SAP NetWeaver
How-To Guide

How To... Flexibly Change Characteristic Values and Comments in BI Integrated Planning

Applicable Releases:

SAP NetWeaver 7.0

IT Practice:

Business Information Management

IT Scenario:

Business Planning and Analytical Services

Version 2.5

November 2009

HOW-TO GUIDE

THE BEST-RUN BUSINESSES RUN SAP™

Document History

Document Version	Description
1.00	First official release of this guide
2.00	Restrictions for SAPNetWeaver EHP 1 added
2.50	Error (missing data declarations) corrected

Typographic Conventions

Type Style	Description
Example Text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, graphic titles, and table titles
`Example text`	File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

Icons

Icon	Description
	Caution
	Note or Important
	Example
	Recommendation or Tip

Table of Contents

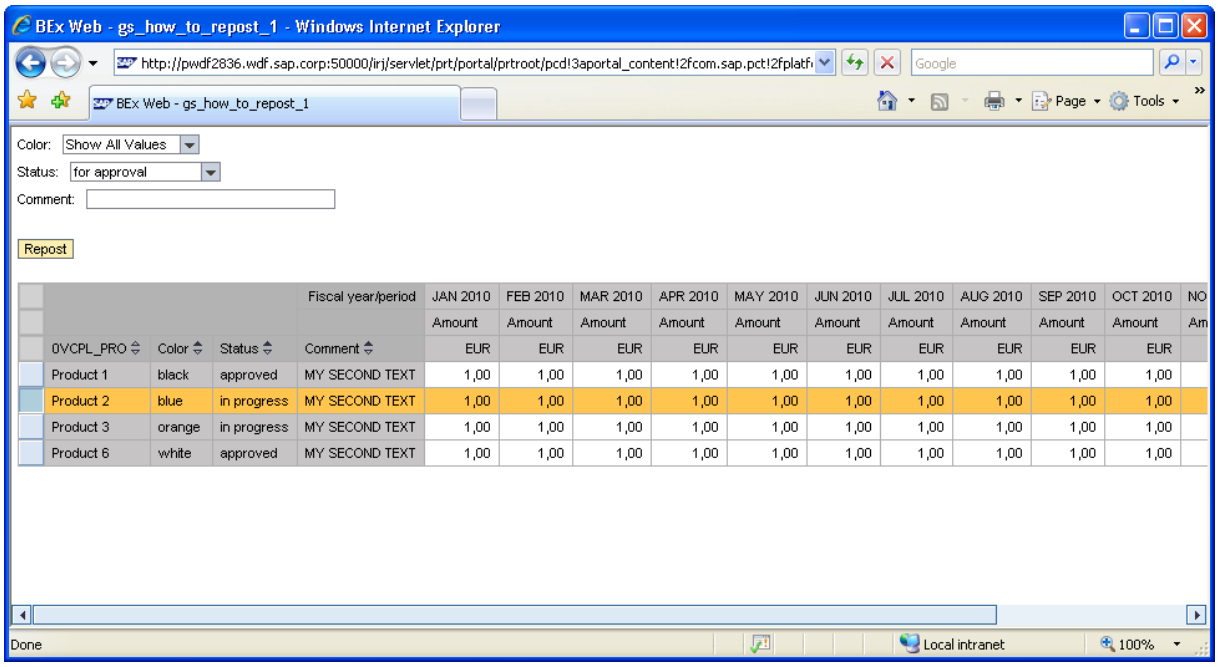
1. Business Scenario

In many BW Planning applications characteristics without master data are used to replace simple comments. Records with new (not yet existing) comments can be either entered directly in new lines (BEx Analyzer) or via a planning function in BEx Web applications. But how can they be changed? As the comment is a characteristic and thus a key field the text cannot be just overtyped by the end user. But a simple standard repost function with an entry field for the new comment will do the job.

Now quite often we have the case that the users do not just want to repost a comment but also want to change other characteristics (like maybe a status flag) as well. In a standard repost function you have to specify which characteristics will be changed and as it is not predictable which ones a user wants to change in one step you will need one planning function for each of the characteristics. The aim of this how to paper is to describe a way how a generic planning function can be implemented that can change any number of characteristics in one step. Thus one planning function for all possible user requirements is enough. We will build up an example for such a planning function in a Web application. This Web application can also be seen as a simple master data planning application as it does not only provide the possibility to change key figures but also characteristic values.

We will also make sure that the planning function will be optimized in regards of performance.

Let us have a look at the example we will build up in this how to paper. We have a web template where we see plan data for different products. Each product has a color, a status, and a comment (all of them are characteristics in the records). The web template offers the possibility to change any of those characteristics. In our example the user wants to change the status for a product. He selects the corresponding row (in this case the row for product 2), chooses the new status (and does not select any color or comment), and presses the button *'repost'*.



The status in the selected row is changed and the drop down box for status is automatically reset to the initial value (no selection). In the same way the user can change the color and/or enter a new comment.

2. Background Information

In this paper we will implement, describe, and use a new planning function type that can repost any number of characteristic. We will use this planning function in a very flexible way.

Let us assume we have a planning application where the user plans different products. Each product can have some attributes – modeled as characteristics in the data record (!); let us use 'color'. Each record has a status and can contain a free comment (modeled as characteristic without master). The user decides which of the values should be changed and is able to change any combination of these characteristics in one step by specifying the source record(s) and the new value(s). If the user does not specify a target value for one of the characteristics then the characteristic will not be changed.

Usually when setting up a repost function where the target is not known the system can run into a performance problem. As no target can be specified in the selection of the planning function the function will have to select all possible targets. In order to make sure that only the source and the target records are selected we use an exit technique in order to restrict the selection to the least possible number of records upon execution.

3. Prerequisites

As we are using an exit to change the data selection for the planning function please have a look at note 1101726 which describes the requirements and the technique itself. You can also have a look at the paper 'How to…Run Planning Functions on Changed Records in BI Integrated Planning' for an actual implementation of the exit.

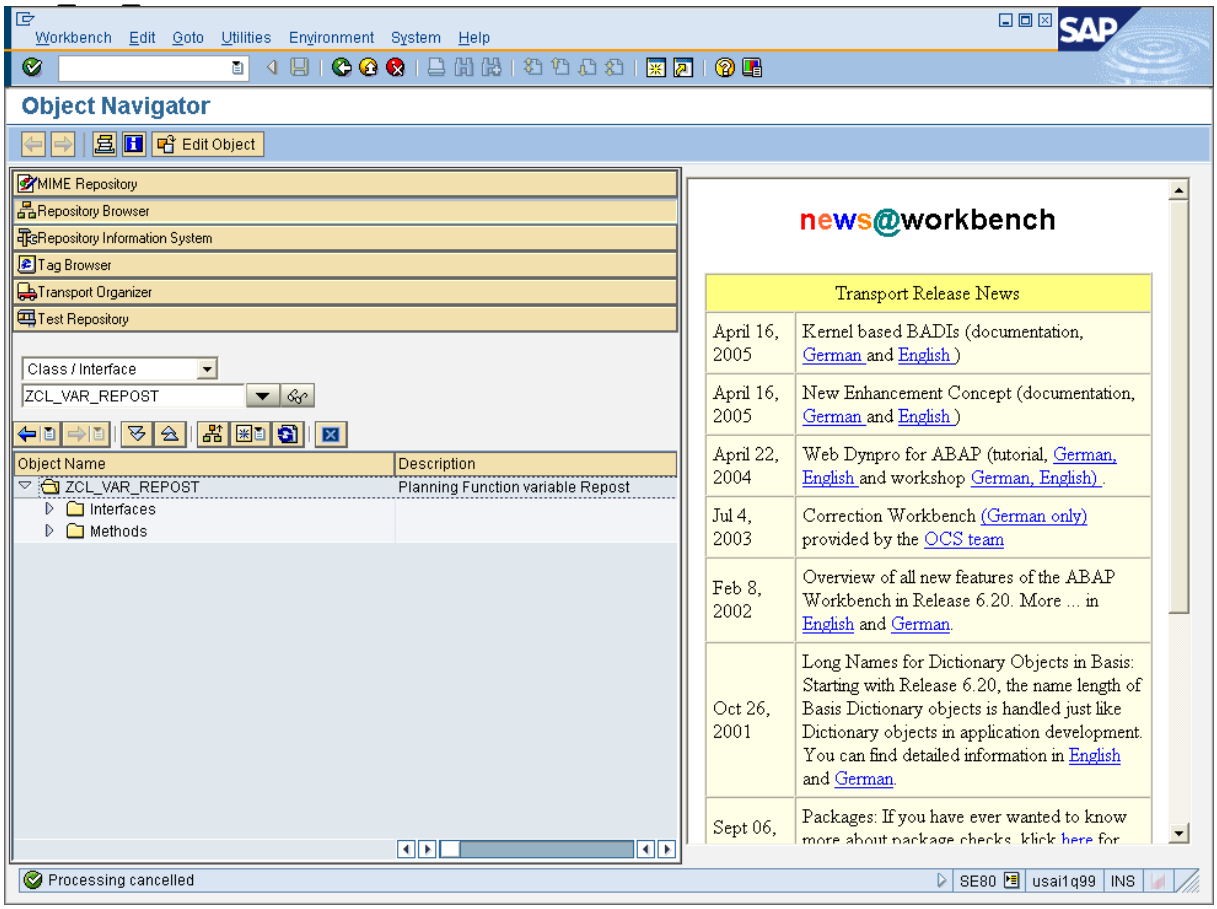
In this how to paper we are using a work around in the case we want to enter new characteristic values/comments that do not exist on the data base yet. Here we rely on the following prerequisites: When entering a string in a text field that is mapped to a variable value NO value check is performed. Thus values that are not yet existing characteristic values can be mapped into the variable. This behavior is changed with EHP1. Thus the work around does not work in the standard setup of a NetWeaver 7.0 EHP 1 system. Notes 1384495, 1387004, and 1368772 describe how this now system behavior can be switched back to the 'old' behavior. Nevertheless it cannot be guaranteed that with EHP 1 and up this workaround will further be supported.

