BEx Web Application Reporting: Modify Cell Contents

Applies To:

SAP BEx Web Application Designer (BW 3.10)

Article Summary

With the help of BW Table Interface, cell formatting and content management can be manipulated by using a custom ABAP class for web reporting. While there are many uses of the Table Interface, this article will focus on delivering a step by step solution to calculate a value for a specific cell on a table item by using values from other fields of that table. For a better understanding, this "how to" provides an example of performing a calculation on the usually untouchable result (or summary) row of a query.

By: Catherine Fan
Title: Technical Consultant
Date: 13 Sept 2005

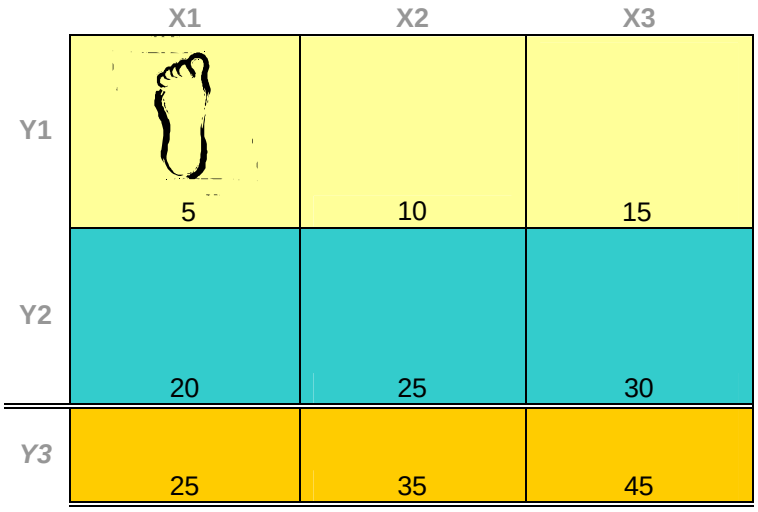
Table of Content

Overview: Background & How It Works

Often times there are requirements to perform a calculation outside of a query. While workbook customization may satisfy this objective, the solution is ends when the query results in over the MS Excel limit of roughly 65K records. By displaying the query in a table item on a web template, this challenge can be overcome. Please remember that this solution meets the requirements for data displayed via the web only.

The standard steps for "populating" a table item on a web template are unaffected in implementing this solution. The real work happens in a custom ABAP class that is called by the table item. This ABAP class is referenced by a one line parameter call on the html page of the web template.

The manner in which the ABAP class works, is illustrated in the example below:



The above table represents the table item with the data provider set to a BEx query. X1, X2 & X3 designate the position of the columns/fields, or the X coordinates. Y1, Y2 & Y2 designate the coordinates of the rows of the report. Y3, in this example, is denoted in italic because it is "special", it represents the summary of rows Y1 & Y2 down the X axis.

Now imagine walking through this table. From this point forward, I'll reference each X & Y intercept as a cell within the table. The first cell: X1, Y1 carries a cell content of the numeric value 10 and the color is yellow . The next cell: X2, Y1 carries a cell content of the numeric value 20 and the color is also yellow… and so forth. Notice that once we've stepped foot in cell X2, Y1, the new cell content is 20 and the value 10 is no longer visible. This concept is exactly how the processing for the custom ABAP class handles "walking" through the table item. It is on a cell by cell basis. The variables and properties/values of each cell are giving you the current state of the current cell.

This can cause confusion because most of us, as BW developers, have taken advantage of ABAP's internal table concept which basically allows you to loop through data within an internal table to make use of values throughout the table across both the X & Y axis. The importance here is that you understand how the ABAP class is processing the table item; the query has already been executed. By using the custom ABAP class, you have the opportunity to

capture/modify cells – ONE AT A TIME – by evaluating the properties of the “current” cell.. the cell of the table item that is being processed by the custom ABAP class.

If you are a seasoned ABAPer, this concept is similar to that of setting screen properties on a table control for dynpro programming: you can use screen attributes such as screen-fieldname to determine which “cell” is being processed.

Remember: the ABAP class is “walking” through the table item of the web template – in the methods, you can access certain properties including the value of each cell.

Getting Started: Step By Step Instruction

For this specific example, our requirement is to perform a simple calculation on the result row. Basically, following the example structure from above, we want to take X1 divide it by X2 and store it in X3 only on result/summary rows (Y3).

