How to create a custom step for GPA
With Solution Manager 7.2 SP3 and higher
Introduction: With the Guided Procedure Authoring in SAP Solution Manager 7.2 you have the possibility to create your own guided procedures to support day-to-day business and IT operations activities, but as well any operation as regular activities in your organization. A guided procedure is a sequence of activities to be carried out in a pre-defined order. Some of these activities can require input values and user interaction, which is not possible via the standard automatic activities. These activities can be implemented using a step of type custom, which can contain its own UI and logic behind it. This guide describes how to create your own custom steps for guided procedures.

The way how to create a custom step didn’t really changed since SAP Solution Manager 7.1, which allow a full compatibility between both releases. All objects created in 7.1 release will still run in 7.2 release.

The code examples used in this guide are part of the package AGS_SISE_GPA_DEMO.

TABLE OF CONTENTS

OVERVIEW OVER CUSTOM STEPS................................................................................................................................................... 2
CREATE CLASS TO IMPLEMENT FUNCTIONALITY IN THE STEP................................................................................................ 3
CREATE METHODS........................................................................................................................................................................... 3
CREATE WEB DYNPRO COMPONENT FOR THE UI OF THE STEP............................................................................................. 6
CREATE CONTEXT, ATTRIBUTES AND METHOD IN COMPONENTCONTROLLER................................................................. 7
CREATE VIEW FOR USER INTERFACE ........................................................................................................................................ 8
GET SCOPE INFORMATION FROM WINDOW.......................................................................................................................... 12
IMPLEMENT METHOD IN COMPONENTCONTROLLER........................................................................................................... 13
IMPLEMENT READ ONLY BEHAVIOR......................................................................................................................................... 18
INTEGRATE CUSTOM STEP IN GUIDED PROCEDURE.................................................................................................................. 22

OVERVIEW OVER CUSTOM STEPS

Within a custom guided procedure, you can create steps that contain manual or automatic activities out-of-the box. But some tasks to be carried out in the step of a guided procedure require specific input values and a more complex logic in the backend. Usually these steps also need an own user interface to collect the input values from the user. These tasks can be modeled using custom steps. A custom step in a guided procedure is a step that contains an own user interface and own logic which is implemented in a backend class.

This guide describes how this custom step are build and what has to be done to make sure the custom UIs and the backend class can integrate in the guided procedure framework smoothly.

To create a custom step, the following steps have to be carried out:

- Create ABAP class that inherits from CL_SISE_ACTIVITY_CUSTOM
• Implement the backend functionality in this class
• Create a Web Dynpro component that implements the interface WD_IF_SISE_STEP

In this guide we will implement a very simple example. We will create an UI where the user can enter the name of an RFC destination. By clicking a button, the backend implementation is called and the RFC destination is checked in the managed system. We then display the result in the message area of the guided procedure and update the log of the step.

The coding to create the custom step described below is available for your reference in the package AGS_SISE_GPA_DEMO in the class CL_GPA_CUSTOM_DEMO and the Web Dynpro component GPA_CUSTOM_UI_DEMO. All coding created by you should be created in the customer namespace nevertheless.

CREATE CLASS TO IMPLEMENT FUNCTIONALITY IN THE STEP

The first step when creating a custom step with an own UI and own functionality, is to create an ABAP class that implements the actual functionality. This class can have different methods that can be called by the UI, e.g. if a button is clicked.

Create a new ABAP class in the customer namespace.

To be able to integrate your class in the guided procedure framework you have to make sure it inherits from the superclass “CL_SISE_ACTIVITY_CUSTOM”. This makes sure the class has the necessary methods and attributes to use the scope of the guided procedure step and to write to the step log.

Create Methods

The functionality of your custom step is provided by different methods. You can have several methods in your class. In our example we need only one method. Make sure that you create all methods that are called from the Web Dynpro component later on as “public”. Otherwise you will not be able to call them. Furthermore, the methods have to be instance methods. The reason is that all inherited methods that are used to interact with the framework are instance methods and hence cannot be called out of a static method.
The next step is to create the parameters for your method. You need at least parameters to hand over the scope of the guided procedure and to return the log and the status of the execution.

