Creation of BI Master Data in Integrated Planning (IP) through Web Layouts

Applies to:

BI-IP 7.0. For more information, visit the Business Intelligence homepage.

Summary

Prior to BI-IP in BPS the user had an option of entering the Planning Data even for the Characteristics for which the master data was not present in the BW system. With BI-IP any attempt to enter the planning data on the characteristic which is not present in BI system fails as the BEx tools do a validation of data entered on the screen against the data present in the BI system and return with the error

The solution below allows user to create the Master Data in the BI system using the Planning Function created in Integrated Planning and hence allowing them for creating any planning data for the characteristics for which earlier there was no master data in the BI system. The same could be used in migration of the Planning application from such scenarios from BPS to IP or to meet any such business requirement in BI-IP implementation

Author: Mahesh Sharma

Company: Knack Systems LLC

Created on: 23 September 2008

Author Bio

Mahesh Sharma is a Senior SAP BI / PI Consultant at Knack Systems LLC. He has a broad experience and exposure of working in various BI, BPS, IP and XI implementation and upgrade projects.

Table of Contents

Scenario

The scenario in discussion is as below. The user is expected to enter “Workload Measure” in the planning application created in IP. Each year user may or may not enter the planning data for a particular “Workload Measure” and also each year there might be a chance of user creating a new “Workload Measure” for which there is no master data present.

Budget Request

Text	Add'l Text	FY 2007 ACT	FY 2008 EST	FY 2009 EST
CLIMBING TREES		34	34	45
Overall Result		34	34	45
SWIM RIVERS		1	2	3

Fig 01. Input Layout using WAD

In the example above “SWIM RIVERS” is not present in the BI system as a master data for “Workload Measure” characteristics and any attempt to save planning data with same would result in the validation error as below in Fig 02.

Incorrect input value: SWIM RIVERS

Validation was not successful

Budget Request

Text	Add'l Text	FY 2007 ACT	FY 2008 EST	FY 2009 EST
CLIMBING TREES		34	34	45
Overall Result		34	34	45
SWIM RIVERS		1	2	3

Fig 02. Error while saving the planning data

We have to provide a solution to the user to enable him to create master data for a workload measure if the same is not present in the BI system as above

Solution

The solution for the above scenario is to allow the user to create the master data for the “Workload Measure” characteristic which is not present in BI system while entering the planning data. In the example above the “Workload Measure” “SWIM RIVERS” is not present in BI system and the same needs to be created in the BI system before any planning data could be saved.

We would achieve the above objective by creating a Planning Function which would take the text entered by the user and would create the same in BI system first (as in Fig. 03) and later allowing user to enter the planning data using the same characteristic. In our example we intend to create “SWIM RIVERS”.

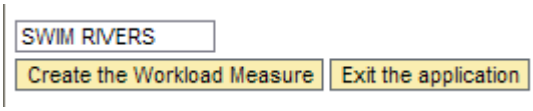


Fig. 03 Web Layout for Planning Function

The steps for the solution above are as below

Step1: Create a Function Module which would accept the Characteristic values and would create the master data in BI system

Step 2: Create a Class to be attached to the Planning Function Type which would call the Function Module created in Step 1 and pass the values entered by the user to the Function Module for creation of master data

Step 3: Create the Planning Function Type for creation of Master Data

Step 4: Create Planning Function in Integrated Planning

Step 5: Create Web layout using as shown in Fig.03.

Step 1: Creation of Function Module

We would be creating Function Module as below of type Normal.

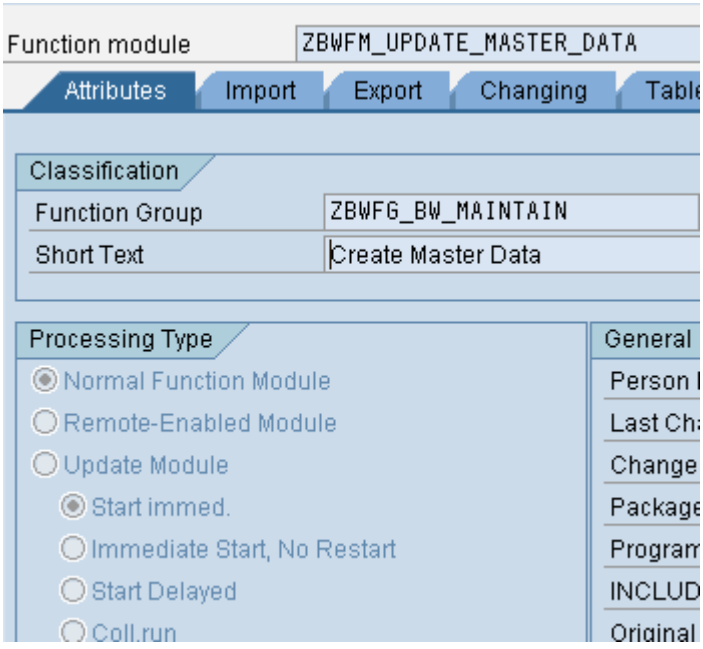
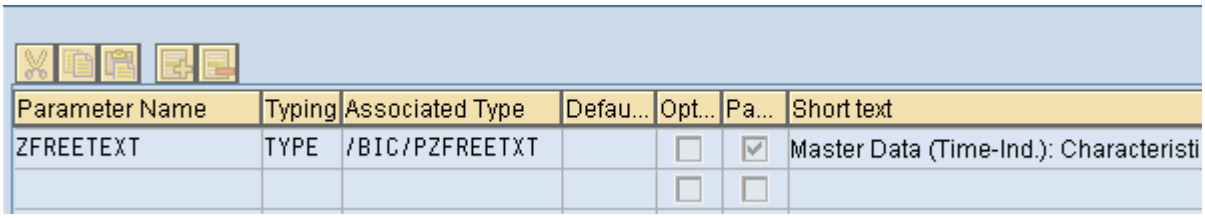


Fig 04: Function Module - Attributes

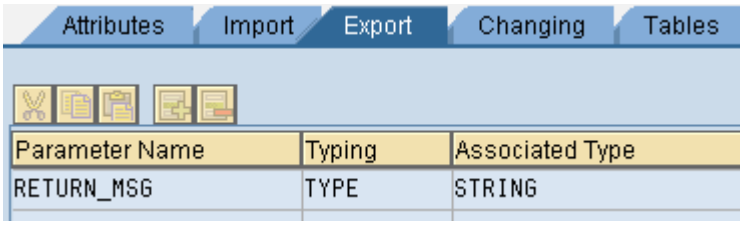
Import parameter for the Function Module is the master table of the InfoObject for which the master data is to be created as below



