Beginner's Guide: Create a Custom 'Copy' Planning Function Type

Applies to:

SAP BW 7.x. For more information, visit the EDW homepage

Summary

This article explains step-by-step, how to set up and create a custom "Copy" planning function in RSPLF1, integrate it in your planning application, test and debug it and finally write it on a transport request.

Author: Jürgen Noe

Company: Kheto Consulting GmbH

Created on: 4 March 2011

Author Bio

Jürgen Noe has over 9 years consulting experience in SAP BW projects and ABAP/OO programming. Jürgen is presently working for Kheto Consulting GmbH, Germany. He's working mainly on SAP BI 7.0 and SAP BO front end tools. He is specialized in SAP BW backend, performance and administration.

Table of Contents

Introduction

Recently I had to write a custom planning function for a customer as part of a project. First I read through the help.sap.com pages and found a nice blog from Marc Bernard how you can upload a flatfile into BI-IP using a custom planning function.

Nevertheless it was a bit crucial until I had found out which parameters in RSPLF1 had to be set and which methods of the interface classes had to be implemented. This document should help you to avoid this obstacles and to give you a step-by-step guide how you can write your custom planning function using RSPLF1, how to test it in your planning application and transport it to your QA/PROD system.

The Business Case

Imagine a small company, which has 5 main customers to which it sells its products. The sales manager recorded the actual sales of this year for each period and each customer in the below Excel, marked in green. He made an ABC-analysis and detected the A, B and C customers. The Top Management decided because of the favourable economic climate to rise the overall target sales to 15% for the next year. This leads to the below sales plan and key figures, marked in blue. But they want to get rid of their Excels and prefer a solution with BI-IP.

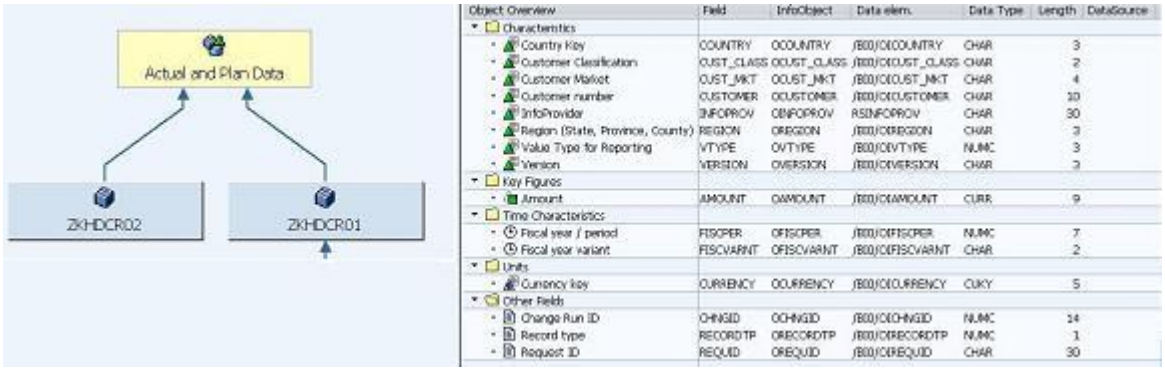
Customer	Period1	Period2	Period3	Period4	Period5	Period6	Period7	Period8	Period9	Period10	Period11	Period12	Actuals
1	10000	11000	12000	8000	7000	9000	8000	10000	12000	11000	10000	12000	120000
2	12000	13000	11000	10000	12000	5000	6000	7000	8000	10000	12000	10000	116000
3	25000	24000	23000	24000	25000	24000	23000	25000	25000	24000	25000	23000	290000
4	20000	20000	20000	19000	18000	17000	19000	21000	19000	18000	20000	18000	229000
5	30000	20000	10000	20000	30000	25000	20000	30000	35000	30000	30000	35000	315000
													1070000
Target:	Actual + 15%												
Customer	Period1	Period2	Period3	Period4	Period5	Period6	Period7	Period8	Period9	Period10	Period11	Period12	Plan
1													138000
2													133400
3													333500
4													263350
5													362250
													1230500

Picture 1: Excel planning sheet

Now let's set up the BW data model and create a custom planning function, which calculates the overall sales key figure and distributes it to the 5 customers accordingly.

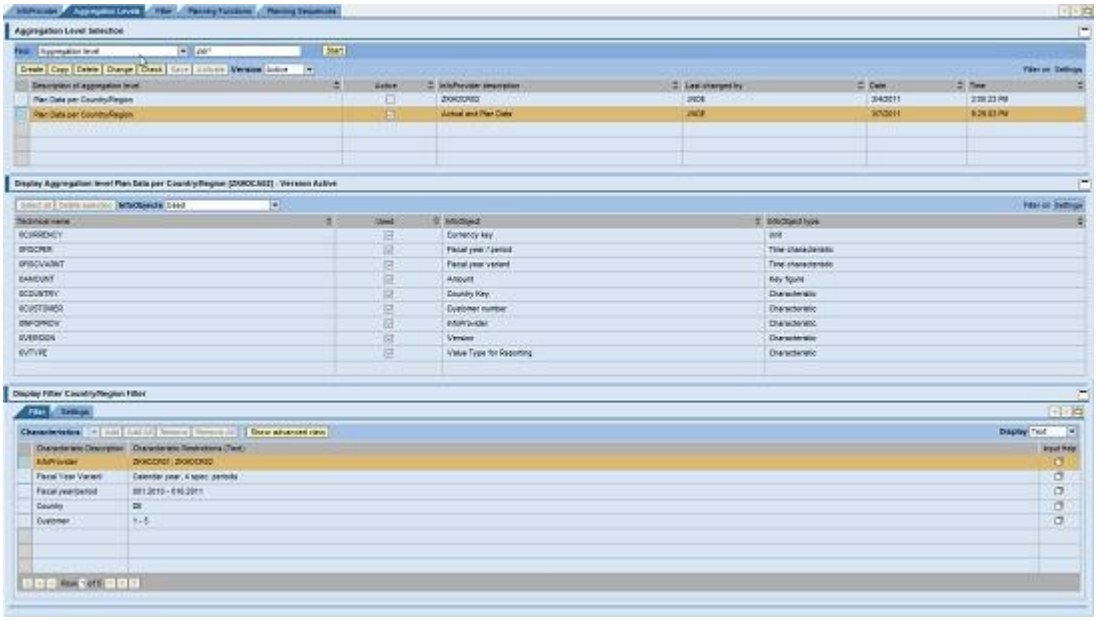
Setting up the environment

To keep it simple, I created a static planning scenario for customer sales of the current year 2010 and next year 2011. I defined a basis cube (ZKHDRC01) for my actual data. The cube holds data for customers in country 'DE' for the period 01.2010 to 12.2010. You can load the data by a flatfile for example. Then I defined a real-time cube (ZKHDRC02) to hold the plan data with the same characteristics and key figures as my actual cube. I combine both cubes in a Multiprovider. See the data model below:



Picture 2: Data modell

Now call TA RSPLAN and set up your aggregation layer and define filter. For details see picture below:



Picture 3: Aggregation level and filter

Now you can define your planning function in a custom class and afterwards make it available in TA RSPLF1.