4. Step-by-Step Procedure

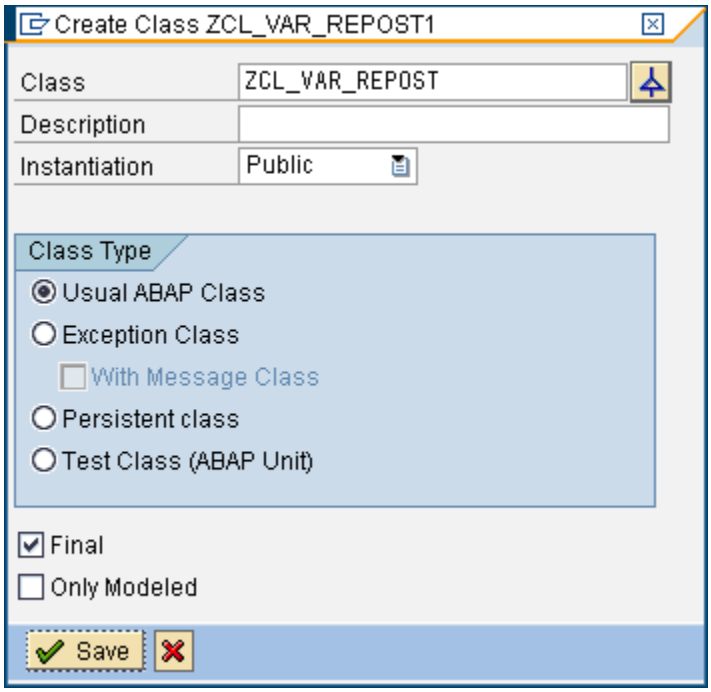
We will build up a BEx Web application that allows a flexible change of characteristic values or comments. First of all we will create the new planning function type. We will then build up the InfoCube, Aggregation Level and the planning function itself. In the end we will crate the Web application and adopt our ABAP exit accordingly.

4.1 Create the New Planning Function Type

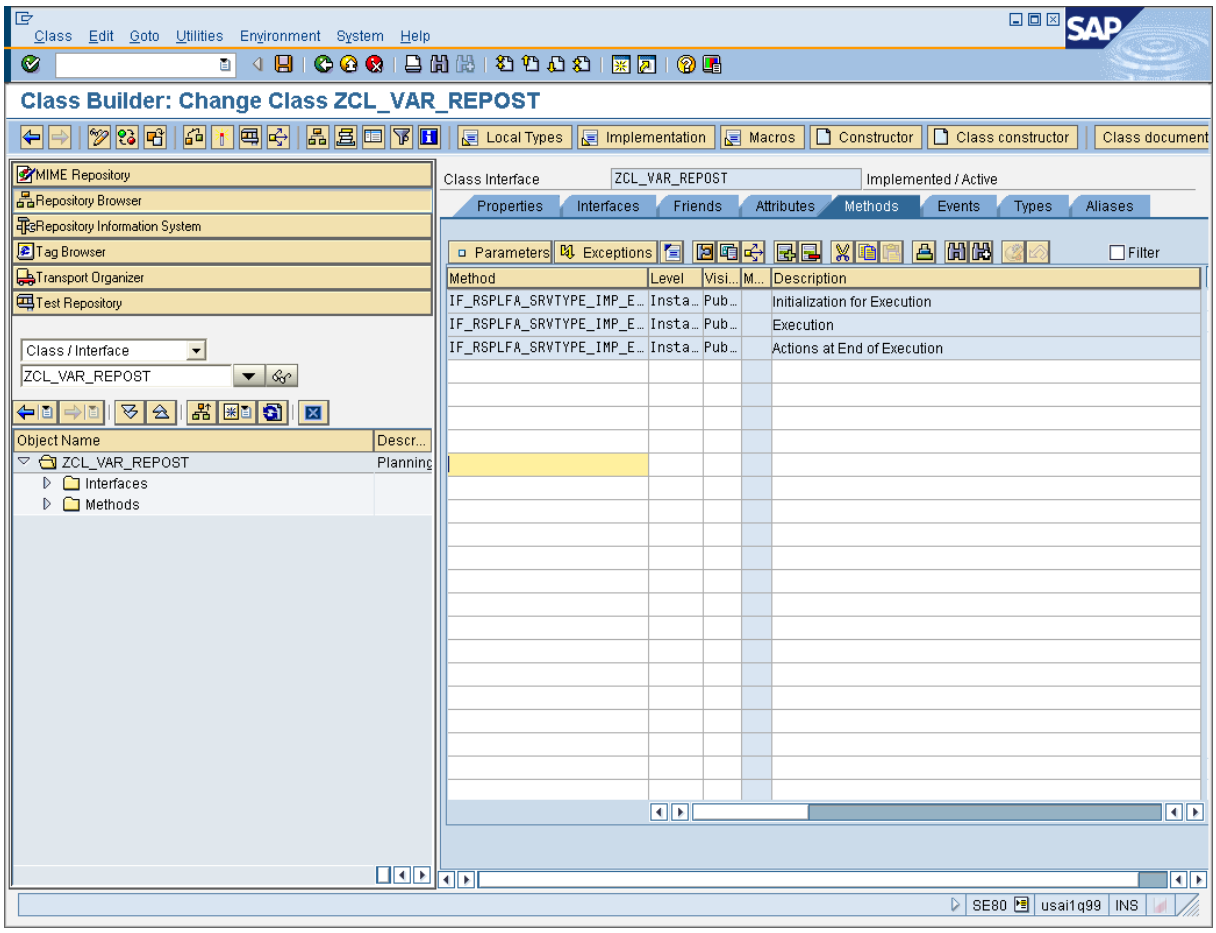
1. First of all we will have to create the class containing the function logic. Log on to your BW system and call the transaction *se80*. In the drop down box choose '*Class/Interface*' and enter a name for the new class containing the logic for the planning function. We used the name '**ZCL_VAR_REPOST**'.



2. Enter return. The system will inform you that the class does not exist yet. Let the system create the class. On the popup enter a description and press '*Save*'.

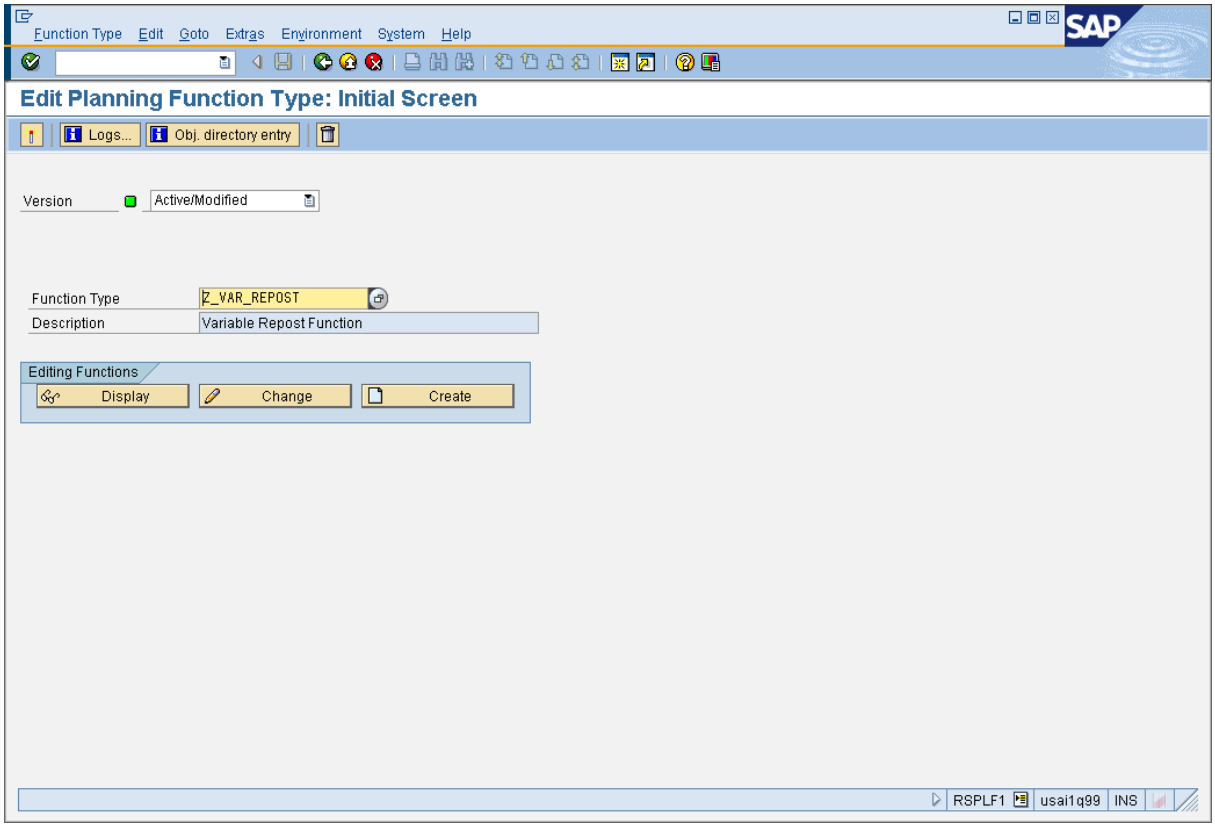


3. Go to the tab '*Interfaces*' and enter the interface '**`IF_RSPLFA_SRVTYPE_IMP_EXEC`**'. Then go to the tab '*Methods*' and double click each of the methods. The system will generate empty methods. Press '*Save*' when asked by the system.

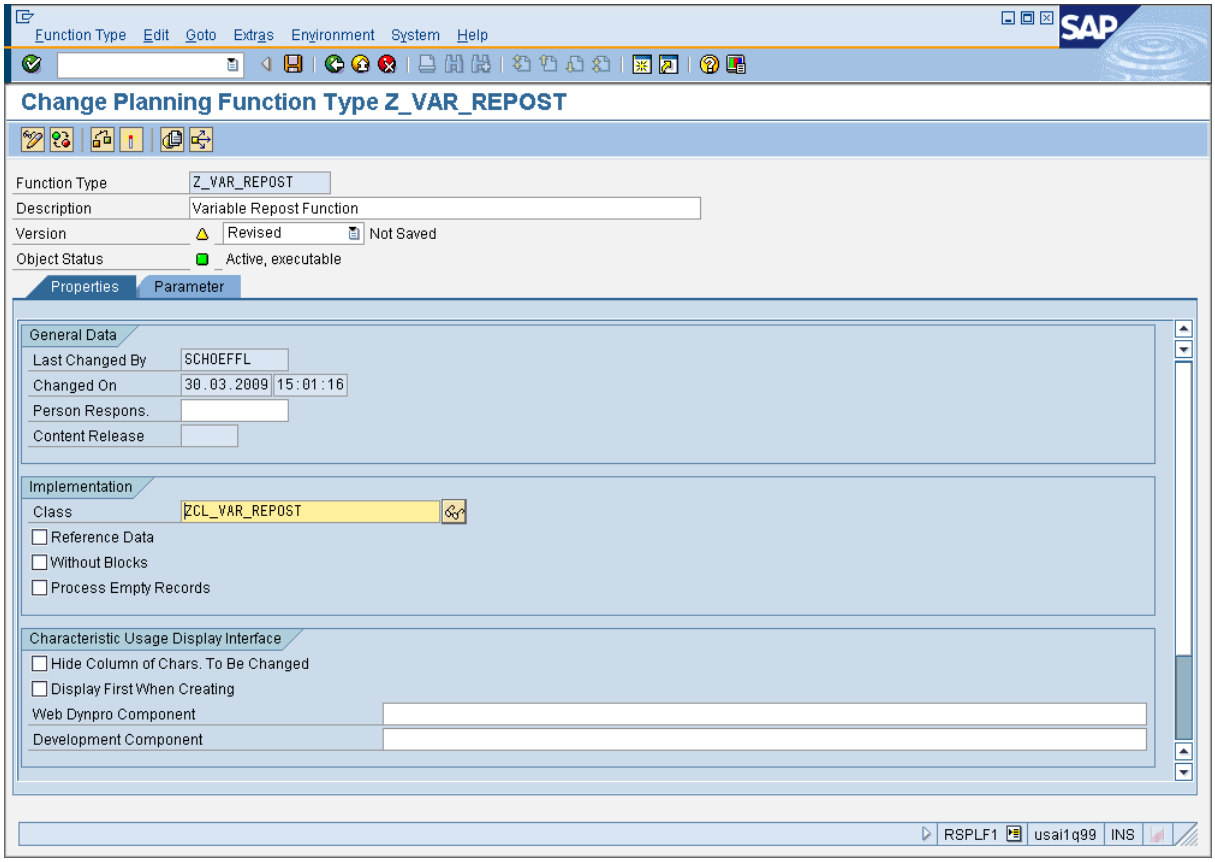


4. Enter the method '**`IF_RSPLFA_SRVTYPE_IMP_EXEC~EXECUTE`**' again and paste the coding from Appendix A. Save and activate the class.