To create parameters, select your method then the tab “Methods” and click “Parameter”.

The parameters “LS_SCOPE_OBJ”, “ET_LOG” and “EV_STATUS” are the minimum required to handle scope and log.

In this example, we need two additional parameters.

Now we can implement the method.
At first, we use the scope object that was handed over from the caller to get the SID of the system we are running the guided procedure for. **method** CHECK_RFC_DESTINATION.

```
DATA lv_dest TYPE rfcdest-rfcdest.
DATA lv_lid TYPE smsy_techsys_name.
DATA lv_string TYPE string.

DATA: lt_rfc TYPE STANDARD TABLE OF
      scdtsysrfc, I ls_rfc_read LIKE LINE OF I lt_rfc,
      lv_rfc_dest TYPE rfcdest.

DATA: lv_check_ok TYPE boolean,
      lv_message TYPE string.

DATA ls_system TYPE ags_sise_s_sysparam.
```

*read system context of the guided procedure from the scope object*  
ls_system = cl_sise_scope_access=>one_obj_to_system( ls_scope_obj ).  
lv_lid = ls_system-lid.

*set to be checked RFC as handed over from the caller*  
lv_dest = lv_rfc_dest.

*get the READ rfc destination of the system the guided procedure runs on*  
SELECT SINGLE * FROM scdtsysrfc

  INTO ls_rfc_read
  WHERE sysname = lv_lid
  AND purpose = 'CUST_SCOUT'.

"#EC CI_GENBUFF"

*if READ RFC destination found*  
IF ls_rfc_read IS NOT INITIAL.

*Check given RFC destination in the managed system*  
CALL FUNCTION '/SDF/CHECK_RFC' DESTINATION ls_rfc_read-rfcdest

  EXPORTING
      lv_destination = lv_dest
  IMPORTING
      ev_check_ok = lv_check_ok
      ev_message = lv_message.

*store the returned status in the exported parameter (will be needed later)*  
ev_rfc_check_ok = lv_check_ok.

*fill the step log depending on the outcome of the RFC test*  
IF lv_check_ok = abap_true.
  MESSAGE s001(gpa_demo) WITH lv_dest lv_lid INTO lv_string.
ELSE.
  MESSAGE e002(gpa_demo) WITH lv_dest lv_lid INTO lv_string.
  ENDIF.
ELSE.
  MESSAGE e003(gpa_demo) WITH lv_lid INTO lv_string.
  ENDIF.

APPEND me->add_sy_to_log( ) TO et_log.
*calculate execution status*

\[
e_v\_status = \text{get\_status\_from\_log}(\ et\_log) .
\]

endmethod

CREATE WEB DYNPRO COMPONENT FOR THE UI OF THE STEP

Now that we have the functionality implemented, we have to implement the UI for the step. At first create a new Web Dynpro component.

The naming convention for the window is W\_MAIN and the name for the view should be V\_MAIN.

To make sure that your UI is compatible with the guided procedure framework and you can add it to a custom step, implement the interface WD\_IF\_SISE\_STEP.

Enter WD\_IF\_SISE\_STEP on the tab “Implemented Interfaces” and press the “Enter” key.

At first the implementation state will show red. Click “Reimplement” to implement the interface.
If you receive a warning, please click the button “Yes”

Implementation Was Only Partially Successful. Implement Remaining Parts Despite Conflicts?

Yes  No  Info  Cancel

Now the implementation state should show green.

Create Context, Attributes and Method in COMPONENTCONTROLLER

To enable the interaction of our class with the Web Dynpro we have to add context to the component controller of the Web Dynpro.

