SAP NetWeaver
How-To Guide

How To... Integrate Custom Formulas into the Formula Builder

Applicable Releases:

SAP NetWeaver 2004

SAP NetWeaver 7.0

IT Practice:

Business Information Management

IT Scenario:

Enterprise Data Warehousing

Version 1.0

April 2008

Document History

Document Version	Description
1.00	First official release of this guide

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

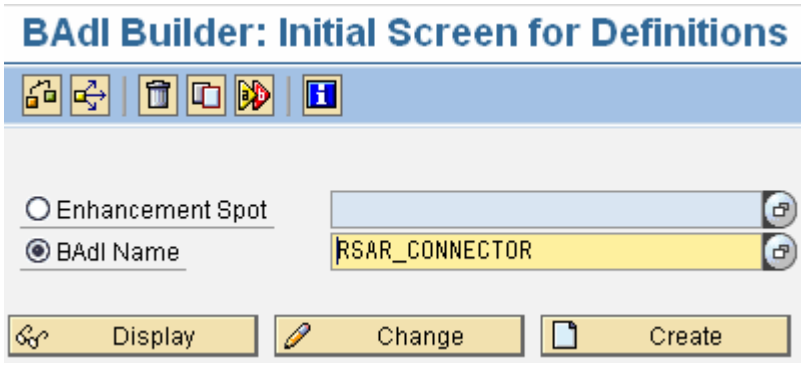
1. Business Scenario

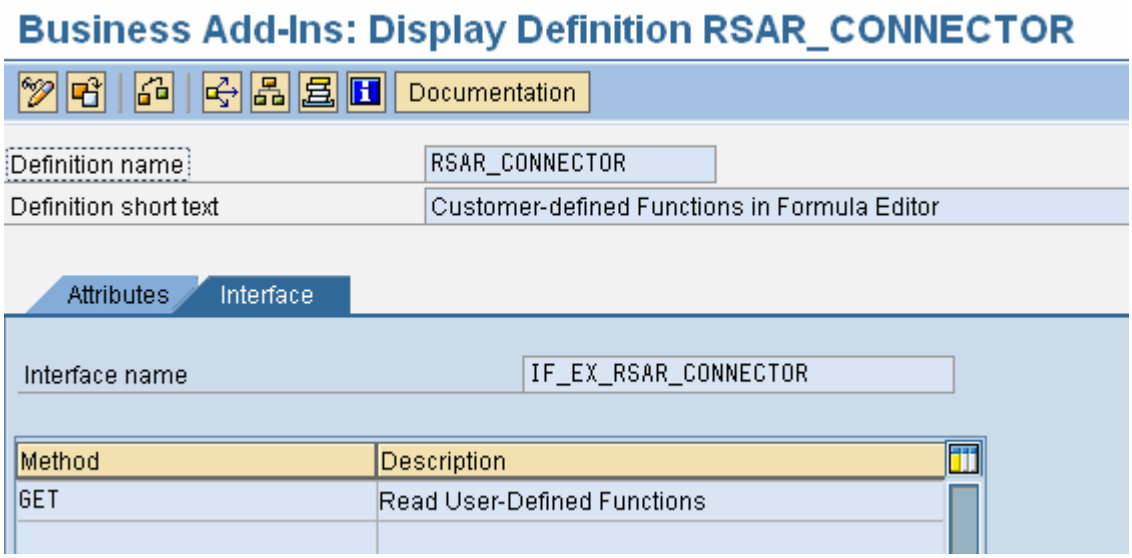
The business scenario for this document is that a customer has a need to transform data coming into the data warehouse, and consistently uses the same data transformation. In this example, it is to transform a timestamp into a date field. Instead of continuously writing the same coding in ABAP routines, the customer has the need to create a reusable formula.

2. Background Information

Custom formulas are introduced into the transformation library of the Formula Builder by means of BAdi (Business Add-In) RSAR_CONNECTOR. The step by step solution that follows will demonstrate how to achieve this.

To view the definition of the BAdi go to transaction SE19 and enter the BAdi name RSAR_CONNECTOR:





Note

The ‘GET’ method is the key to the solution. It is called when you list all the formulas that are available. We will enter all our custom methods within the ‘GET’ method making them available at runtime to the BI developer.

