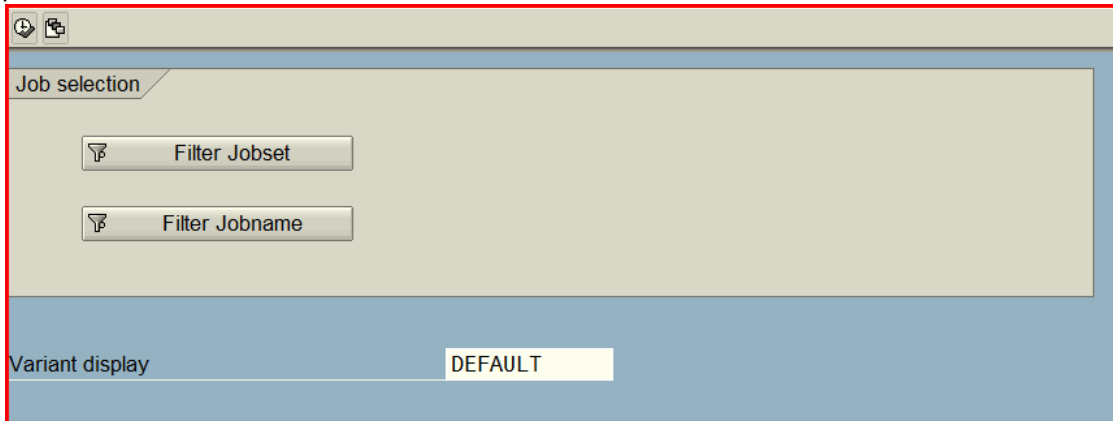


Utilisation de la sélection dynamique dans un programme ABAP

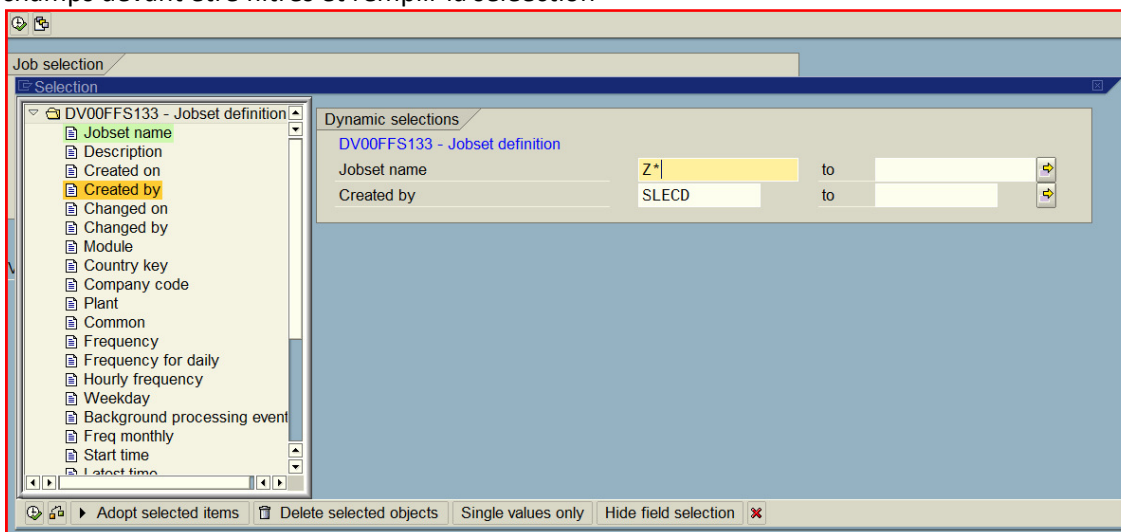
Note : Version utilisé pour les tests : 4.6C

But : Laisser l'utilisateur mettre les filtres qu'il veut pour la sélection des données. Ces filtres portent ici sur 2 tables. Ce sont sur ces tables que la sélection de donnée porte dans l'ABAP...

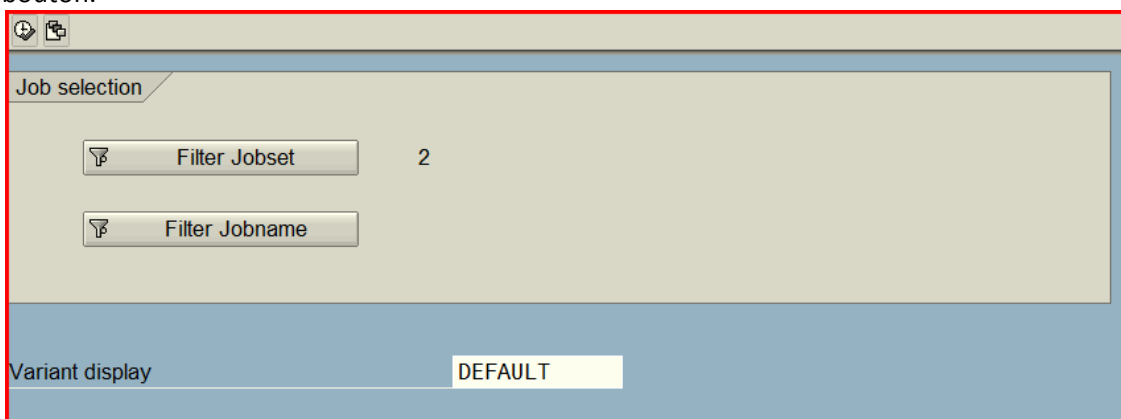
Au départ de la transaction, on a 2 boutons qui permettent d'appeler les dialogues de sélection dynamique pour les 2 tables :