Create a custom class

The easiest way to create your own class is take an existing one and copy it to your namespace.

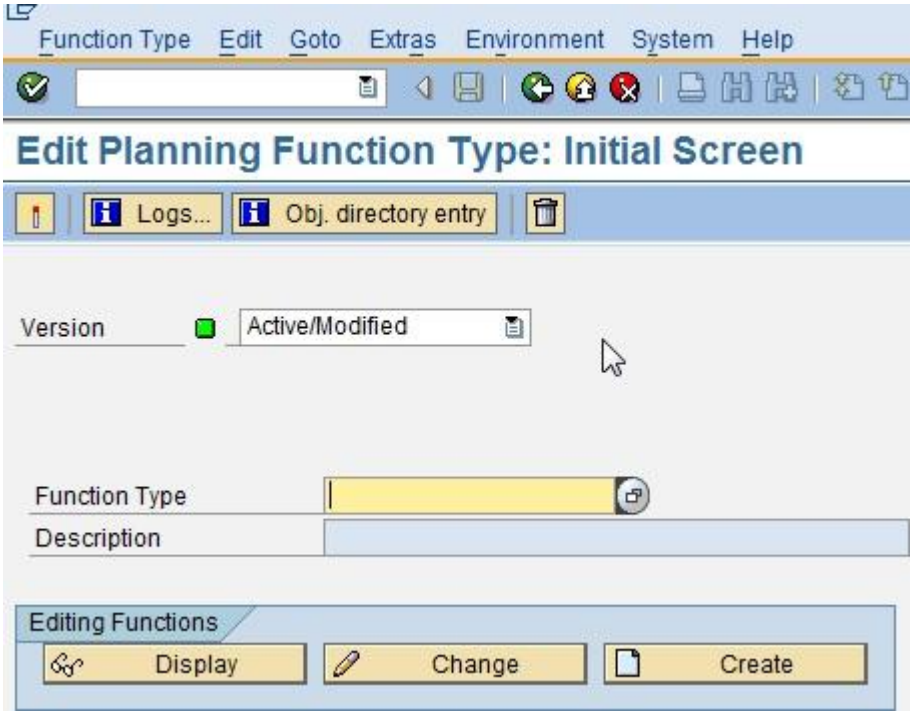
In SE 24 enter CL_RSPLFC* and choose from the existing classes. We want to copy the data from actual to plan and modify the key figure values. Class CL_RSPLFC_COPY is sufficient for the moment.

Copy the class to ZCL_PLFC_COPY and activate it.

All necessary methods have been implemented and we only need to adopt the code to our needs.

For the first test now create your custom planning function type in RSPLF1.

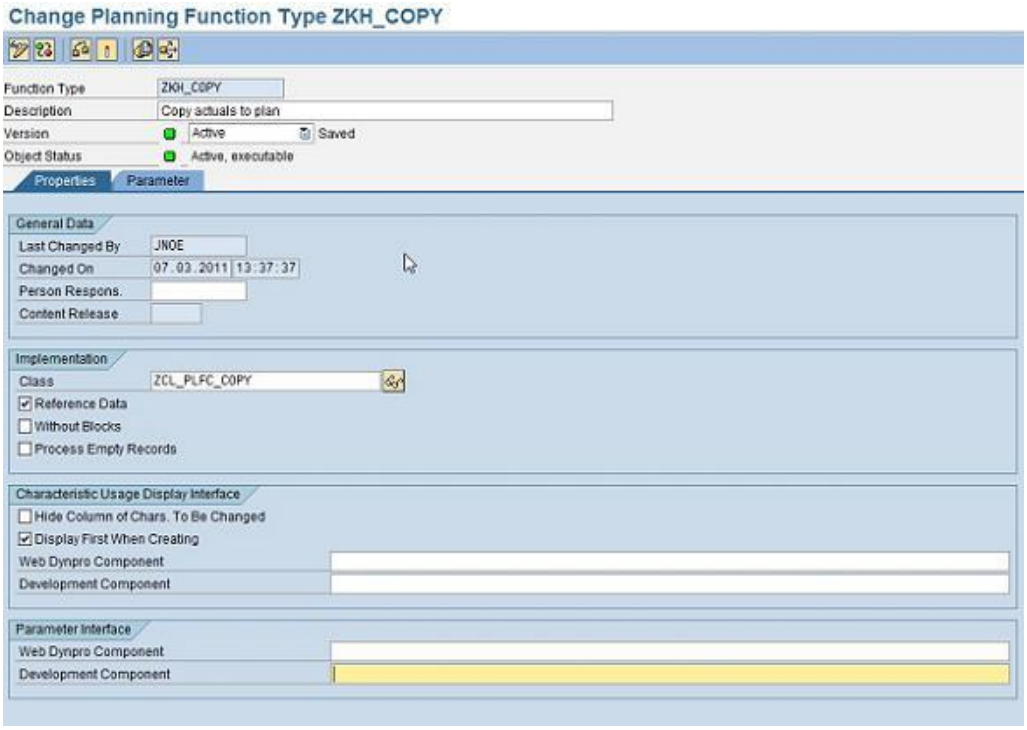
Call Transaction RSPLF1



Picture 4: RSPLF1

Enter e.g. ZKH_COPY as name and press "CREATE" button.