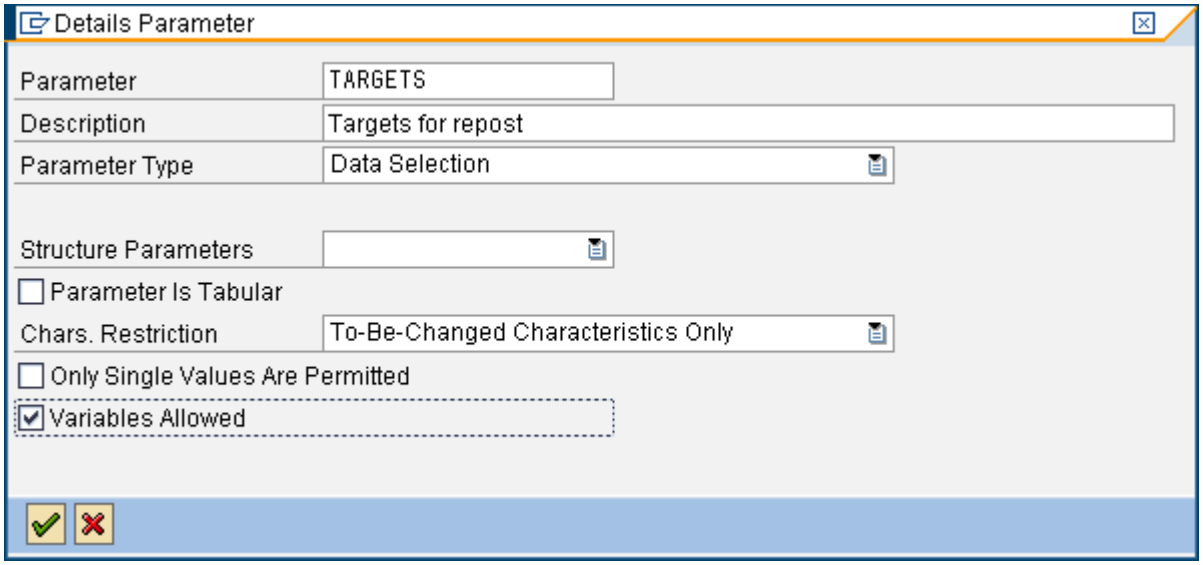
5. Now we can create the planning function type. Therefore call the transaction *rsplf1*. Enter a name (we have chosen '**`Z_VAR_REPOST`**') for the new planning function type and press '*Create*'.



6. Enter a description for the planning function type and enter the name of our class.



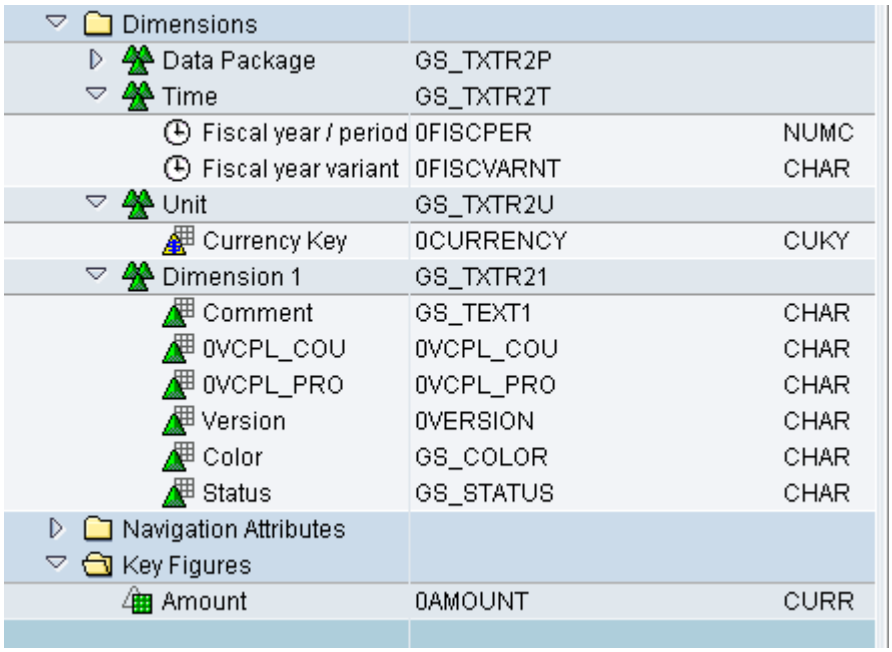
7. Go to the tab '*Parameter*'. Call the context menu on the entry '*Parameter*' and create a new parameter. Choose the name '**TARGETS**' and fill the parameters exactly the same way as in the screen shot. Press '*ok*' and activate your planning function type (use the icon or `Ctrl+F3`).



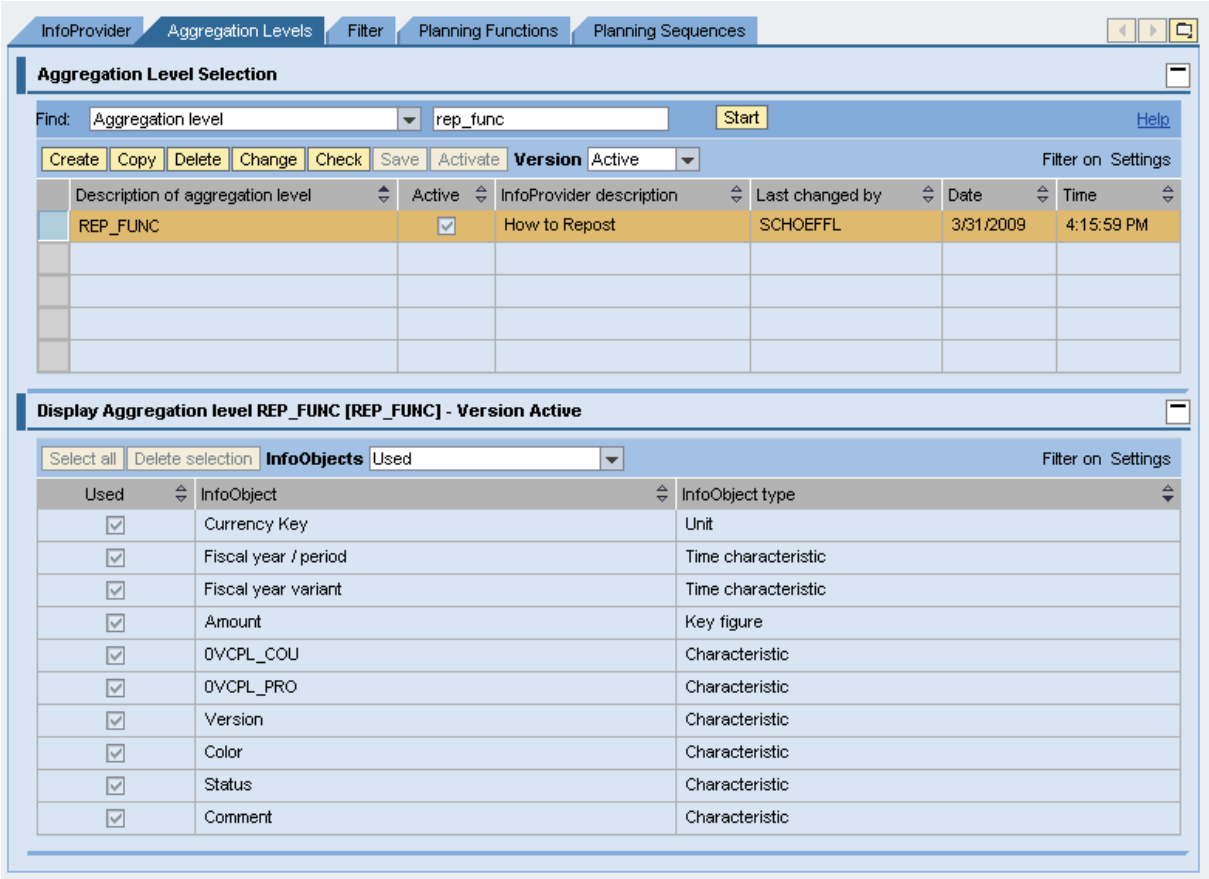
4.2 Create the InfoCube, Aggregation Level and Planning Function

1. Go to the transaction rsa1 and create a new real-time InfoCube. You can as well use an existing InfoCube and use your own characteristics accordingly. In our case the InfoCube contains – among others – the characteristics 'Product', 'Color', 'Status', and 'Comment'. Comment is a characteristic that has no master data and is used as a 'free text field'. All the characteristics except 'Comment' marked as '*Characteristic is InfoProvider*' so we can later use queries to display the characteristic values.

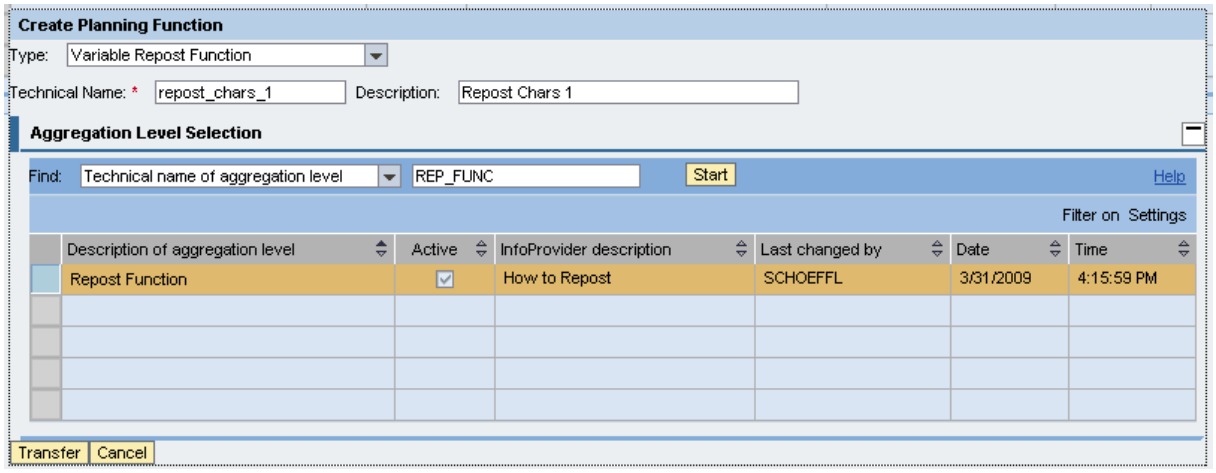
   You can already load some sample data into the InfoCube.



