

# Using ABAP-OO in BI-Transformations through Custom Class



## Applies to:

BI 7.0 BW 3.5. For more information, visit the [Business Intelligence homepage](#).

## Summary:

In BI 7.0, BI Transformation routines must use ABAP-OO . Going a step further, using custom class gives not only tremendous advantages of OO, but also through a logical segregation of ABAP code from BI design. This is illustrated with a practical application. Custom classes can also be used on earlier versions of BI.

**Author:** Varad Desikan

**Company:** Capgemini(US)LLC

**Created on:** 4 July 2009

## Author Bio

Varad Desikan is Manager, BI with the East Business Unit of Capgemini (US) LLC

## Table of Contents

|  |                                     |
|--|-------------------------------------|
| 1. Introduction .....                                    | 3                                   |
| 2. Case Application .....                                | 3                                   |
| 3. Calling the Custom Class in the Transformation: ..... | 3                                   |
| 3.1 Start Routine: .....                                 | 3                                   |
| 3.2 End Routine: .....                                   | 5                                   |
| 4.0 Construction of the Custom Class Object:.....        | 5                                   |
| 4.1 Communication between Transformation and Class ..... | 5                                   |
| 4.2 Class Attributes .....                               | 9                                   |
| 4.3 Methods .....  | 9                                   |
| 4.4 Parameters.....                                      | 10                                  |
| 4.5 Brief on methods used in the case example .....      | 11                                  |
| 5. Conclusion .....                                      | 12                                  |
| Related Content.....                                     | 13                                  |
| Copyright.....   | <b>Error! Bookmark not defined.</b> |

## 1. Introduction

In BI 7.0, BI Transformation routines must use ABAP-OO and all new constructs by SAP follow ABAP-OO. Indeed, as per the methodology promoted by SAP, ABAP-OO should be the right path for any ABAP development whether in ECC or BI. It has been a common practice to use function modules(standard and custom) within start routines to avoid repetitive code. Going further in this process, using custom class gives tremendous advantage not only in terms of advantages of OO viz., reliability, re-usability, information hiding etc. but also in the speed of development and advantage of segregation of code and BI Modelling. This is illustrated with a real application approach. Indeed, custom classes can also be used on earlier versions of BI, such as BW 3.5.

## 2. Case Application

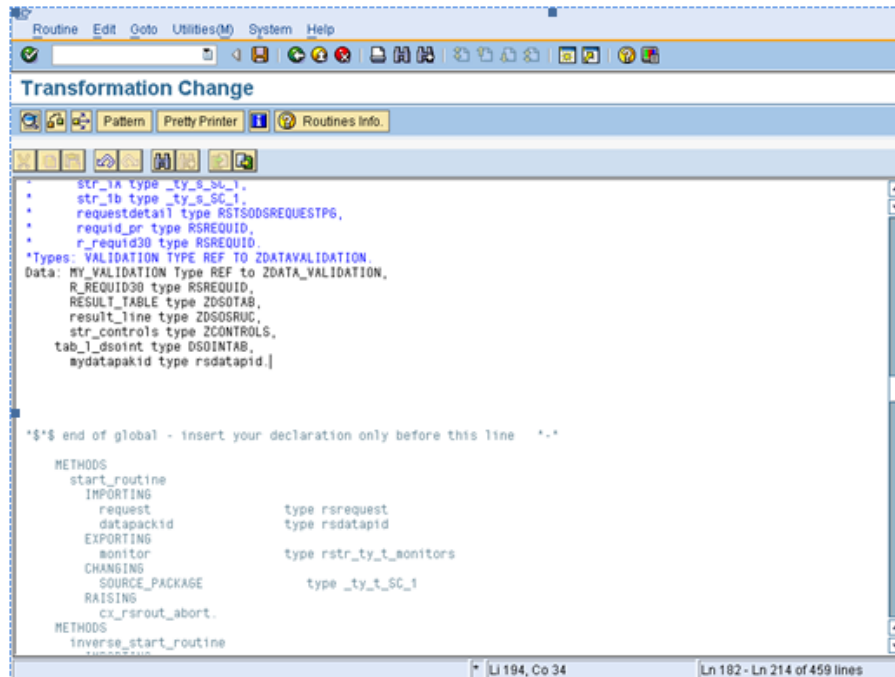
The application is: Custom validation of flat file inputs to load GL DSO for balances from different companies of a multi-national, into the Acquisition Layer. There are different flat files for Actual, and Plan(of different types).The file is accepted or rejected in this layer and only accepted files move into the Integration layer. This necessitates building exhaustive and complex validation routines, within the transformation for each type of transaction . The process is off-loaded into a custom class, which is instantiated in the Transformation. The custom class takes the source-package as the input and returns the modified source-package with error codes or success codes and the uploaded request is processed subsequently by a separate ABAP error-handler. The development of the custom class and usage in the different transformations is illustrated in the following.