Parameter Name	Typing	Associated Type	Defau...	Opt...	Pa...	Short text
ZFREETEXT	TYPE	/BIC/PZFREETXT		☐	☑	Master Data (Time-Ind.): Characteristi
				☐	☐	

Fig 05: Function Module – Import parameters

Export parameters is any message that needs to returned

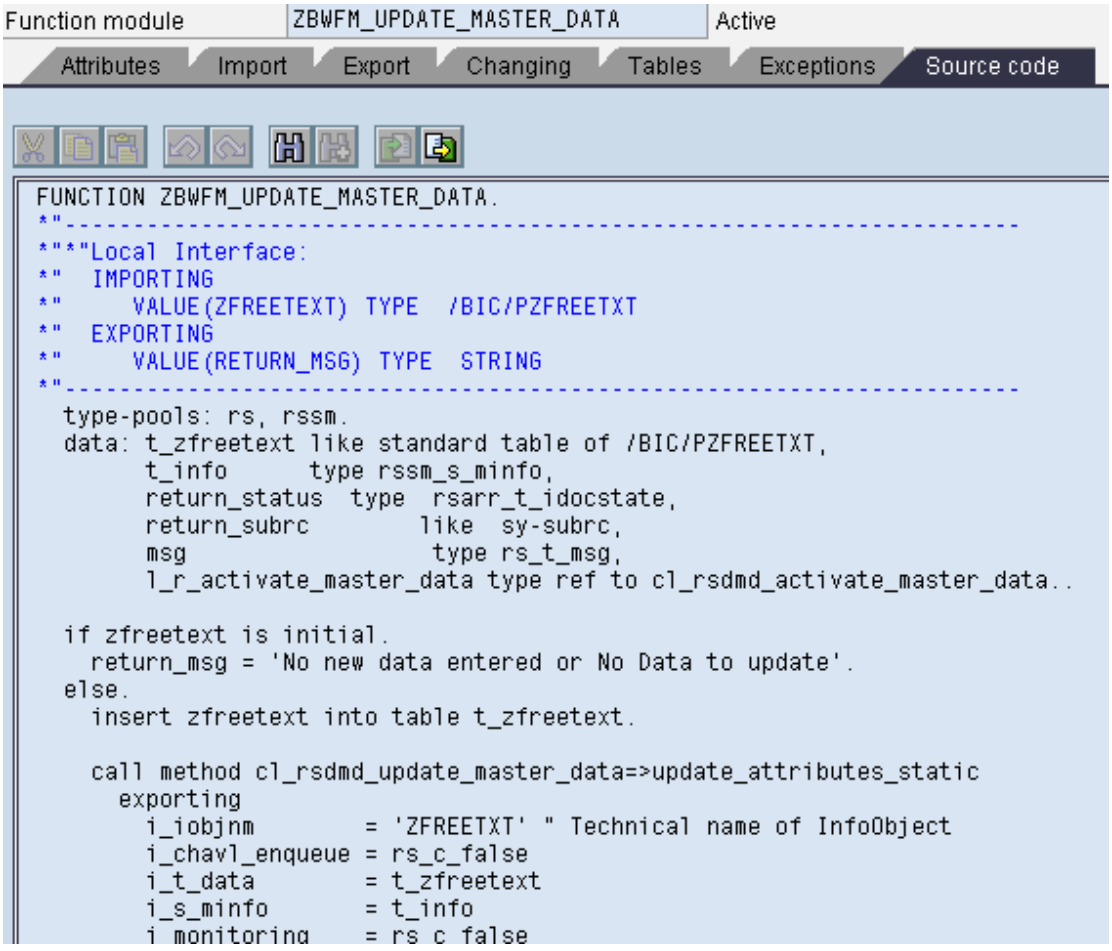


Attributes | Import | Export | Changing | Tables

Parameter Name	Typing	Associated Type
RETURN_MSG	TYPE	STRING

Fig 06: Function Module – Export parameters

Finally, the Source Code. The same is attached in Appendix. Here “ZFREETEXT” is the Technical Name of InfoObject in question.



Function module | ZBWFM_UPDATE_MASTER_DATA | Active

Attributes | Import | Export | Changing | Tables | Exceptions | Source code

```
FUNCTION ZBWFM_UPDATE_MASTER_DATA.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(ZFREETEXT) TYPE  /BIC/PZFREETXT
*"  EXPORTING
*"     VALUE(RETURN_MSG) TYPE  STRING
*"----------------------------------------------------------------------
  type-pools: rs, rssm.
  data: t_zfreetext like standard table of /BIC/PZFREETXT,
        t_info      type rssm_s_minfo,
        return_status  type  rsarr_t_idocstate,
        return_subrc        like  sy-subrc,
        msg                  type rs_t_msg,
        l_r_activate_master_data type ref to cl_rsdmd_activate_master_data..

  if zfreetext is initial.
    return_msg = 'No new data entered or No Data to update'.
  else.
    insert zfreetext into table t_zfreetext.

    call method cl_rsdmd_update_master_data=>update_attributes_static
      exporting
        i_iobjnm        = 'ZFREETXT' " Technical name of InfoObject
        i_chavl_enqueue = rs_c_false
        i_t_data        = t_zfreetext
        i_s_minfo       = t_info
        i_monitoring    = rs_c_false
```

Fig 07: Function Module source code

Step 2: Creation of Class

A class which can be attached to the Planning Function type would be created. The details of the class are as below.

