BPS Function Module Parameter Reference Guide

INDEX

1) Exit Planning Functions - function modules

a) Import Parameters:

- I_AREA
- I_PLEVEL
- I_METHOD
- I_PARAM
- I_PACKAGE

- IT_EXITP
- ITO_CHASEL
- ITO_CHA
- ITO_KYF

i) I_AREA

This is the technical name of the area from which the function is called.

ii) I_PLEVEL

This is the technical name of the level from which the function is called

iii) I_METHOD

This is the technical name of the planning method called

iv) I_PARAM

This is the technical name of the parameter group used

v) I_PACKAGE

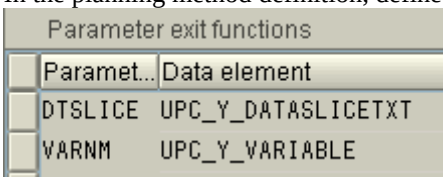
This is the technical name of the package from which the function is called

vi) IT_EXITP

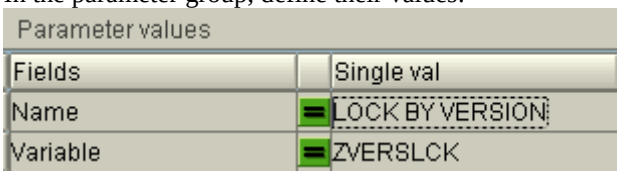
These are the parameter values provided in the parameter group.
The values are stored in a table. This table contains two fields – "PARNM" and "CHAVL".
PARNM is the name of the parameter, as specified in the planning method definition.
CHAVL is the value of the parameter, as specified in the planning parameter group.

For example, there is a function with two parameters, the first parameter is the name of the data slice and the second parameter is the name of the variable used in the data slice.

In the planning method definition, define the parameters:



In the parameter group, define their values:

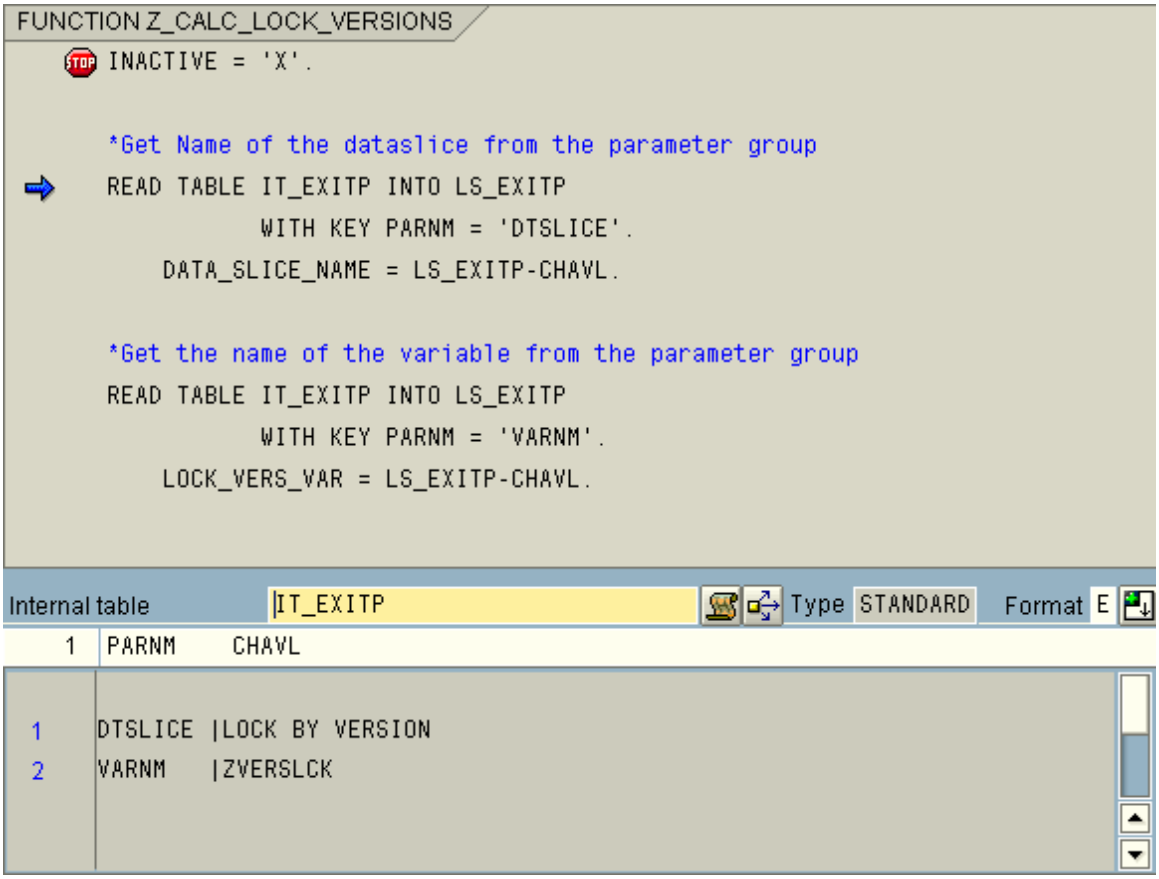


In the function, this is how to access the data values:

```
*Get Name of the dataslice from the parameter group
READ TABLE IT_EXITP INTO LS_EXITP
           WITH KEY PARNM = 'DTSLICE'.
    DATA_SLICE_NAME = LS_EXITP-CHAVL.

*Get the name of the variable from the parameter group
```

```
    READ TABLE IT_EXITP INTO LS_EXITP
               WITH KEY PARNM = 'VARNM'.
                  LOCK_VERS_VAR = LS_EXITP-CHAVL.
     ---------------------------------------------------------------------
-
```

Here is the trace of how these values are read at runtime:

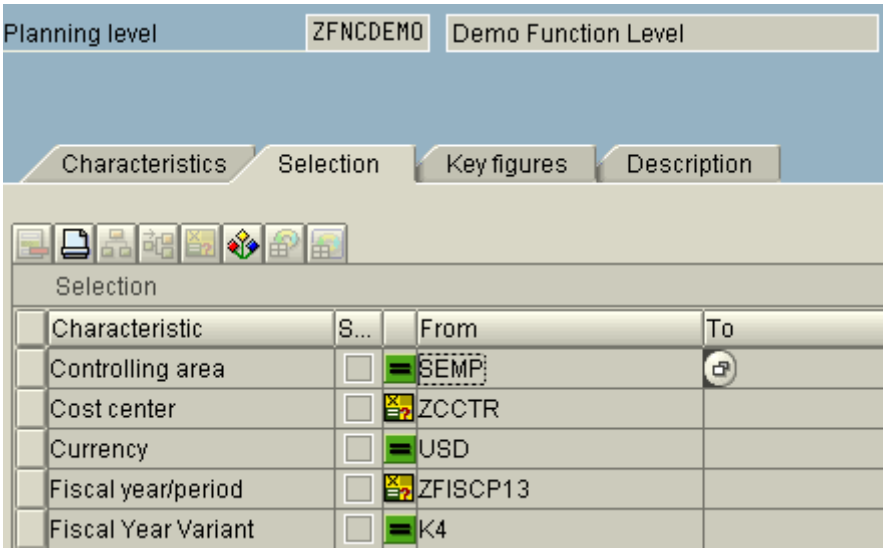


vii) ITO_CHASEL

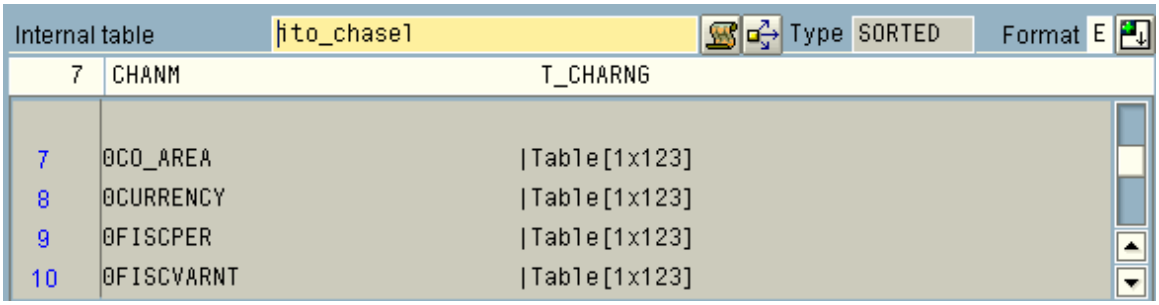
A table with the selection values for characteristics. This table contains all the selection criteria contained in the level, and additional restrictions by the data packages from the characteristics not specified in the planning method as “fields to be changed”.

When a function is called all of the characteristic values ranges, specified in the level and package are transferred to ITO_CHASEL:

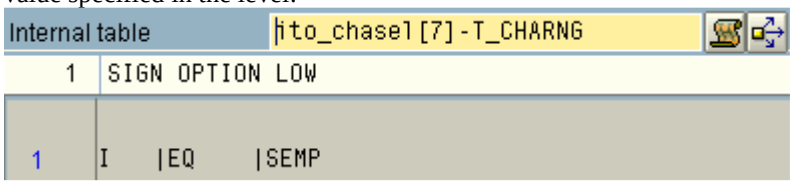
For example, the example level contains the following selection criteria:



Notice that the selections for controlling area, currency and fiscal year variant are static. When a function in this level is called, those static values are placed in the field ITO_CHASEL:
Here is the table ITO_CHASEL with all of the characteristics listed and a corresponding selection table for each (T_CHARNG).



