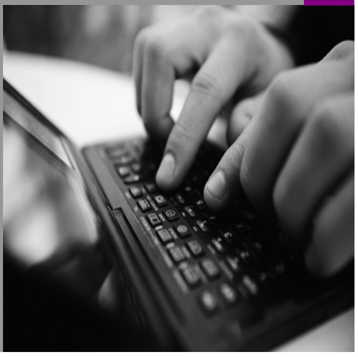
How-to Guide
SAP NetWeaver '04



How To... Run Planning Sequences on Save and other events (WEB)

Version 1.00 – September 2004

Applicable Releases:
SAP NetWeaver '04
(BW 3.5 SPS 5)

THE BEST-RUN BUSINESSES RUN SAP

1 Scenario

When creating a user interface for planning there is often a need to execute planning sequences automatically on certain events. For example a planning sequence should be executed when pressing the save button in order perform a currency conversion.

When transferring SAPGui based planning folders to a Web Interface Application you find that the execution of functions *before layout display* and *on save* are not supported. By following through this document these features are added and planning folders can be transferred more easily.

Please keep in mind that such kinds of automatic functions will typically increase the response time. In particular the layout related execution occurs quite frequently (on each submit of data or server roundtrip).

2 Introduction

This document describes how to enhance the Web Interfaces to automatically trigger a global planning sequence …

- … by pressing the save button.
- … before a layout is executed.

To enable this feature a mapping of planning sequences to layouts and save buttons is required. One option given is to reuse the standard function button for carrying the sequence names. Following this approach you get things like F4-help etc. for free and you stay within the standard customizing.

The solution uses the Web Interface Extension (the so called 'exit class') to enhance the input and output processing of the Web Interface.

Once a planning sequence has been executed you will see the status in the message tray. In case of errors the complete list of single functions is shown and the following step is not executed.

Assignments using function buttons

The assignment of a planning sequence is done by adding a function button to the Web Interface. The attribute contains the name of the global planning sequence whereas the component name follows the rule indicated below:

Component Name	Suffix	Name of function button
MyLayout	_before	MyLayout_before
SaveButton	_save	SaveButton_save

3 The Step By Step Solution

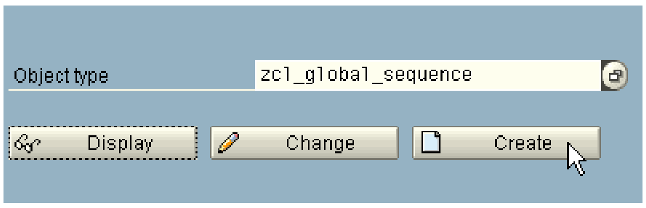
The step-by-step solution comes with two sections: first we have to enhance the standard input and output processing of the Web Interface. This is done on the backend and described in section 3.1. Additionally we have to adjust the customizing of the Web Interfaces in order to enable the new functionality for a specific Web Interface. This is described in section 3.2 .

3.1 Enhance input and output processing

Some technical background information:

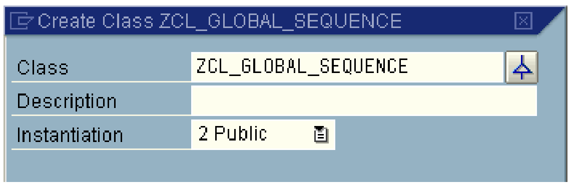
- When creating a Web Interface using the Web Interface Builder (transaction bps_wb) the system generates a BSP page based on the customizing given. The BSP event handlers are implemented in the class cl_upwb_bsp_appl.
- By creating subclasses and using the redefinition capability of ABAP OO, the standard behavior can be changed and new functionalities added.
- During the input processing the system creates a list of tasks to be done (which components to use and which actions to perform) and dispatches the work to the different component. In order to add the validation feature, we redefine the dispatcher method.

1. Call transaction SE24, enter the new class name and press Create.



Object type zcl_global_sequence

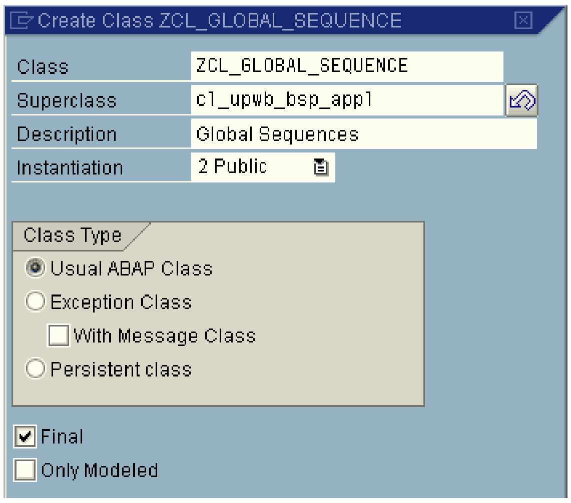
2. Press the Button on the pop-up to get the additional input field for the super class.