## 3. Calling the Custom Class in the Transformation:

The build in the Transformation to call the custom class is illustrated here:

### 3.1 Start Routine:

In the Global Area, define the Class under Data: definition



```

Routine  Edit  Goto  Utilities(M)  System  Help
Transformation Change
Pattern  Pretty Printer  Routines Info.
*
* str_1a type _ty_s_sc_1.
* str_1b type _ty_s_sc_1.
* requestdetail type RSTSDSREQUESTPS.
* request_pr type RSREQUEST.
* r_request30 type RSREQUEST.
*Types: VALIDATION TYPE REF TO ZDATAVALIDATION.
Data: MY_VALIDATION Type REF to ZDATA_VALIDATION.
R_REQUEST30 type RSREQUEST.
RESULT_TABLE type ZDSOTAB.
result_line type ZDSOSRUC.
str_controls type ZCONTROLS.
tab_1_dsoint type DSOINTAB.
mydatapakid type rsdatapid.]
*$$$ end of global - insert your declaration only before this line  *-*
METHODS
start_routine
IMPORTING
request                type rsrequest
datapackid            type rsdatapid
EXPORTING
monitor               type rst_rty_t_monitors
CHANGING
SOURCE_PACKAGE        type _ty_t_sc_1
RAISING
cx_rsrout_abort.
METHODS
inverse_start_routine

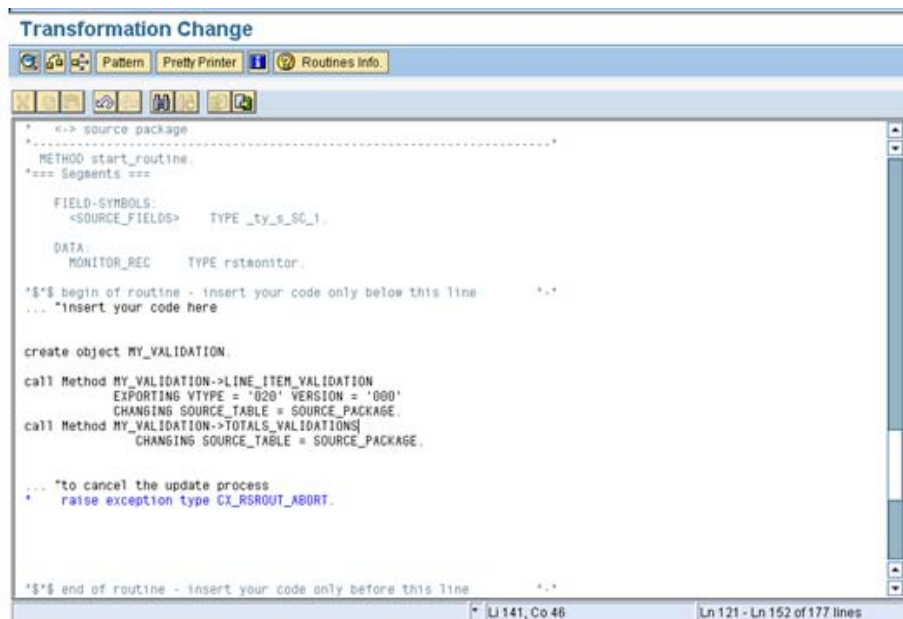
```

Screen 1: Global Area definition of class

In the Transformation Method, where the user code is written,

First, 'Instantiate' the class by 'Create Object'.