Here is the selection table entry for 0CO_AREA, notice that this entry corresponds to the value specified in the level:



The table ITO_CHASEL also contains the selection for the current data packet of the fields not specified to be changed.

How many sub-packets should be created with which records depends on the entries in the list of *'fields to be changed'*. You can see this list on the detail screen when you create the planning function.

The following example shows a scheme used to create the sub-packets. Sample data:

Cost Center	Fiscal Period	Amount
1000	200401	20
1000	200402	10
1001	200401	5

1001	200402	7

- The list of *'fields to be changed'* contains the characteristic for fiscal Period (0FISCPER).

All rows are bundled in one packet that share the same characteristic values – except for the cost center and key figures – hence, in our example there would be two packets with two rows each.

1000	200401	5
1000	200402	7

1001	200401	20
1001	200402	10

In each packet the characteristic Cost Center has only one value. This is feasible for operations that use different fiscal periods. (e.g., spreading amounts…)

Internal table	ito_chasel[5]-T_CHARNG
1	SIGN OPTION LOW
1	I |EQ |1100-0342

However, the entry for fiscal period contains the range of values specified in the level, since this characteristic was not included in the fields to be changed:

Internal table	ito_chasel[9]-T_CHARNG
1	SIGN OPTION LOW HIGH
1	I |BT |2004001 <2008012

Referencing data from ito_chasel in a planning function is a two step process. First, get the line of the characteristic selection table (ito_chasel) for the desired characteristic. Then read the individual selection table for that characteristic.

The following ABAP code sample gets the cost center value for a given data package:

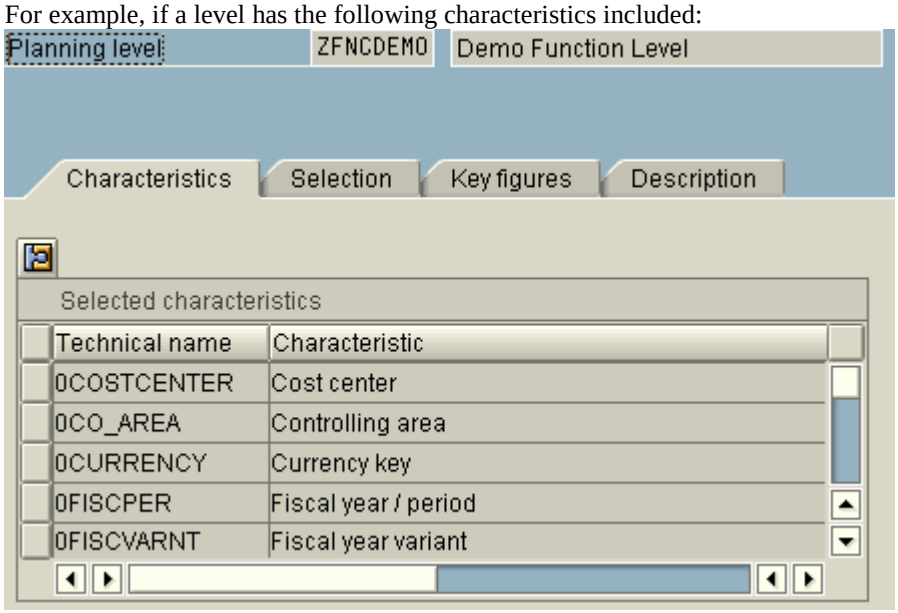
```
Data: L_COSTCENTER type /BIO/OICOSTCENTER,  "Cost center value
      LS_CHASEL TYPE  UPC_YS_CHASEL, "Line of characteristic selection table
      LT_CHARNG TYPE UPC_YT_CHARNG,  "Selection table for char value
```

```
LS_CHARNG TYPE UPC_YS_CHARNG,     "line of selection table for char value

        *Get the cost center of the data begin processed
        READ TABLE ITO_CHASEL
            WITH KEY CHANM = '0COSTCENTER'
            INTO LS_CHASEL.

        *Get the selection criteria cost center to be processed
        READ TABLE LS_CHASEL-T_CHARNG INDEX 1 INTO LS_CHARNG.

        *Read the cost center from the selection criteria
         L_COSTCENTER = LS_CHARNG-LOW.
```