Create Class ZCL_GLOBAL_SEQUENCE

Class	ZCL_GLOBAL_SEQUENCE
Description	
Instantiation	2 Public

3. Complete the pop-up as shown in the screenshot and press Save.

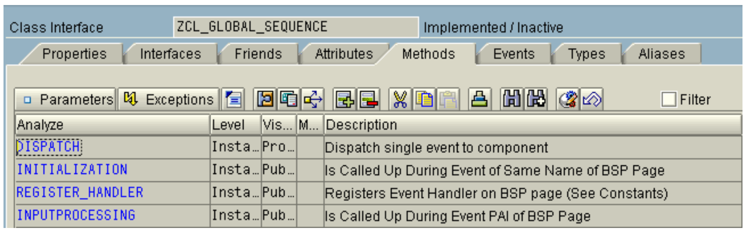

Create Class ZCL_GLOBAL_SEQUENCE

Class	ZCL_GLOBAL_SEQUENCE
Superclass	cl_upwb_bsp_appl
Description	Global Sequences
Instantiation	2 Public

Class Type
- [✓ Usual ABAP Class]
- Exception Class
  - With Message Class
- Persistent class

- [✓ Final]
- Only Modeled

4. Now you should see this screen. Select the method Dispatch and press the button (when using SAP Gui 6.40) to redefine the method.


Class Interface ZCL_GLOBAL_SEQUENCE Implemented / Inactive

Analyze	Level	Vis...	M...	Description
DISPATCH	Insta...	Pro...		Dispatch single event to component
INITIALIZATION	Insta...	Pub...		Is Called Up During Event of Same Name of BSP Page
REGISTER_HANDLER	Insta...	Pub...		Registers Event Handler on BSP page (See Constants)
INPUTPROCESSING	Insta...	Pub...		Is Called Up During Event PAI of BSP Page

5. Now an editor opens showing the following default ABAP code. Delete the coding and insert the ABAP coding given in section 4.1 Source Code: Method dispatch of this How To.

```
method DISPATCH.
*CALL METHOD SUPER->DISPATCH
*  EXPORTING
*     IS_HANDLER =
*  RECEIVING
*     R_SUBRC    =
*     .
endmethod.
```

6. Note: When executing a planning sequence the status is shown in error and success case. By uncommenting … the status is shown in error case only.

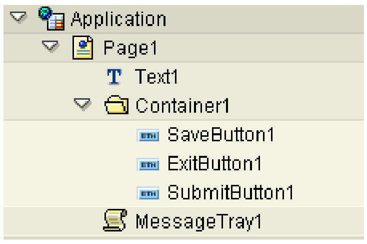
```
* If you like to see message in error cases only, uncomment the
if ... endif
*   IF l_subrc <> 0.
    lr_component->mo_msg_log->add( EXPORTING it_bapiret =
lt_msg_log ).
*   ENDIF.
```

7. After activating the class (press button) we have now created a subclass of the standard implementation and we have done the back-end part of the enhancement. Congratulations!

3.2 Assignments using function buttons

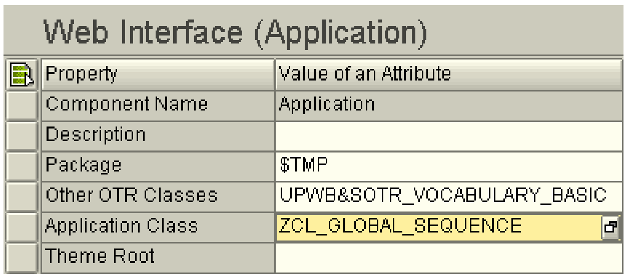
1. Start the Web Interface Builder using transaction BPS_WB.

2. Open the Web Interface to be enhanced. The application used for this example has the name *Application*.



3. Double click on the root node to open the property section for the *Application*.

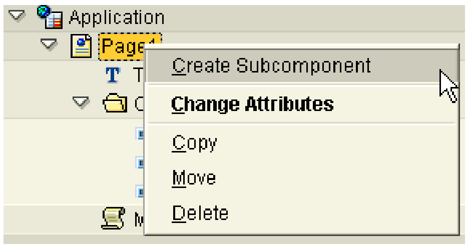
   Replace the application class cl_upwb_bsp_appl by zcl_global_sequence, the name of the subclass created before.