When you instantiate, the Constructor method of the Class is invoked(if it is included as a method)



```

Transformation Change
Pattern  Pretty Printer  Routines Info.
*
* <-> source package
*-----
METHOD start_routine.
*==== Segments ====
FIELD-SYMBOLS:
<SOURCE_FIELDS>  TYPE _ty_s_sc_1.
DATA:
MONITOR_REC      TYPE rstaonitor.
*$$$ begin of routine - insert your code only below this line  *-*
... "insert your code here
create object MY_VALIDATION.
call Method MY_VALIDATION->LINE_ITEM_VALIDATION
EXPORTING VTYPE = '020' VERSION = '000'
CHANGING SOURCE_TABLE = SOURCE_PACKAGE.
call Method MY_VALIDATION->TOTALS_VALIDATIONS
CHANGING SOURCE_TABLE = SOURCE_PACKAGE.
... "to cancel the update process
* raise exception type CX_RSRROUT_ABORT.
*$$$ end of routine - insert your code only before this line  *-*

```

Screen 2: Instantiation of the Class &amp; calling the methods

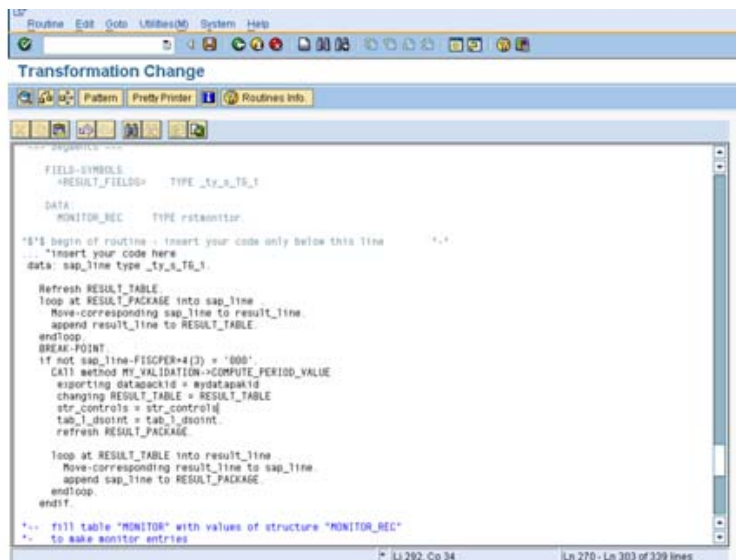
Then, Call the methods of the Class as required; referring to Screen 2, the entire source\_package is passed to the Class-method and it returns with the 'Changed' source\_package besides the controls updated in a defined structure.

The first Call is made to the method LINE\_ITEM\_VALIDATION passing the parameters vtype = '010' for value type for actual(transformation to Actuals DSO) and version which specifies type of actual(eliminated, uneliminated), the datapakid to address special treatment for the last datapack.

The second call to method TOTALS\_VALIDATION invokes validation of the totals(at end of the last package), where accumulated totals are validated for various criteria.