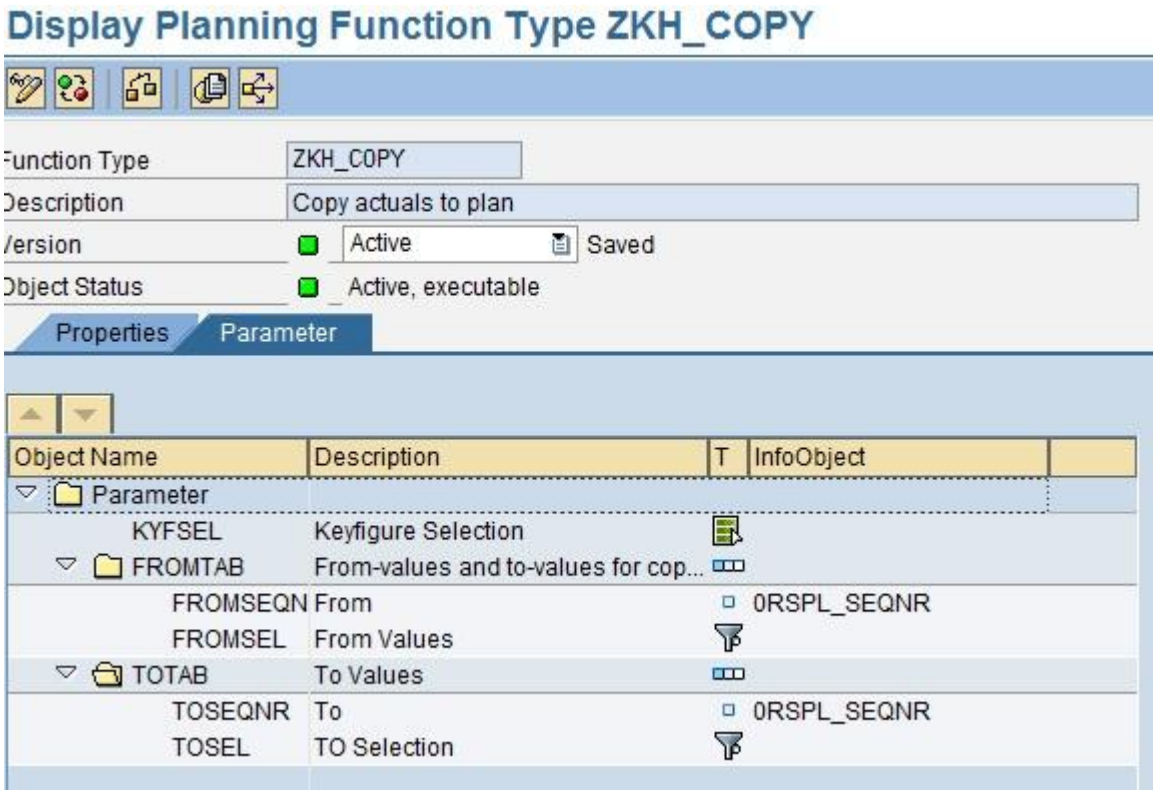
In the next screen make following entries:



Picture 5: RSPLF1 settings

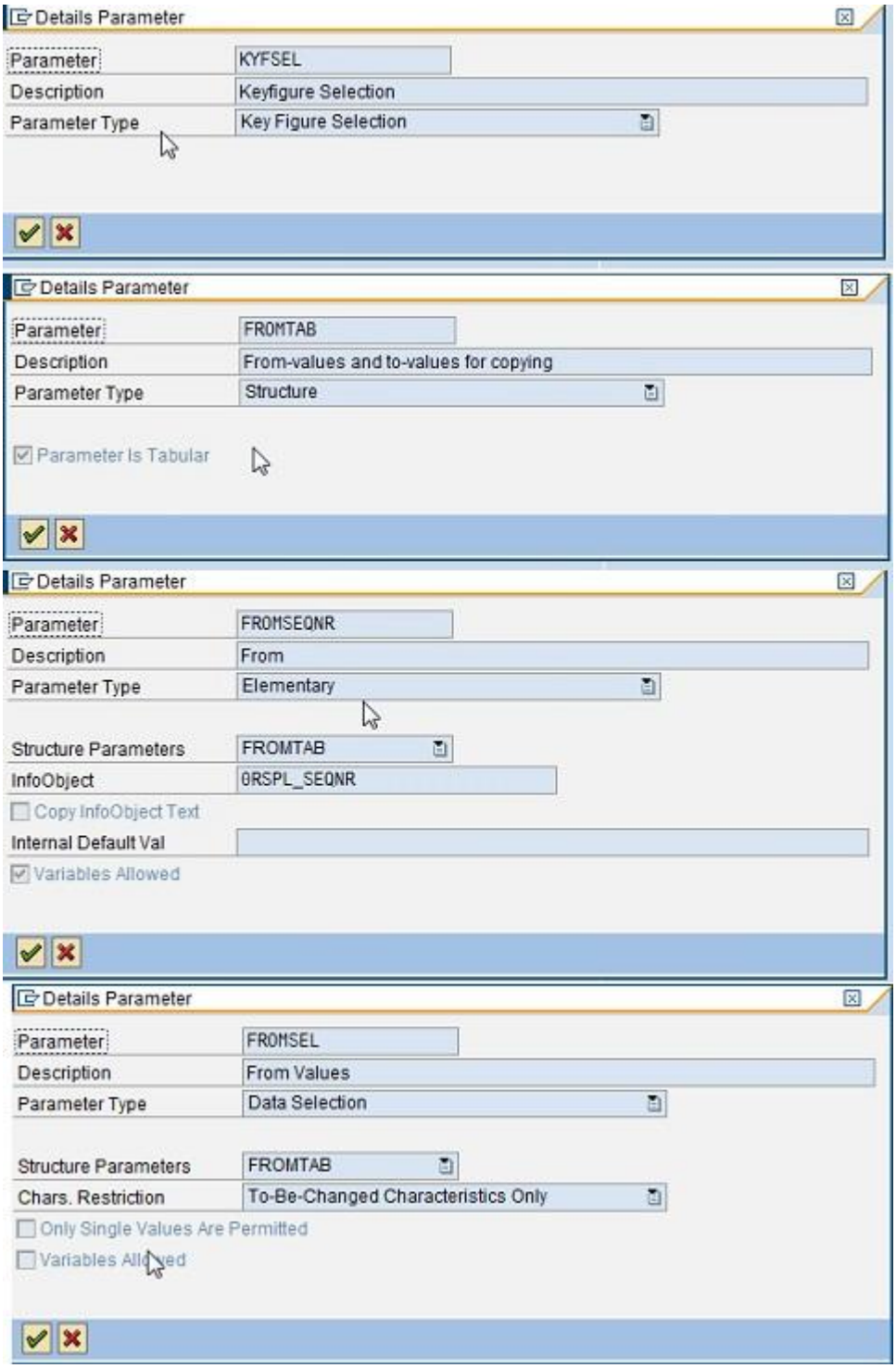
Because we checked "Reference Data" we have to implement interface IF_RSPLFA_SRVTYPE_IMP_EXEC_REF in our class later. Otherwise we have to implement interface IF_RSPLFA_SRVTYPE_IMP_EXEC.