Les 2 boutons fonctionnant de la même manière, nous ne verrons que le premier :
Après avoir cliquer sur le 1^{er} bouton, le dialogue standard s'ouvre. On étend le premier nœud (avec la description de la table) pour afficher la liste des champs de ladite table. Il ne reste qu'à double cliquer sur les champs devant être filtrés et remplir la sélection



Au retour du dialogue de sélection dynamique, on affiche le nombre de champs sélectionné à coté du bouton.



Voyons maintenant le code ABAP (la définition des données globales est donnée à la fin) :

Ecran de sélection (utilisation d'un pushbutton dans une ligne d'écran de sélection standard):

```
SELECTION-SCREEN: BEGIN OF BLOCK b01 WITH FRAME TITLE text-tt1,
                  SKIP,
                  BEGIN OF LINE,
                  PUSHBUTTON 05(20) seloptjs USER-COMMAND seljs,
                  COMMENT    30(15) selcntjs,
                  END OF LINE,
                  SKIP,
                  BEGIN OF LINE,
                  PUSHBUTTON 05(20) seloptjn USER-COMMAND seljn,
                  COMMENT    30(15) selcntjn,
                  END OF LINE,
                  SKIP.
SELECTION-SCREEN: END OF BLOCK b01,
```

Les textes des boutons est fait dans l'événement programme :

```
INITIALIZATION.
  MOVE : '@4G@ Filter Jobset'(bt1)  T0 seloptjs,
        '@4G@ Filter Jobname'(bt2) T0 seloptjn,
```

Les compteurs selcntjs et selcntjn sont mise à jour dans le « PBO » de l'écran de sélection (par l'utilisation de 2 variables global v_active pour le nombre de champs filtré et v_sav_ucomm qui permet de savoir sur quelle table le filtre vient d'être appliqué) :

```
AT SELECTION-SCREEN OUTPUT.
CASE v_sav_ucomm.
  WHEN 'SELJS'.
    IF v_active IS INITIAL.
      CLEAR: selcntjs.
    ELSE.
      WRITE v_active T0 selcntjs LEFT-JUSTIFIED.
    ENDIF.
  WHEN 'SELJN'.
    IF v_active IS INITIAL.
      CLEAR: selcntjn.
    ELSE.
      WRITE v_active T0 selcntjn LEFT-JUSTIFIED.
    ENDIF.
```

Gestion de la sélection dynamique :

Ceci ce fait dans le « PAI » de l'écran de sélection en contrôlant le code ok (à noter la sauvegarde du code ok pour l'utilisation dans le « PBO » afin de mettre à jour le compteur...) :

```
AT SELECTION-SCREEN.
  CLEAR v_active.
  CASE sy-ucomm.
    WHEN 'SELJS'.
      PERFORM f_fill_selection TABLES i_rangjs
                                USING 'ZBJOBSET'
                                       v_selidjs.
    WHEN 'SELJN'.
      PERFORM f_fill_selection TABLES i_rangjn
                                USING 'ZBJOBNAME'
                                       v_selidjn.
  ENDCASE.
  v_sav_ucomm = sy-ucomm.
```

Voyons maintenant comment appelé l'écran standard de sélection dynamique via l'utilisation de la routine F_FILL_SELECTION avec plusieurs paramètre : le nom de la table à filtré, la variable pour l'ID de la sélection dynamique et la table qui contiendra les filtres.

```

FORM f_fill_selection TABLES pi_ranges LIKE i_rangjs
                        USING p_name
                        p_selid TYPE rsdynsel-selid.

*--- Prepare free selection on table
REFRESH i_tables.
w_tables-prim_tab = p_name.
APPEND w_tables TO i_tables.

IF p_selid IS INITIAL.
*--- Init free selection dialog
CALL FUNCTION 'FREE_SELECTIONS_INIT'
EXPORTING
    expressions = i_expr
IMPORTING
    selection_id = p_selid
    expressions = i_expr
TABLES
    tables_tab = i_tables
EXCEPTIONS
    OTHERS = 1.
ENDIF.

*--- Display free selection dialog
CALL FUNCTION 'FREE_SELECTIONS_DIALOG'
EXPORTING
    selection_id = p_selid
    title = 'Selection'
    status = 1
    as_window = 'X'
IMPORTING
    expressions = i_expr
    field_ranges = pi_ranges[]
    number_of_active_fields = v_active
TABLES
    fields_tab = i_fields
EXCEPTIONS
    OTHERS = 1.

ENDFORM.                    " f_fill_selection