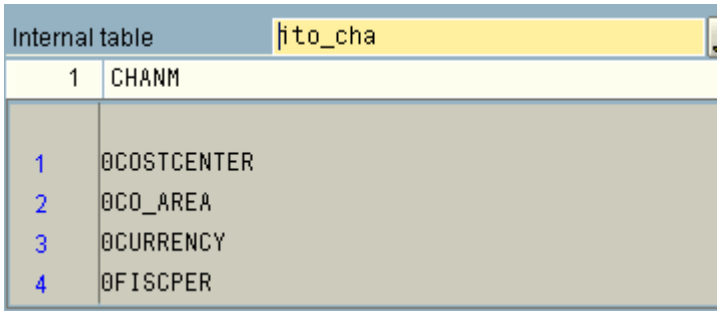
viii) ITO_CHA

Contains the technical names of all the characteristics processed in the function. This corresponds to the characteristics included in the level.

For example, if a level has the following characteristics included:



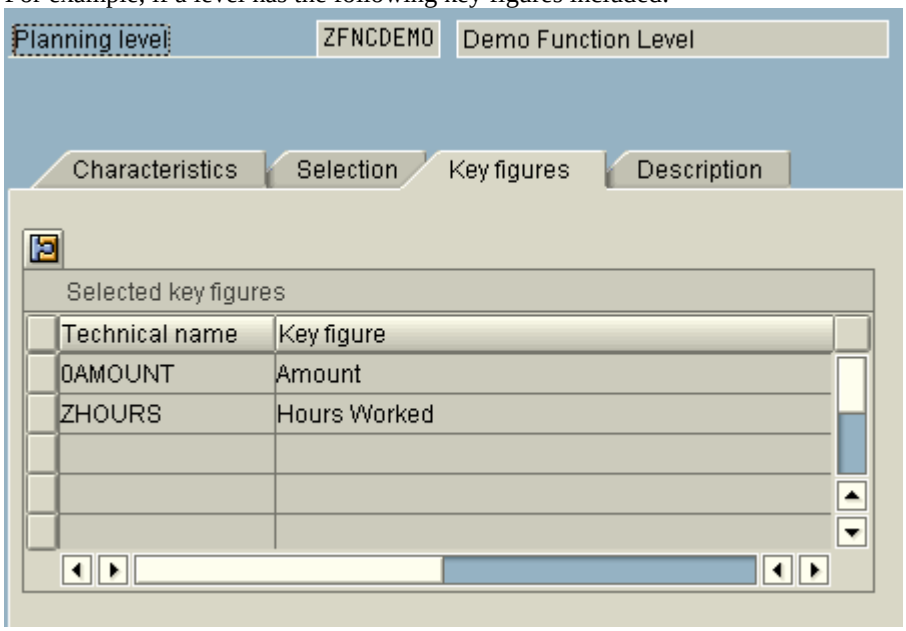
The technical names of these characteristics are also available in the table “ITO_CHA”:



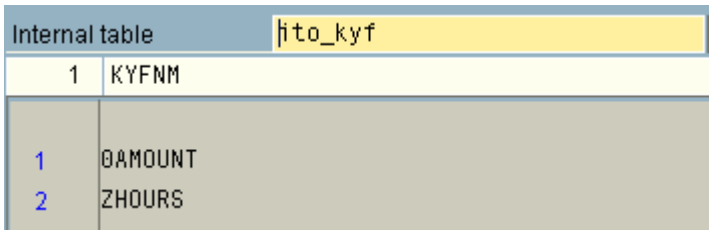
ix) ITO_KYF

Contains the technical names of all the key figures processed in the function. This corresponds to the key figures included in the level.

For example, if a level has the following key figures included:



The table ito_kyf will contain the technical names of these key figures:



b) Export Parameters

i) ET_MESG

ET_MESG is the BPS error message table to be filled with any error messages to be returned to the end user in the functions application log. (In the case of planning folders, this log is displayed by clicking the “F9” button at the end of a function).

ET_MESG contains the following fields:

- MSGID – the ID of the message class from which the message is from
- MSGTY – Message Type
  - Three possible values

  - "E" – Error, this signifies a procedural failure
  - "W" – Warning
  - "I" – Informational message
- MSGNO - The number of the message from the message class to be sent
- MSGV1 - The first variable of the message
- MSGV2 - The second variable of the message
- MSGV3- The third variable of the message
- MSGV4- The third variable of the message

When sending a message to an end user, you can use a pre-provided error message from a message class in the system.
Example:
The following ABAP code sends a standard BPS error message saying the value of the variable specified in the ABAP variable "l_source_var" cannot be determined.

```
------------------------------------------------------------
    Data: ls_mesg  TYPE upc_ys_mesg,
          l_source_var  TYPE upc_y_variable.

    IF sy-subrc <> 0.
       * Values of variable &1 cannot be determined
      ls_mesg-msgty  =  'I'.
      ls_mesg-msgid  = 'upc_fw'.
      ls_mesg-msgno  = '001'.
       ls_mesg-msgv1  = l_source_var.
       APPEND Ls-msg TO ETO_MESG.
              EXIT.
     ENDIF.
------------------------------------------------------------
```

You can also retrieve a message from the standard system messages, by appending the contents of the system log structure (SYST) to the message (ETO_TABLE). This is extremely helpful after executing a function that returns a system message.