3. Prerequisites

There are no prerequisites for this How-To-Guide, however the following reading material will help if you are not familiar with ABAP Classes and Method:

Reference 1 - ABAP Business Development and Service Provisioning

Reference 2 - Creating, Editing, and Deleting Enhancement Implementations

Reference 3 - The Transformation Library and Formula Builder Use

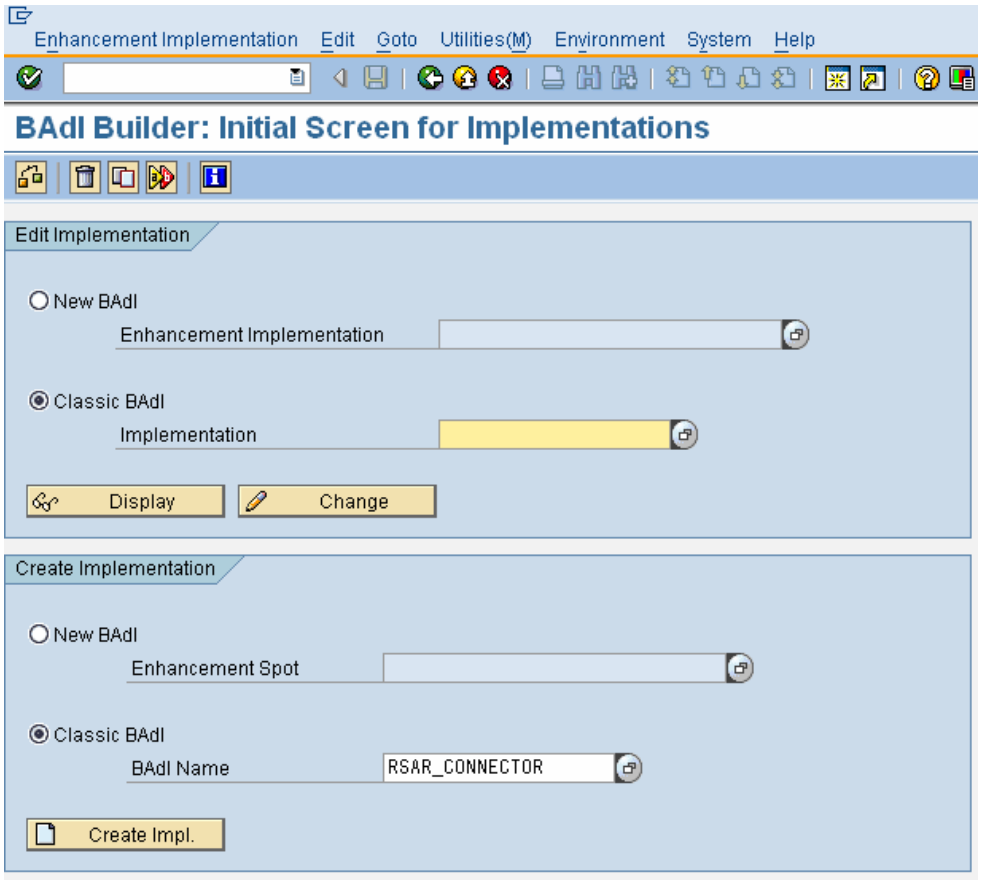
4. Step-by-Step Procedure

The guide starts you off by creating a BAdi implementation. This implementation will generate a class which we will use to store our custom functions (implemented as methods) in. When the custom methods are written and activated, we can then reference them in our BAdi implementation. Once linked, we will have them available in the Formula Builder.

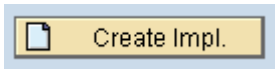
> **CAUTION**
>
> The guide does not deal with error handling for e.g. exception handling. It is important to incorporate proper error handling in a production scenario.

4.1 Create an implementation for BAdi RSAR_CONNECTOR

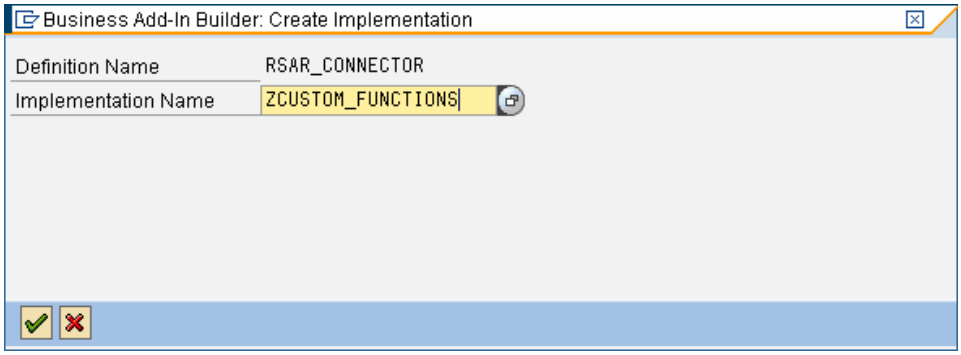
1. Go to transaction SE19. In the 'Create Implementation' section, change the radio button to 'Classic BAdi' and enter 'RSAR_CONNECTOR' as the BAdi name.