The original Copy-Function needs a parameter for the key figures named KYFSEL and for the changing parameters named FROMTAB and TOTAB. These parameters are implemented in the method "CREATE_RULES" of class "ZCL_PLFC_COPY". The parameter check is done in method IF_RSPLFA_SRVTYPE_IMP_CHECK~CHECK_PARAM_SET. Create these parameters on the next screen:



Picture 6: Parameters of planning function type

For the setting of each parameter see the next screenshot below:



Picture 7: Parameter settings

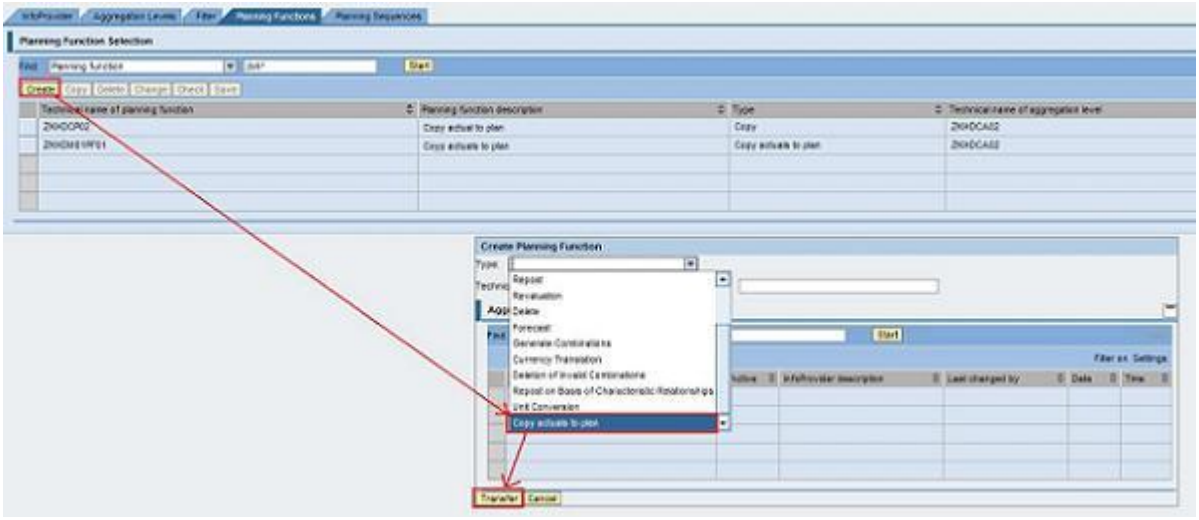
The TOTAB parameters are created accordingly to the FROMTAB parameters.

Activate it.

Integrate function in your planning application

Now you can integrate your custom function into your planning application.

Goto RSPLAN, choose your aggregation layer and filter. Switch to tab "Planning functions". Press "Create" a new planning function.

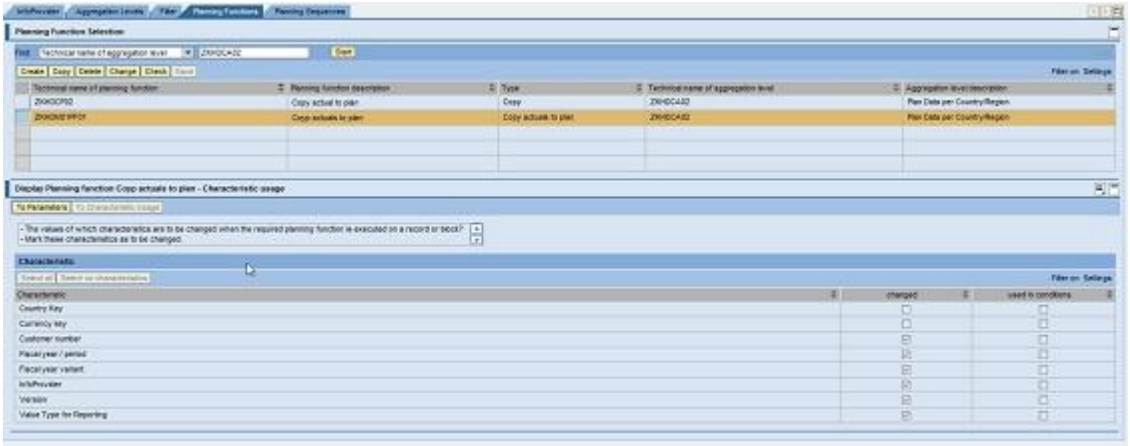


Picture 8: Create custom planning function

In the dropdown box you find all available planning function types. The standard planning function types appear at the beginning, the custom planning function types at the end.

Enter e.g. ZKHDM01PF01 as technical name and "Copy actuals to plan" as description for your planning function. Then press "Transfer".