Example:
The following code snippet opens a job, if the job cannot be opened, an error message is returned and appended to the BPS message table.

```
-------------------------------------------------------------------------------------------
    Data: ls_mesg  TYPE upc_ys_mesg.

    CALL FUNCTION 'JOB_OPEN'
      EXPORTING
        JOBNAME                = 'BUNDLE_EXECUTE'
     IMPORTING
       JOBCOUNT               = '2'
     EXCEPTIONS
       CANT_CREATE_JOB        = 1
       OTHERS                 = 2.

    IF SY-SUBRC NE 0.
      CLEAR ls_mesg.
      MOVE-CORRESPONDING syst TO ls_mesg.
      APPEND ls_mesg TO et_mesg.
      EXIT.
                 ENDIF.
-------------------------------------------------------------------------------------------
```

It is also possible to right your own custom messages, by using the message class 'upf' and message number '001'. This class provides an empty message whose entire contents

are specified in the first message variable. Custom messages must be limited to 50 characters or less (the specification for the field msgv1 in the table ETO_MESG).

Here is an example of a custom message from a BPS function:

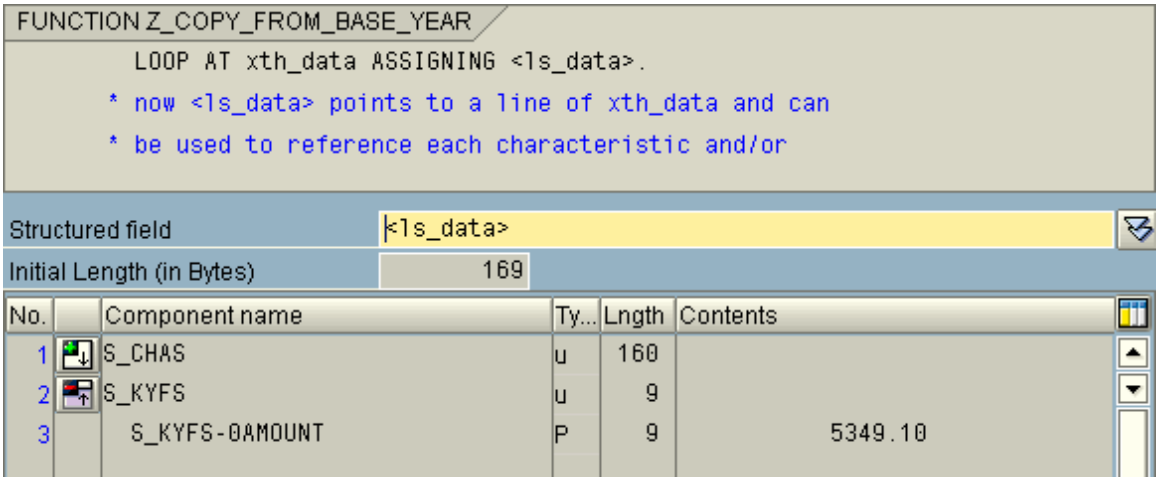
```
------------------------------------------------------------
Data: ls_mesg  TYPE upc_ys_mesg.
            ls_mesg-msgty  = 'W'.
            ls_mesg-msgid  = 'upf'.
            ls_mesg-msgno  = '001'.
             ls_mesg-msgv1  = “Something odd occurred”.
            APPEND ls_mesg TO et_mesg.
------------------------------------------------------------
```

c) Tables to be changed

i) XTH_DATA

This is the table containing the data from the data packages. These are the actual transactional data records in the BPS buffer for manipulation in the function.

The table XTH_DATA is divided into two components: S_CHAS and S_KYFS. S_CHAS contains the characteristic values for a data record and S_KYFS contains the key figure values for a record. Regardless of the characteristics and key figures specified in the level, the structure S_CHAS will contain all of the characteristics contained in the InfoCube and S_KYFS will contain all of the key figures contained in the InfoCube. However, if you attempt to edit a characteristic or key figure value NOT specified in the planning level and package, the function will error.



XTH_DATA is fed into the function as a hash table. This means that static field selection is not possible, since the fields are not previously defined. In order to access a field in S_CHAS or S_KYFS you will have to use dynamic ABAP selection.