For the methods in our class, we’ll need download these text files containing the code for our methods to be used later.


INIT METHOD

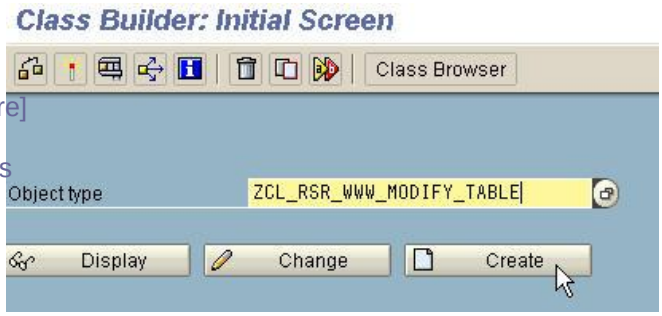

DATA_CELL METHOD

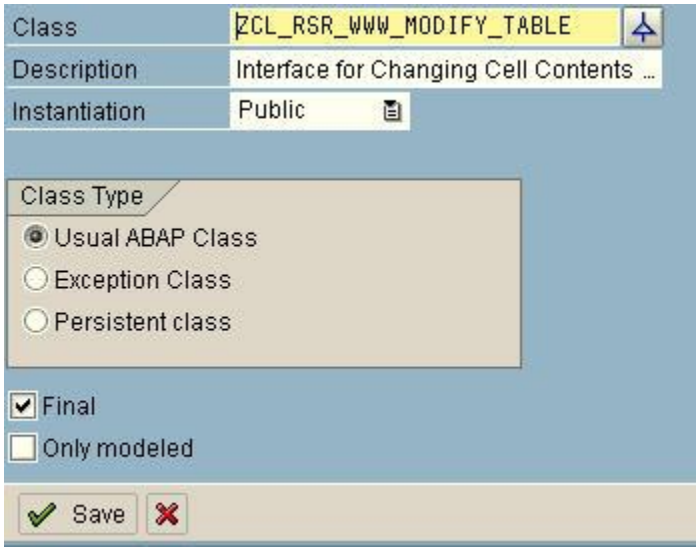

STUCTURE_CELL METHOD

Task 1: Steps for Creating your custom ABAP class

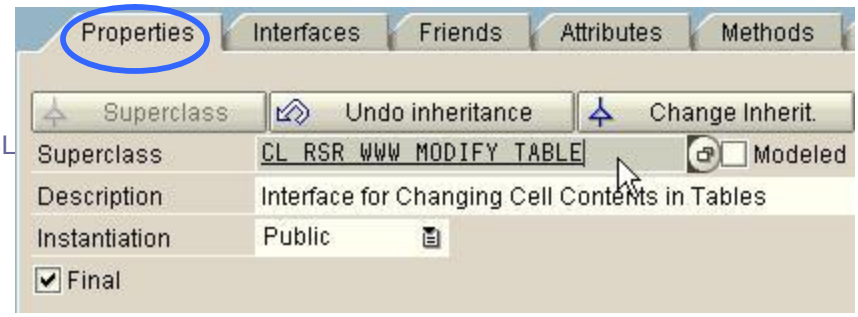
The first thing we’ll want to do is create our custom class which will process the table item from the web template.

1-1. Go to transaction “SE24”
1-2. Create Object Type: [enter your custom class name here]
1-3. Click the “Create” Button
1-4. Enter a description & accept the creation screen defaults
1-5. Click “Save”



