOOP.

ENDFORM.                                " AFFICHE
*&---------------------------------------------------------------------*
```

```
*&      Form  W_TXT
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*----------------------------------------------------------------------*
FORM W_TXT.

  CALL FUNCTION 'WRITE_FORM'
       EXPORTING
            ELEMENT = 'E1'
            WINDOW  = 'SIGNE'
*    IMPORTING
*         PENDING_LINES =
  EXCEPTIONS
       ELEMENT       = 1
       FUNCTION      = 2
       TYPE          = 3
       UNOPENED      = 4
       UNSTARTED     = 5
       WINDOW        = 6
       OTHERS        = 7.
ENDFORM.                               " W_TXT
*&---------------------------------------------------------------------*
*&      Form  AVANT_SEL
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*----------------------------------------------------------------------*
FORM AVANT_SEL.

LOOP AT SCREEN.

 IF ( ( SCREEN-GROUP4 = '001' ) OR ( SCREEN-GROUP4 = '002' )
    OR ( SCREEN-GROUP4 = '003' ) )
    AND ( ( SCREEN-GROUP3 = 'HGH' ) OR ( SCREEN-GROUP3 = 'VPU' ) ) .
  SCREEN-ACTIVE = 0 .
  MODIFY SCREEN.
 ENDIF.

 IF  ( ( SCREEN-GROUP4 = '004' ) OR ( SCREEN-GROUP4 = '005' )
     OR ( SCREEN-GROUP4 = '006' ) OR ( SCREEN-GROUP4 = '007' )
     OR ( SCREEN-GROUP4 = '008' ) OR ( SCREEN-GROUP4 = '009' )
     OR ( SCREEN-GROUP4 = '010' ) OR ( SCREEN-GROUP4 = '011' )
     OR ( SCREEN-GROUP4 = '012' ) OR ( SCREEN-GROUP4 = '013' )
     OR ( SCREEN-GROUP4 = '014' ) ) .

  SCREEN-ACTIVE = 0 .
  MODIFY SCREEN.
 ENDIF.

ENDLOOP.

DD_VBELN-HIGH   = '!'  .
DD_VBELN-LOW    = '!'  .
DD_VBELN-SIGN   = 'I'  .
```

```
DD_VBELN-OPTION = 'EQ' .
APPEND DD_VBELN .

ENDFORM.                    " AVANT_SEL
*&---------------------------------------------------------------------*
*&      Form  APRES_SEL
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*----------------------------------------------------------------------*
FORM APRES_SEL.
IF DD_VBELN-LOW = '!' OR DD_VBELN-HIGH = '!'  .
MESSAGE E015 WITH TEXT-015.
ENDIF.

MESSAGE I014 WITH TEXT-014.
MESSAGE S013 WITH TEXT-013.


ENDFORM.                    " APRES_SEL
```

Voici des trucs du HELP SAP :

```
 Classification of Key Words by Type

   Declarative key words

     These declare or define the following data objects:

     CONSTANTS            Definition of constants

     CREATE               Generates an external object

     DATA                 Internal fields, field strings,
                           internal tables, common areas

     FIELD-GROUPS         Field groups for extract sorting

     INCLUDE              Include a structure or type

     FIELD-SYMBOLS        Variable work fields

     LOCAL                Local use of global fields in subroutines

     PARAMETERS           Internal fields set by the user after program start

     RANGES               Selection criterion that can only be set internally

     SELECT-OPTIONS       Selection criterion set by the user after program
                           start for data selection purposes