The following is an example of using ABAP to get field values from a record. We select the value of the field 0COORDER (internal order). Note that any characteristic field whose values are manipulated, must be specified in the planning function parameter “fields to be changed”.

```
-------------------------------------------------------------
   FIELD-SYMBOLS:  <ls_data>  TYPE ANY,
                  <ch_struct> TYPE ANY,
                      <cl_coorder> TYPE ANY.

   LOOP AT xth_data ASSIGNING <ls_data>.
   * now <ls_data> points to a line of xth_data and can
   * be used to reference each characteristic and/or
   * keyfigure contained within.


   * choose first structure which contains the characteristics.
     ASSIGN COMPONENT 'S_CHAS' OF STRUCTURE <ls_data> TO
   <ch_struct>.

   *Assign fields for value assignment in the record (ls_data)

   * choose the characteristic '0COORDER' (use uppercase)
     ASSIGN COMPONENT '0COORDER' OF STRUCTURE <ch_struct> TO
   <cl_coorder>.

   *<cl_coorder> now contains the value of the internal order in the
   record
   *...
ENDLOOP
-------------------------------------------------------------
```

To manipulate a key figure value, the key figure component of XTH_DATA must be selected. The following is an ABAP example of the selection of the key figure 0AMOUNT. The function doubles the amount for all records in XTH_DATA:

```
   -------------------------------------------------------
   FIELD-SYMBOLS:  <ls_data>  TYPE ANY,
                  <ch_struct> TYPE ANY,
                   <cl_amount> TYPE ANY.

   *Process the data records
     LOOP AT xth_data ASSIGNING <ls_data>.
   * now <ls_data> points to a line of xth_data and can
   * be used to reference each characteristic and/or
   * keyfigure contained within.

   * choose second structure which contains the keyfigures
     ASSIGN COMPONENT 'S_KYFS' OF STRUCTURE <ls_data> TO
   <kf_new_struct>.

   *Assign the key figure amount to the field symbol for manipulation
     ASSIGN COMPONENT '0AMOUNT' OF STRUCTURE <kf_new_struct> TO
   <cl_amount>.

   *double the amount
   <cl_amount> = <cl_amount> * 2.

   ENDLOOP.
   -------------------------------------------------------------
```

Notice that in all examples, the function loops through all records inputed into the function. This is because; do to the hash nature of the table XTH_DATA, it is impossible to select certain records for processing. All records must be looped through for processing.

The number of records contained is XTH_DATA is determined by the number of characteristics contains in the “fields to be changed” parameter of the planning function specification.
The following example shows a scheme used to create the sub-packets.
Sample data:

0FISCPER	Cost Center	Amount
2005001	1000	20
2005001	1001	10
2005002	1000	5
2005002	1001	7

- The list of *'fields to be changed'* is empty.
  The exit function is called up four times – each time exactly one row is passed. This setting might be helpful, for example, when you want to change the key figures of a row without making references to other rows.

  2005001 1000 5

  2005001 1001 7

  2005002 1000 20

  2005002 1001 10
- The list of *'fields to be changed'* contains the characteristic Cost Center.
  All rows are bundled in one packet that share the same characteristic values – except for the cost center and key figures – hence, in our example there would be two packets with two rows each. This characteristic is also called a calculation characteristic.

  2005001 1000 5
  2005001 1001 7

  2005002 1000 20
  2005002 1001 10

  In each packet the characteristic 0FISCPER has only one value. This is feasible for operations that use different cost centers. (e.g., offsetting entries)
- The list of *'fields to be changed'* contains all characteristics.
  Only in this case is a packet created, which contains all rows and hence permits operations that require the creation of references between all entries.

  20050011000 20
  20050011001 10
  20050021000 5
  20050021001 7

2) Exit Planning Function – Initialization Function Modules

Initialization function modules are functions that are performed before the actual planning function. While a planning function may be run several times, depending on the number of data packets to be processed, an initialization function is only run once per planning function execution. Initialization function modules are run regardless of whether or not records exist for a package selection; planning functions are only run if records exist or have been created in the initialization function.

The two most common uses for initialization functions are:

- Initialize global variable and tables for later use
- Generating records

Any custom global ABAP variable or tables can be initialized from an initialization function. For example, for an escalation function, we must get the escalation rates from an external database table. This table is loaded into system memory for use in the planning function.

In the following example, the function fills the global internal table “GT_ZESCATE” with values from the ODS “ZESCRATE”:

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

REFRESH GT_ZESCRATE.

Select * From /BIC/AZESCRATE00
      APPENDING TABLE GT_ZESCRATE .

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

Initialization functions can also be used to generate records. This is especially important when we cannot guarantee that the planning level and package selection already have records. Therefore, for the planning function to run, we must create records.