```

Observons le contenu de quelques éléments à l'exécution, lorsqu'on à appuyé sur le premier bouton :

Internal table	i_tables	Type	STANDARD	Format	E
1	PRIM_TAB	SEC_TAB		PRIM_FNAME	
1	ZBJOBSET				

Après l'appel de la première fonction, la variable P_SELID vaut DS000001

Après l'appel de la 2^{ème} fonction, et remplissage comme donnée en exemple plus haut, on a :

Field names	Field contents
i_expr	Table[1x40]
i_fields	Table[2x76]
pi_ranges[]	Table[1x40]
v_active	2
SY-SUBRC	0
SY-TABIX	14
SY-DBCNT	1

Dont la table avec la liste des champs filtrés

Internal table			
i_fields		Type	Format
1	TABlename	FIELDNAME	TYPE
1	ZBJOBSET	ZJJOBSET	
2	ZBJOBSET	ZJUSER	

Celle contenant la table des expressions correspondantes à la sélection

```
*--- Display free selection dialog
CALL FUNCTION 'FREE_SELECTIONS_DIALOG'
```

Structured field				
i_expr[1]				
Length (in bytes)				
40				
No.	Component name	Ty...	Lngh	Contents
1	TABlename	C	30	ZBJOBSET
2	EXPR_TAB	h	8	Table[3x132]

Internal table			
i_expr[1]-EXPR_TAB		Type	Format
1	LOGOP ARITY	FIELDNAME	OPTION LOW
1	AND	2	
2		0 ZJJOBSET	CP Z*
3		0 ZJUSER	EQ SLECD

Et celle contenant l'équivalent sous forme de ranges

Internal table		Type	Format
pi_ranges[]		STANDARD	E
1	TABlename	FRANGE_T	
1	ZBJOBSET	Table[2x40]	

Internal table		Type	Format
pi_ranges[1]-FRANGE_T		STANDARD	E
1	FIELDNAME	SELOPT_T	
1	ZJJOBSET	Table[1x93]	
2	ZJUSER	Table[1x93]	

Internal table		Type	Format
pi_ranges[1]-FRANGE_T[1]-SELOP		STANDARD	E
1	SIGN OPTION	LOW	HIGH
1	I	CP Z*	

Récupération de données :

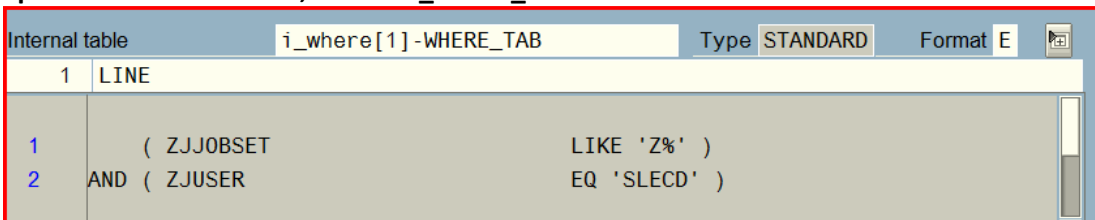
A partir de la table des ranges, on peut construire la clause WHERE de notre instruction SELECT avec le code suivant (contenu dans une routine) :

```
FORM f_build_selection.

  FIELD-SYMBOLS <wa> TYPE rsdswhere.

*--- Selection on ZBJOBSET
REFRESH i_where.
CALL FUNCTION 'FREE_SELECTIONS_RANGE_2_WHERE'
  EXPORTING
    field_ranges = i_rangjs
  IMPORTING
    where_clauses = i_where.
READ TABLE i_where INTO w_where WITH KEY tablename = 'ZBJOBSET'.
APPEND LINES OF w_where-where_tab TO i_where_tab.
...
ENDFORM.
```

Après exécution du code, la table i_where_tab contient ceci :



LINE	WHERE CLAUSE
1	(ZJJOBSET LIKE 'Z%')
2	AND (ZJUSER EQ 'SLECD')

Qui est directement utilisable comme cela :

```
*--- Fill data table
SELECT * INTO CORRESPONDING FIELDS OF TABLE i_data
  FROM zbjoset
  WHERE (i_where_tab).
```

(A noter que dans ce code, on agrège les clauses des 2 tables ensemble pour une sélection sur une vue des 2 tables directement...)

Définition des principales données globales utiles.

On doit tout d'abords déclarer le TYPE-POOLS RSDS.

```
*--- Internal table
DATA : i_rangjs TYPE rsds_trange,
      i_where_tab TYPE TABLE OF rsdswhere,
      i_tables TYPE TABLE OF rsdstabs,
      i_fields TYPE TABLE OF rsdsfields,
      i_expr TYPE rsds_texpr,
      i_where TYPE rsds_twhere,
      ...

*--- Working area
      w_where TYPE rsds_where,
      w_tables TYPE rsdstabs,
      ...

*--- Data
      v_repid TYPE syrepid,
      v_selidjs TYPE rsdynsel-selid,
      ...
      v_active TYPE i,
      v_sav_ucomm TYPE syucomm.
```