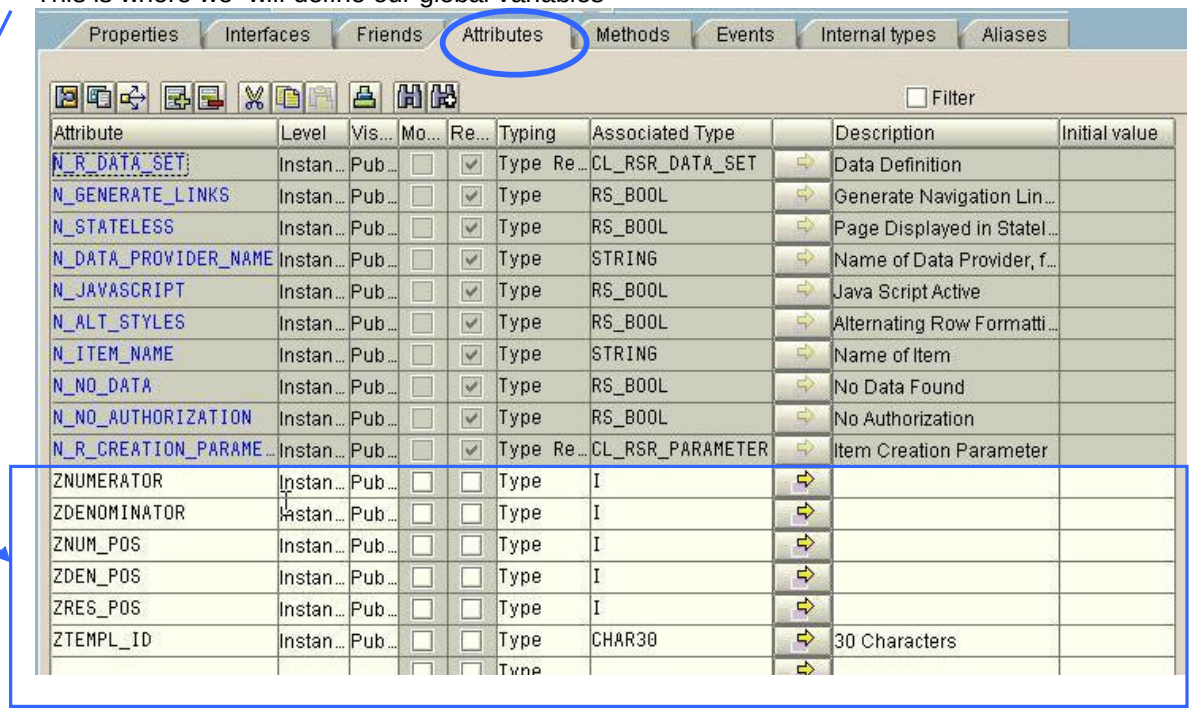


1-6. Enter Package & Transport information
1-7. Go to the Properties tab
1-8. Click the “Superclass” Button
1-9. Enter “CL_RSR_WWW_MODIFY_TABL
1-10. Save the class



1-11. Click the “Attributes” tab

This is where we will define our global variables



Before we create the global variables, we need to consider what fields we need to perform our calculation for this particular query.

In this example, we will need to create 2 value variables which will store the values of the fields (numerator and denominator) to be used in our calculation. Of course, before we can even capture those values, we need to determine the location (X coordinate) of those fields.

To make this process more dynamic, we can use a custom table to carry the field descriptions that will be used in the query as well as the role they will play in our calculation. (This will allow us to use the same class for multiple queries that will require the same calculation and allow for the field description to change without having to maintain the code). See appendix 1 on how to create a custom table. In order to use this same class to perform the same calculation on a different query using different fields, we’ll want to capture the web template technical name in our custom table as part of the key so our class can determine which fields should be used based on the query. To make this work, we’ll need a global variable for the template id as well.

First, we make an evaluation of the fields that we need for the calculation. For our example, we need to know which fields from the query will be used as the numerator, denominator and the result (or answer). For our calculation, we are calculating the percentage of sales which result in returns.

In our custom table, we'll enter fields:

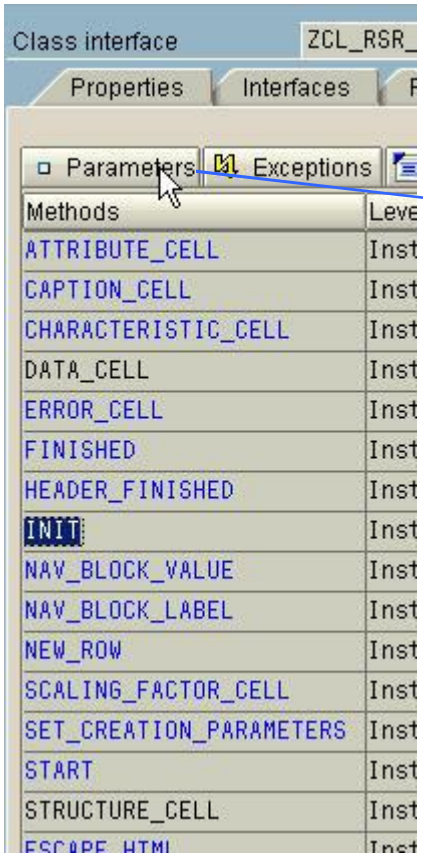
Web Template technical name	**Field_job**	**Field_txt (from query)**
ZCALC_QRY	NUM	Return Qty
ZCALC_QRY	DEN	Total Qty Sold
ZCALC_QRY	ANS	%Returns

