Modeling Data Transformations in SAP NetWeaver Business Intelligence

(based on SAP NetWeaver 2004 technology)

Version 1.0
August 17, 2006

Table of Contents:

BI Data Transformation

The BI data transformation process enables you to consolidate, cleanse, and integrate data. You can semantically synchronize data from heterogeneous sources. When you load data from one BI object into a further BI object, the data is passed through a BI data transformation, which converts the fields of the source into the format of the target.

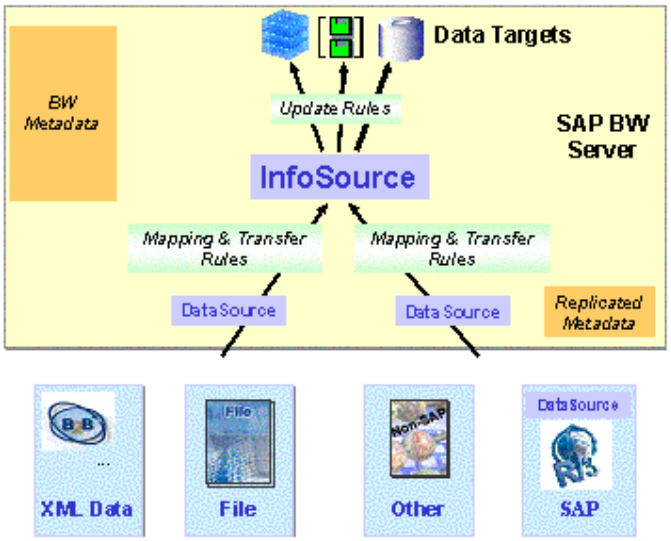

Figure 1: Data flow in BI systems

Up to release SAP NetWeaver 2004 there are two types of BI data transformations displayed in Figure 1:

- Transfer rules

  Transfer rules specify how the data (key figures, time characteristics, characteristics) is transferred to InfoSources from the transfer structure of a DataSource. You are therefore connecting a DataSource with an InfoSource.

- Update rules

  Update rules specify how the data (key figures, time characteristics, characteristics) is updated to data targets (that is InfoProviders) from the communication structure of an InfoSource. You are therefore connecting an InfoSource with an InfoProvider.

  An update rule must be specified for each key figure and the corresponding characteristics of the InfoCube. For a DataStore object (also known as ODS object) it must be specified for the data and key fields, and for an InfoObject it must be specified for the attribute and key fields.

Important Note

As of release SAP NetWeaver 2004s the concept of transfer and update rules is simplified by using a single BI Content object type, simply called *transformation*. There, you create a transformation between a source and a target. The BI objects DataSource, InfoSource, DataStore object, InfoCube, InfoObject and InfoSet serve as source objects. The BI objects InfoSource, InfoObject, DataStore object and InfoCube serve as target objects.

While this document is based on SAP NetWeaver BI 2004, it is nevertheless of general interest because the main BI data modeling ideas remain the same across these two releases. Only Section 4 dealing with routines (for example the start routine) in transformations will be modified when this document is updated for SAP NetWeaver BI 2004s.

1 General Do's and Don'ts for BI Data Transformation

- Generally, a data transformation should be performed within the BI system, if it is not necessary to transform the data already in the backend system.
    - Business logic of the applications and application processes, which is needed for the data extraction process itself, should be implemented in the DataSources.
    - Transformations beside this (that is, the reporting logic) should be done on BI side.
    - Quantity / currency / time conversions for reporting reasons should be done on BI side
- Use the generic transformation features like formula builder, time-, unit- and currency-conversion, master data read. [see section 2 and 3]
- Do not refer to generated DDIC structures like /BIC/*. Use dynamic coding instead. [see section 4.3]
- Do not aggregate or cumulate key figures in data packages of start routines if you can not make sure that all records with identical aggregation keys are in the same data package during the data upload. [see section 4.1]
- In general it is preferable to apply transformations as early as possible in order to reuse the data for several targets. Better use transfer rules (ONE transformation) if you have to transform data to several InfoProviders (the same transformation for EACH InfoProvider).
- Try to avoid data transformation of business relevant data before updates to the (Enterprise) Data Warehouse Layer are performed. This is needed in order to get a 1:1 mapping between backend system and (Enterprise) Data Warehouse Layer.
- Do not add new records to a data package during the update to an InfoProvider. Normally it is better to change the data model to achieve the same results.
- Encapsulate your coding for routines [see section 4.6]

2 Data Transformation – Scenarios

There are three main usages for data transformation in SAP NetWeaver BI:

1. Modifications, enhancements, or grouping of the loaded records before writing them into the InfoProvider which will be used for the reporting afterwards.
2. Conditional updates to the InfoProvider in order to make sure that only needed records will be written to the InfoProvider.
3. Transformation of unit, time, or currency-code related records to a uniform entity, which allows for integrative reporting on all these records.