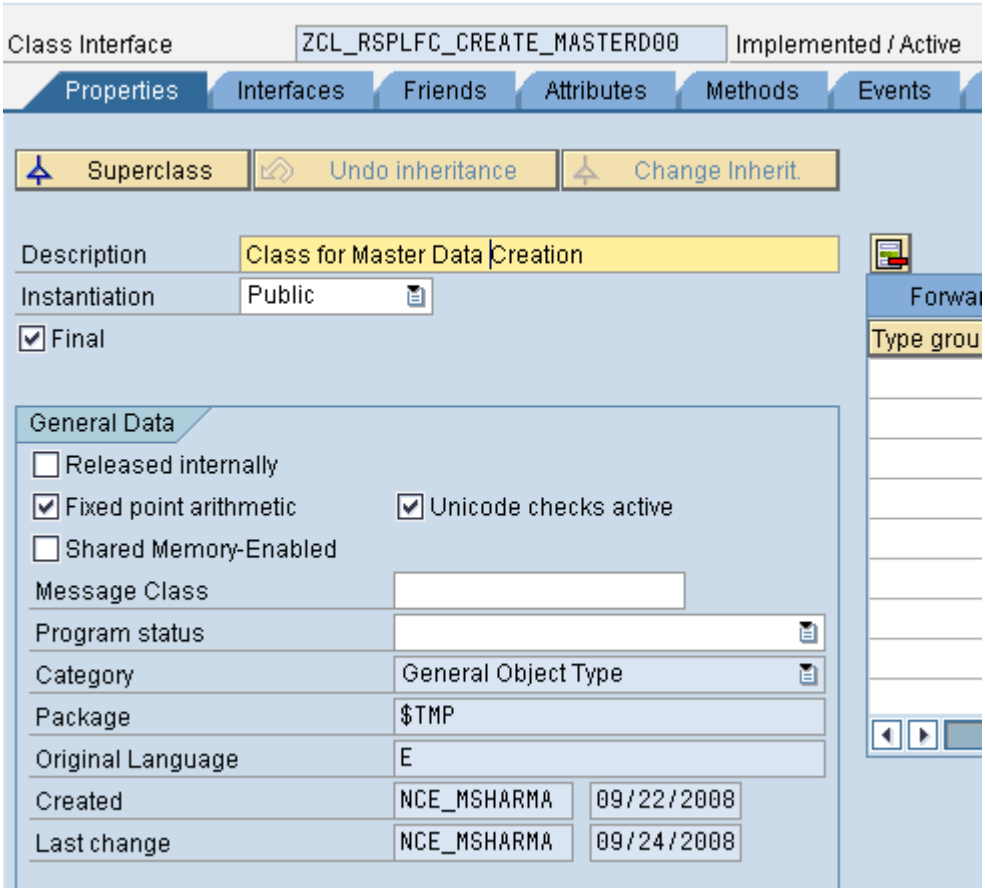


Fig 08: Class properties

We would be using the SAP delivered interface in the class “IF_RSPLFA_SRVTYPE_IMP_EXEC_REF”. The interface inserted allows us to pass the parameter values from the Planning Function to the Method in the Class.

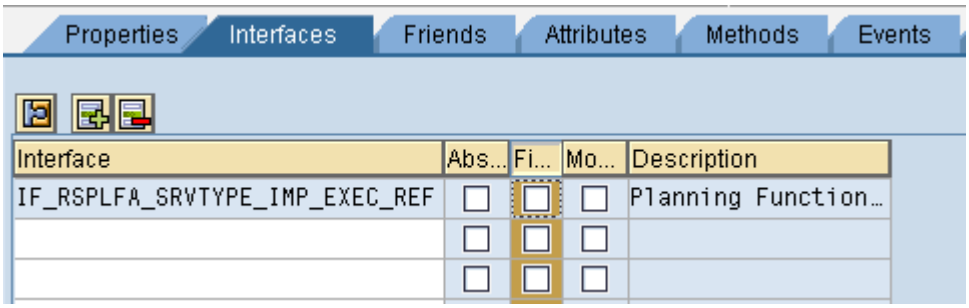


Fig 09: Interface IF_RSPLFA_SRVTYPE_IMP_EXEC_REF inserted to the class

Following are the attributes needed to pass the values from Web Layout to the method in the class

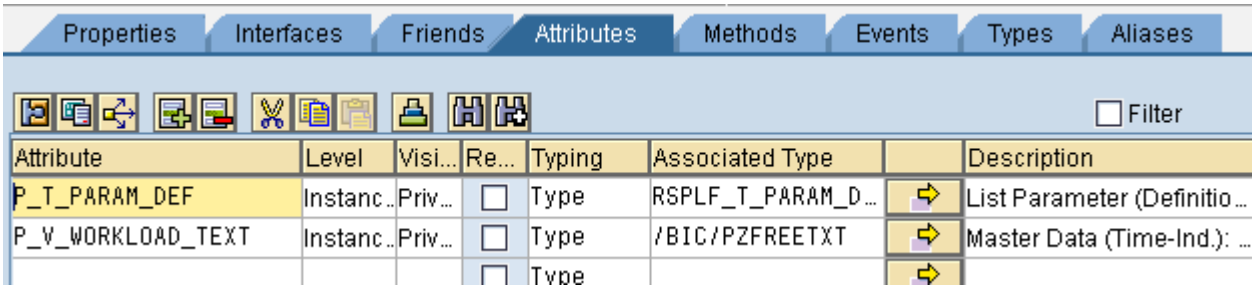


Fig 10: Attributes declared for the class

In our example we would only be using two methods from the available methods below.

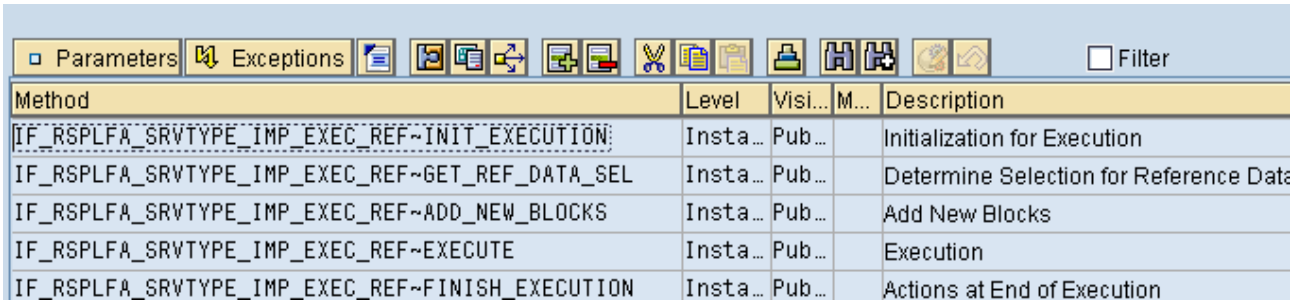


Fig 11: Methods present in the class after inserting the interface IF_RSPLFA_SRVTYPE_IMP_EXEC_REF

The methods used would be INIT_EXECUTION and EXECUTE

The method INIT_EXECUTION is used to pass the values from the web layout to the class attributes as below. The Source Code is attached in the appendix.