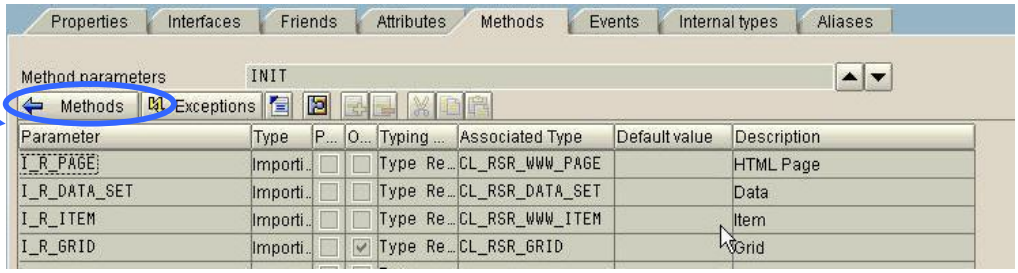
We also need to create our variables which will carry the X coordinates of all 3 fields as well as the values of our numerator and denominator. A variable to store the result/answer field is unnecessary since we can store this directly into the current cell value.

To allow the ABAP class to determine the web template id from the calling web template, we are going to redefine the method "INIT".

Before we modify the method, let's take a look at the parameters which are available to this method.

1-12. Click the methods tab
1-13. Place cursor on Method "INIT"
1-14. Click the "Parameters" button