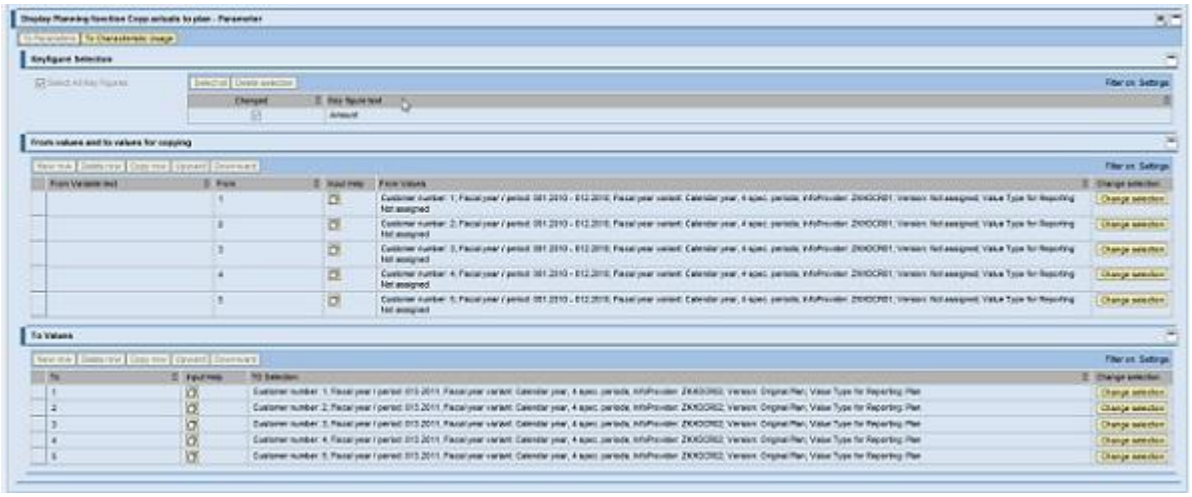
In our business case we have to change Version, Value type, fiscal period, customer and Infoprovider. We choose as version "000"for original plan and as value type "020" for plan. The fiscal period will be changed from [01.2010; 12.2010] to [13.2011]. Infoprovider will be changed from "ZKHDRC01" to "ZKHDRC02", which means in fact copying data from one Infoprovider to another. Set these characteristics as changeable:



Picture 9: Set changeable parameters

The check for changeable characteristics is done in method IF_RSPLFA_SRVTYPE_IMP_CHECK~CHECK_CHAR_USAGE.

Switch now to parameters screen and enter the values for From- and To- Values with "Change selection" button. To make life easy for the moment just change period [01.2010;12.2010] to 13.2011. Create for each customer a new selection row. You can use the "Copy row" button to create similar lines and change customer. Your result should look like the one on the next screen:



Picture 10: Choose data selection for parameters

As you can see, the interface looks different to the original copy function. The parameters FROMTAB and TOTAB are put together in one table and you have only one "Change selection" button per row. To achieve this, you can define your own webdynpro interface in the parameter interface of your custom planning function type. See picture 5 for the option. I will give you an example how you can create your own parameter screen using Webdynpro in a follow-up article.

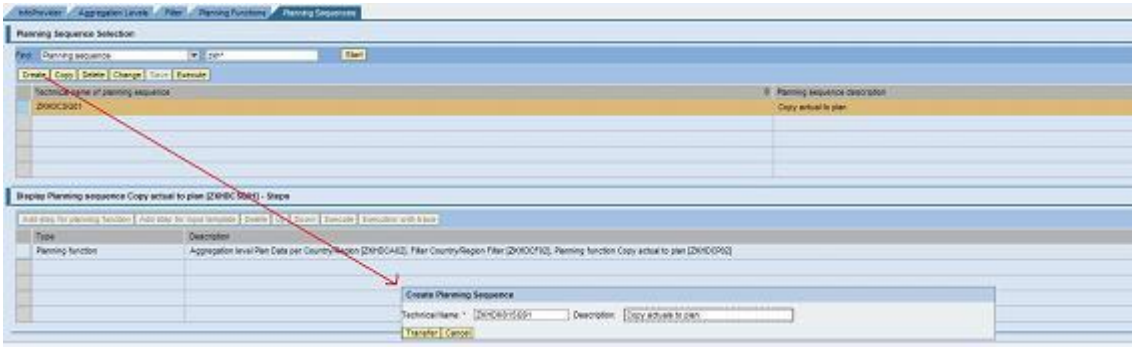
The planning function will be executed for each customer and a new row for each customer will be generated in the results.

Save this planning function.

If you check your planning function at the top of the screen, no errors should be reported.

Create a planning sequence

Now create a planning sequence in RSPLAN and press "Transfer".



Picture 11: Create Planning sequence

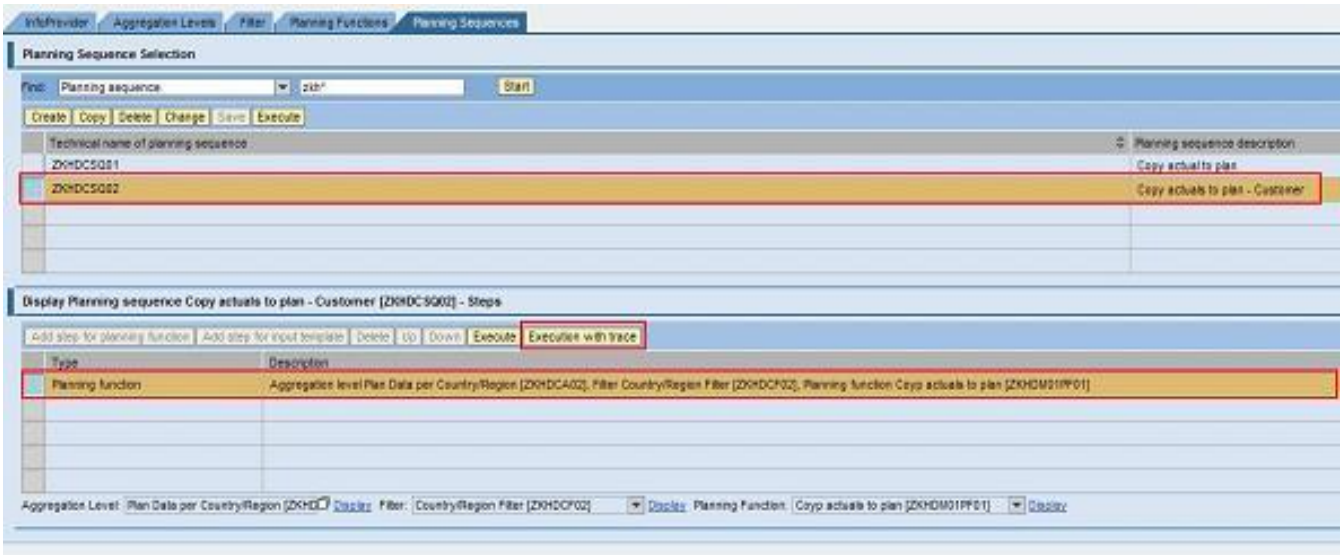
Add planning function by pressing "Add step for planning function" to planning sequence. Enter a technical name and a description. Press button "Transfer". Choose filter and planning function created in the step before in the appropriate drop-down boxes:

Picture 12: Add Planning function to Planning sequence

Save your planning sequence.