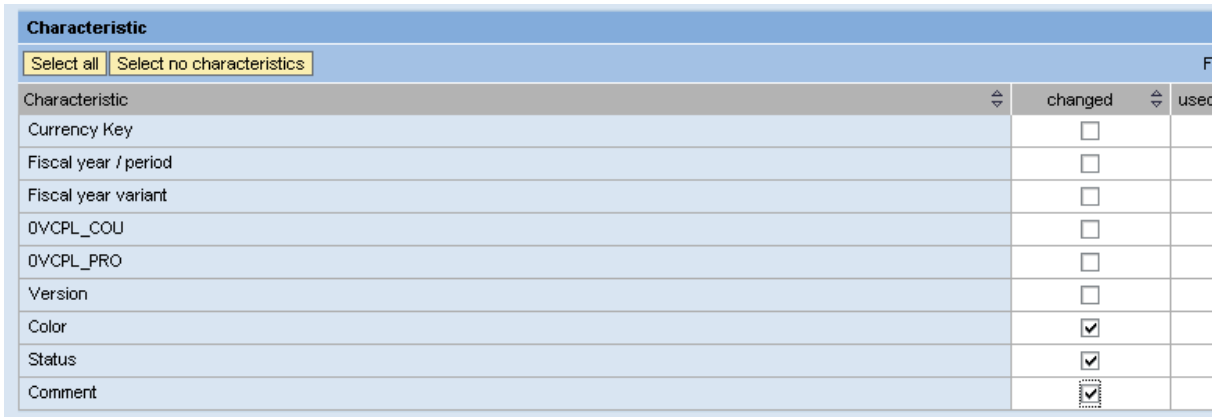
2. Now we create an aggregation level on which we will do our planning. As in our example we are using a very simple InfoCube we create an aggregation level using all characteristics. Go to the transaction *rsplan* and call the planning modeler. Choose your InfoCube and create the aggregation level. Do not forget to activate the aggregation level.



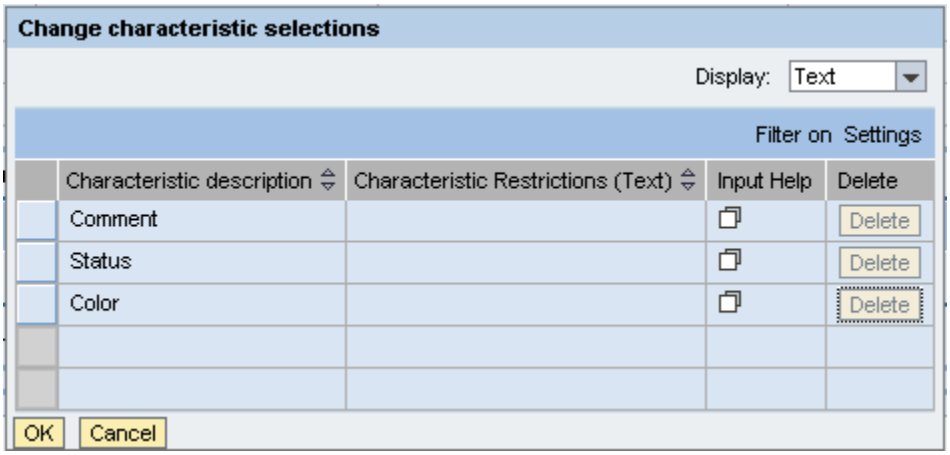
3. Within the aggregation level we create a planning function. Press the button '*Create*'. Choose our new planning function type and enter a name for the planning function.



4. Press '*Transfer*'. Go to the tab '*To Characteristic Usage'* and select all the characteristics a user might want to change in the Web template using the new function. In our case we do not want to give the user the option to change the country, version, or product in a given record, but it should be able to change 'Color', 'Status', and 'Comment'. We select those characteristics.

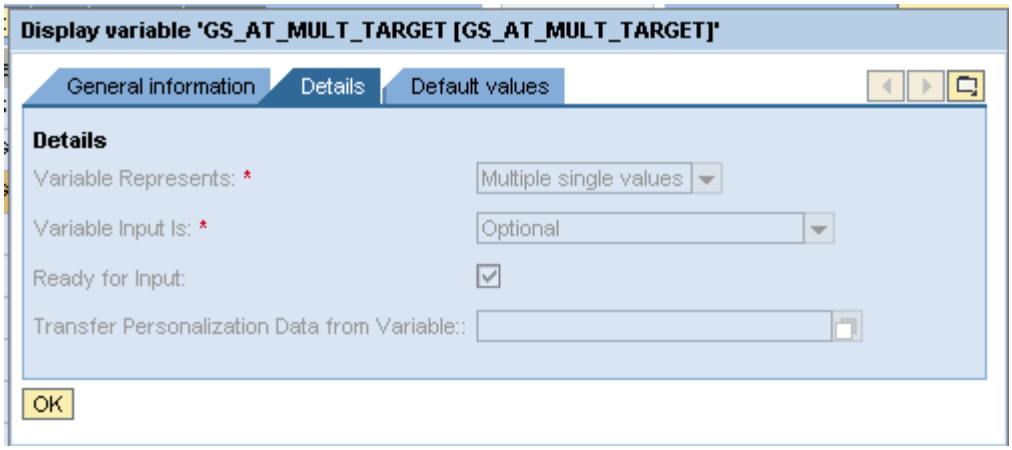


5. Now choose '*To Parameters'* and create the '*Targets for repost'* by pressing '*Change'*. As we want to be able to fill the targets by user interaction from the Web application as a selection we enter a variable for each of the characteristics.

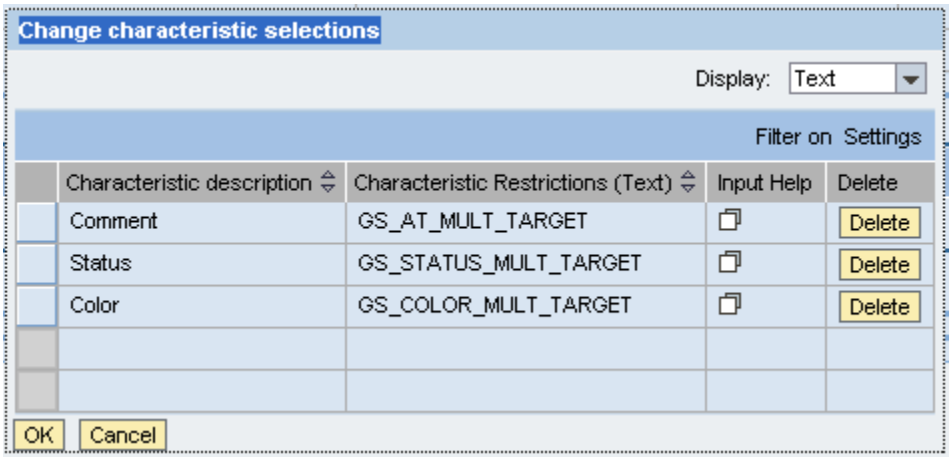


6. The target of a repost is in our case always a single value. Nevertheless for ease of programming the variable for each of the characteristics has to be defined as a variable of

multiple single values! It has to be '*optional*' and '*ready for input*'. It is crucial that the variables are defined as described, as the coding in the planning function relies on that.



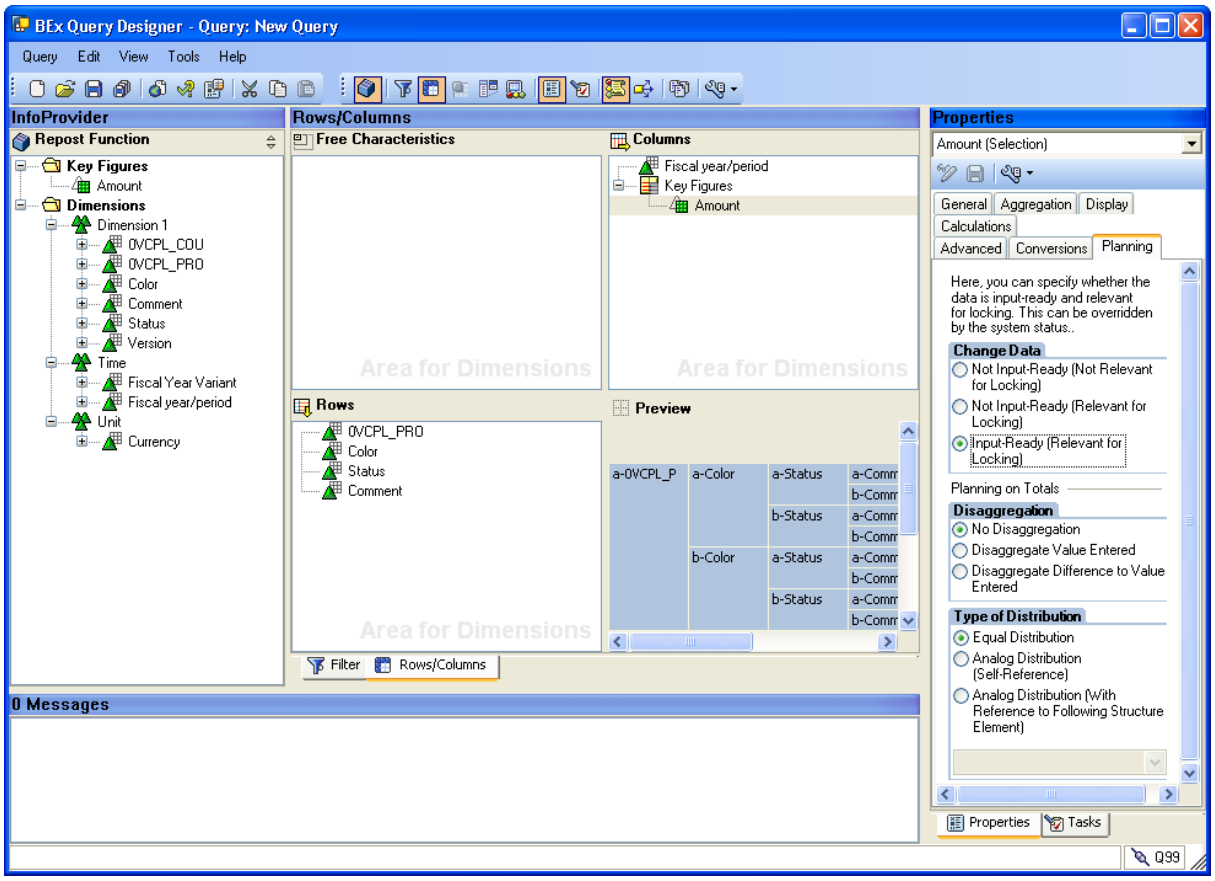
7. For each of the characteristics we have now selected a variable as characteristic restriction. Now save the planning function.