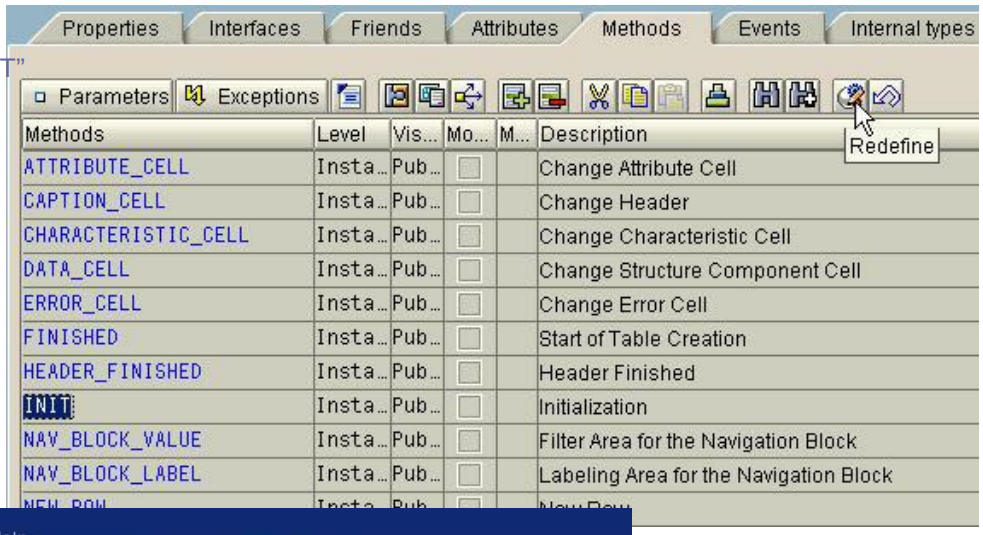




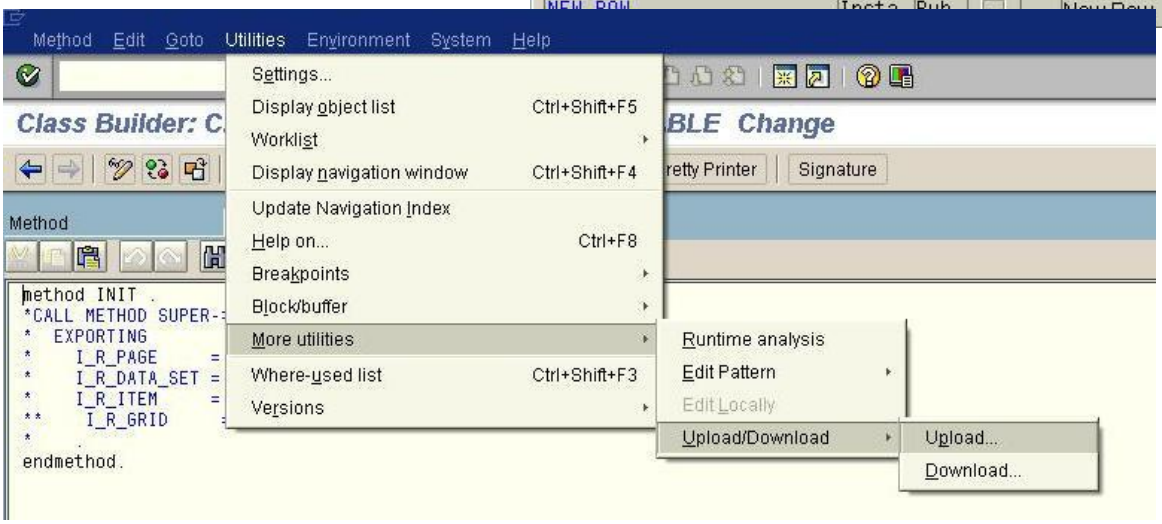
From method "INIT", we will take advantage of the Parameter, "I_R_PAGE" to capture the web template name and use it globally.

To return to the methods view to redefine the code,
1-15. click the methods button above the parameters.

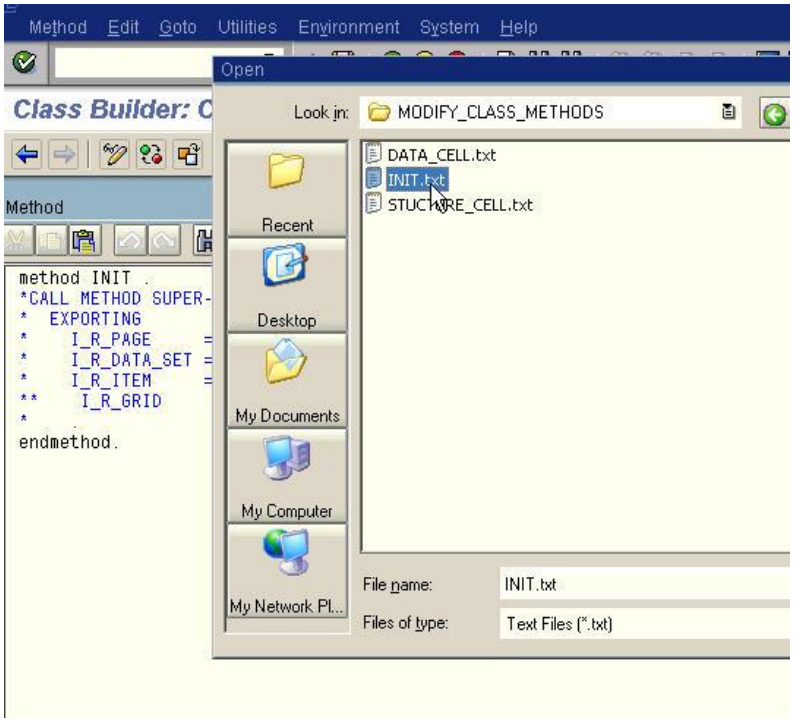
1-16. Place cursor on Method “INIT”
1-17. Click the “Redefine” button



1-18. Follow menu path:
Utilities → More Utilities → Upload/Download → Upload



1-19. Navigate to the previously unzipped folder and select the corresponding file name with the same Method name
1-20. Save and Activate the method



1-21. Return to the “Methods” tab by using the green arrow back button and repeat steps 16-20 for the remaining methods.

As before, to see the available parameters for each method – click the parameters button above the methods on the methods tab while cursor is on the method you wish to see.