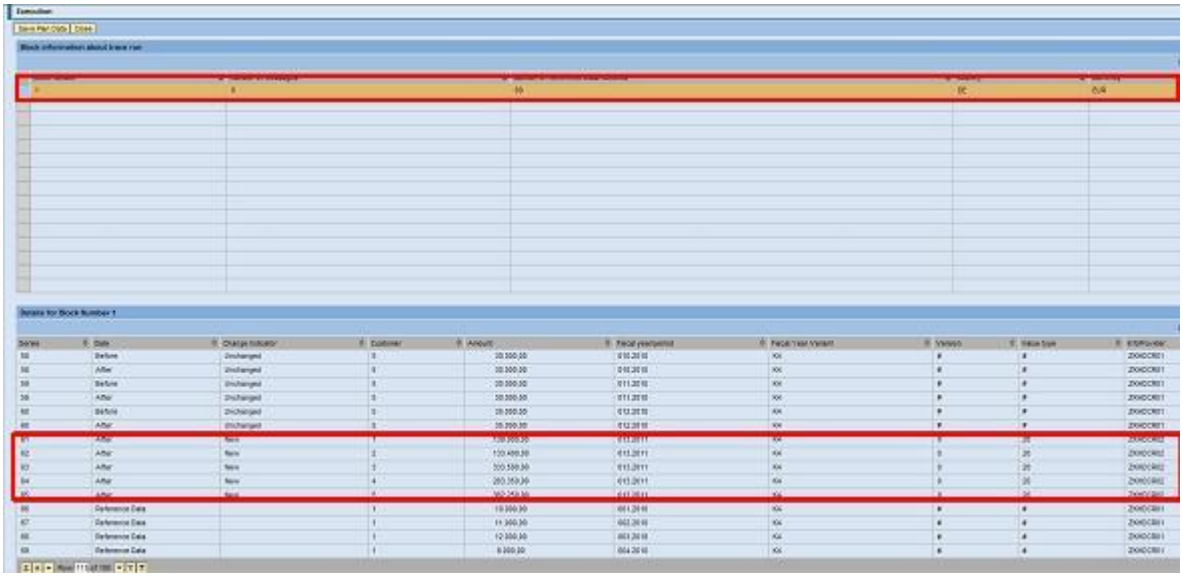
Execute Planning Sequence

Now run your planning sequence. Select the planning sequence. Then select the planning function. Choose "Execution with trace".



Picture 13: Execution with trace

You're planning function should execute now and generate new records for customers. The data for periods [01.2010;12.2010] should be copied to period 13.2011. You can view the generated data and see the results:



Picture 14: Results after execution

With Button "Save plan data" the generated data will be written to your plan cube. A new open request will be created.

Change code in planning function

Now we can do the changes to the code. The key figure calculation is done in method IF_RSPLFA_SRVTYPE_IMP_EXEC_REF~EXECUTE.

The new or changed lines are marked red.