8. Based on the aggregation level already used for the planning function create an input enabled query that uses the characteristics 'Product', 'Color', 'Status', and 'Comment' in the rows.

   Make sure that you restrict the characteristics that are not contained in the rows or columns to single values. Otherwise the query will not be input enabled. Also make sure that the key figure is set to 'input ready'.

   In our query we also switch on the zero suppression for empty rows and do not show any result lines.

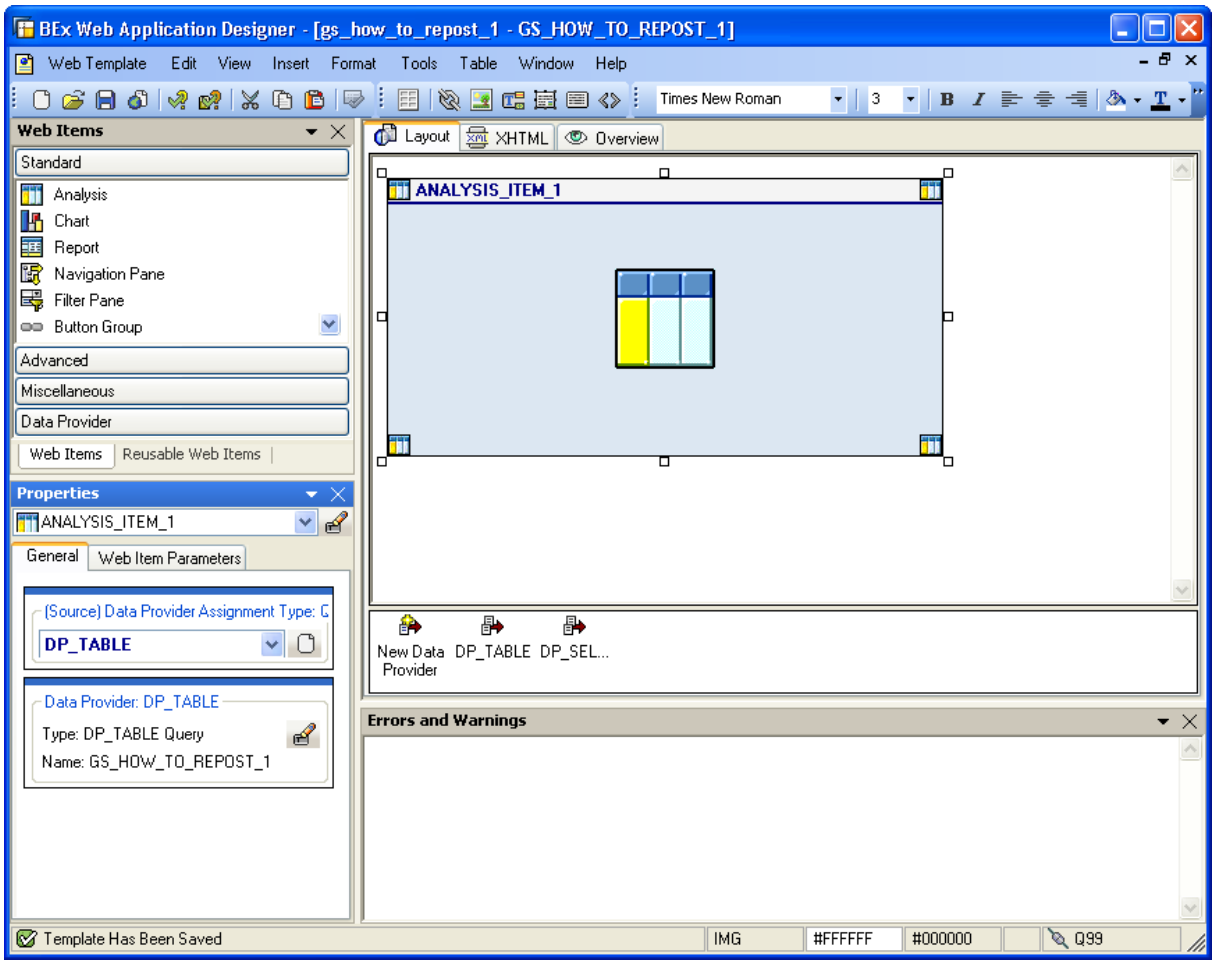


4.3 Create the BEx Web Application

1. Open the BEx Web Application Designer and create a new web template. In the web template create two data providers both containing the query we have just created. Call the first data provider **`'DP_TABLE'`** (or any other name you like) and the second one **`'DP_SELECTION'`**.

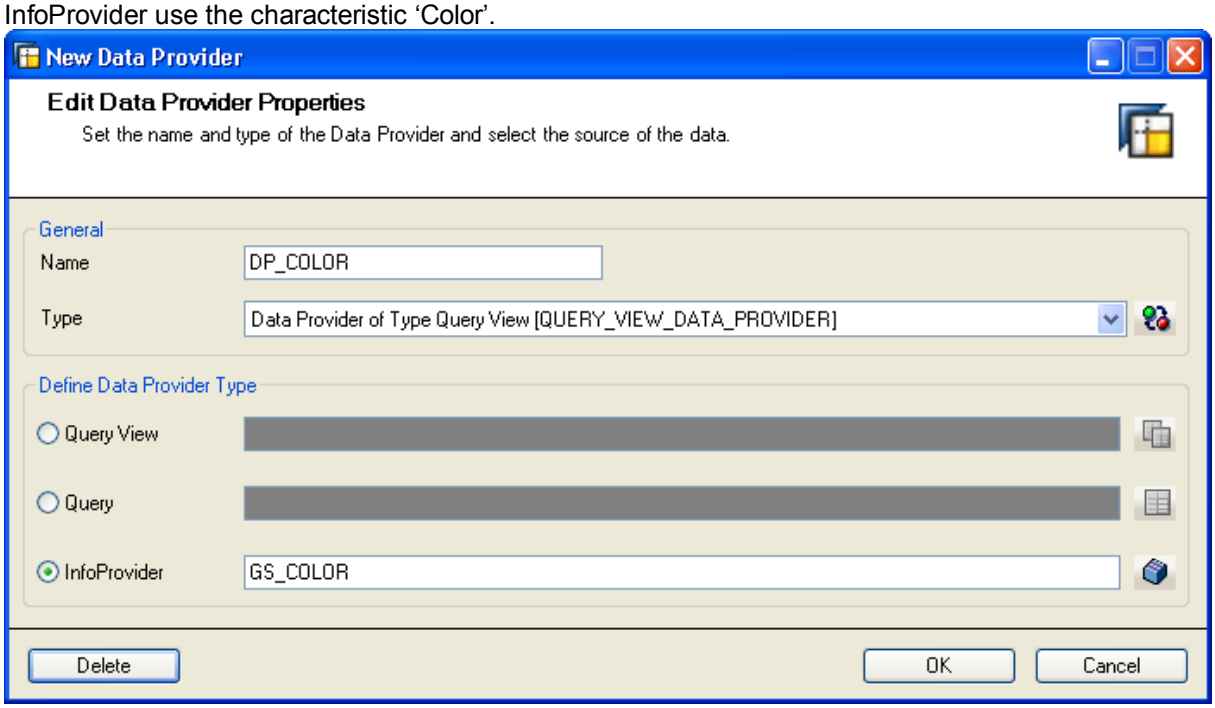
   For the data provider **`'DP_TABLE'`** create an analysis item in the web template. The other data provider will be used for transmitting the selection to the planning function.

   In the analysis item enable the row selection. We will use the row selection for specifying which records should be changed by our planning function. If the user should be able to select multiple records in which the characteristic values should be changed then choose the option *'Multiple'*. Otherwise choose *'Single'*.

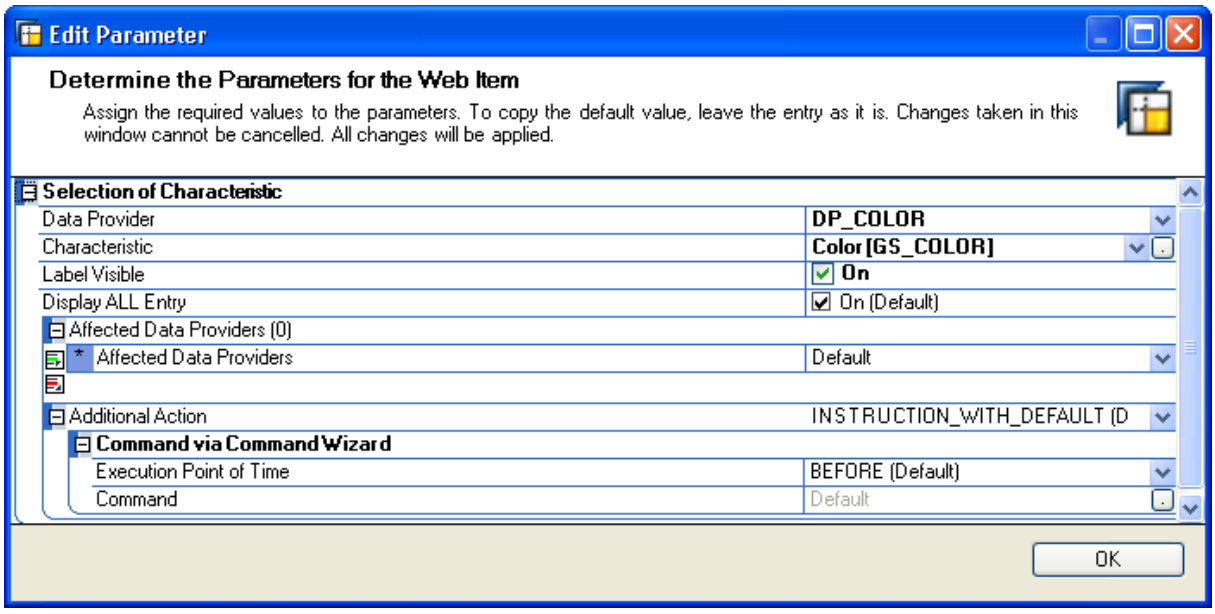


2. We now need screen elements to specify the target values for the characteristics. For the characteristics ‘Color’ and ‘Status’ we will use drop down boxes, for the characteristic ‘Comment’ we want to enable the end user to freely enter a text and thus will create an input field.

Create a new data provider. Call it **'DP_COLOR'** and choose the type '*InfoProvider*'. As InfoProvider use the characteristic 'Color'.