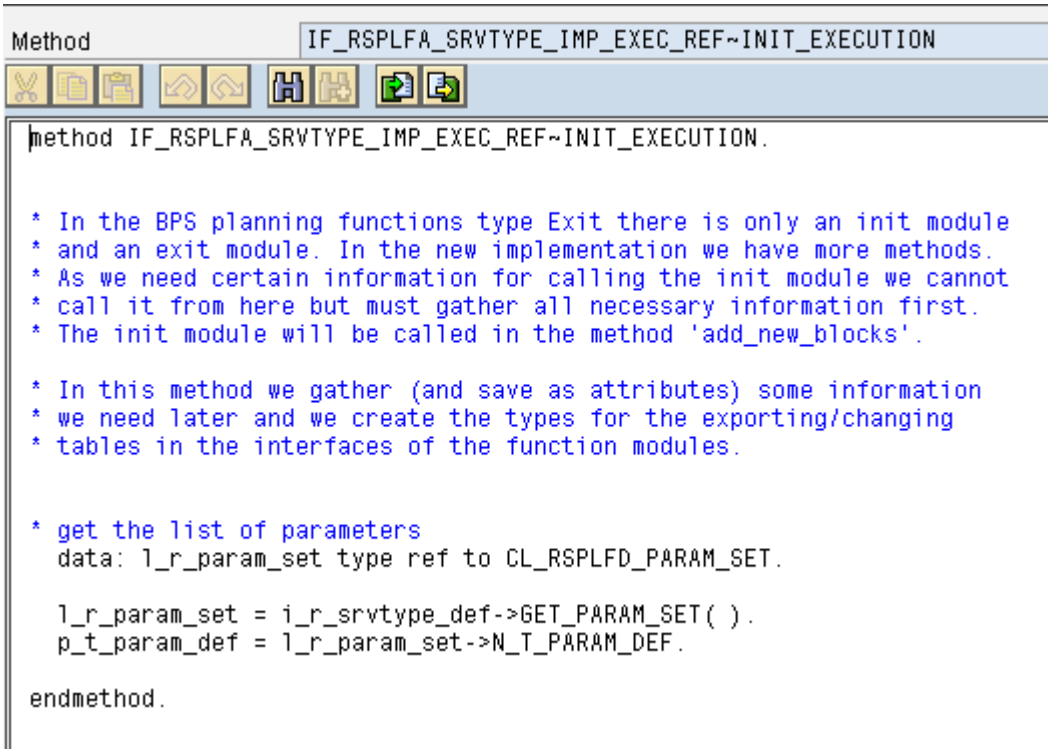


Fig 12: Implementation of method INIT_EXECUTION

The method EXECUTE is used to read the values entered by the user on the layout and pass the same to the function module for creation of master data. The Source Code is attached in the appendix

Method IF_RSPLFA_SRVTYPE_IMP_EXEC_REF~EXECUTE

```
method IF_RSPLFA_SRVTYPE_IMP_EXEC_REF~EXECUTE.

data : ZV_WORKTEXT  like table of P_V_WORKLOAD_TEXT.
Data : ZL_WORKTEXT Type ZFREETEXT.


* loop for debugging

* endloop for debugging

* get the values of all parameters!
  data: lt_exitp  TYPE  UPF_YT_EXITP,
        ls_exitp type upf_ys_exitp,
        l_s_param_def type RSPLF_R_PARAM_DEF,
        l_r_param type ref to IF_RSPLFA_PARAM_ELEM.

  data: l_area type upc_y_area.

* if you do not need any parameters other than the function module
* names you can delete this coding
  loop at p_t_param_def into l_s_param_def.
* we do not insert the name of the function modules
* Yes we do <- Jürgen Schwab
*    check l_s_param_def->n_name <> P_C_NAME_EXIT_PARAM.

    check not l_s_param_def->N_IS_ELEMENTARY is initial.
```

Fig 13: Implementation of method EXECUTE

Save and activate the class.

Step 3: Creation of Planning Function Type

For creation of Planning Function Type execute the transaction RSPLF1.

Create the Planning Function Type as below. Make sure you have selected “Reference Data” on the properties Tab.

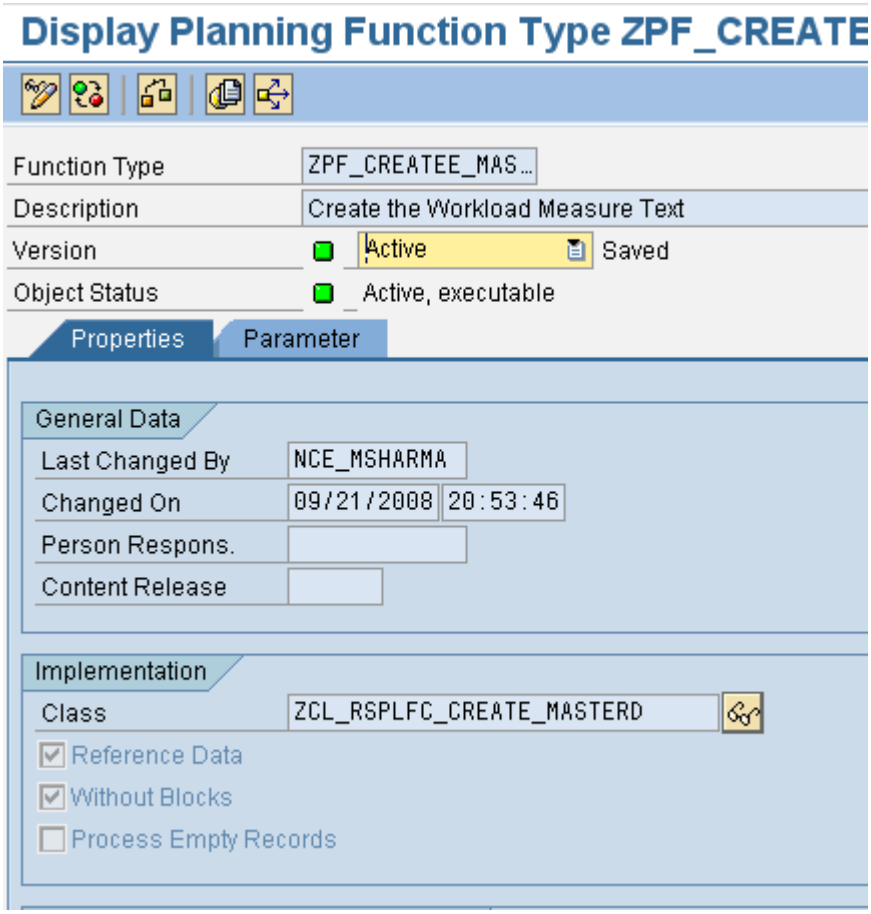


Fig 14: Planning Function Type properties

On the Parameter Tab insert a parameter as below:

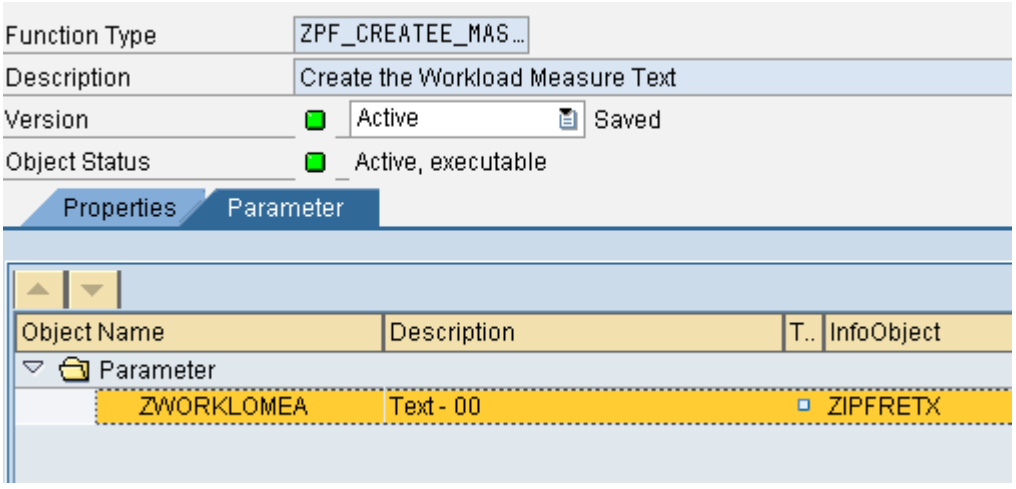


Fig 15: Parameters created for the Planning Function Type

The parameter “ZWORKLOMEA” is of type Elementary and refers to the InfoObject ZIPFRETX as below.

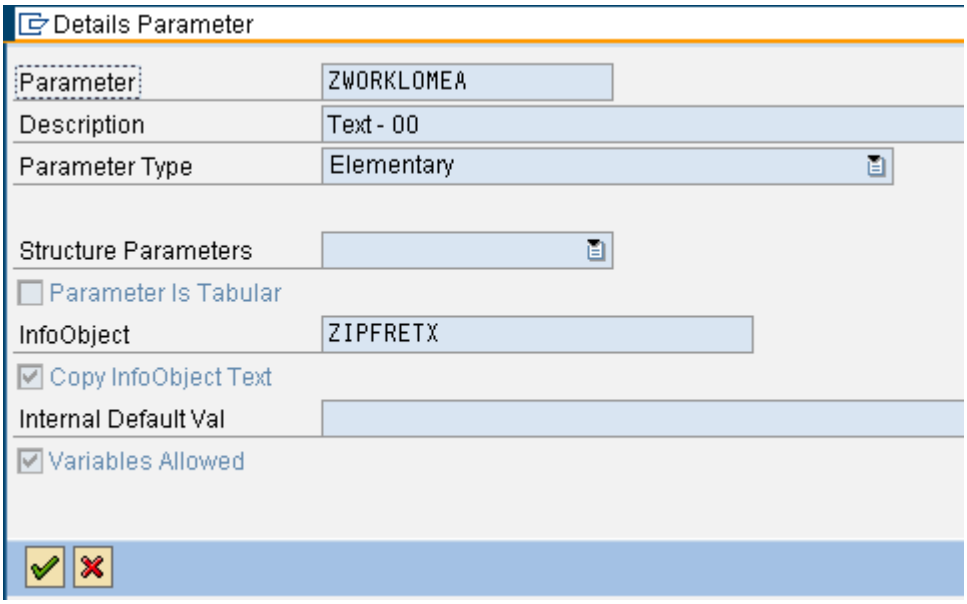


Fig 16: Parameter definition for the parameter inserted in Planning Function Type

The InfoObject ZIPFRETX is a new InfoObject of copied from the InfoObject ZFREETXT. The only difference is that ZIPFRETX is w/o master data. The same is needed as we cannot use ZFREETXT in the Planning Function Type as any record entered will the fail the validation again. So we are using another InfoObject to read the values entered by the user.