Replace the existing code with this code:

```
METHOD IF_RSPLFA_SRVTYPE_IMP_EXEC_REF~EXECUTE.
  DATA: l_rulepos TYPE rsplf_rulepos,
        l_ts_rule TYPE t_ts_rule,
        l_s_rule TYPE t_s_rule,
        l_r_data_wa TYPE REF TO data,
        l_r_block_line type ref to data,
        l_r_msg TYPE REF TO cl_rsplfu_msg,
        l_t_msg TYPE rsplf_t_msg,
        l_s_msg TYPE rsplf_s_msg,
        l_dummy TYPE c,                                    "#EC NEEDED
        l_r_i_msg TYPE REF TO cl_rsplfu_msg,
        l_r_dta TYPE REF TO cl_rsd_dta,
        l_t_dta_pro TYPE rsd_t_dta_pro,
        l_s_dta_pro TYPE rsd_s_dta_pro,
        l_s_iobj TYPE rsplf_s_iobjnm,
        l_th_iobj TYPE rsplf_th_iobj,
        l_r_s_chas TYPE REF TO data,
        l_s_sel TYPE rspls_s_sel,
        l_s_rng TYPE rspls_s_rng,
        l_t_contr_msg  TYPE if_rspls_cr_types=>tn_t_mesg,
        l_s_contr_msg  TYPE if_rspls_cr_types=>tn_s_mesg,
        l_is_valid     TYPE rs_bool.

  FIELD-SYMBOLS: <s_data_wa> TYPE ANY,
                 <s_data_wa2> TYPE ANY,
                 <s_block_line> type any,
```

```
                  <th_chas> TYPE HASHED TABLE,
                  <customer> type /bi0/oicustomer,
                  <amount> type /bi0/oiamount,
                  <s_chas> TYPE ANY,
                  <cha> TYPE ANY.

  l_r_i_msg ?= i_r_msg.
  CREATE OBJECT l_r_msg.

* ==== get or create p_r_th_chas for generation of combinations
  IF p_r_th_chas IS INITIAL.
    l_r_dta = cl_rsd_dta=>factory( p_infoprov ).
    l_r_dta->dta_get_info( IMPORTING e_t_dta_pro = l_t_dta_pro ).

    LOOP AT l_t_dta_pro INTO l_s_dta_pro WHERE iobjtp = 'CHA' OR iobjtp = 'TIM' OR
iobjtp = 'UNI'.
      l_s_iobj-iobjnm = l_s_dta_pro-iobjnm.
      INSERT l_s_iobj INTO TABLE l_th_iobj.
    ENDLOOP.
    l_r_s_chas = cl_rsplfr_controller=>p_r_cr_controller->n_r_structure->create_data(
).
    ASSIGN l_r_s_chas->* TO <s_chas>.
    CREATE DATA p_r_th_chas LIKE HASHED TABLE OF <s_chas>
                            WITH UNIQUE KEY (l_th_iobj).
  ENDIF.

  IF p_r_object_creation is initial.
   create object p_r_object_creation exporting i_t_char_usage = p_t_char_usage
                                               i_r_cr_controller =
cl_rsplfr_controller=>p_r_cr_controller.

  endif.

  ASSIGN p_r_th_chas->* TO <th_chas>.
  CLEAR <th_chas>.
  l_r_s_chas = cl_Rsplfr_controller=>p_r_cr_controller->n_r_structure->create_data(
).
  ASSIGN l_r_s_chas->* TO <s_chas>.
* ==== get or create rule
  l_rulepos = i_r_param_set->get_rulepos( ).
  READ TABLE p_ts_rule TRANSPORTING NO FIELDS WITH KEY rulepos = l_rulepos.
  IF sy-subrc <> 0.
    l_ts_rule = create_rules( i_r_param_set = i_r_param_set i_r_msg = l_r_msg ).
    INSERT lines of l_ts_rule INTO TABLE p_ts_rule.
    IF l_r_msg->is_filled( ) = rs_c_true.
      l_t_msg = l_r_msg->get_t_msg( ).
      LOOP AT l_t_msg INTO l_s_msg.
        MESSAGE ID l_s_msg-msgid TYPE l_s_msg-msgty NUMBER l_s_msg-msgno
                WITH l_s_msg-msgv1 l_s_msg-msgv2 l_s_msg-msgv3 l_s_msg-msgv4
                INTO l_dummy.
        i_r_msg->add_msg( ).
      ENDLOOP.
      RETURN.
    ENDIF.
  ENDIF.
```

```
* ==== clear work areas
  CREATE DATA l_r_data_wa LIKE LINE OF i_th_ref_data.
  ASSIGN l_r_data_wa->* TO <s_data_wa>.
  CREATE DATA l_r_data_wa LIKE LINE OF i_th_ref_data.
  ASSIGN l_r_data_wa->* TO <s_data_wa2>.

* ==== clear receivers
  LOOP AT c_th_data INTO <s_data_wa>.
    LOOP AT p_ts_rule INTO l_s_rule WHERE rulepos = l_rulepos.
      IF abap_true = cl_rsplfu_work_area=>check_conditions( i_ts_sel = l_s_rule-
t_to_sel
                                                            i_s_data = <s_data_wa> ).
        l_s_rule-r_work_area->clear( CHANGING c_s_data = <s_data_wa> ).
        MODIFY TABLE c_th_data FROM <s_data_wa>.
        EXIT.
      ENDIF.
    ENDLOOP.
  ENDLOOP.

* for performance optimation the controller might call this function type only once
with all data
* therefore we fill the block line from the reference data.
  create data l_r_block_line like i_s_block_line.
  assign l_r_block_line->* to <s_block_line>.

* ==== for every ref data element, do for every receiver_part the copying
  LOOP AT i_th_ref_data INTO <s_data_wa2>.
    move-corresponding <s_data_wa2> to <s_block_Line>.
    LOOP AT p_ts_rule INTO l_s_rule WHERE rulepos = l_rulepos.
      <s_data_wa> = <s_data_wa2>.
      IF abap_true = cl_rsplfu_work_area=>check_conditions( i_ts_sel = l_s_rule-
t_from_sel
                                                            i_s_data = <s_data_wa> ).

        l_s_rule-r_work_area->complement_clear( CHANGING c_s_data = <s_data_wa> ).

        IF l_s_rule-to_sel_is_single = abap_false.
*         .... no restriction to single values => create allowed combinations
          p_r_object_creation->create( EXPORTING i_r_cr_controller =
cl_rsplfr_controller=>p_r_cr_controller
                                                 i_s_block_line = <s_block_line>
                                                 i_tsx_seldr = l_s_rule-
tsx_seldr_repr_of_to
                                                 i_r_msg = l_r_msg
                                       IMPORTING e_th_chas = <th_chas> ).
          IF abap_true = l_r_msg->contains_error( ).
            me->msg_with_context( EXPORTING i_r_msg_src = l_r_msg
                                            i_s_rule = l_s_rule
                                  CHANGING  c_r_msg_trgt = l_r_i_msg ).
            RETURN.
          ENDIF.

          LOOP AT <th_chas> ASSIGNING <s_chas>.
            MOVE-CORRESPONDING <s_chas> TO <s_data_wa>.
            COLLECT <s_data_wa> INTO c_th_data.
          ENDLOOP.
```

```
        ELSE.
*         .... restriction to single values => check the combination (much faster)
          MOVE-CORRESPONDING <s_block_line> TO <s_chas>.
          LOOP AT l_s_rule-t_to_sel INTO l_s_sel.
            ASSIGN COMPONENT l_s_sel-chanm OF STRUCTURE <s_chas> TO <cha>.
*           precondition:  l_s_sel-t_rng contains just one line with sign = 'I' and
opt = 'Eq'
            LOOP AT l_s_sel-t_rng INTO l_s_rng.
              <cha> = l_s_rng-low.
              EXIT.
            ENDLOOP.
          ENDLOOP.

          cl_rsplfr_controller=>p_r_cr_controller->check( EXPORTING i_s_chas =
<s_chas>
                                    IMPORTING e_t_mesg = l_t_contr_msg
                                              e_is_valid = l_is_valid ).
          IF l_is_valid = abap_false.
*           create context without real message
            me->msg_with_context( EXPORTING i_r_msg_src = l_r_msg
                                            i_s_rule = l_s_rule
                                  CHANGING  c_r_msg_trgt = l_r_i_msg ).
            LOOP AT l_t_contr_msg INTO l_s_contr_msg.
              MESSAGE ID l_s_contr_msg-msgid TYPE 'E' NUMBER l_s_contr_msg-msgno
                    WITH l_s_contr_msg-msgv1 l_s_contr_msg-msgv2
                         l_s_contr_msg-msgv3 l_s_contr_msg-msgv4
                    INTO l_dummy.
              l_r_i_msg->add_msg( ).
            ENDLOOP.
            RETURN. " all messages are error. => planning function stops.
          ENDIF.

*         combination valid -> store kyfs to receiver.
          MOVE-CORRESPONDING <s_chas> TO <s_data_wa>.
**           recalculate the keyfigure and redistribute
             assign component '0CUSTOMER' of structure <s_data_wa> to <customer>.
             assign component '0AMOUNT' of structure <s_Data_wa> to <amount>.
             <amount> = <amount> * '1.15'.

*            if <customer> = '0000000001'.
*               <amount> = <amount> * 105 / 100.
*            elseif <customer> = '0000000002'.
*               <amount> = <amount> * 105 / 100.
*            elseif <customer> = '0000000003'.
*               <amount> = <amount> * 115 / 100.
*            elseif <customer> = '0000000004'.
*               <amount> = <amount> * 110 / 100.
*            elseif <customer> = '0000000005'.
*               <amount> = <amount> * 115 / 100.
*            endif.
          COLLECT <s_data_wa> INTO c_th_data.
        ENDIF.

*        p_r_object_creation->create( EXPORTING i_index      = 0
*                                               i_s_new_line = <s_data_wa>
*                                               i_ts_sel     = l_s_rule-t_to_sel
```

```
*                                                 i_r_msg      = l_r_msg
*                                       CHANGING  c_th_data    = c_th_data ).
*          IF l_r_msg->is_filled( ) = rs_c_true.
*            l_t_msg = l_r_msg->get_t_msg( ).
*            LOOP AT l_t_msg INTO l_s_msg.
*              MESSAGE ID l_s_msg-msgid TYPE l_s_msg-msgty NUMBER l_s_msg-msgno
*                      WITH l_s_msg-msgv1 l_s_msg-msgv2 l_s_msg-msgv3 l_s_msg-msgv4
*                      INTO l_dummy.
*              i_r_msg->add_msg( ).
*            ENDLOOP.
*            RETURN.
*          ENDIF.
      ENDIF.
    ENDLOOP.
  ENDLOOP.
ENDMETHOD.
```