At first, we add context. Context is needed for all information that is passed to or received from the UI components of the Web Dynpro to the backend implementation. In our case the user will have an input field in the UI to enter the name of the RFC destination he wants to check. The context is added on the tab “Context” of the component controller. We create a new context node with one attribute of the type String for the RFC destination name.
Next, we need to create an attribute for the activity of the guided procedure. Every custom step has a generic activity that carries the scope of the guided procedure which is needed to jump to the correct managed system and also an activity ID which is needed to be able to assign the log for the step. One and the same guided procedure can run for different managed systems. For every run the guided procedure is instantiated, and the activity gets an ID. This ID makes sure that for every instance (every managed system) the correct log records are displayed.

Create a new attribute as reference to ZCL_GPA_CUSTOM_DEMO.

At last we have to create the method that will later on call our backend class. This is done on the tab methods. For now, it is enough to just create the method. We will implement it later on.

Now you should activate the complete Web Dynpro component to make sure all attributes and methods are available in the subsequent steps.

Create View for User Interface

The actual UI components have to be added to the view of the Web Dynpro component.
Double click the view to open it. At first, we will add the context from the component controller to the context of the view. Then we add UI components and bind them to the context. Drag and drop the context from the component controller to the view context.

Now we can create the UI.

When creating the UI, make sure you don’t forget to bind the context to the UI elements. In our case we have to bind the input field IF_RFCNAME to the context RFC_NAME. This is done in the properties of the input field element.

Click the binding button for the property “value”
Select the context element and confirm the dialog.

To call the method in the backend class when you click on the button, you have to create an action for it. To do this click on the “New” button in the button element properties in the property “onAction”.

Create a new action.

To implement the action, double-click on it.
All that has to be done here is to call the method we created in the component controller before.

That's all for the view for now.

**Get Scope Information from Window**

In the next step we will use the main window of the Web Dynpro component to get the generic activity of the step which contains the scope the guided procedure runs in.

This is implemented in the DEFAULT inbound plug or directly in the method HANDLEDEFAULT. Double-click on the method to move to the coding.
Because the Web Dynpro component implements the interface WD_IF_SISE_STEP, the method imports several parameters. The most interesting parameter for us is the parameter R_STEP. This contains the activities information we are interested in.

From the R_STEP parameter we receive all activities. For most of all the custom steps this will be only one. Then we read this one activity and assign a reference to it to the AV_ACTIVITY attribute we created on the component controller before.

```java
method HANDLEDEFAULT .

  data: lt_activities type cl_sise_step_abs => tt_activity,
        ls_activity like line of lt_activities,
        lv_total_act type i.

try.
  " Only Get Custom Activities " lt_activities = r_step->
    get_activities_by_type( iv_act_type = 'C' ).

  lv_total_act = lines( lt_activities ). "number of activites in the step"
  if lv_total_act = 1. " this will be true for 99% of all custom steps"
      read table lt_activities into ls_activity index 1.
      " if you have several activities in this step =>
      " you need an activity attribute in the for every activity
      " but usually you would only have one activity per custom step
      endif.

  catch cx_sise_model. " Simple Setup: Data Layer Exception"
      " Check exception and perform action or logging of error"
      endtry. endmethod.
```

Activate your Web Dynpro component to be sure everything is available in the next steps.

**Implement Method in COMPONENTCONTROLLER**

Now we need to implement the method we created in the component controller before. In this method we have to extract the managed system from the activity, call the method in the backend class, write the result in the message area of our guided procedure and update the log.

At first, we read the input provided in the input field in our UI. The easiest way to do this is using the Web Dynpro code wizard, to be opened via the button.

Select the context attribute and the operation "Read"
The code is generated automatically.

```
" data for input read - generated
DATA lo_nd_rfc_context TYPE REF TO if_wd_context_node.

DATA lo_el_rfc_context TYPE REF TO if_wd_context_element.
DATA ls_rfc_context TYPE wd_this->element_rfc_context.
DATA lv_rfc_name TYPE wd_this->element_rfc_context->rfc_name.

/*Read input from input field on GUI
 * navigate from <CONTEXT> to <RFC_CONTEXT> via lead selection
 lo_nd_rfc_context = wd_context->get_child_node(name = wd_this->wctx_rfc_context).

/* handle non existent child
CHECK lo_nd_rfc_context IS NOT INITIAL.

/* get element via lead selection
 lo_el_rfc_context = lo_nd_rfc_context->get_element().
/* handle not set lead selection
 CHECK lo_el_rfc_context IS NOT INITIAL.

/* get single attribute
 lo_el_rfc_context->get_attribute( 
   EXPORTING
   name = "RFC_NAME"
   IMPORTING
   value = lv_rfc_name ).
```

