Conditional Execution of Planning Functions in BI-IP in a Planning Sequence using the After_Burn_Selection Exit

Applies to:

BW 7.X, BI-Integrated Planning. For more information, visit the Business Intelligence homepage.

Summary

Executing a planning sequence means that every single planning function contained in the planning sequence is executed one after the other. This paper shows how a conditional execution of planning functions inside a planning sequence can be achieved. It is possible to skip the execution of certain planning functions of a planning sequence based on a customer defined criterion. This behavior can be used to emulate IF THEN ELSE ENDIF or CASE ENDCASE conditions in planning sequences. We utilize the AFTER_BURN_SELECTION exit to achieve this behavior, which is described in detail in note 1101726 “Changing the active filter of planning functions”, although in a different context.

Author: Matthias Nutt

Company: SAP (Switzerland) AG

Created on: 12 May 2010

Author Bio

Matthias Nutt works for SAP (Switzerland) AG. He is a professional consultant since 2002 and works in the area of BW in close co-operation with the development and support team. His work spans a variety of roles, including project manager, technical project manager, consulting, development, primary and development support.

Table of Contents

Introduction

A planning sequence in BI-IP 7.X is a sequence of planning functions. Planning sequences can be defined in the planning modeler (Transaction RSPLAN). Every single step of a planning sequence is defined by a (internal) step id, an aggregation level, a filter and a planning function. During the execution of the planning sequence each single step is executed. It is not possible to skip a single step. One planning function is executed after the other. The approach discussed in this paper does not change this fundamental behavior but it demonstrates how “skipping” a step can be simulated by implementing the AFTER_BURN_SELECTION exit. Indeed the planning framework tries to start the planning function but by raising an artificial error message it is possible to stop the execution of the planning function before it is really started which leads to the illusion that the planning function was not executed at all.

The technique which allows that has been introduced by note 1101726 “Changing the active filter of planning functions”. In that note, the AFTER_BURN_SELECTION exit is used to change the active filter of a planning function to allow the planning function to work only on changed records. Here we show another example how this exit can be used to fulfill our requirement.

The basic idea is to implement the AFTER_BURN_SELECTION exit in such a way that

1. A customer defined criterion is evaluated to check if a planning function should be executed or not
2. If the criterion is true an artificial error message is issued in the AFTER_BURN_SELECTION which causes the system to stop the execution of the planning function. If the criterion is false, no artificial message is raised and the execution of the planning function continues normally.

In the remaining section an outline of this technique is given. It starts by activating the exit, followed by a simple sample implementation. Later on, a simple IF THEN ELSE ENDIF sample is explained.

IMPORTANT NOTE: Please be aware that the purpose of the AFTER_BURN_SELECTION is to allow changing the active selection. This has been done to allow planning functions to run only on changed records as described in the above mentioned note. This functionality is now available in the standard (see note 1356805 for further details). In fact, this how-to-paper misuses the AFTER_BURN_SELECTION exit in a way it was not designed for. Please note as well that the SAP support organization cannot handle customer messages about the implementation of “skipping” planning functions. Please post your questions in SDN in the business planning forum.

Activating the AFTER_BURN_SELECTION exit

The core idea is to use the AFTER_BURN_SELECTION exit. This exit is not an exit in terms of a standard SAP customer exit which can be implemented using transactions SMOD/CMOD. In fact, the exit is a link to a customer defined function module which has a certain interface (see below). To use the exit it must be activated in a different way. An entry in table RSADMIN needs to be maintained (see note 1101726 for further details). Adding the entry can be achieved by executing report SAP_RSADMIN_MAINTAIN. Please enter RSPLF_SELECTION_EXIT into parameter OBJECT. For parameter VALUE, please enter the name of a function module (in the customer name space) that should be used as exit function. In this paper, the exit function has the name Z_SELECTION_EXIT.

The function module must have the following interface:

```
FUNCTION z_selection_exit.
*"*"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
*"      EMPTY
*"      ERROR
```

A simple sample implementation for “skipping” the execution of a planning function

In the following we assume that the planning sequence with the name PLSEQ has been defined in the planning modeler. Planning sequence PLSEQ has 2 steps. The first step uses aggregation level AGGLVL01, filter FILT01 and planning function PLFUNC01. The second step uses aggregation level AGGLVL02, filter FILT02 and planning function PLFUNC02.

Furthermore, we need a customer based criterion if a planning function should be executed or not. In this example a very simple criterion is used: Planning function PLFUNC01 should only run on the first day of each month. Planning function PLFUNC02 should run all the time. Or in other words: planning function PLFUNC01 should be “skipped” if the planning sequence is not running on the first day of a month.