2. Press the 'Create Impl.' Button.



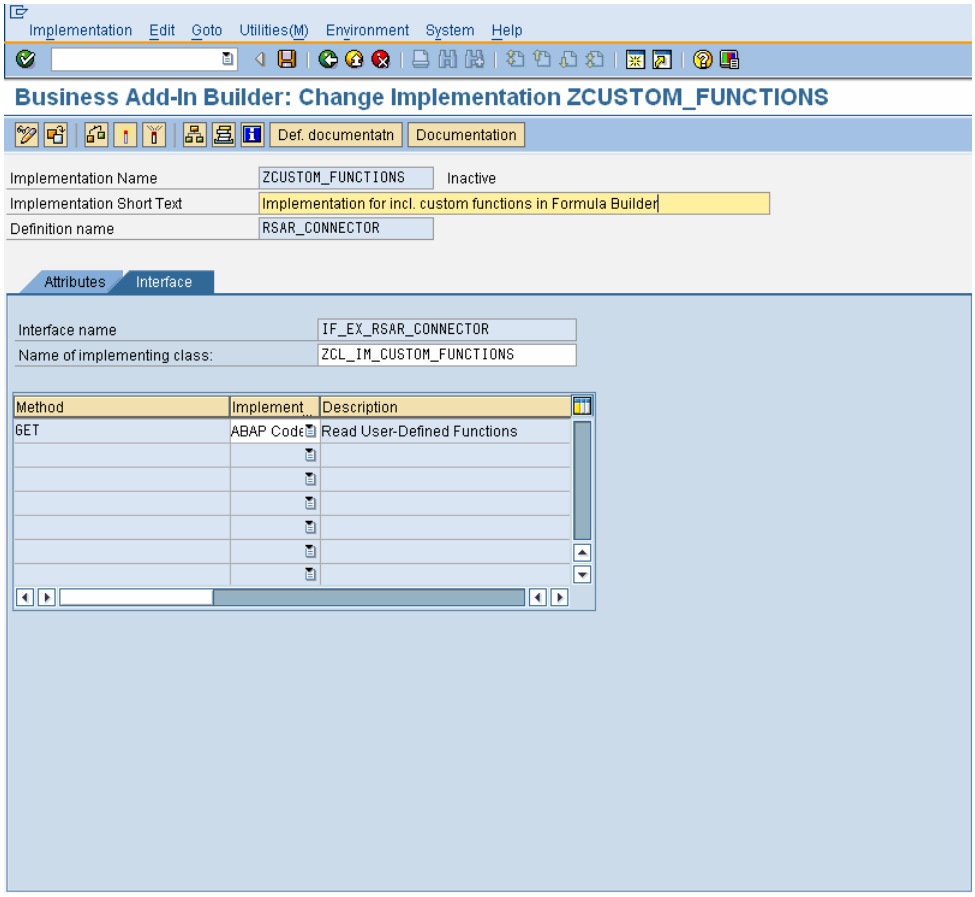
3. Give the implementation a suitable name for e.g. ZCUSTOM_FUNCTIONS, and then press the continue button ( ).