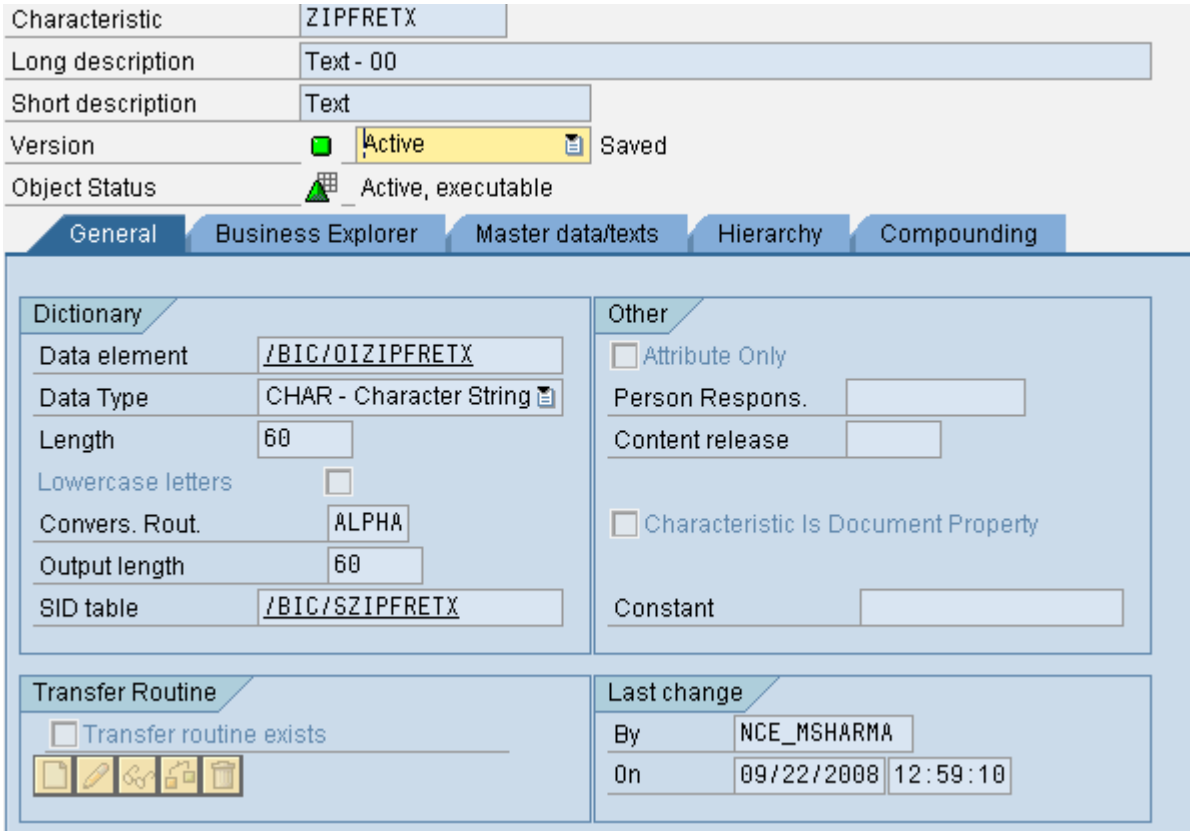


Fig 17: InfoObject definition for the InfoObject ZIPFRETX

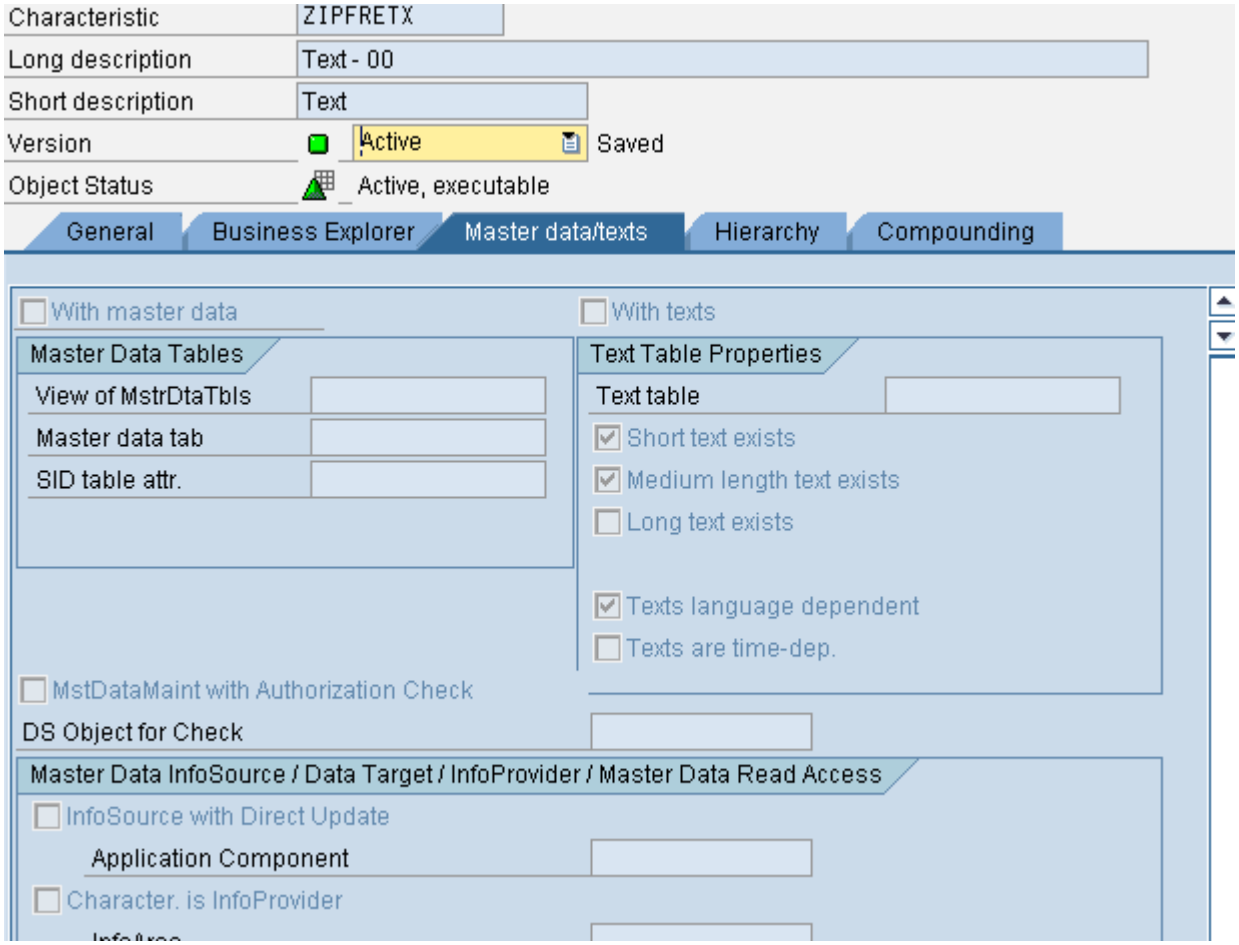


Fig 18: Master Data properties for the InfoObject ZIPFRETX

Step 4: Creation of Planning Function

The next step is to create the Planning Function in Planning Modeler. Our Planning Function Type created in Step 3 would be available in the drop down list while selecting the Planning Function Type.

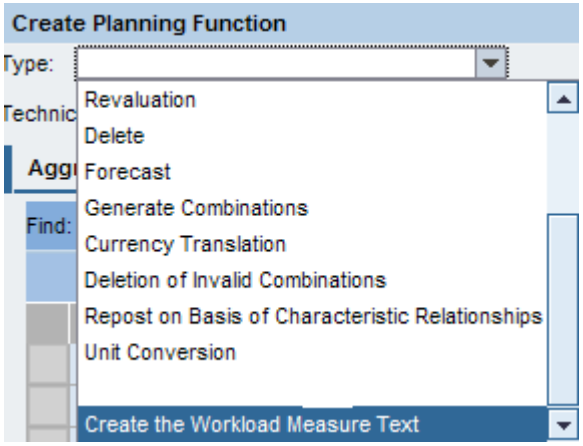


Fig 19: Functions available in Planning Modeler

As below we assign the values to the Parameter using the a Variable “IPFREETXT” and save the Planning Function