The ABAP code below implements this scheme: The case statement checks the name of the planning function. If the name of the planning function is equal to PLFUNC01, the system retrieves the current day from the system date. If the day is not equal to 01, which corresponds to the first day in the month, the system is raising an artificial error. This artificial error stops the remaining execution of the planning function. The planning framework stops the execution of the current planning function and continues with the next planning function in the planning sequence. If the name of the planning function is not equal to PLFUNC01, nothing happens in the exit. In this case, the planning function is executed as normal. Please note that although an error message is raised using the raising statement, the error message is a success message (type S). This is necessary because we do not want to stop the execution of the planning sequence which would be the case if a message of type E is raised.

```
FUNCTION z_selection_exit.
*"*"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
 *"      EMPTY
 *"      ERROR
           CASE I_SERVICE.
           WHEN ‘PLFUNC01’.
                IF sy-datum+6(2) <> ‘01’.
*                   raise artificial error to stop the execution of the planning function
                    MESSAGE S001(UPF) WITH ‘Skipping planning function: ‘ I_SERVICE RAISING error.
                ENDIF.
           WHEN OTHERS.
*               do nothing
           ENDCASE
ENDFUNCTION.
```

The simple example above shows how a conditional execution of a planning function in a planning sequence can be achieved. Planning function PLFUNC01 is only executed on the first day in a month. In fact, the BI-IP planning framework is trying to start them all but for planning function PLFUNC01 the AFTER_BURN_-SELECTION exit tests a condition. Raising the error message inhibits the remaining execution of the planning function by the planning framework. As the check in the AFTER_BURN_SELECTION exit is one of the first things done during the execution of a planning function, the effect is that the planning function does not seem to be executed. Especially the expensive part of the planning function is not executed (e.g. reading data, execution of the planning function, checking the data consistency and the data slices).

In the next chapter, the simple example is further extended to simulate an IF THEN ELSE ENDIF condition.

Modeling IF THEN ELSE ENDIF for “skipping” the execution of planning functions

Based on the simple example above modeling an IF THEN ELSE ENDIF statement is straight forward. All that must be done is to split up the IF THEN ELSE ENDIF statement in two simple IF ENDIF statements. The first IF statement uses the original condition and the second IF statement uses the negated condition.

Example:

IF A = 01 THEN B ELSE C ENDIF is split up into two simple conditions

IF A = 01 THEN B ENDIF followed by IF not (A = 01) THEN C ENDIF

This shows that it is straight forward to split up IF THEN ELSE ENDIF statements into two simple statements; a technique which is used in this chapter.

In the previous chapter, planning function PLFUNC01 should be executed on the first day of a month. This represents the THEN branch in our example. Here we enrich the first example by stating that planning function PLFUNC02 should be executed on all days except the first day of the month. This represents the ELSE branch of the IF statement. The condition used is to check if the current day is equal to 01.

Stating this in more technical terms leads to the following statement:

```
IF current day = 01 THEN
    Execute planning function PLFUNC01
ELSE
    Execute planning function PLFUNC02
ENDIF
```

Restating this using a “skipping” wording leads to:

```
IF current day <> 01 THEN
    Skip the execution of planning function PLFUNC01
ELSE
    Skip the execution of planning function PLFUNC02
ENDIF
```

Transforming this into two separate IF statements leads to:

```
IF current day <> 01 THEN
    Skip the execution of planning function PLFUNC01
ENDIF
IF current day = 01 THEN
    Skip the execution of planning function PLFUNC02
ENDIF
```

And this is exactly what is going to be implemented in this example.

It is clear that these two conditions need to be checked for each planning function. For planning function PLFUNC01, the system checks if the day is not equal to 01. If the condition is true, an artificial error is raised which stops the execution of the planning function. This is equivalent to run the planning function only on the first day of a month. For planning function PLFUNC02, the inverse condition is checked: if the day is equal to 01, the artificial error is raised and the execution of PLFUNC02 is stopped.

```
FUNCTION z_selection_exit.
*"*"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
 *"      EMPTY
 *"      ERROR
             CASE I_SERVICE.
             WHEN ‘PLFUNC01’.
                     IF sy-datum+6(2) <> ‘01’.
*                            raise artificial error to stop the execution of the planning function
                            MESSAGE S001(UPF) with ‘Skipping planning function: ‘ I_SERVICE RAISING error.
                     ENDIF.
             WHEN ‘PLFUNC02’.
                     IF sy-datum+6(2) = ‘01’.
*                            raise artificial error to stop the execution of the planning function
                            MESSAGE S001(UPF) with ‘Skipping planning function:  ‘ I_SERVICE RAISING error.
                     ENDIF.
             WHEN OTHERS.
*                    do nothing
             ENDCASE
ENDFUNCTION.
```

By introducing different conditions for each planning function, we are able to model IF THEN ELSE ENDIF statements. If the model is extended further, CASE ENDCASE like constructs can be created. Nevertheless the system is only checking one specific condition for each single planning function. The IF THEN ELSE ENDIF or CASE ENDCASE semantic is derived from choosing the conditions carefully and implementing them the right way. Please note again that although an error message is raised, the system uses a success message.

Miscellaneous

This paper describes the “skipping” technique in the context of a planning sequence. As we can see from the ABAP code above, the information which planning sequence is currently running is not evaluated. This means that “skipping” as described above is independent of the planning sequence and relies only on the name of the planning function. If you have for example two planning sequences PLSEQ1 and PLSEQ2 which both contain planning function PLFUNC01, it will be skipped in both planning sequences. If you do not want skipping PLFUNC01 in planning sequence PLSEQ2, you first need to copy PLFUNC01 to PLFUNC03. In the next step you can replace PLFUNC01 with PLFUNC03 in PLSEQ2.

Related Content

Note 1101726 “Changing the active filter of planning functions”

Note 1356805 “Planning functions: Deltas”

For more information, visit the Business Intelligence homepage.

Copyright