The following initialization exit function code creates a dummy record to run a planning function with. This is especially helpful if the planning function initializes a non-bps process, instead of manipulating records:

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

FIELD-SYMBOLS: <f> TYPE ANY.

*
*Create one dummy combination
*If we don't do this, the function won't run since the second function
*will not be executed at all, if there is now transactional data in the
*level.


data: lr_wa     TYPE REF TO data.
FIELD-SYMBOLS: <s_chas>   TYPE ANY,
               <ref>  TYPE ANY.

* create structure for the selection criteria
create data lr_wa like line of eto_chas.
assign lr_wa->* to <s_chas>.

* the keys are now in S_CHAS.
COLLECT <s_chas> INTO eto_chas.

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

a) Import Parameters:

i) I_AREA

ii) I_PLEVEL

iii) I_METHOD

iv) I_PARAM

v) I_PACKAGE

vi) IT_EXITP

vii) ITO_CHASEL

viii) ITO_CHA

ix) ITO_KYF

The import parameters for initialization functions are the same as for planning functions. For more information, please reference the section "Exit Planning Functions – Function Modules".

b) Export Parameters

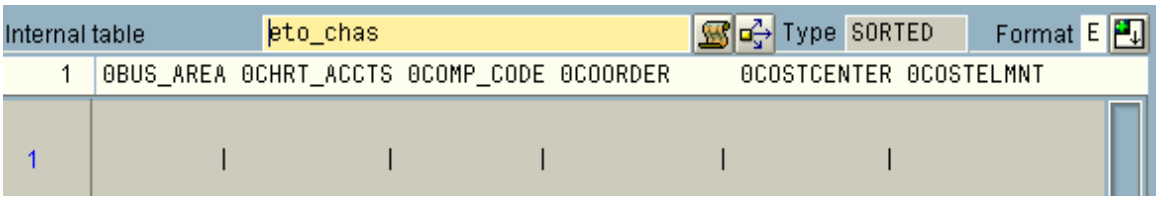
- ETO_CHAS
- ET_MESG

i) ET_MESG

The parameter ET_MESG for initialization functions are the same as for planning functions. For more information, please reference the section "Exit Planning Functions – Function Modules".

ii) ETO_CHAS

ETO_CHAS is an initially blank table with fields for all characteristics in the planning area. The suggested use for ETO_CHAS, is to fill this table with characteristics selections allowed in the planning package, but not already included in the given data. By filling ETO_CHAS with just one line of blank data, such as in the example at the beginning of this section, we can force the planning function to run, regardless if actual data exists.



3) Variable Exit Functions

a) Import Parameters:

- I_AREA
- I_VARIABLE
- I_CHANM
- ITO_CHANM

i) I_AREA

This is the technical name of the area for the variable.

ii) I_VARIABLE

This is the technical name of the variable.

iii) I_CHANM

The technical name of the characteristic used in the variable. However, now that variables can contain more than one characteristic, this field is no longer reliable.

iv) ITO_CHANM

A table containing the technical names of the characteristic(s) contained in the variable.

b) Export Parameters

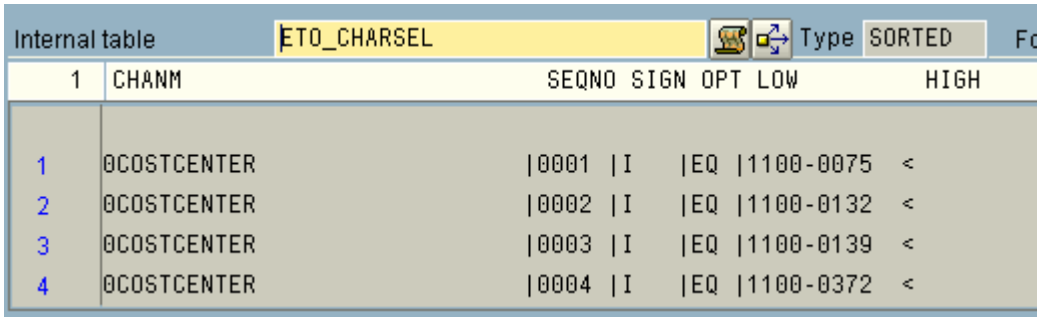
i) ETO_CHARSEL

ETO_CHARSEL is the selection table for the variable. This selection table is end-user specific and valid only for the current read of the variable.