Instead of enlarging the sales expectation for each customer for 15%, you could also enlarge it dependent on the customer, e.g. 5% for A customers, 10% for B Customers and 15% for C customers. The code for this example is commented out in colour red.

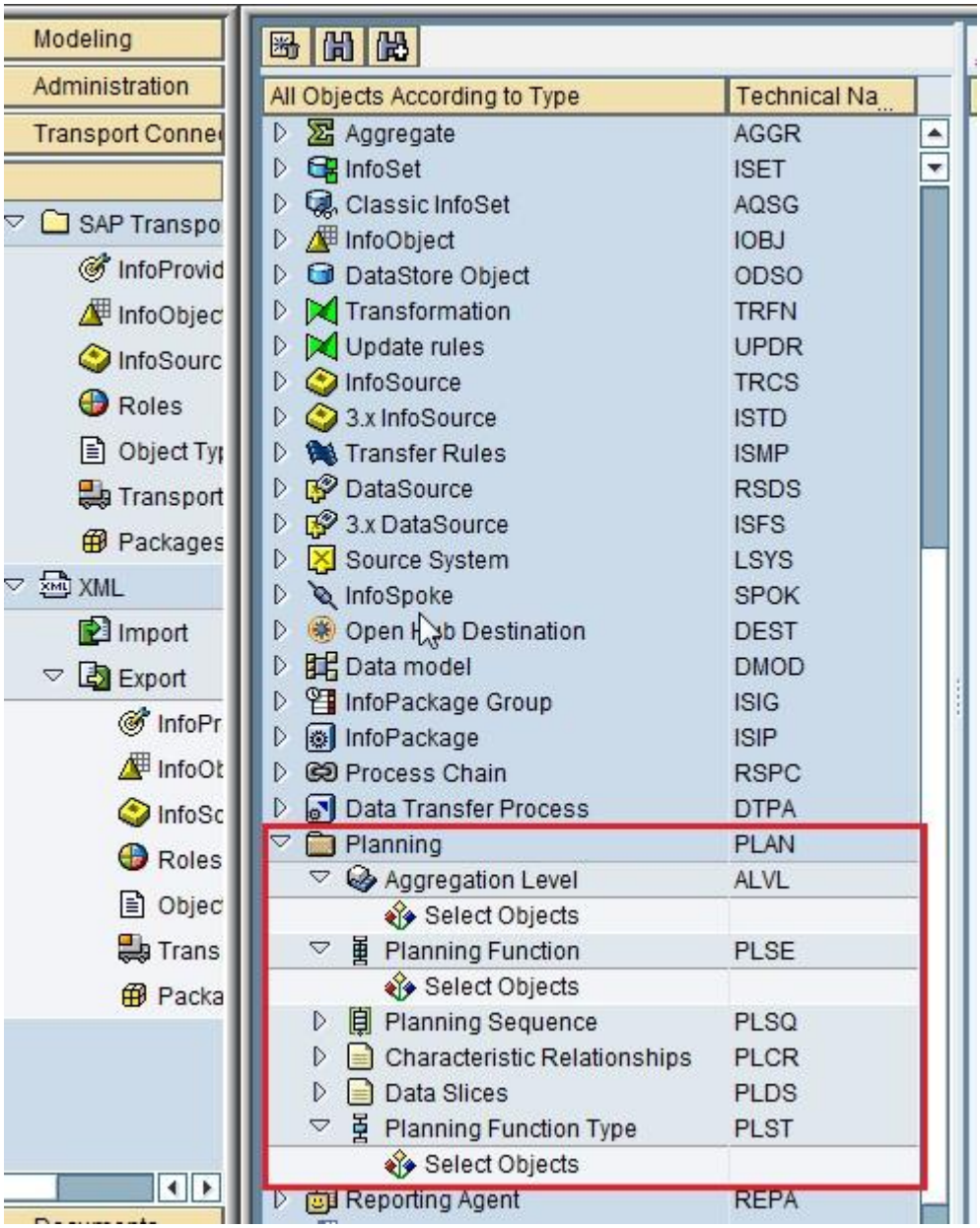
You can use the method IF_RSPLFA_SRVTYPE_IMP_EXEC_REF~INIT_EXECUTION to do initializations, checks or anything else before the planning function will be executed and data read from your source Infoprovider.

Debug and Test

Debugging is very easy. Just set a break-point on the method and in your planning sequence press button "Execute with Trace". The debugger will stop at your breakpoint and you can debug your code until it works fine. But you have to restart your planning application to see the changes of your planning function. Also if you change the settings of your custom planning function type in RSPLF1, you have to restart your browser session to work with your latest version.

Transport

You simply go to the transport connection in your Data Warehouse Workbench and search for Planning:



Picture 15: Planning objects in transport connection

Here you can select which objects you want to transport. You should collect your aggregation level, your planning function, your planning sequence and your planning function type. Finally you have to make sure that your class is written on the transport request also.

Related Content

Implementing Planning function types

How-to…Load a file into SAP Netweaver Integrated Planning (Part 2)

For more information, visit the EDW 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.