4. This will take you to the definition of the implementation.

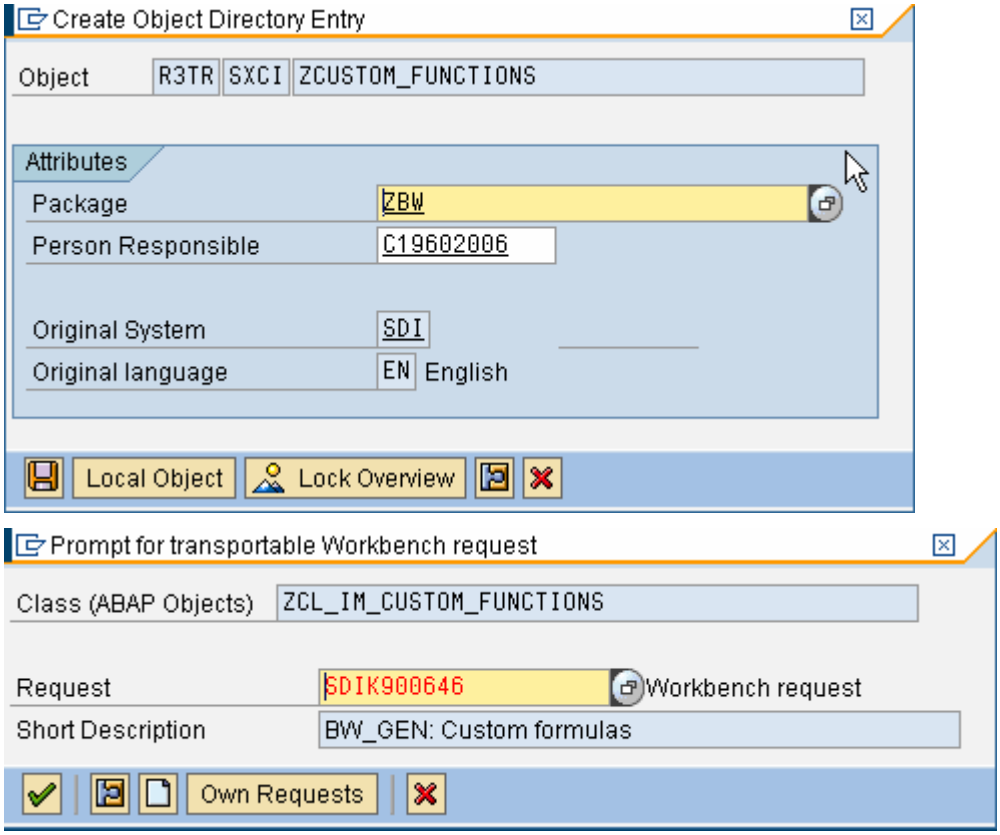
   Note

   Remember the name of the implementing class you can see in the screenshot (ZCL_IM_CUSTOM_FUNCTIONS), we will use this later to store our methods.



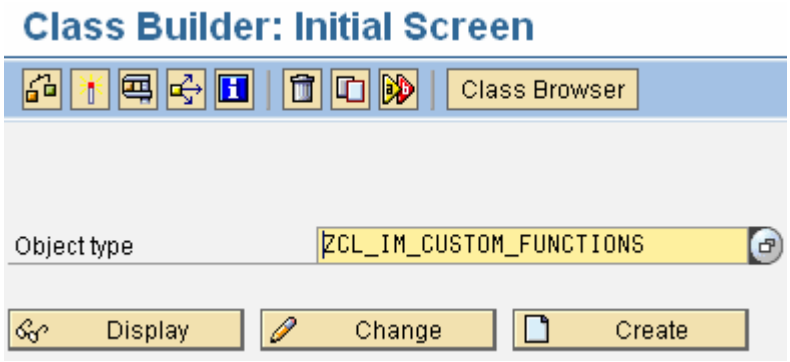
5. Press the activate button on the toolbar in order to activate your implementation and its underlying objects.

6. Assign your implementation to a transportable package (for e.g. ZBW) and include it in a transport request.



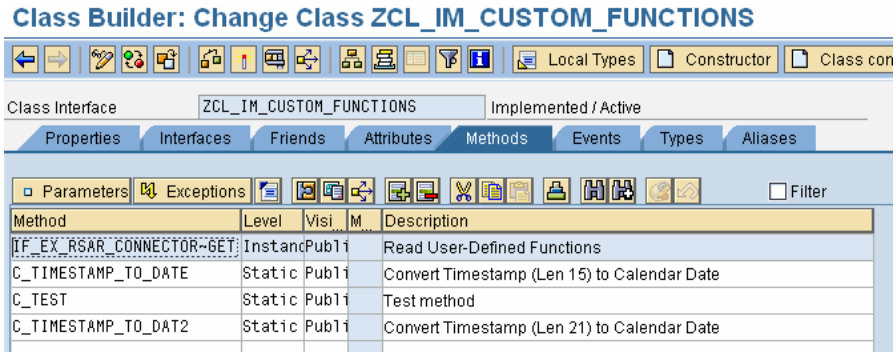
4.2 Create custom methods i.e. custom formula

1. Go to transaction SE24 in order to edit Class ZCL_IM_CUSTOM_FUNCTIONS. Alternatively, instead of using the class generated from creating the implementation, you could implement your own class with methods.



2. Press the ‘Change’ button.

3. On the ‘Methods’ tab you can implement as many custom methods as you wish. In the screenshot 3 custom methods are visible: C_TIMESTAMP_TO_DATE; C_TEST and C_TIMESTAMP_TO_DAT2.