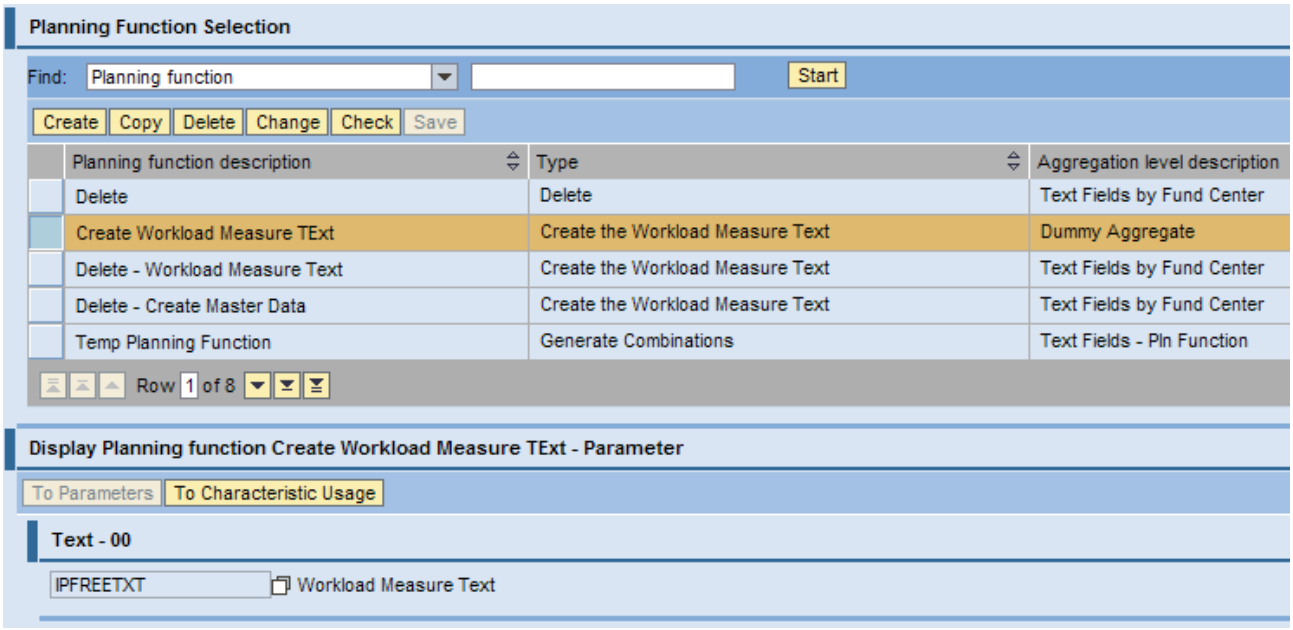


Fig 20: Planning Function definition for the creation of master data

Note: As shown above we have created a Dummy Aggregate on which we have built the planning function. In order to implement the solution, we created a Dummy Cube with one Dimension and one Key Figure. Also the cube is need not be loaded with any data.

The reason for the above approach is the fact that Planning Function tries to read all the rows from cube while execution and even though there is no logic where the change in values is happening for the data all the data from the aggregate would be read. To improve the runtime of Planning Function we run the same against a dummy InfoCube which has no data in it.

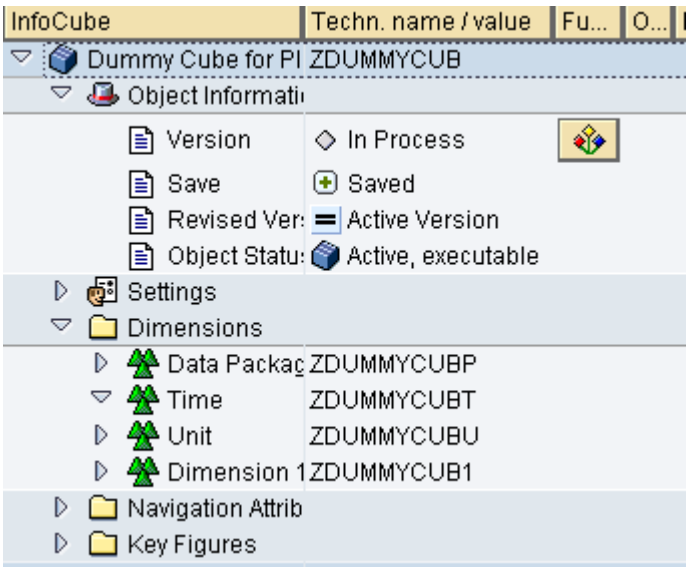


Fig 21: InfoCube definition for the dummy InfoCube used in the Planning Function

Step 5: Creation of Web Layout

Now we need to create a Web Template mentioned in the Fig. 03.

The Web Items used for the same are “Input Field” and “Button Group” as below

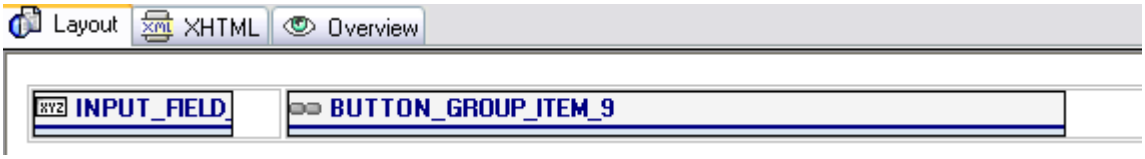


Fig 22: Web Layout for the template created in Web Application Designer

While configuring the Button Group we need to do binding of the data entered in the Input Filed to the variable in the Planning Function as below.

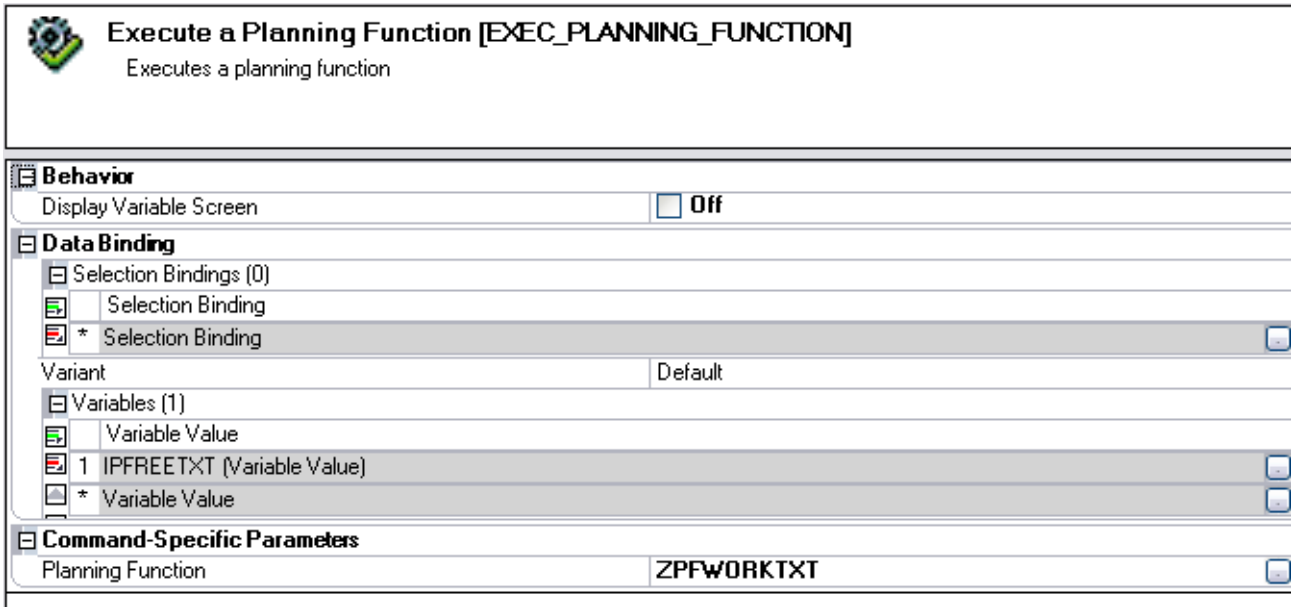


Fig 23: Data binding between the command and the web items

Result

As discussed in Scenario, now when the user tries to enter planning data and system returns with an error message, the user can now first create the master data and enter the planning data thereafter.