### 3.2 End Routine:

In the End Routine, the class is used to transform the Result\_Package. In the case, period value for each line is computed from Totals for Current period and previous period.



```

Routine Edit  Odb  Utilities  System  Help
Transformation Change
Pattern  Print/Printer  Routines Info
FIELDS-SYMBOLS
<RESULT_FIELDS>  TYPE _ty_s_RS_T
DATA
  MONITOR_REC  TYPE rskontor.
*-*- begin of routine - insert your code only below this time  *-*-
... *insert your code here
data: sap_line type _ty_s_RS_T.

Refresh RESULT_TABLE.
loop at RESULT_PACKAGE into sap_line
  Move-corresponding sap_line to result_line.
  append result_line to RESULT_TABLE.
endloop.
BREAK-POINT
if not sap_line-FISCPER+4(3) = '000'
  call method MY_VALIDATION->COMPUTE_PERIOD_VALUE
    exporting datapackid = mydatapackid
    changing RESULT_TABLE = RESULT_TABLE
    str_controls = str_controls@
    tab_1_dsort = tab_1_dsort@
  refresh RESULT_PACKAGE.

loop at RESULT_TABLE into result_line
  Move-corresponding result_line to sap_line
  append sap_line to RESULT_PACKAGE
endloop.
endit.
*-*- fill table "MONITOR" with values of structure "MONITOR_REC"
*-*- to make monitor entries
  
```

Screen 3 Call to a method in the End Routine

## 4. Construction of the Custom Class Object:

This case example involves passing the entire package: source\_package or result\_package to the Class and getting back the changed package

### 4.1 Communication between Transformation and Class

Create 2 structures which map to the source structure of the data source and the DSO(or Cube) structure to which the data is loaded by the Data source; the source structure is for use in the Start Routine and the DSO structure for the End Routine.

The screenshot shows the SAP Dictionary: Maintain Structure interface. The structure is ZTRANSTRUCT, which is active. The short description is 'Dataseource' and 'Source Structure'. The 'Components' tab is selected, showing a list of predefined types. The table below is a representation of the data shown in the screenshot.

| Component       | RTy                      | Component type | Data Type | Length | Decim | Short Description  |
|-----------------|--------------------------|----------------|-----------|--------|-------|--------------------|
| /BIC/CSUNIT     | <input type="checkbox"/> |                | CHAR      | 6      | 0     |                    |
| /BIC/ACUNIT     | <input type="checkbox"/> |                | CHAR      | 4      | 0     |                    |
| /BIC/FISCPER    | <input type="checkbox"/> |                | CHAR      | 7      | 0     |                    |
| VTYPE           | <input type="checkbox"/> |                | CHAR      | 3      | 0     |                    |
| VERSION         | <input type="checkbox"/> |                | CHAR      | 3      | 0     |                    |
| /BIC/CURRENCY   | <input type="checkbox"/> |                | CUKY      | 5      | 0     |                    |
| UNIT            | <input type="checkbox"/> |                | UNIT      | 3      | 0     |                    |
| /BIC/FSITEM     | <input type="checkbox"/> |                | CHAR      | 12     | 0     |                    |
| /BIC/USRCO      | <input type="checkbox"/> |                | CHAR      | 4      | 0     |                    |
| PROFIT_GTR      | <input type="checkbox"/> |                | CHAR      | 6      | 0     |                    |
| /BIC/FUNCAREA   | <input type="checkbox"/> |                | CHAR      | 16     | 0     |                    |
| /BIC/COSTCENTER | <input type="checkbox"/> |                | CHAR      | 18     | 0     |                    |
| /BIC/LCMTI      | <input type="checkbox"/> |                | CHAR      | 18     | 0     |                    |
| /BIC/USDAMI     | <input type="checkbox"/> |                | CHAR      | 18     | 0     |                    |
| /BIC/QUANTITY   | <input type="checkbox"/> |                | CHAR      | 18     | 0     |                    |
| /BIC/G_ERRTYP   | <input type="checkbox"/> |                | CHAR      | 3      | 0     |                    |
| REFREG          | <input type="checkbox"/> |                | NUMC      | 4      | 0     |                    |
| RECORD          | <input type="checkbox"/> | RSARECORD      | INT4      | 10     | 0     | Data record number |

#### Screen 4 Source Structure

For the DSO structure, the DSO dictionary structure can also be used.