Now we need to create an instance of our backend class, since we want to use the instance method to check the RFC destination. At first, we create a data object for it.

```
" Instance of backend class
DATA lr_gpa_back_imp TYPE REF TO cl_gpa_custom_demo.
```

And then we create the instance. The parameter lv_id is mandatory to create the instance. You can give it any value you want.
The next step is to read the system ID from the scope the guided procedure is running in and then call the back-end implementation to check the given RFC destination.

At first, we create the data objects for input and output of the backend method.

```abap
" define data fields for the scope
DATA lv_lid TYPE smsy_techsys_name.
DATA ls_obj_guid TYPE ags_sise_t_scope_object,
   ls_system TYPE ags_sise_s_system,
   ls_activity TYPE ags_sise_s_activity.

" define data fields for return values of the backend method
DATA ls_log TYPE ags_sise_tt_log,
   ls_status TYPE ags_sise_activity_status,
   lv_check_ok TYPE boolean.
```

And then we read the scope from the activity attribute. A guided procedure can run for more than one managed objects at a time. So, we have to loop over the scope table and for each managed object get the SID.

```abap
* Read scope and loop over all systems in scope
  ls_obj_guid = wd_this->av_activity->if_sise_activity_scope->get_configuration_objects( abap_true ).
  LOOP AT ls_obj_guid INTO ls_obj_guid.
    ls_system = ci_sise_scope_access->one_obj_to_system( ls_obj_guid ).
    lv_lid = ls_system->lid.
    IF ls_system->sys_type EQ 'ABAP'.
      EXIT.
    ENDIF.

Then we call the backend method.

```abap
" call backend method to check RFC destination
lr_gpa_back_imp->check_rfc_destination:
    EXPORTING
    ls_scope_obj = ls_obj_guid " Scope object structure
    iv_rfc_dest = lv_rfc_name
    IMPORTING
    tt_log = lt_log " Simple Setup: Log Structure
    ev_status = lv_status " Simple Setup: Activity Status
    ev_rfc_check_ok = lv_check_ok " Boolean Variable (X-True, ~False, Space=Unknown)
```

Now we want to display the result of our test in the message area of the guided procedure. For this we need a message manager. To create this, we will use again the Web Dynpro code wizard.
We have to add two more parameters to hand over the name of the RFC destination and the SID. Then we create the message in the message area depending on the status of the return value `lv_check_ok` than indicates whether the RFC check was successful or not. To set the status of the guided procedure step to the correct value, we set the parameter `lv_status`. 
The last step is now to write the result in the log area of the guided procedure step as well. The log is written using a method in the activity object and the parameters returned from the backend method. This is also the end of our loop!

In a very last step we refresh the log area to make sure the log gets displayed.

```abap
" write message to message area of guided procedure
DATA lo_api_controller TYPE REF TO if_wd_controller.
DATA lo_message_manager TYPE REF TO if_wd_message_manager.
DATA lv_p1 TYPE syst-msgv1.
DATA lv_p2 TYPE syst-msgv2.

lv_p1 = lv_rto_name.
lv_p2 = lv_sid.

lo_api_controller = wd_this->wd_get_api( ).
lo_message_manager = lo_api_controller->get_message_manager( ).