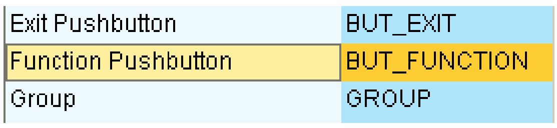
Web Interface (Application)

Property	Value of an Attribute
Component Name	Application
Description	
Package	$TMP
Other OTR Classes	UPWB&SOTR_VOCABULARY_BASIC
Application Class	ZCL_GLOBAL_SEQUENCE
Theme Root	

4. Right click on the page node to add a new subcomponent.

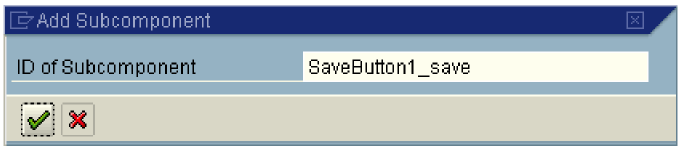


5. Chose the text component.



Exit Pushbutton	BUT_EXIT
Function Pushbutton	BUT_FUNCTION
Group	GROUP

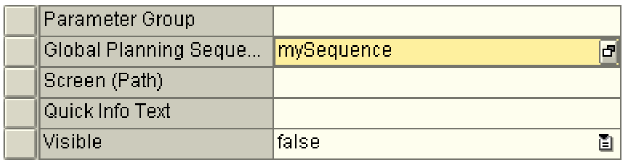
6. Concatenate the component name of the save button with the suffix _save (or the name of the layout component with the suffix _before).



Add Subcomponent

ID of Subcomponent	SaveButton1_save

7. Set the property Visible to false and choose a global planning sequence.



Parameter Group	
Global Planning Seque...	mySequence
Screen (Path)	
Quick Info Text	
Visible	false

8. (Re-) Generate the planning application by pressing button.

4 Appendix

4.1 Source Code: Method dispatch

```
METHOD dispatch.

  CONSTANTS:
    c_layout_class   TYPE string VALUE 'LAYOUT',
    c_layout_method  TYPE string VALUE 'API_GET_DATA',
    c_save_class     TYPE string VALUE 'BUT_SAVE',
    c_save_method    TYPE string VALUE 'SUBMIT'.

  DATA:
    lr_component TYPE REF TO if_upwb_c_component,
    lr_convert   TYPE REF TO if_upwb_c_component,
    lr_lay       TYPE REF TO if_upwb_c_layout2,
    lr_func      TYPE REF TO if_upwb_c_but_function,

    l_class     TYPE string,
    l_id        TYPE string,
    l_id_seq    TYPE string,

    l_bundle  TYPE upf_y_bundle,
    l_subrc   LIKE sy-subrc,
    lt_msg_log TYPE TABLE OF bapiret2.

  lr_component ?= is_handler-component.
  l_class       = lr_component->get_class( ).
  l_id          = lr_component->get_id( ).

* Do we have the right component and method for starting sequence?
  IF l_class = c_layout_class AND is_handler-method = c_layout_method.
    lr_lay ?= lr_component.
    IF lr_lay->is_ready_for( cl_upwb_bsp_appl=>ce_get_data ) = cb_true.
      CONCATENATE l_id '_before' INTO l_id_seq.
    ENDIF.
  ENDIF.

* Do we have the right component and method for starting sequence?
  IF l_class = c_save_class AND is_handler-method = c_save_method.
    CONCATENATE l_id '_save' INTO l_id_seq.
  ENDIF.

  IF NOT l_id_seq IS INITIAL.

* Get name of sequence from function component
    lr_convert = cl_upwb=>get_component( l_id_seq ).
    IF lr_convert IS BOUND.
      lr_func ?= lr_convert.
      l_bundle = lr_func->get_gl_sequence( ).
    ENDIF.

* If no sequence is found, l_bundle is initial
    IF NOT l_bundle IS INITIAL.
      CALL FUNCTION 'API_SEMBPS_GLSEQUENCE_EXECUTE'
        EXPORTING
          i_sequence = l_bundle
        IMPORTING
          e_subrc    = l_subrc
        TABLES
          etk_return = lt_msg_log.

* If you like to see message in error cases only, uncomment the if ... endif
*   IF l_subrc <> 0.
      lr_component->mo_msg_log->add( EXPORTING it_bapiret = lt_msg_log ).
*   ENDIF.

*     Valid from BW3.5 SP5 onwards
      cl_upwb=>raise_buffer_changed( 'buffer' ).
*     Valid until BW3.5 SP4 and for SEM 3.1, 3.2, 3.5.
*     cl_upwb=>set_parameter( param = 'buffer_changed' value = 'X' ).
      IF l_subrc <> 0. r_subrc = 10. ENDIF.
    ENDIF.
  ENDIF.

  IF r_subrc IS INITIAL.
    CALL METHOD super->dispatch
      EXPORTING
        is_handler = is_handler
      RECEIVING
        r_subrc    = r_subrc.
  ENDIF.

ENDMETHOD.
```