**Dictionary: maintain structure**

Structure: ZDSOSTRUC  
Short Description: DSO structure

Attributes Components Entry help/che

Predefined Typ

| Component      | RTy                      | Component type    |
|----------------|--------------------------|-------------------|
| RECORD         | <input type="checkbox"/> | /BI0/OIRECORD     |
| /BIC/CSUNIT    | <input type="checkbox"/> | /BIC/OICSUNIT     |
| /BIC/ACUNIT    | <input type="checkbox"/> | /BIC/OIACUNIT     |
| /BIC/FSITEM    | <input type="checkbox"/> | /BIC/OIFSITEM     |
| PROFIT_CTR     | <input type="checkbox"/> | /BIC/OIPROFIT_CTR |
| /BIC/FUNCGAREA | <input type="checkbox"/> | /BIC/OIFUNC_AREA  |
| /BIC/USRCD     | <input type="checkbox"/> | /BIC/OIUSRCD      |
| FISCPER        | <input type="checkbox"/> | /BI0/OIFISCPER    |
| VTYPE          | <input type="checkbox"/> | /BI0/OIVTYPE      |
| VERSION        | <input type="checkbox"/> | /BI0/OIVERSION    |
| /BIC/FSCHART   | <input type="checkbox"/> | /BIC/OIFSCHART    |
| RECORDMODE     | <input type="checkbox"/> | RODMUPDMOD        |
| /BIC/CSPRTNR   | <input type="checkbox"/> | /BIC/OICSPRTNR    |
| /BIC/ERRTYP    | <input type="checkbox"/> | /BIC/OIERRTYP     |
| CURTYPE        | <input type="checkbox"/> | /BI0/OICURTYPE    |
| AMOUNT         | <input type="checkbox"/> | /BI0/OIAMOUNT     |
| /BIC/KUSDAMI   | <input type="checkbox"/> | /BIC/OIKUSDAMI    |

Screen 5: DSO structure

Create 2 table types based on the 2 structures.

**Dictionary: Maintain Table Type**

Table Type: ZTRANSTABTYP Active

Short text: Table type for Data Source Structure

Attributes | **Line Type** | Initialization and Access | Key

Line Type: ZTRANSSTRUC

Predefined Type

Data Type: [ ]

No. of Characters: 0 Decimal Places: 0

Reference type

Name of Ref. Type: [ ]

Reference to Predefined Type

Data Type: [ ]

Length: 0 Decimal Places: 0

Screen 6: Source table-type

Table Type | Edit | Goto | Utilities(M) | Environment | System | Help

**Dictionary: Maintain Table Type**

Table Type: ZDSOTABTYP Active

Short text: Table Type for DSO STRUC

Attributes | **Line Type** | Initialization and Access | Key

Line Type: ZDSOSTRUC

Predefined Type

Data Type: [ ]

No. of Characters: 0 Decimal Places: 0

Reference type

Name of Ref. Type: [ ]

Reference to Predefined Type

Data Type: [ ]