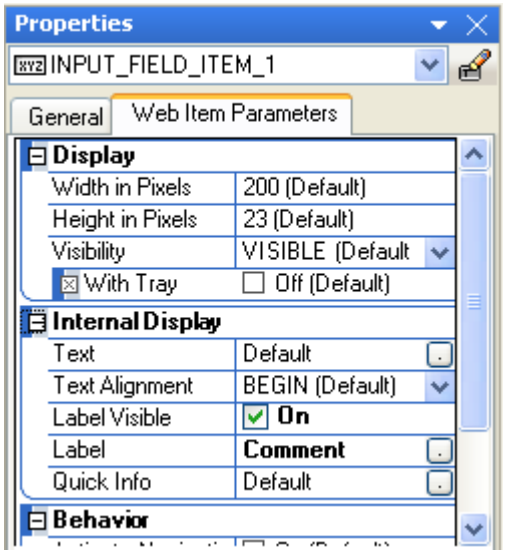


Now create a drop down box in the web template using the data provider **'DP_COLOR'**. Make sure you switch o the option *'Display ALL Entry'*.

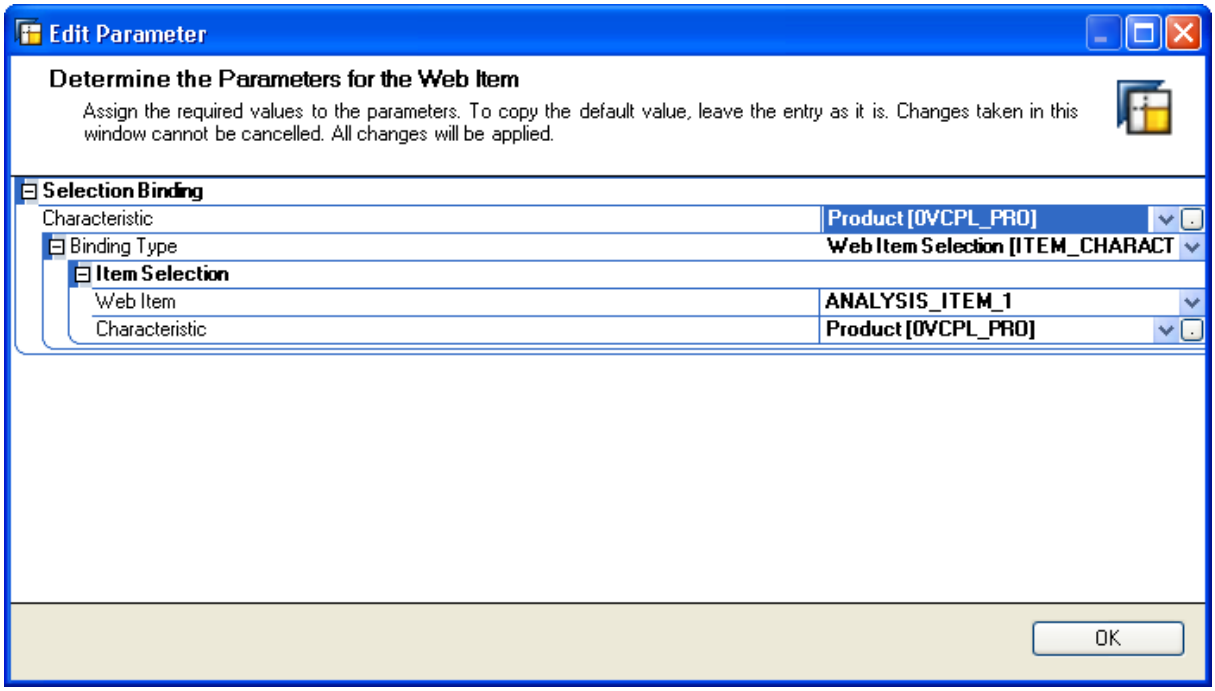


In the same way create a data provider and a drop down box for the characteristic 'Status'.

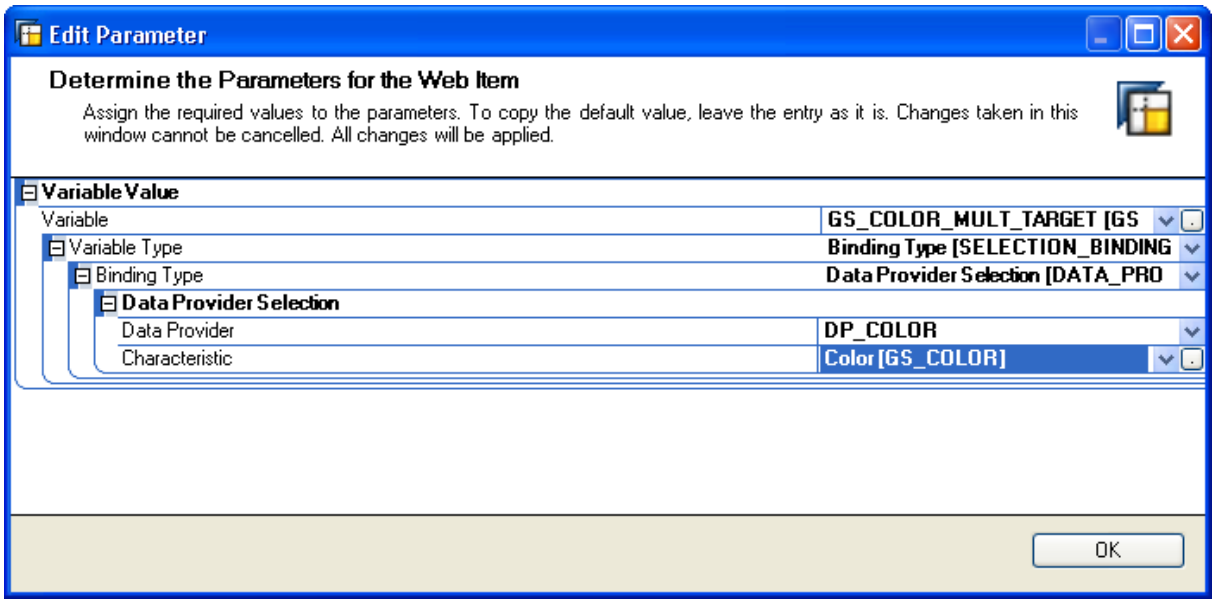
For the characteristic 'Comment' just create a standard input field. Create a label and call it **'Comment'**.



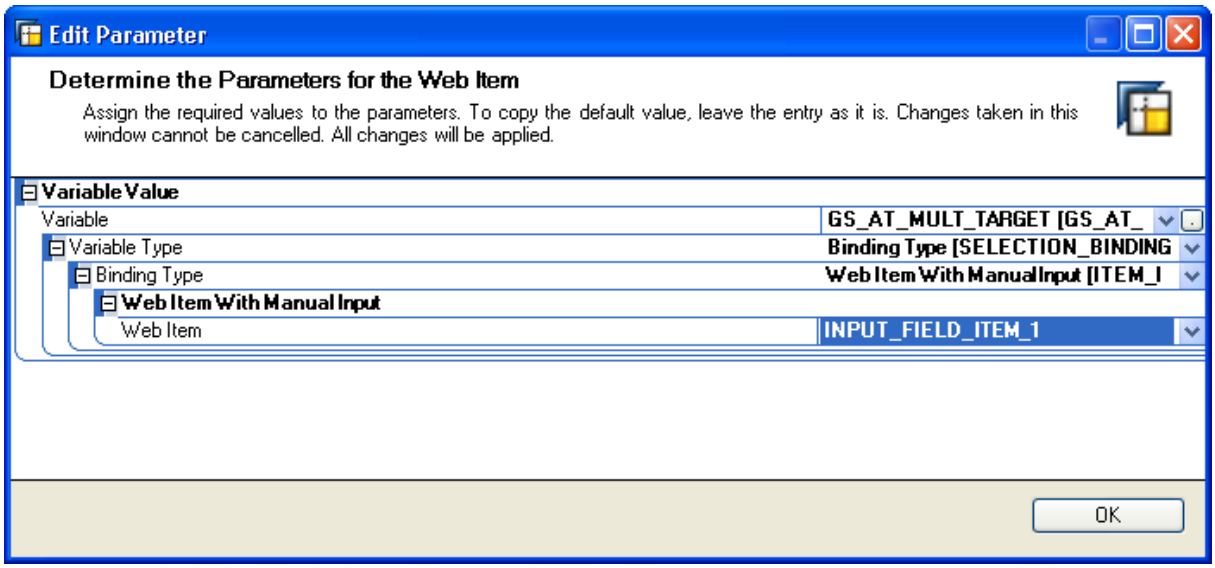
3. Now create a button for executing the planning function. Create a button group and in there a button called the button **`'Repost'`**. We need a number of commands executed once the button is pressed. As first command create a command type *'Set Filter Values Using Different Sources'*. As a target choose the data provider **`'DP_SELECTION'`**. As characteristic choose 'Product', as binding type *'Web item selection',* as web item choose the name of your analysis item, and as target characteristic choose again 'Product'.



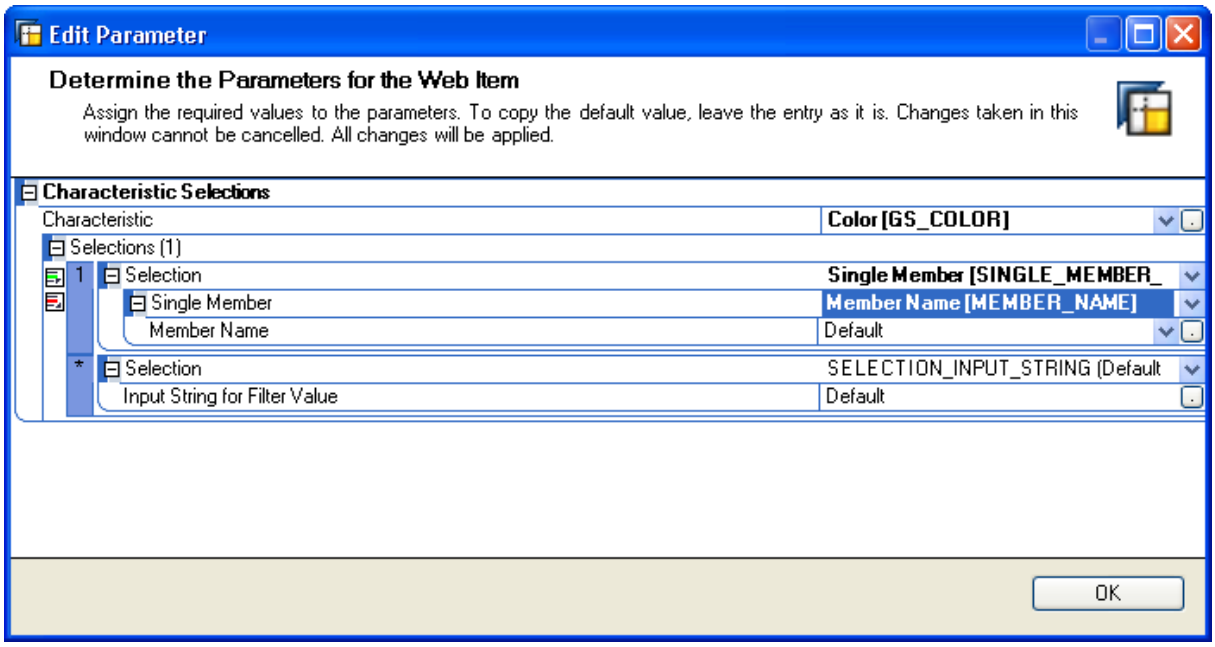
Proceed in the same way for the characteristics 'Color', 'Status', and 'Comment'. When done press the button *'Next Command'* and *'Insert'*. As command type choose *'Execute Planning Function (Simple)'*. Use **`'DP_SELECTION'`** as data provider and enter the technical name of your planning function. Now fill the variables in the planning function by mapping. The variables for 'Color' and 'Status' should be filled from the corresponding data provider.



