How To...

Routines within Transformations

Version 1.00 – February 2006

Applicable Releases:
SAP NetWeaver 2004s
(BI Capabilities of SAP NetWeaver 2004s)
1 Business Scenario

You have been tasked to manipulate and transform the data flowing through your data warehouse. You were asked to do the following:

1) You need to delete a select number of records within the data package based upon a particular pattern that the records being loaded have:
2) Populate a field based upon values of other fields being passed in the load process.
3) Derive an additional field for the target record structure

In order to accomplish these tasks it has been determined that you need to invoke the use of routines within the transformations of SAP NetWeaver 2004s Business Intelligence.

For purposes of our example, (use of these routines is of course not limited to this particular example) your company is implementing FI-GL and loading data from the FI_GL_1 DataSource in SAP R/3. You are loading this data into the level 1 DataStore Object 0FIGL_O06. See diagram below:

This transformation will be enhanced via routines by performing the following actions:

1) All records that do not have either a value for Debit Postings or Credit Postings will be deleted from the data package in the start routine
2) The Debit/Credit Indicator field in the target structure will be populated in an individual characteristic routine.
3) An additional Plan/Actual field will be populated in the end routine.

2 Introduction

Within the BI Capabilities for SAP NetWeaver 2004s there is a new methodology for manipulating data as it moves through your SAP Business Information Warehouse. This methodology is called Transformations.

Transformations are made up at least one transformation rule.
Transformation rules map any number of source fields to at least one field in the target. You can use different rules types for this.

The different rule types available within a transformation rule are laid out below:
- Constants – A field can be filled directly with a value that is predetermined
- Formula – Formula Builder and Transformation Library
- Reading Master Data – Derive values off of existing master data
- Time Update – When performing a time update, an automatic time conversion and a time distribution are available
- Unit of Measure Conversion / Currency Translation
- Initial – Leave a fields value blank
- **Routines** – Custom code which you have written that will determine based on the programmed logic what the eventual value(s) of a field will be.

In this how-to paper we will be focus on how these routines can be used within our data warehouse to meet different needs.

The different types of routines that are available within Transformations are listed below:

- **The Start Routine**
- Routine for updating Key Figures
- **Routine for updating Characteristics**
- **End Routine**
- Expert Routine

The three transformation routine types **highlighted** above will be those which we will address today in our scenario.

**NOTE:** Routines as they existed in previous releases are no longer subroutines, they are ABAP Objects methods. Similar to the migration from procedural to object-oriented programming seen with the move from CMOD Customer Exits to BaDI’s in SAP BI we are seeing this same move here with the transition from Update/Transfer Rules to Transformations.
3 Step by Step Solution

In order to eliminate all zero debit and credit records coming through in the data package a transformation is needed.

3.1 Create a Transformation

1. Right Click on the target object and select the Create Transformation option.

2. Create the relevant direct mappings by dragging and dropping the source field to their relevant targets.
3. Now save the transformation rule group.

3.2 Create a Start Routine

1. From change mode in the Transformation click on the Create start routine button.

2. From within the start routine there are two sections of code to be filled.
   a. Global Section (optional)
   b. Local Section

Navigate to the Local section

NOTE: Unlike within a subroutine where you would see the fields being passed into and out of the subroutine in the first line of the form declaration, within methods these fields are defined in the definition of the method within this class. (see appendix 4.1 for a detailed description of the start routine definition)
3. Tasked to eliminate all records that have neither a debit nor a credit, the first and only step is insert a delete statement
   - The source package is filtered.
   The Start Routine is now complete

4. Save the Start Routine and enter back into change mode for the Transformation. Save the Transformation as well.
   - There now exists a pencil on the start routine icon which is indicative of the fact that a start routine exists.

3.3 Create a Routine for Updating Characteristics

1. As dictated by the customer the next step is to populate the Debit/Credit indicator with a value of ‘D’ if there is a debit posting on the record and a ‘C’ if there is a credit posting on the record.
   - Right-Click on the Debit/Credit Indicator field within the rule group and click on the Rule Details button.
2. Give a Description to the rule being created.

3. Now the source fields:
   UMSOL – Total Debit Postgs
   UMHAB – Total Credit Postgs
   need to be added to the rule so they can be accessed within the routine. Add the two fields and hit the Green OK button.

**NOTE:** You could also assign the fields to the rule via drawing a link to the rule box in the netgraph UI of the transformations.
4. Within this piece of code the logic needs to be added to derive either a ‘D’ or a ‘C’ for our result field.

**NOTE:** Please see the appendix 4.2 for detailed description of the characteristic routine definition.
5. A conditional statement needs to be created that determines whether the debit posting field has a value (triggering the population of the result with a ‘D’) or the credit posting field has a value (triggering the population of the result with a ‘C’).