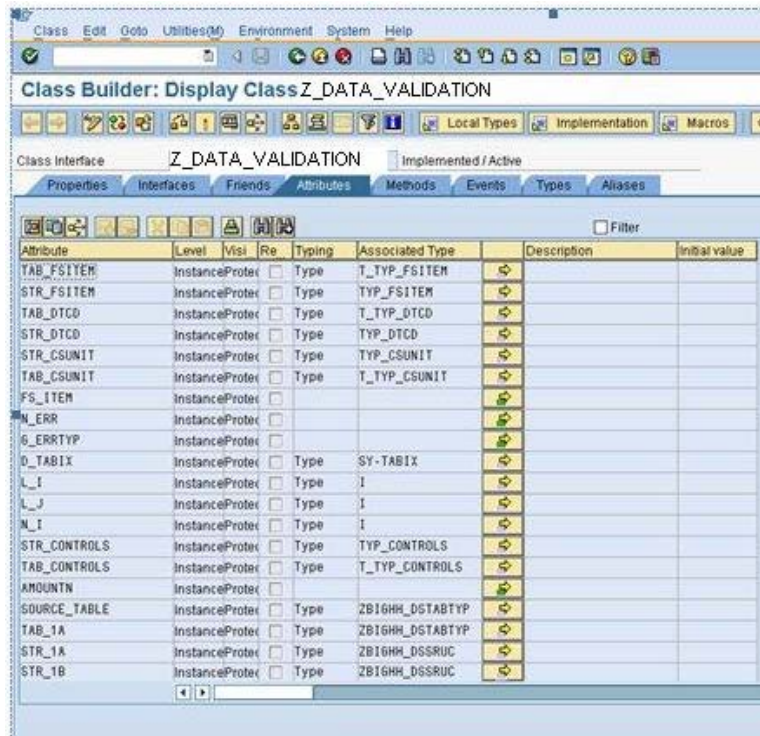
Length: 0 Decimal Places: 0



## Screen 7: DSO table type

### 4.2 Class Attributes

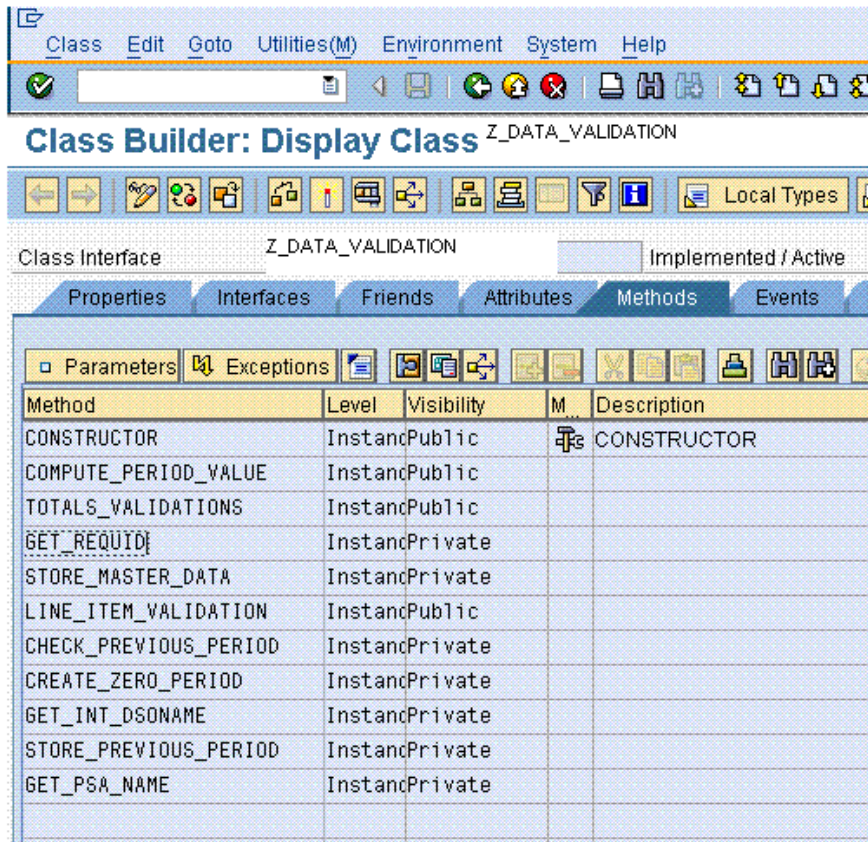
In the Class Object, define your Data objects within Attributes after defining custom local type within Types; this is the Global data that is accessed by the different Methods.



## Screen 8: Class Attributes

### 4.3 Methods

Private Methods are built for processes which are sub-processes(similar to function modules which carry out specific tasks) that can be carried out within the Class.