The variable for 'Comment' should be filled from the input item.

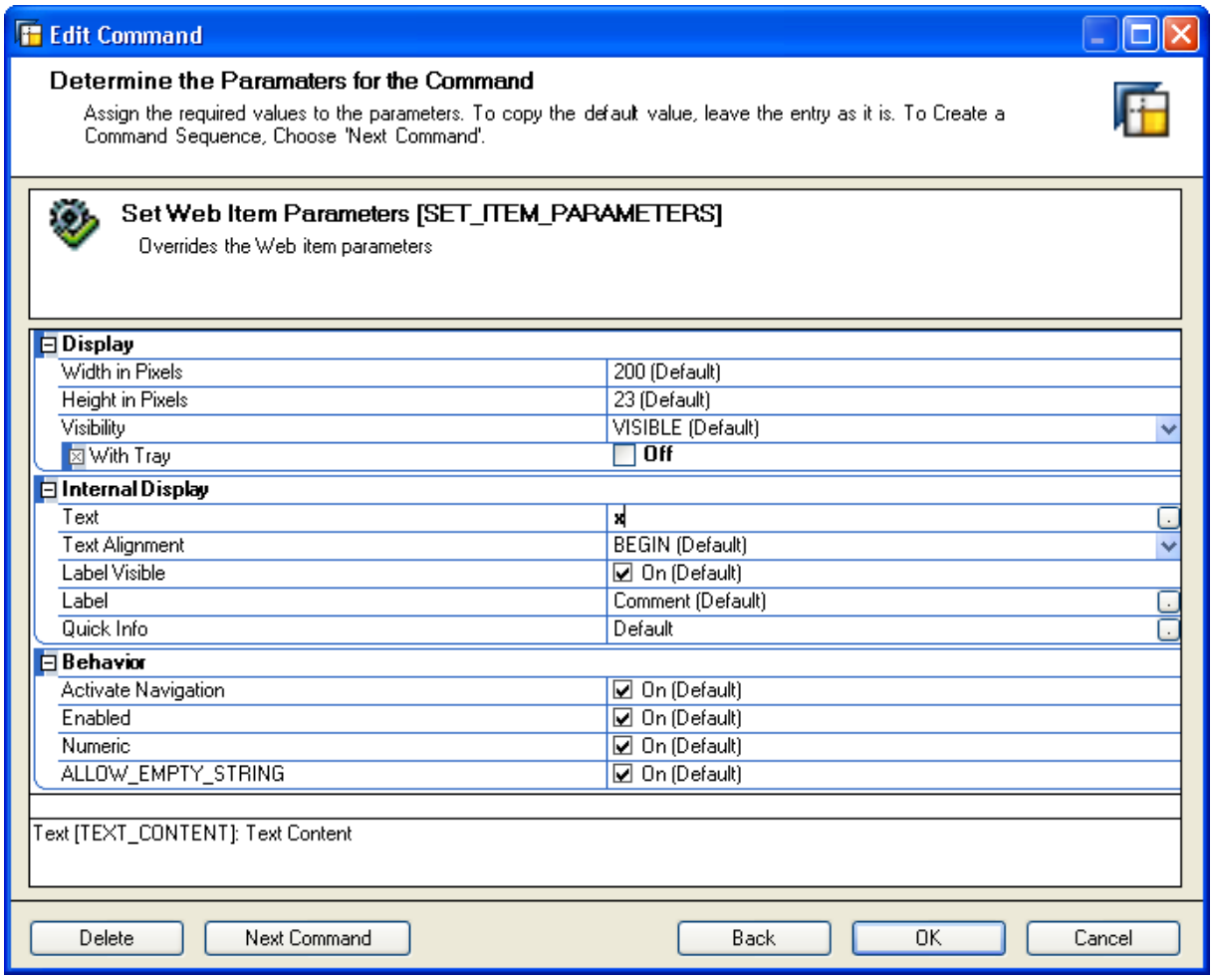


As once we have executed the planning function we want the selection in the drop down boxes and the input field to be cleared we insert some more commands. Now choose the command *'Set Filter Values'*. The target data provider is 'DP_COLOR'. For the characteristic 'Color' set a single member selection and leave the member (which is the characteristic value) empty. Create another command for the data provider 'DP_STATUS' in the same way.

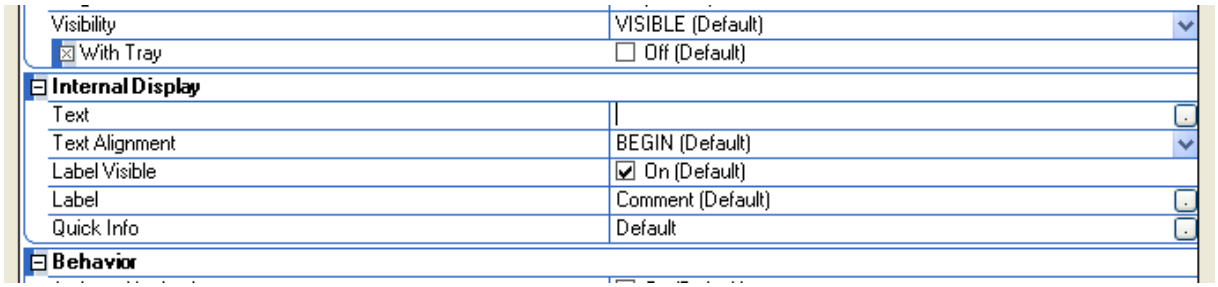


We also want to clear the input field once the planning function has been executed. Because of some internal buffering mechanism in the Web application runtime we have to use a little trick here. We have two options to do that. We could use Java script to clear the input field directly. But then all commands of the command sequence we have just created have to be done in Java script. It is possible to create this Java script commands via a wizard but changes to the commands have to be done in the java script directly or all the commands have to be created in the wizard again. As this is quite clumsy we use another work around.

Create a new command *'Set Web Item Parameters'* in the command sequence. As web item choose the input item. Unfortunately setting the text field to am empty value does not clear the input field as the internal buffering of the web runtime runs into a problem. Thus we first set an arbitrary value (here we use `'x'`).



We create another command *'Set Web Item Parameters'* for our input field and this time we set an empty text. Now the field is cleared once we execute the repost function (and the command sequence).



Save you web template.

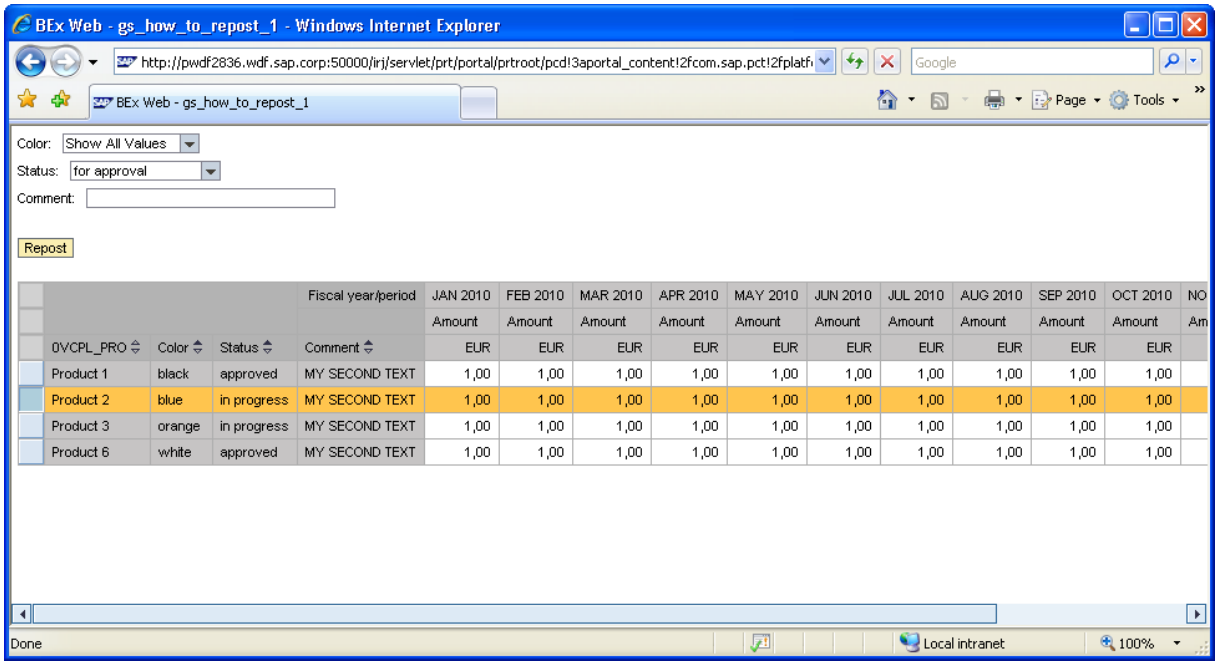
4.4 Adapt the ABAP Exit for Changing the Selection

If you test you web layout now you will run into an error sent from the planning function. The selection for the planning function comes from the data provider that is restricted by the row selection. As you want to change some of the characteristic values obviously the new value specified in the drop down box or the input field is not contained in the row selection and thus not in the selection for the planning function. One could just enlarge the selection to make sure any target value is in the selection. The negative side of this approach is that many records that should not be changed are then selected by the planning function which has a negative impact on the performance. Thus we want to make sure that the least possible number of records is selected. This is why we use the data provider restricted by the row selection. Nevertheless the target value must be contained in the selection. We ensure this by enlarging the selection using an ABAP exit.

The technique for the ABAP exit is described in note 1101726 and in the paper 'How To…Run Planning Functions on Changed Records in BI Integrated Planning'. Please proceed as described there and create and register a function module that enhances the selection by the target values of the repost function. You will find the relevant coding in Appendix B. You will have to adapt the coding by using the names of the variables you have used in your setup. If you already have implemented such an exit add a check for the name of the planning function and adapt the coding in such a way that the exit implementation only enhances the selection when we are running the repost function.