- The debit and credit postings are checked for values if the debit posting has a value not equal to zero and the credit posting value is equal to zero we assign the value ‘D’, for debit to the debit/credit indicator.

- On the other hand, if the credit posting’s value is not equal to zero and the debit posting’s value is, the value ‘C’, for credit is assigned to the debit/credit indicator.

6. The last step is to catch an exception if both the credit and debit fields have a value this is an error so a message needs to be written to the monitor and we will raise an exception to stop the load.
7. Save the Characteristic Routine and Transfer the values back to the Rule Group. Save your Transformations.

3.4 Create End Routine

1. The final routine to be created is the end routine, this routine will populate the Plan/Actual Indicator. The routine will read the R/3 value type field and if the value being passed is a 10 (Actual), the value ‘A’ will be assigned to the Plan/Actual indicator. If the value type has the value 20 (Plan), the value ‘P’ will be assigned, otherwise the indicator will remain in its initial state.

Begin by clicking on the create button for the End Routine.

2. The end routine to be populated looks very similar to the start routine (see appendix 4.3 for details of the method definition’s interface). The result_package needs to be looped through where the R/3 value types are either plan (20) or actual (10) value types.
   - The code here is looping through the result_package into the field symbol <result_fields> provided by the method only for records that have the value types 10 or 20.
3. If the value type of a given record is 10 the plan/actual indicator receives the value ‘A’ for actual. If the value type is 20 the value passed to the plan/actual indicator is ‘P’ for plan.

- The conditional case statement inserted evaluates the R/3 value type and based on its value gives the appropriate value to the plan/actual indicator.

4. Save the end routine. Save and activate the Transformation.

3.5 Examine DataSource

1. Before starting the data load to test the new Transformation logic the dataSource should be examined.

- The DataSource has been loaded with a limited set of records. Only records for G/L Account 0000453000 for the year 1998 have been loaded.

- From the DataSource, it can be determined that the records for period 7 through 16 will be deleted. It can also be determined that all of the records are credit postings and are actuals due to the R/3 value type of 10 shown.
3.6 Execute Data Transfer Process & Verify Results

1. The next step is to execute the data package.
   - From within the DataWarehousing Workbench identify the data transfer process created to load the 0FIGL_O06 DataStore Object right click and hit display.
   - Navigate to the execute tab.
   - Choose the execute button.

2. Verify within the Data Transfer Process Monitor that the load into the DataStore Object 0FIGL_O06 was successful.

3. Validate that the data and the transformation logic performed as expected.
   - Six records should have loaded for the first six periods of 1998 for GL/Account 000453000.
   - The six records should all be marked with a Debit/Credit indicator ‘C’
   - All six records should also have a value of ‘A’ for the plan/actual indicator.
4  Appendix A: Example Transformations Code

4.1  START_ROUTINE

INTERFACE

METHODS
    start_routine
IMPORTING
    request                  type rsrequest
    datapackid               type rsdatapid
EXPORTING
    monitor                  type rstr_ty_t_monitors
CHANGING
    SOURCE_PACKAGE              type _ty_t_SC_1
RAISING
    cx_rsrout_abort.

• REQUEST – Request ID of load in progress
• DATAPACKID – data package ID (i.e. 1, 2, 3)
• MONITOR – Messaging mechanism for Transformations, place informational and error messages inside the structure, it will then be displayed from within the monitor.
• SOURCE_PACKAGE – (formerly DATA_PACKAGE) Contains all of the data being passed into the Transformation from the specified source
• CX_RSROUT_ABORT – Class Exception when raised will cause the transformation process to halt.

4.2  ROUTINE FOR UPDATING CHARACTERISTICS

INTERFACE

METHODS
    compute_0FI_DBCRIND
IMPORTING
    request                  type rsrequest
    datapackid               type rsdatapid
    SOURCE_FIELDS              type _ty_s_SC_1
EXPORTING
    RESULT                   type _ty_s_TG_1-FI_DBCRIND
    monitor                  type rstr_ty_t_monitor
RAISING
    cx_rsrout_abort
    cx_rsrout_skip_record.

• REQUEST – Request ID of load in progress
• DATAPACKID – data package ID (i.e. 1, 2, 3)
• SOURCE_FIELDS – Structure containing the values of the field(s) specified as source fields when creating the rule.
• RESULT – Parameter that receives the desired result value for the characteristic.
• MONITOR – Messaging mechanism for Transformations, place informational and error messages inside the structure, it will then be displayed from within the monitor.
• CX_RSROUT_ABORT – Class Exception when raised will cause the transformation process to halt.
- **CX_RSROUT_SKIP_RECORD** – Class Exception when raised will cause the transformation process to skip a record.