✓	INIT
	STRUCTURE_CELL
	DATA_CELL

1-22. Save and activate the entire class

For a further understanding of what each method is doing, let’s take a look at the key components of the code:

STRUCTURE_CELL METHOD: This is where we will locate the X coordinate of our fields

```
    SELECT * FROM ZTABLE
      INTO TABLE it_ztab
     WHERE TEMPLID = ztempl_id.

* load itabs to hold full description text of column
  LOOP AT it_ztab INTO wa_ztab.
    CASE wa_ztab-field_job.
      WHEN 'DEN'.
        IF zden_pos IS INITIAL.
          SPLIT wa_ztab-field_txt  AT SPACE INTO TABLE it_text_den.
        ENDIF.
      WHEN 'NUM'.
        IF znum_pos IS INITIAL.
          SPLIT wa_ztab-field_txt  AT SPACE INTO TABLE it_text_num.
        ENDIF.
      WHEN 'ANS'.
        IF zres_pos IS INITIAL.
          SPLIT wa_ztab-field_txt  AT SPACE INTO TABLE it_text_res.
        ENDIF.
    ENDCASE.
  ENDLOOP.

ENDIF.

* check denominator position
CLEAR: w_not_pos, w_left.
LOOP AT it_text_den INTO w_text.
  SEARCH i_text+w_left FOR w_text.
  IF SY-SUBRC = 0. "match!
    w_left = STRLEN( w_text ).
    w_left = w_left + SY-FDPOS.
    w_not_pos = 'x'.
    CONTINUE. "get next word of description
  ELSE.  "no match!!
    CLEAR w_not_pos.
    EXIT.
  ENDIF.
ENDLOOP.
IF NOT w_not_pos IS INITIAL.
  zden_pos = I_X. "set x coordinate of denominator
  FREE it_text_den.
ENDIF.
```

This step is optional; it’s meant to keep the functionality as dynamic as possible. Allowing 2 things:
1. to use the same class for multiple queries
2. to allow the field description on the query to change without having to maintain the code

Parameter which carries the description of the field

Parameter which carries current X Coordinate

The reason we cannot do a direct comparison (ie. w_text = i_text) is because sometimes, we want the query to show:

Total Qty
Sold

Where sold is on it’s own line. ABAP reads that with spacing in between Qty and sold. In order to accommodate this, we’re using the internal table logic.

Without using the itab logic and if you knew it were just one line on the description, you could use:

DATA_CELL METHOD: This is where we find the X coordinates of our variables & capture there values

```
IF c_cell_style = 'SAPBEXaggData'.
  CASE i_x.
    WHEN znum_pos.
      znumerator = i_value.
    WHEN zden_pos.
      zdenominator = i_value.
    WHEN zres_pos.
      IF znumerator <> 0.
        c_cell_content = ( znumerator / zdenominator ) * 100.
      ENDIF.
  ENDCASE.
ENDIF.
```

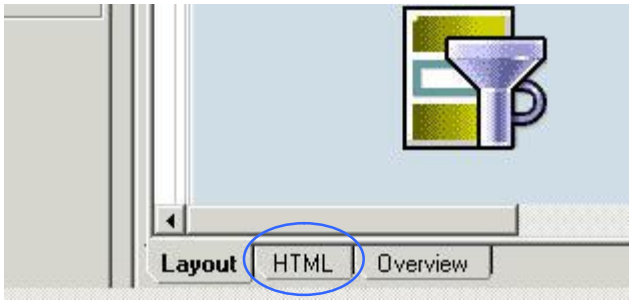
This cell style is used for the result row… allowing us to format only result rows

Parameter which carries the value of the cell

Where we overwrite the contents of the cell

Task 2: Call custom ABAP method from web template

2-1. Go to the web application designer
2-2. Open your web template
2-3. Create a table item and assign the data provider to your query
2-4. Click the “html” tab