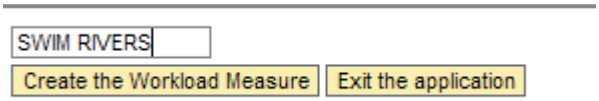


Fig 24: Input screen available for the user to create master data

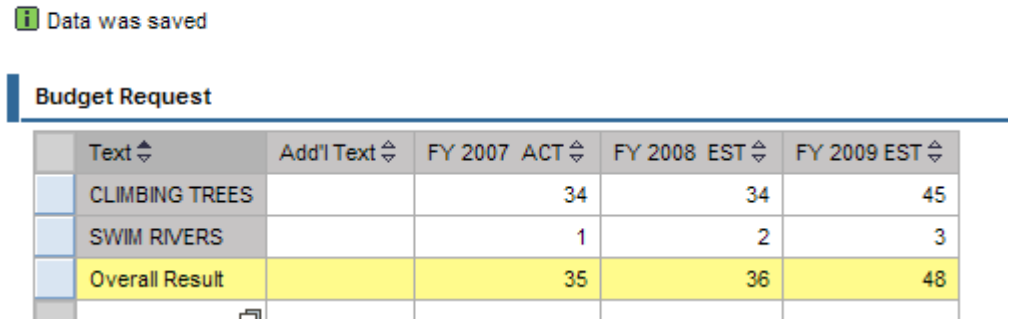


Fig 25: Planning layout result after entering the planning data

Appendix

Source code for Function Module

```
type-pools: rs, rssm.
data: t_zfreetext like standard table of /BIC/PZFREETXT,
      t_info      type rssm_s_minfo,
      return_status  type  rsarr_t_idocstate,
      return_subrc        like  sy-subrc,
      msg                 type rs_t_msg,
      l_r_activate_master_data type ref to cl_rsdmd_activate_master_data..

if zfreetext is initial.
  return_msg = 'No new data entered or No Data to update'.
else.
  insert zfreetext into table t_zfreetext.

  call method cl_rsdmd_update_master_data=>update_attributes_static
    exporting
      i_iobjnm        = 'ZFREETXT' " Technical name of InfoObject
      i_chavl_enqueue = rs_c_false
      i_t_data        = t_zfreetext
      i_s_minfo       = t_info
      i_monitoring    = rs_c_false
    changing
      c_t_idocstate   = return_status.

  if sy-subrc <> 0.
    return_msg = 'Error happened'.
  else.
    return_msg = 'Process Completed Successfully'.

    create object l_r_activate_master_data
      exporting
        i_iobjnm            = 'ZFREETXT'
        i_p_q_have_to_exist = rs_c_false.

    if sy-subrc = 0.
      l_r_activate_master_data->activate(
              importing
                e_subrc = return_subrc
              changing
                c_t_msg = msg ).
    endif.
  endif.

endif.
```

Source code for the Class

Method : INIT_EXECUTION.

```
method IF_RSPLFA_SRVTYPE_IMP_EXEC_REF~INIT_EXECUTION.


* In the BPS planning functions type Exit there is only an init module
* and an exit module. In the new implementation we have more methods.
* As we need certain information for calling the init module we cannot
* call it from here but must gather all necessary information first.
* The init module will be called in the method 'add_new_blocks'.

* In this method we gather (and save as attributes) some information
* we need later and we create the types for the exporting/changing
* tables in the interfaces of the function modules.


* get the list of parameters
  data: l_r_param_set type ref to CL_RSPLFD_PARAM_SET.

  l_r_param_set = i_r_srvtype_def->GET_PARAM_SET( ).
  p_t_param_def = l_r_param_set->N_T_PARAM_DEF.

endmethod.
```

Method : EXECUTE

```
method IF_RSPLFA_SRVTYPE_IMP_EXEC_REF~EXECUTE.

data : ZV_WORKTEXT  like table of P_V_WORKLOAD_TEXT.
Data : ZL_WORKTEXT Type ZFREETEXT.


* loop for debugging

* endloop for debugging

* get the values of all parameters!
  data: lt_exitp  TYPE  UPF_YT_EXITP,
        ls_exitp type upf_ys_exitp,
        l_s_param_def type RSPLF_R_PARAM_DEF,
        l_r_param type ref to IF_RSPLFA_PARAM_ELEM.

  data: l_area type upc_y_area.

* if you do not need any parameters other than the function module
* names you can delete this coding
  loop at p_t_param_def into l_s_param_def.
* we do not insert the name of the function modules
* Yes we do <- Jürgen Schwab
*    check l_s_param_def->n_name <> P_C_NAME_EXIT_PARAM.

    check not l_s_param_def->N_IS_ELEMENTARY is initial.

    l_r_param = i_r_param_set->GET_PARAM_ELEM( l_s_param_def->n_name ).

    if l_s_param_def->n_name = 'ZWORKLOMEA'.
      ls_exitp-PARNM =  l_s_param_def->n_name.
      l_r_param->get_value( IMPORTING e_value = ls_exitp-chavl ).
      ZL_WORKTEXT-/BIC/ZFREETXT = ls_exitp-chavl.
```

```
    endif.

  endloop.

CALL FUNCTION 'ZBWFM_UPDATE_MASTER_DATA'
  EXPORTING
    ZFREETEXT         = ZL_WORKTEXT
* IMPORTING
*   RETURN_MSG        =
```

endmethod.

Related Content

Getting Improved Performances in BW-BPS / BI-IP When Using Enhancements Trough the ABAP Technology (Exits and Classes)

Maintaining BI Master Data with Visual Composer

Implementing Planning Function Types

For more information, visit the Business Intelligence homepage.

Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.