### 4.3 END ROUTINE

#### INTERFACE

```plaintext
METHODS
  end_routine
IMPORTING
  request                  type rsrequest
  datapackid               type rsdatapid
EXPORTING
  monitor                  type rstr_ty_t_monitors
CHANGING
  RESULT_PACKAGE              type _ty_t_TG_1
RAISING
  cx_rsrout_abort.
```

- **REQUEST** – Request ID of load in progress
- **DATAPACKID** – data package ID (i.e. 1, 2, 3)
- **MONITOR** – Messaging mechanism for Transformations, place informational and error messages inside the structure, it will then be displayed from within the monitor.
- **RESULT_PACKAGE** – Contains all of the data being passed out of the Transformation after the different Rule Groups have been processed, transforming the data.
- **CX_RSROUT_ABORT** – Class Exception when raised will cause the transformation process to halt.
### Appendix B: Examples of Routines SAP BW 3.x -> BI Capabilities of NetWeaver 2004s

#### BW 3.X

```prog
PROGRAM UPDATE_ROUTINE.
*$*$ begin of global - insert your declaration only below this line "-" * TABLES: ...
* DATA: ...
*$*$ end of global - insert your declaration only before this line *-
```

Here we can see the global data area of the BW 3.X systems. These are simply variables created within the program outside of any forms (subroutines).

#### BI Capabilities of SAP NetWeaver 2004s

```prog
 PROGRAM trans_routine.
*$*$ begin of routine - insert your code only below this line "-" * fill the internal tables "MONITOR" and/or "MONITOR_RECNO", * to make monitor entries
```

- **CLASS routine DEFINITION**
- **PUBLIC SECTION**

```prog
**TYPES:**
** BEGIN OF _ty_s_SC_1,**
* Field: KTOPL Chart of Accts.
  KTOPL TYPE C LENGTH 4,
* Field: SAKNR G/L Account No..
  SAKNR TYPE C LENGTH 10,
* Field: BUKRS Company Code.
  BUKRS TYPE C LENGTH 4,
* Field: GSBER Business Area.
  GSBER TYPE C LENGTH 4,
* Field: WRTTP Value Type.
  WRTTP TYPE N LENGTH 3,
* Field: VERSN Version.
  VERSN TYPE C LENGTH 3,
* Field: FISCPER Fiscal year/period.
  FISCPER TYPE N LENGTH 7,
* Field: FISCVAR Fiscal year variant.
  FISCVAR TYPE C LENGTH 2,
* Field: CURTYPE Currency type.
  CURTYPE TYPE C LENGTH 2,
* Field: CURRENCY Currency.
  CURRENCY TYPE C LENGTH 5,
* Field: UMSOL Total debit postings.
  UMSOL TYPE P LENGTH 9 DECIM
```

- **ALs**
- **KUMSL Accumulated balance.**
  KUMSL TYPE P LENGTH 9 DECIM
- **RECORD Record Number.**
  RECORD TYPE RSARECORD,
- **END OF _ty_s_SC_1.**

**TYPES:**

<table>
<thead>
<tr>
<th>ty_t_SC_1</th>
<th>TYPE STANDARD TABLE OF ty_s_SC_1</th>
</tr>
</thead>
</table>

---

Start Routines
* if abort is not equal zero, the update process will be canceled
   ABORT = 0.

* end of routine - insert your code only before this line

ENDFORM.

The global data area of the new Transformations is a bit different; the data is technically created within the Private Section of the local ‘routine’ class. This does not affect the way someone defines and uses the variables created within the global section. Variables can be used just as they were in SAP BW 3.X.

NOTE:

If converting a 3.X routine to 04s’ and the form (subroutine) has been defined in the global data area, to continue using these subroutines, you can do the following.

1) either convert them to local private methods
2) create a subroutine pool (in se38) and execute these subroutines by using the perform “subroutine” in program “your subroutine pool name”. statement.
3) convert the subroutine logic into a Function Module
Here you can insert your code for the start routine. In order to abort as you would in a BW 3.X system go ahead and instead of changing the variable abort to a non-zero value, you raise the class exception cx_rsrout_abort. For example,

RAISE EXCEPTION TYPE cx_rsrout_abort.

NOTE: To learn more about the RAISE EXCEPTION command reference transaction ABAPHELP

METHOD start_routine.
*==== Segments ===*
FIELD-SYMBOLS:
   <SOURCE_FIELDS>    TYPE _ty_s_SC_1.
*$$ begin of routine -
insert your code only below this line
*--*
... "insert your code here
*$$ end of routine -
insert your code only before this line
*--*
ENDMETHOD.

METHOD inverse_start_routine.
*$$ begin of inverse routine -
insert your code only below this line
*--*
... "insert your code here
*$$ end of inverse routine -
insert your code only before this line
*--*
ENDMETHOD.