This table contains the following fields:

- CHANM – the characteristic for the selection. Must be a characteristic specified in the variable definition.
- SEQNO – the sequence number of the variable selection value. No two values or ranges can have the same sequence number. Numbers are ascending, starting with 1.
- SIGN – Include or exclude the specified values in the selection, in BPS this is almost always include
  - Possible Values
    - "I" (Include)
    - "E" (Exclude)
- OPT – Whether the selection value is a single value or a range of values.
  - Possible Values
    - "EQ" (Equal, for single values – if the option is equal no value is included for the field "HIGH"; the single value is entered in the field "LOW")
    - "BT" (Between, signifies a range, the high and low values of this range are specified in the fields "LOW" and "HIGH")
- LOW – The selection value (if the field "OPT" contains "EQ") or the low end of the value (if the field "OPT" contains "BT")
- HIGH – The high end of a selection value range (only filled if the field "OPT" contain the value "BT")

Here is what an example ETO_CHARSEL table looks like at runtime. The following example is a cost center variable filled with four values:



Internal table ETO_CHARSEL Type SORTED

	CHANM	SEQNO	SIGN	OPT	LOW	HIGH
1	0COSTCENTER	0001	I	EQ	1100-0075	
2	0COSTCENTER	0002	I	EQ	1100-0132	
3	0COSTCENTER	0003	I	EQ	1100-0139	
4	0COSTCENTER	0004	I	EQ	1100-0372	

Every time the variable is read, this function is called, so if the value will consistent, regardless of other variable and data changes, it may make sense to create a buffer situation, where the value of the variable is stored in a static ABAP variable, for future reads during the planning session.

In the following code example, to get authorization variables, the function calculates the variable selection table only once, save it in a static ABAP table and fills ETO_CHAS from the static ABAP table in memory (PTO_VARSEL) for the remainder of the planning session.

```
FUNCTION Z_SET_VAR_CCAUT_VAL.
*"----------------------------------------------------------
*"*"Local interface:
*"  IMPORTING
*"     REFERENCE(I_AREA) TYPE  UPC_Y_AREA
*"     REFERENCE(I_VARIABLE) TYPE  UPC_Y_VARIABLE
*"     REFERENCE(I_CHANM) TYPE  UPC_Y_CHANM OPTIONAL
*"     REFERENCE(ITO_CHANM) TYPE  UPC_YTO_CHA
*"  EXPORTING
*"     REFERENCE(ETO_CHARSEL) TYPE  UPC_YTO_CHARSEL
*"----------------------------------------------------------------
  CONSTANTS:
*Specifiy the company code variable
    L_COMPCODE TYPE /BIO/OICOMP_CODE  VALUE '1100'.

*Buffer values stored in program memory
  STATICS:
     pto_varsel TYPE upc_yto_charsel,
     p_variable TYPE upc_var-var,
     p_area TYPE upc_y_area,
     p_first_read LIKE boole-boole VALUE 'X'.


  DATA:
      l_subrc LIKE sy-subrc ,
      ls_return LIKE bapiret2,
       L_AUTH_CHARSEL TYPE  UPC_YTO_CHARSEL.


*After the first read, the value of the variable will ALLOWS be
*taken from the buffer
*first read is determined by whether the flag “p_first_read” is
*checked
  IF  p_first_read IS INITIAL
      AND p_area = i_area
      AND p_variable = i_variable.
```

```
        eto_charsel = pto_varsel.

      return.

*****CALCULATE THE VALUE OF ETO_CHAS*************************
****This is done only during the first call
****** to the variable function
  ELSE.

    REFRESH ETO_CHARSEL.

*Get the authorization values restricted by the company code
      PERFORM GET_AUTH_VALUES USING L_COMPCODE
                              CHANGING
                                    eto_CHARSEL.


*Remove the first read flag
      CLEAR p_first_read.

* -- no error occured => store results to buffer
    p_area = i_area.
    p_variable = i_variable.
    pto_varsel = eto_charsel.
  ENDIF.

ENDFUNCTION.
```

c) Exceptions

i) FAILED

A variable exit does not have a message return parameter, however, in the case of a function failure, a message can still be sent, if it is attached to an exception. This is done using the following syntax:

```
MESSAGE <type><number>(<class>) WITH [MSGV1] [MSGV2] [MSGV3]  RAISING
exception FAILED.
```

For Example:

```
  *Get source Variable value

    CALL FUNCTION 'Z_VARIABLE_GET_DETAIL'
      EXPORTING
        i_area     = i_area
        i_variable = L_SOURCE_VAR
        i_buffer   = ' '
      IMPORTING
        e_subrc    = l_subrc
        eto_varsel = eto_charsel.
    IF l_subrc <> 0.
  * Values of variable &1 cannot be determined
  * raise error message
      MESSAGE i136(upc_fw) WITH L_VAR RAISING FAILED.
    ENDIF.
```

Additional Guides from SAP

The following How-To Documents from SAP provide some valuable tips for BPS programming:

1_How_to_variables_exit.pdf

EXIT_FUNCTION_BPS.doc