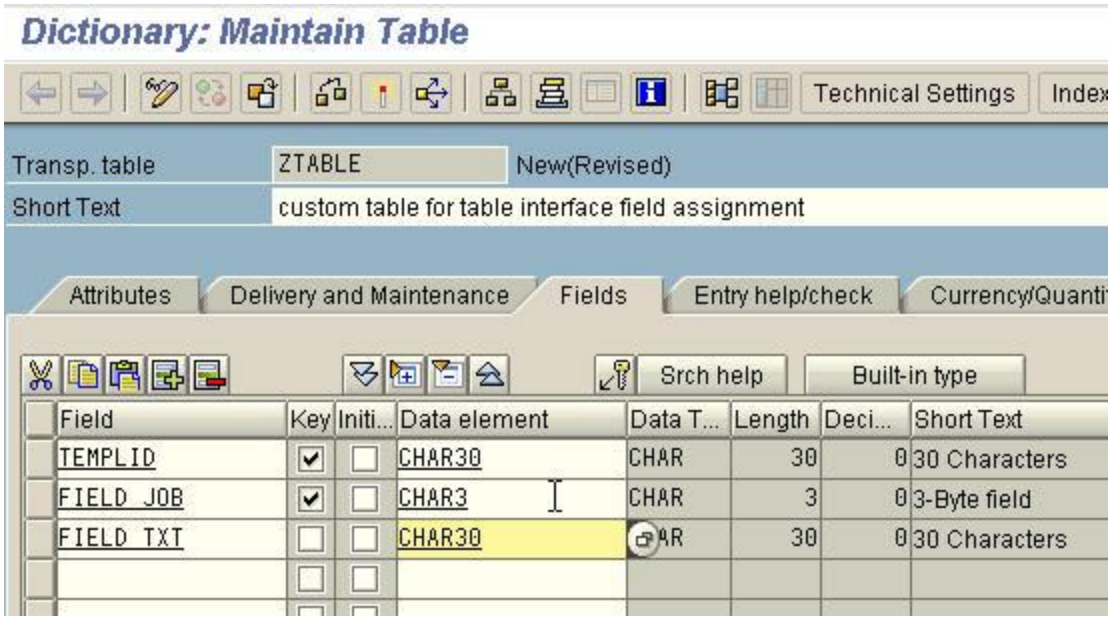
2-5. Find the table item object and add :

<param name=”MODIFY_CLASS” value=”[enter your custom class name here]”/>

APPENDIX 1: Creating a custom table to hold the field descriptions

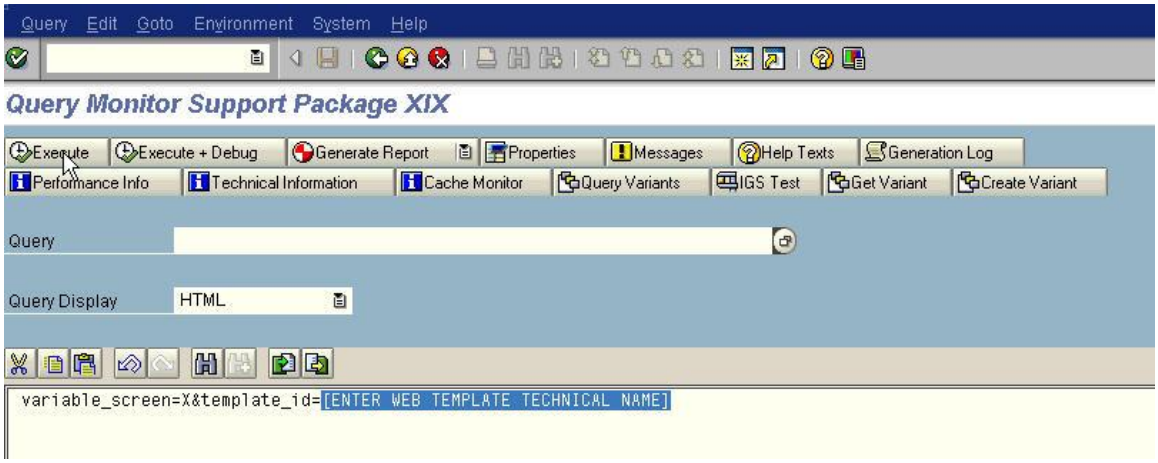
A1-1. Go to transaction “SE11”
A1-2. Follow the steps from SAP documentation on creating a table ->
http://help.sap.com/saphelp_nw04/helpdata/en/cf/21eb6e446011d189700000e8322d00/content.htm

Be sure to select: *Display/Maintenance allowed*



APPENDIX 2: Debugging the Custom ABAP Class

A2-1. Set a break-point in your method which you'd like to debug
A2-3. Run transaction "RSRT2"
A2-4. Enter your web template technical name
A2-5. Click Execute



Disclaimer & Liability Notice

This document may discuss sample coding, which does not include official interfaces and therefore is not supported. 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 of the code and methods suggested here, and anyone using these methods, is doing it under his/her own responsibility.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of the technical article, including any liability resulting from incompatibility between the content of the technical article and the materials and services offered by SAP. You agree that you will not hold SAP responsible or liable with respect to the content of the Technical Article or seek to do so.