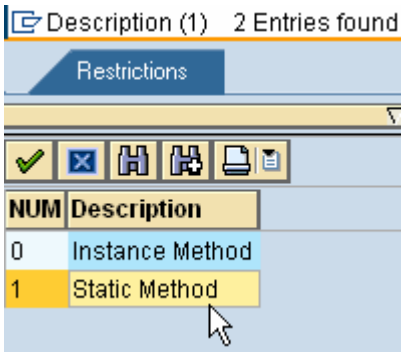


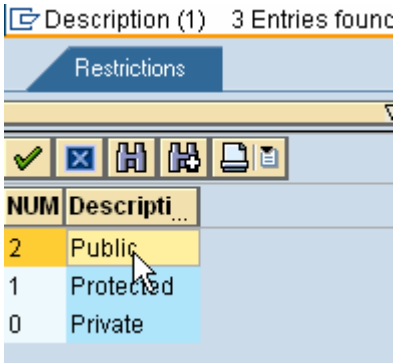
Important

Methods must have the following attributes when you create them:

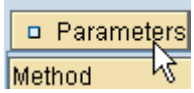
- They are declared static and public.
- They can only have importing, exporting, and returning parameters. Changing parameters are not permitted.
- They can only have one exporting or returning parameter.
- Exporting parameters cannot have a generic type.

4. Give your method a meaningful technical name and description, and declare it as Static and Public.

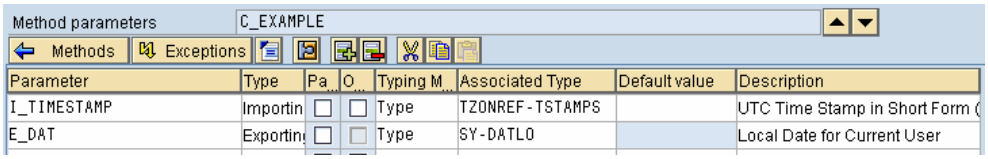




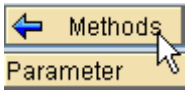
5. Once your method has a name and attributes, proceed to defining its parameters by pressing the ‘Parameters’ button.



6. Importing refers to the parameter that you pass to the method for execution and exporting refers to the parameter that is passed back once the method has executed. In our example, we pass an ‘Importing’ parameter I_TIMESTAMP (timestamp) which is then converted to a calendar day and passed back via exporting parameter E_DAT.

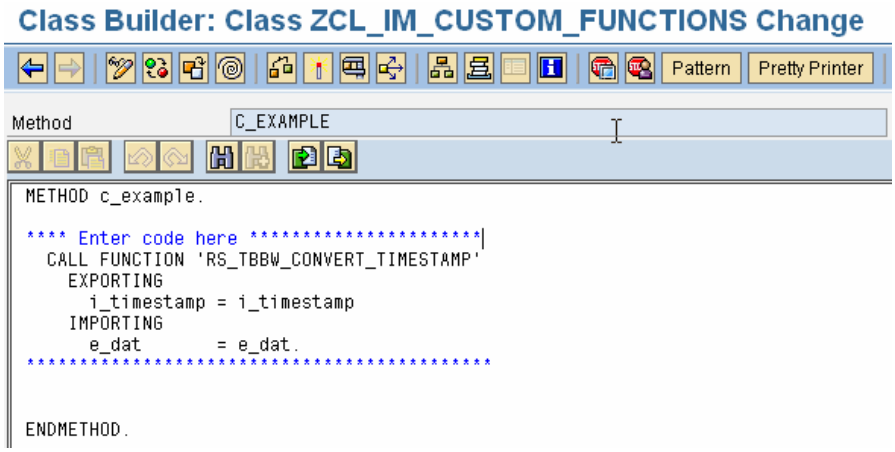


7. Press the ‘Method’ button to return back to the main method definition.



8. Double-click on the method name or press the  button to view the method code.

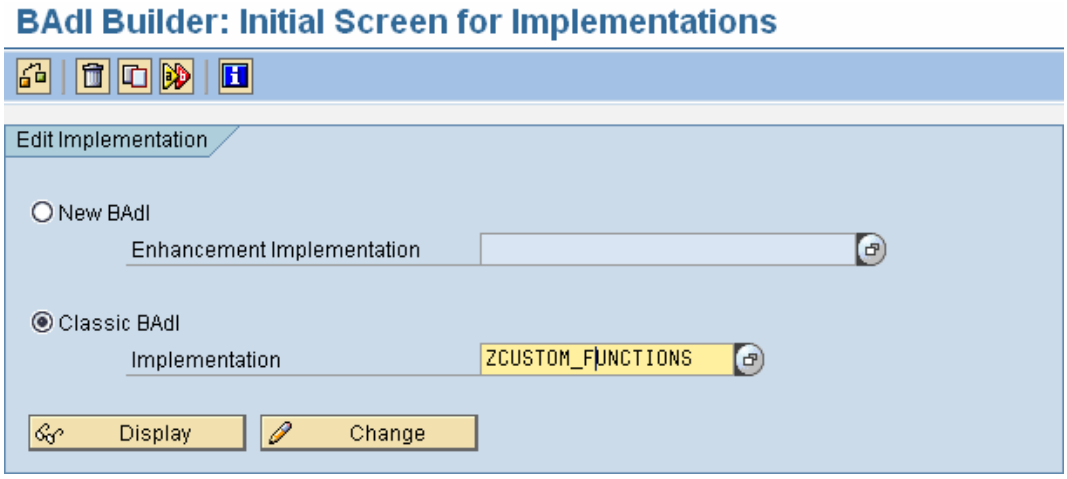
9. Enter code between the sections “METHOD <method name>. ENDMETHOD.” Remember to use your parameters exactly as you have named them on the parameter definition screen for your method. In the screenshot you will see that we pass the timestamp to function “RS_TBBW_CONVERT_TIMESTAMP” which converts it to a date.