IF lv_check ok = abap_true.
" report result in message area
lo_message_manager->report_t100_message( EXPORTING
  msgid = 'GPA_DEMO' " Message Identification
  msgno = '001' " Message Number
  msgty = 'S' " Message Type
  p1 = lv_p1 " 1st parameter
  p2 = lv_p2 " 2nd parameter
).
" set step status
lv_status = 'S'.
ELSE.
  lo_message_manager->report_t100_message( EXPORTING
    msgid = 'GPA_DEMO' " Message Identification
    msgno = '002' " Message Number
    msgty = 'E' " Message Type
    p1 = lv_p1 " 1st parameter
    p2 = lv_p2 " 2nd parameter
  ).
" set step status
lv_status = 'E'.
ENDIF.
```

The last step is now to write the result in the log area of the guided procedure step as well. The log is written using a method in the activity object and the parameters returned from the backend method. This is also the end of our loop!
it_log = lt_log
iv_status = lv_status
iv_scope_obj = ls_obj_guid
)
   CATCH cx_sise_model.
   CATCH cx_sise_log.
ENDTRY.

ENDLOOP.

"refresh log area  wd_this
>fire_refresh_log_evt( EXPORTING
   r_activity = wd_this->av_activity
).

IMPLEMENT READ ONLY BEHAVIOR
As you know you can switch in a guided procedure between “Edit” and “Read Only” mode. You can use this
also for your custom step. If you don’t implement this functionality your UI elements will be in read mode all
the time, independent from the guided procedure status.

The framework makes the implementation of this behavior very easy.

As you have seen the component controller contains context called MODE_CONTROL. This is the context
used to steer the read only behavior.

At first you have to drag this context to the view context as done before with the RFC context.

Now you have to bind the READ_ONLY context to the respective fields, in our case the button and the input
field. For the input field you do this via the property “readOnly”. Click the binding button and select the
context READ_ONLY.
Select the context READ_ONLY and confirm the dialog box.

That's it for the input field.

For the button you have to change the property “enabled”.
For the button you have to invert the attribute. The reason is easy to calculate. You want the button to be "enabled" if the Guided procedure is not "READ_ONLY".

That's it for the button.

Now we have to implement the method that is triggered when the "Read Only" event is fired. It is the method SET_CHANGE_MODE in the component controller. Double-click the method to implement it.
The method has one input parameter IV_READ_ONLY.

We need to set this input parameter in the context. To do this we again use the Web Dynpro code wizard that implements it for us. Select the READ_ONLY attribute from the context and choose “Set”.
Now we have to complete the generated code by adding error handling and assigning the IV_READ_ONLY input value to the context attribute.

```java
METHOD set_change_mode .

DATA lo_nd_mode_control TYPE REF TO if_wd_context_node.
DATA lo_el_mode_control TYPE REF TO if_wd_context_element.
DATA ls_mode_control TYPE wd_this->element_mode_control.
DATA lv_read_only TYPE wd_this->element_mode_control read_only.

" navigate from <CONTEXT> to <MODE_CONTROL> via lead selection
lo_nd_mode_control = wd_context->get_child_node( name = wd_this->wdctx_mode_control ).

" handle non existant child
CHECK lo_nd_mode_control IS NOT INITIAL.

" get element via lead selection
lo_el_mode_control = lo_nd_mode_control->get_element( ).

" handle not set lead selection
CHECK lo_el_mode_control IS NOT INITIAL.

* @TODO fill attribute
  * lv_read_only = 1.

* set single attribute
lo_el_mode_control->set_attribute( 
  name = 'READ ONLY' 
  value = lv_read_only ).

ENDMETHOD .
```

That's it for the coding. Activate the whole Web Dynpro component to use it in the Guided Procedure.

**INTEGRATE CUSTOM STEP IN GUIDED PROCEDURE**

The last step is to integrate your new custom step in a custom guided procedure. Call the transaction GPA_ADMIN to open the guided procedures browser.

To add the manual activity, go into your standard step and select the tab 'Custom UI'.

Enter the name of the Web Dynpro component and the view you want to display.
Now you can test your guided procedure by clicking on the “Preview” button.

Enter an RFC destination you want to test and click the “Check” button. You see that the message is written in the message area and the log is updated. The step rating is calculated and displayed in the roadmap.

Also test that the read-only functionality works by clicking on “Read Only”.
After switching to read-only mode the input field and the button are disabled.