     SELECTION-SCREEN     Selection screen layout

     STATICS              Static data objects in subroutines
```

TABLES	SAP tables
TYPE-POOL	Combine types and constants together in a type group
TYPE-POOLS	Include a type group
TYPES	Type definitions

Event key words

These specify a processing event:

AT LINE-SELECTION	After line selection
AT PFn	After pressing a function key
AT SELECTION-SCREEN	After input on the selection screen
AT USER-COMMAND	After input in the command field
END-OF-PAGE	After output of the last line of a page
END-OF-SELECTION	At the end of the program
GET	After reading a new record from a logical database
INITIALIZATION	Before output of the selection screen (not in background processing)
START-OF-SELECTION	At the beginning of the program
TOP-OF-PAGE	At the start of a new page
TOP-OF-PAGE DURING LINE-SELECTION	At the start of a new page in a details list

Control key words

These control the processing flow:

AT FIRST ... ENDAT	Execute processing block within a LOOP before processing single lines
AT NEW ... ENDAT	Control group end during LOOP
AT END OF ... ENDAT	Control group end in LOOP
AT LAST ... ENDAT	Execute processing block within a LOOP after processing single lines
AT fg ... ENDAT	Processing specific to record type in LOOP
CALL	Call processing (program, function module, screen)
CASE ... WHEN ... ENDCASE	Case distinction

CHECK	Selection condition, leave loops and subroutines
CONTINUE	Exit current loop pass within a DO, WHILE, LOOP or SELECT loop
DO ... ENDDO	Loop processing
EXEC SQL ... ENDEXEC	Execute a Native SQL statement
EXIT	Leave loops or subroutines
FORM ... ENDFORM	Definition of a subroutine
IF ... ELSE ... ENDIF	Conditional processing

The following relational operators can be used:

=,	EQ,	(equal)
<>, ><,	NE	(not equal)
>,	GT	(greater than)
<,	LT	(less than)
>=, =>	GE	(greater than or equal)
<=, =<	LE	(less than or equal)
O, Z, M		(bit comparison)
CO, CN, CA, NA, CS, NS ,CP, NP		(string comparison)
BETWEEN...AND		(range condition)
IS INITIAL		(check initial value)
IS REQUESTED		(existence of actual parameter)
IN		(set condition)

LEAVE	Leave program processing, go to a transaction, list or menu
LOOP ... ENDLOOP	Loop on a table or extract dataset
MODULE ... ENDMODULE	Definition of a dialog module
ON CHANGE OF ... ENDON	Processing on field change
PERFORM ... USING	Call a subroutine
SELECT ... ENDSELECT	Read database tables
STOP	End selection
WHILE ... ENDWHILE	Loop processing

Operational key words

These perform operations on the data at certain times (events) and under certain conditions:

ADD	Addition of fields
ADD-CORRESPONDING	Addition of matching fields in field strings

```
APPEND                Append an entry to an internal table

ASSIGN                Assign a field to a field symbol

AUTHORITY-CHECK       Check authorization

BACK                  Return to top of page

BREAK-POINT           Set a breakpoint

CALL                  Call a transaction, screen, dialog module
                       or function module

CLEAR                 Reset to initial value

COLLECT               Include an entry in an internal table

COMMIT                Perform database changes

COMPUTE               Calculation
```

The following functions can be used:

```
ABS      Absolute amount
ACOS     Arc cosine
ASIN     Arc sine
ATAN     Arc tangent
CEIL     Smallest integer value
COS      Cosine
COSH     Hyperbola cosine
EXP      Exponential function (base e)
FLOOR    Greatest integer value
FRAC     Decimal part
LOG      Natural logarithm (base e)
LOG10    Logarithm (base 10)
SIGN     Sign
SIN      Sine
SINH     Hyperbola sine
SQRT     Square root
STRLEN   Length of a character string
TAN      Tangent
TANH     Hyperbola tangent
TRUNC    Whole number part
```

```
CONCATENATE           Join character strings

CONDENSE              Condense a field

DELETE                Delete lines

DIVIDE                Division of fields

DIVIDE-CORRESPONDING  Division of matching fields in field strings

EDITOR-CALL           Call the SAP editor

EXPORT                Export data objects
```

EXTRACT	Write a record to an internal dataset
FILL	Fill a field string
FORMAT	Set format for list output
FREE	Release memory space reserved for an internal table
GET	Read parameters, cursor, etc.
HIDE	Note line-related contents
IMPORT	Import data objects
INCLUDE	Insert program components
INSERT	Insert operations
MESSAGE	Output messages
MODIFY	Change a table or list lines
MOVE	Transfer field contents
MOVE-CORRESPONDING	Transfer matching fields in field strings
MULTIPLY	Multiplication of fields
MULTIPLY-CORRESPONDING	Multiplication of matching fields on field strings
NEW-LINE	Set a new line (place cursor on a new line)
NEW-PAGE	Set a new page (place cursor on a new page)
POSITION	Position on a variable column
PRINT-CONTROL	Determine print format
PUT	Trigger GET event
READ	Read tables and databases, (direct) read of list lines
REFRESH	Refresh internal table of a screen
REPLACE	Replace character strings
RESERVE	Reserve output lines on current page
SEARCH	Find character strings in tables
SET	Set processing parameters
SHIFT	Move field contents
SKIP	Set blank lines, position on line
SORT	Sort an internal table or extract dataset

```
    SPLIT                Split a character string

    SUBMIT               Start another program

    SUBTRACT             Subtraction of fields

    SUBTRACT-CORRESPONDING
                         Subtraction of matching fields in field strings

    SUM                  Sum during LOOP

    TRANSFER             Sequential data output

    TRANSLATE            Convert to upper/lower case

    UPDATE               Online database update

    ULINE                Underscore

    UNPACK               Unpacked with leading zeros

    WINDOW               Define screen section

    WRITE                Output


  Comments

    *                    In first column: Whole line is comment

    "                    In any column: Remainder of line is comment


  Include other program components

    INCLUDE              Inserts the program component stored under the
                          specified name at the current position

    PERFORM form(prog)   Calls the subroutine form of the program prog
                         (external PERFORM)

Retour déb.->
    Structure documentation ABAP/4
```