10. Activate your method.

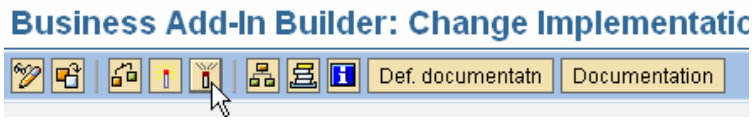
4.3 Integrate custom methods into your implementation

1. Go back to transaction SE19, and in the ‘Edit Implementation’ section, change the radio button to ‘Classic BAdi’ and enter BAdi name ‘ZCUSTOM_FUNCTIONS’.

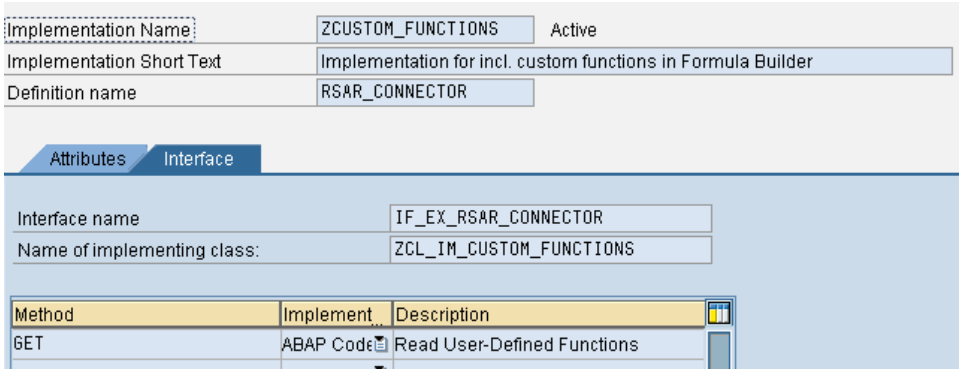


2. Press the ‘Change’ button.

3. Make sure to inactivate your BAdi implementation as you cannot edit active implementations.



4. Double-click on the ‘GET’ method on the ‘Interface’ tab in order to incorporate your custom methods as formulas.

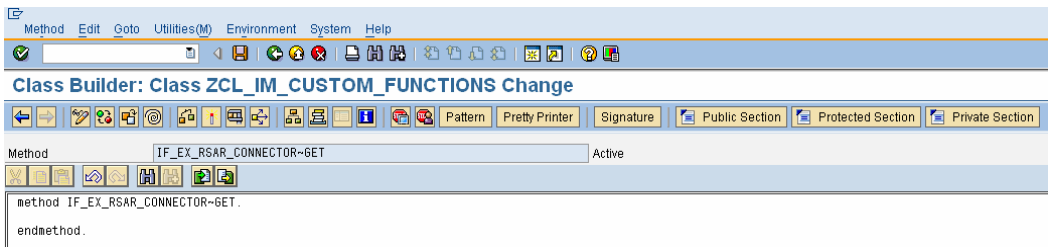


Note

The ‘GET’ method allows us to do two things:

- Firstly we get to create new categories for custom formulas (optional)
- Secondly we get to assign custom formulas to those new groupings or alternatively to the default grouping.