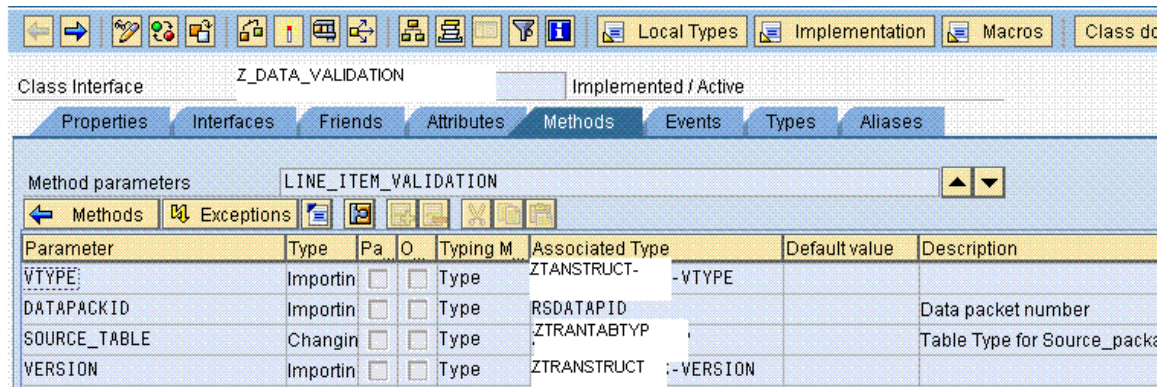


## Screen 9 Class Methods

Public Methods are built for all the processes that are required to be carried out on the Source Package(Start routine) or the Result Package(End Routine)(or even Transfer Routines between fields) from the Transformation. They receive and pass data between the Transformation and the Class.

### 4.4 Parameters

The Public methods use the parameters based on structures and table types as stated in 4.1, for Source Structure and Source package, to pass the contents of the package or a line of the package as required. There can be additional parameters which identify the source and specific characteristics within the transformation. Parameter Types for the public method should be known outside the class in the Data dictionary.



Screen10 Public method Parameters(associated type must be known outside the class)

Parameter types of the Private class may be defined within the class only.

#### 4.5 Brief on methods used in the case example

To get an idea of the advantage and usage of the custom class, here is a brief on the different methods in the example

| Method                | Type of method | Objective  |
|-----------------------|----------------|--|
| CONSTRUCTOR           | Public         | Calls the Private method STORE_MASTER_DATA                                       |
| STORE_MASTER_DATA     | Private        | Stores master data as internal tables  |
| GET_PSA_NAME          | Private        | Gets the PSA table name based on value type & version                            |
| GET_REQUID            | Private        | Gets the max. Request id and max. datapakid of the PSA table                     |
| GET_INT_DSONAME       | Private        | Composes the DSO table name of integration layer based on value type and version |
| LINE_ITEM_VALIDATION  | Public         | Carries out validation at the line item level                                    |
| TOTALS_VALIDATIONS    | Public         | Carries out validation on the totals at the end of the request                   |
| STORE_PREVIOUS_PERIOD | Private        | Stores the previous period cumulative values from the Integration layer          |
| CHECK_PREVIOUS_PERIOD | Private        | Checks if previous period exists in the Integration layer                        |
| COMPUTE_PERIOD_VALUE  | Public         | Computes period value based on cumulative values of current and previous period  |
| CREATE_ZERO_PERIOD    | Private        | Creates zero period values for next year at end of year                          |

Testing: you can debug the class using DTP debugging feature or you may also build a simple Test report program which uses the custom structure and custom table and calls the custom class, to test.

## 5. Conclusion

The Custom class can be used for any Transformation routines. As they are maintained independent of the transformations, any changes to the routines are handled independent of the BI-Workbench build. The structure of the class itself provides considerable self-documentation, if it is split into small self contained private and public methods. For e.g. there can be one class object which manages all transformation routines for a Logistics application; separate methods are built for specific transformations while common methods are used by all transformations within the application.

## Related Content

How to Routines within Transformations

<https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/6090a621-c170-2910-c1ab-d9203321ee19>

[Simulating and Debugging DTP Requests](#)

ABAP Objects

[http://help.sap.com/saphelp\\_470/helpdata/en/d3/2e974d35c511d1829f0000e829fbfe/frameset.htm](http://help.sap.com/saphelp_470/helpdata/en/d3/2e974d35c511d1829f0000e829fbfe/frameset.htm)

For more information, visit the [Business Intelligence homepage](#).

## Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. 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 the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

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