ENDCLASS.

Here you can insert your code for the start routine. In order to abort as you would in a BW 3.X system go ahead and instead of changing the variable abort to a non-zero value, you raise the class exception cx_rsrout_abort. For example,

RAISE EXCEPTION TYPE cx_rsrout_abort.

NOTE: To learn more about the RAISE EXCEPTION command reference transaction ABAPHELP

METHOD start_routine.
*==== Segments ===*
FIELD-SYMBOLS:
   <SOURCE_FIELDS>    TYPE _ty_s_SC_1.
*$$ begin of routine -
insert your code only below this line
*--*
... "insert your code here
*$$ end of routine -
insert your code only before this line
*--*
ENDMETHOD.

METHOD inverse_start_routine.
*$$ begin of inverse routine -
insert your code only below this line
*--*
... "insert your code here
*$$ end of inverse routine -
insert your code only before this line
*--*
ENDMETHOD.

ENDCLASS.
PROGRAM trans_routine.

*--------------------------------------------
*-------------------------------*
*       CLASS routine DEFINITION
*--------------------------------------------
*-------------------------------*
*--------------------------------------------
*-------------------------------*
CLASS routine DEFINITION.
PUBLIC SECTION.

TYPES:
BEGIN OF _ty_s_SC_1,
*      Field: KTOPL Chart of Accts.
  KTOPL           TYPE C LENGTH 4,
*      Field: SAKNR G/L Account No..
  SAKNR           TYPE C LENGTH 10,
*      Field: BUKRS Company Code.
  BUKRS           TYPE C LENGTH 4,
*      Field: GSBER Business Area.
  GSBER           TYPE C LENGTH 4,
END   OF _ty_s_SC_1.

BEGIN OF _ty_s_TG_1,
*      InfoObject: ZPLACTUAL Plan/Actual Indicator.
/BIC/ZPLACTUAL           TYPE /BIC/OI,
END   OF _ty_s_TG_1.

PRIVATE SECTION.

TYPE-POOLS: rsd, rstr.

METHODS
  compute_ZPLACTUAL
  IMPORTING
  request                        type rsreq
  uuest                          type rstid
  datapackid                     type rsdat
  apid                          SOURCE_FIELDS type ty_s_SC_1
  _s_SC_1
  EXPORTING
  RESULT                        type ty_s_TG_1-
  /BIC/ZPLACTUAL
  monitor                      type rstr ty_t_monitor
  Raising
  cx_rsargout_abort
  cx_rsargout_skip_record.

In both 3.X and 2004s' to set a value for a given characteristic the value of result needs to be set.

However as in the start routine above in order to skip a line or abort you need to raise exception. For example:

04s – raise exception type cx_rsrout_abort.

To skip a record in 2004s' insert the following:
raise exception type cx_rsrout_skip_record.
METHODS

invent_ZPLACTUAL
IMPORTING
  i_r_selset_outbound TYPE R EF TO cl_rsmds_set
  i_th_fields_outbound TYPE H ASHED TABLE
  i_r_selset_outbound_complete TYPE R EF TO cl_rsmds_set
  i_r_universe_inbound TYPE R EF TO cl_rsmds_universe
CHANGING
  c_r_selset_inbound TYPE R EF TO cl_rsmds_set
  c_th_fields_inbound TYPE H ASHED TABLE
  c_exact TYPE r
  s_bool.
ENDCLASS.

"routine DEFINITION

*--------------------------------------------
-------------------------*
*       CLASS routine IMPLEMENTATION
*--------------------------------------------
-------------------------*
*--------------------------------------------
-------------------------*
CLASS routine IMPLEMENTATION.

METHOD compute_ZPLACTUAL.

DATA:
  MONITOR_REC TYPE rsmonitor.

"insert your code only below this line
... "insert your code here
*--
fill table "MONITOR" with values of structure "MONITOR_REC"
*-- to make monitor entries
... "to cancel the update process
* raise exception type CX_RSROUT_ABORT.
... "to skip a record"
* raise exception type CX_RSROUT_SKIP_RECORD.

* result value of the routine
RESULT = .

"insert your code only before this line
*--
ENDMETHOD.

"compute_ZPLACTUAL

*----------------------------------------------------------*
This subroutine needs to be implemented only for direct access (for better performance) and for the Report/Report Interface (drill through).

The inverse routine should transform a projection and a selection for the target to a projection and a selection for the source, respectively.

If the implementation remains empty all fields are filled and all values are selected.

METHOD invert_ZPLACTUAL.

\$\$ begin of inverse routine - insert your code only below this line\$\$

... "insert your code here

\$\$ end of inverse routine - insert your code only before this line \$\$

ENDMETHOD.  "invert_ZPLACTUAL ENDCLASS.