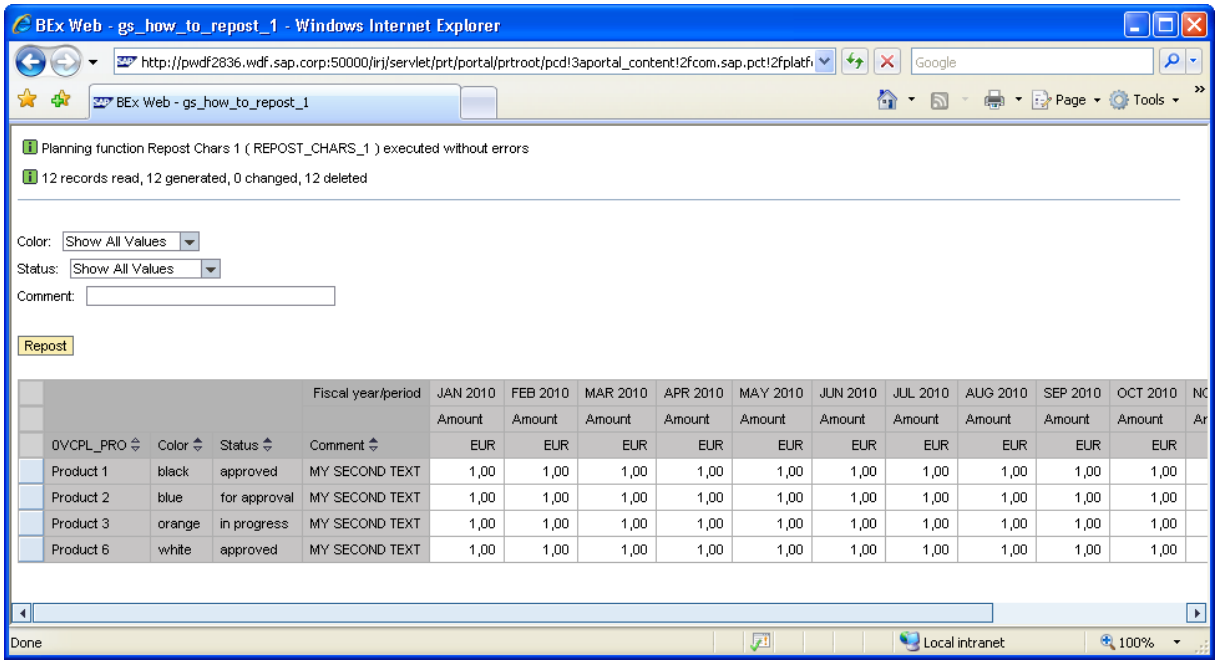
4.5 Test Your Web Template

Your web template is now finished. Start the template if not yet done create some data. Say now in our example we want to change the status in the second row. Select the second row and select a status from the drop down box. Do not change the drop down box for color or the text field for the comment.



Now run the planning function. Only the selected row is changed. As you can see from the messages only the necessary records have been selected. The drop down box for status is reset

to the original state. You can go on and change any of the characteristics (one after the other or all in one go) as you like.



5. Appendix

Appendix A – Coding for the method 'IF_RSPLFA_SRVTYPE_IMP_EXEC~EXECUTE'

```
method IF_RSPLFA_SRVTYPE_IMP_EXEC~EXECUTE.

  DATA:
    l_r_data    TYPE REF TO data,
    l_r_param_data_sel type ref to IF_RSPLFA_PARAM_DATA_SEL,
    l_t_sel type RSPLF_T_CHARSEL,
    l_t_sel_2 type RSPLF_T_CHARSEL,
    l_s_sel type RSPLF_S_CHARSEL,
    l_s_sel_2 type RSPLF_S_CHARSEL,
    l_last_iobjnm type RSIOBJNM,
    l_tabix type i.

  FIELD-SYMBOLS:
    <f>         TYPE ANY,
    <l_s_data>  TYPE ANY,
    <l_th_data> TYPE ANY TABLE.

* create the work areas
  CREATE DATA l_r_data LIKE LINE OF c_th_data.
  ASSIGN l_r_data->* TO <l_s_data>.

  CREATE DATA l_r_data LIKE c_th_data.
  ASSIGN l_r_data->* TO <l_th_data>.

  CALL METHOD I_R_PARAM_SET->GET_PARAM_DATA_SEL
    EXPORTING
      I_PARAM_NAME       = 'TARGETS'
    RECEIVING
      R_R_PARAM_DATA_SEL = l_r_param_data_sel.

  CALL METHOD L_r_PARAM_DATA_SEL->GET_T_SEL
    RECEIVING
      R_T_SEL = l_t_sel.

  clear l_last_iobjnm.

  sort l_t_sel.

*check whether we have new SINGLE ENTRIES and delete other entries
  loop at l_t_sel into l_s_sel.
* if we have a new iobjnm
    check l_s_sel-iobjnm <> l_last_iobjnm.
* check whether the next entry is for the same characteristic
    l_tabix = sy-tabix + 1.
    read table l_t_sel index l_tabix into l_s_sel_2.

    if sy-subrc <> 0 or ( sy-subrc = 0 and l_s_sel_2-iobjnm <> l_s_sel-iobjnm ).
* no next of the same characteristic.
      if l_s_sel-sign = 'I' and l_s_sel-opt = 'EQ'.
        insert l_s_sel into table l_t_sel_2.
      endif.
    endif.
    l_last_iobjnm = l_s_sel-iobjnm.
  endloop.

  l_t_sel = l_t_sel_2.
```

```
  clear <l_th_data>.
  LOOP AT c_th_data INTO <l_s_data>.
    loop at l_t_sel into l_s_sel.
      assign component l_s_sel-iobjnm of structure <l_s_data> to <f>.
      <f> = l_s_sel-low.
    endloop.
    Collect <l_s_data> INTO <l_th_data>.
  ENDLOOP.

  c_th_data = <l_th_data>.

endmethod.
```

Appendix B – Coding for the ABAP exit changing the selection

```
FUNCTION Z_DELTA_SEL2.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     REFERENCE(I_INFOPROV) TYPE  RSINFOPROV
*"     REFERENCE(I_SERVICE) TYPE  RSPLF_SRVNM
*"     REFERENCE(I_SELOBJ) TYPE  RSZCOMPID
*"  CHANGING
*"     REFERENCE(C_T_CHARSEL) TYPE  RSPLF_T_CHARSEL
*"  EXCEPTIONS
*"      SELECTION_EMPTY
*"----------------------------------------------------------------------


  DATA: l_r_plan_buffer TYPE REF TO if_rsplfa_plan_buffer,
        l_t_charsel TYPE rsplf_t_charsel,
        l_s_charsel TYPE rsplf_s_charsel,
        l_r_th_data TYPE REF TO data,
        l_r_mesg TYPE REF TO if_rsplfa_msg,
        l_lines TYPE i.

  data: l_t_range TYPE RSPLS_T_RNG,
        l_s_range like line of l_t_range,
        l_chanm TYPE RSIOBJNM.



  FIELD-SYMBOLS: <l_s_data> TYPE ANY,
                 <l_country> TYPE ANY,
                 <l_prod> TYPE ANY,
                 <l_th_data> TYPE HASHED TABLE.


  case i_service.

   when 'REPOST_CHARS_1'.

      l_t_charsel = c_t_charsel.

* we have to enlarge the selection by the target of the repost function
      clear l_t_range.
      clear l_s_charsel.

      CALL METHOD CL_RSPLFR_VAR_CONT=>GET_VAR_VALUE
        EXPORTING
          I_VARIABLE = 'GS_COLOR_MULT_TARGET'
*     I_VSHIFT   = 0
*     I_TS_CMP   =
```

```
        IMPORTING
          E_T_RANGE  = l_t_range
*     E_VTYPE    =
           E_CHANM    = l_chanm
*     E_FOUND    =
          .

       describe table l_t_range lines l_lines.
       if l_lines = 1.
         read table l_t_range index 1 into l_s_range.
         if l_s_range-sign = 'I' and l_s_range-opt = 'EQ'.
           move-corresponding l_s_range to l_s_charsel.
           l_s_charsel-IOBJNM = l_chanm.

* does the entry exist already?
           read table l_t_charsel from l_s_charsel transporting no fields.
           if sy-subrc <> 0.
             insert l_s_charsel into table l_t_charsel.
           endif.
         endif.
       endif.


       clear l_t_range.
       clear l_s_charsel.
       clear l_lines.

       CALL METHOD CL_RSPLFR_VAR_CONT=>GET_VAR_VALUE
         EXPORTING
           I_VARIABLE = 'GS_STATUS_MULT_TARGET'
*     I_VSHIFT   = 0
*     I_TS_CMP   =
         IMPORTING
           E_T_RANGE  = l_t_range
*     E_VTYPE    =
           E_CHANM    = l_chanm
*     E_FOUND    =
          .

       describe table l_t_range lines l_lines.
       if l_lines = 1.
         read table l_t_range index 1 into l_s_range.
         if l_s_range-sign = 'I' and l_s_range-opt = 'EQ'.
           move-corresponding l_s_range to l_s_charsel.
           l_s_charsel-IOBJNM = l_chanm.
* does the entry exist already?
           read table l_t_charsel from l_s_charsel transporting no fields.
           if sy-subrc <> 0.
             insert l_s_charsel into table l_t_charsel.
           endif.
         endif.
       endif.

       clear l_t_range.
       clear l_s_charsel.
       clear l_lines.

       CALL METHOD CL_RSPLFR_VAR_CONT=>GET_VAR_VALUE
         EXPORTING
           I_VARIABLE = 'GS_AT_MULT_TARGET'
*     I_VSHIFT   = 0
*     I_TS_CMP   =
         IMPORTING
           E_T_RANGE  = l_t_range
*     E_VTYPE    =
           E_CHANM    = l_chanm
```

```
*     E_FOUND     =
            .

       describe table l_t_range lines l_lines.
       if l_lines = 1.
         read table l_t_range index 1 into l_s_range.
         if l_s_range-sign = 'I' and l_s_range-opt = 'EQ'.
           move-corresponding l_s_range to l_s_charsel.
           l_s_charsel-IOBJNM = l_chanm.
* does the entry exist already?
           read table l_t_charsel from l_s_charsel transporting no fields.
           if sy-subrc <> 0.
             insert l_s_charsel into table l_t_charsel.
           endif.
         endif.
       endif.

       clear l_t_range.
       clear l_s_charsel.
       clear l_lines.

       if l_t_charsel is initial or c_t_charsel = l_t_charsel.
         MESSAGE i001(rsplf)
           WITH 'No source and target specified' RAISING selection_empty.
       else.
         c_t_charsel = l_t_charsel.

       ENDIF.

  endcase.

ENDFUNCTION.
```

www.sdn.sap.com/irj/sdn/howtoguides