5. Initially the method will be empty.



6. The base format of the code should look like the code that follows. The logic is as follows:

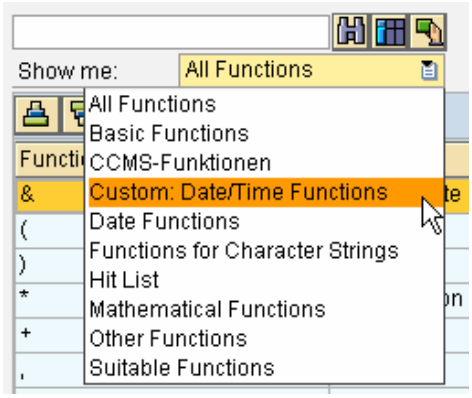
- When i_key is blank, you can use it to define new categories for formulas. In this section appending to c_operands creates a new category.
- When i_key is populated, it is referring to a category name in which you want to place the custom formulas. In this section appending to c_operands creates a new function within that category.

```
METHOD IF_EX_RSAR_CONNECTOR~GET.
//This is a code sample block
  Data: l_function  type SFBEOPRND.
  Case i_key.
  * Importing parameter: key with function category
  When ‘ ‘.
  * Use this section to declare new groupings of formulas
  When ‘CUSTOM’.
  * default category
  Endcase.
 ENDMETHOD.
```

Tip

The default category is CUSTOM. If you choose not to create your own categories then all custom formulas should be placed in the section when i_key is ‘CUSTOM’.

7. Code Sample 1 → Creating a category for functions called ‘Custom: Date/Time Functions’.




```
METHOD IF_EX_RSAR_CONNECTOR~GET.
      Data: l_function  type SFBEOPRND.
      CASE i_key.
        WHEN space.
        l_function-descriptn =  'Custom: Date/Time Functions'.
* Description of category
        l_function-tech_name = 'C_TIME'.
* Name of category in uppercase letters
```

```
        APPEND l_function TO c_operands.
*** Coding Continues for Formulas ***
    . . .
      ENDCASE.
ENDMETHOD.
```

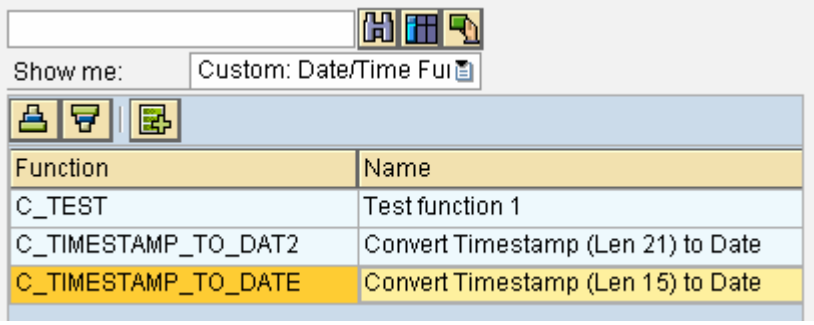
Tip

Each new category needs the following two fields populated:

1. descriptn → This is the description that will be displayed within the Formula Builder
2. tech_name → This is technical name that will uniquely identify the category in the list of categories

The fields, class and method, must NOT be populated as they are only applicable when loading new formulas.

8. Code Sample 2 → Adding formulas within the custom category C_TIME. In this code sample we have loaded 3 custom formulas: C_TIMESTAMP_TO_DATE, C_TIMESTAMP_TO_DAT2 and C_TEST.




```
METHOD if_ex_rsar_connector~get.
  DATA: l_function TYPE sfbeoprnd.
    * Structure with the description of the function
CASE i_key.
    WHEN space.
*** Coding Continues for Categories ***
    WHEN 'C_TIME'.
      CLEAR l_function.
      l_function-tech_name = 'C_TIMESTAMP_TO_DATE'.
      l_function-descriptn = 'Convert Timestamp (Len 15) to Date'.
      l_function-class     = 'ZCL_IM_CUSTOM_FUNCTIONS'.
      l_function-method    = 'C_TIMESTAMP_TO_DATE'.
      APPEND l_function TO c_operands.
**********************************************************************
      CLEAR l_function.
```

```
        l_function-tech_name = 'C_TIMESTAMP_TO_DAT2'.
        l_function-descriptn = 'Convert Timestamp (Len 21) to Date'.
        l_function-class     = 'ZCL_IM_CUSTOM_FUNCTIONS'.
        l_function-method    = 'C_TIMESTAMP_TO_DAT2'.
        APPEND l_function TO c_operands.
*********************************************************************
        CLEAR l_function.
        l_function-tech_name = 'C_TEST'.
        l_function-descriptn = 'Test function 1'.
        l_function-class     = 'ZCL_IM_CUSTOM_FUNCTIONS'.
        l_function-method    = 'C_TEST'.
    APPEND l_function TO c_operands.
    * ... further descriptions
      ENDCASE.
    ENDMETHOD.
```

Important

Each new formula needs the following four fields populated:

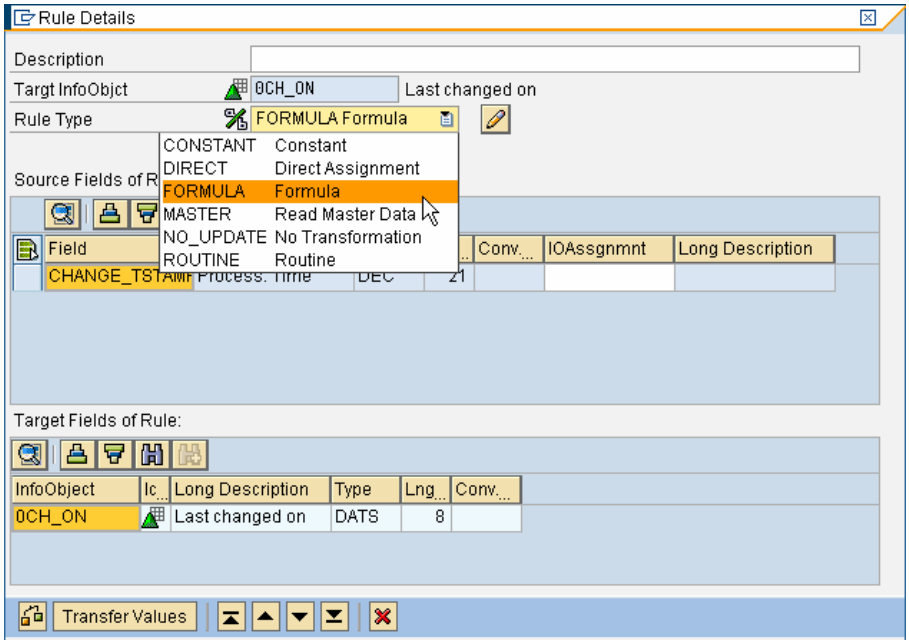
1. descriptn → This is the description that will be displayed within the Formula Builder
2. tech_name → This is technical name that will uniquely identify the formula in the list of formulas
3. class → This is the ABAP class in which the formula method is implemented
4. method → This is the method within the ABAP class mentioned above in which the formula is implemented

9. Activate the GET method using the activate button on the toolbar.

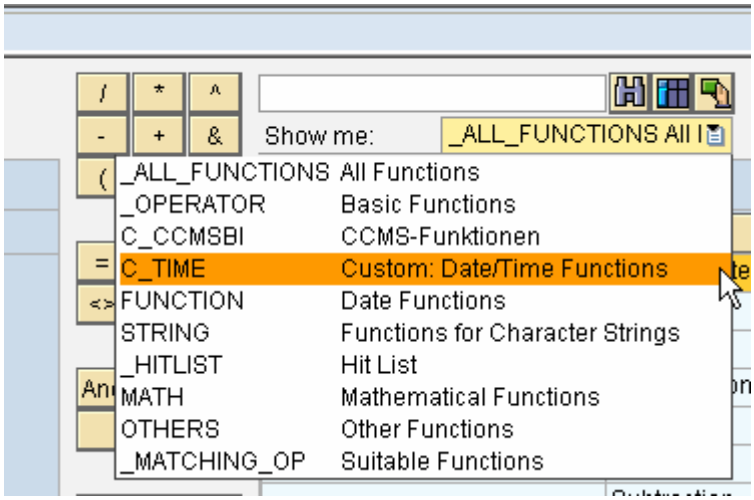
10. Go back one screen and activate your implementation.

4.4 View in Formula Builder

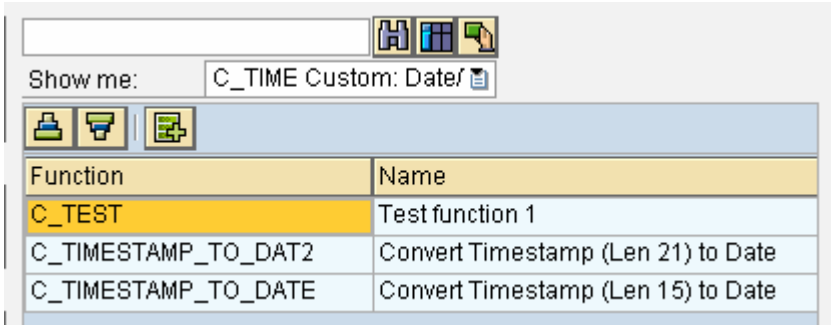
1. To access your custom formulas, enter the Formula Builder either from your transformation, update rules or transfer rules.



2. In the list of formula categories, select our custom category C_TIME.



3. All our custom formulas assigned to that category are now available.



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