As an overview of possible transformation scenarios, some of the most popular transformation scenarios will be explained in this section.

2.1 Scenario 1: On-The-Fly Transformation

The easiest way to perform a data transformation during the update of an InfoProvider is to use the standard BI transformation features. You should use these transformations if you would like to perform simple calculation for key figures on record level.

ABAP routines should be used only if the transformation can not be done by using constants, currency/unit conversion or formulas.

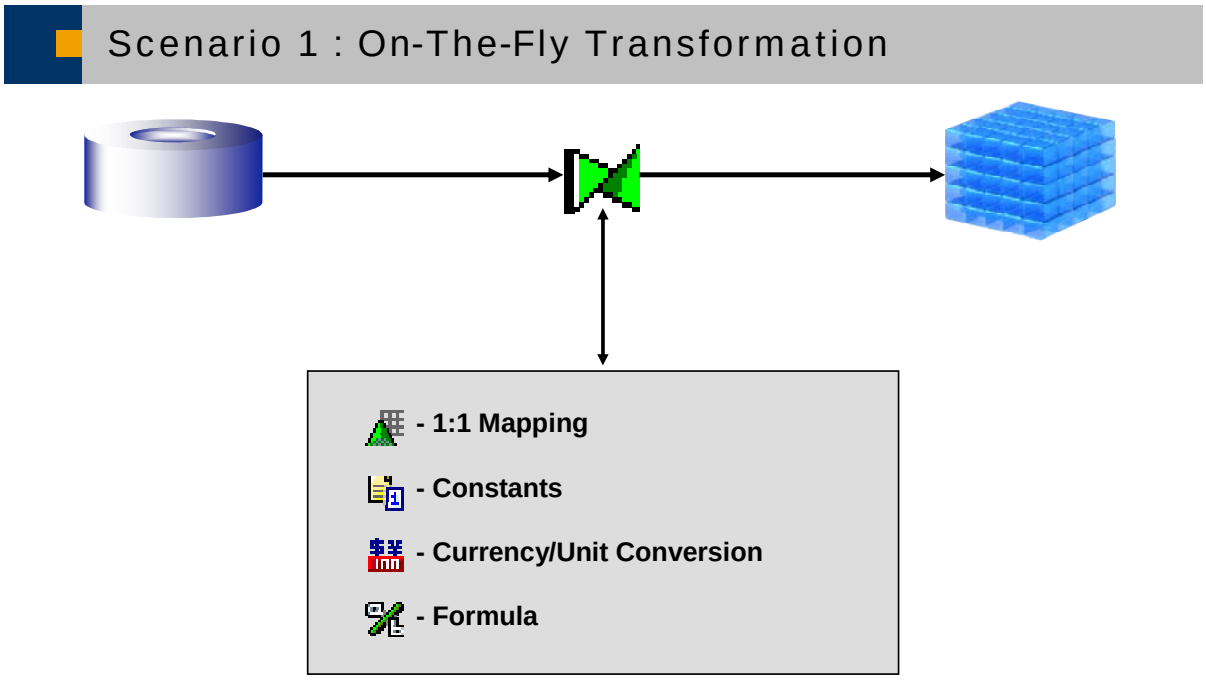

Figure 2: On-the-fly transformation

2.1.1 Constants

If a characteristic or key figure should have a constant value for every record, then it is not necessary to create your own ABAP routine to assign this constant value to the InfoObject. In the transfer rules you have the opportunity to assign constants directly to InfoObjects.

2.1.2 Currency / Unit Conversion

For enterprise wide reporting it is often necessary to adjust reporting objects to uniform units of measure and currency codes. This conversion can be done during data upload into the InfoProvider or during query execution. Please have a look at Section 3 for a detailed explanation of the different conversion possibilities.

2.1.3 Formula

Formulas are the best and easiest way to do on-the-fly transformations of single records for specific key figures or characteristics.

You can assign a formula to an InfoObject and use this formula in the update rules or transfer rules. With the help of the formula builder you can calculate new values for key figures or implement simple conditional updates with IF-statements. For calculation or IF-statements, all values in the fields/InfoObjects that make up the communication structure or transfer structure (if the formula will be created in an update rule or a transfer rule, respectively) can be used.

In the example below, the key figure 0ORA_COUNT gets the value “0”, if 0PSTNG_DATE (Posting Date) is older than 01.01.2000. If not, 0ORA_COUNT gets the value “1” for the current record.

Scenario 1 : On-The-Fly Transformation - Formula

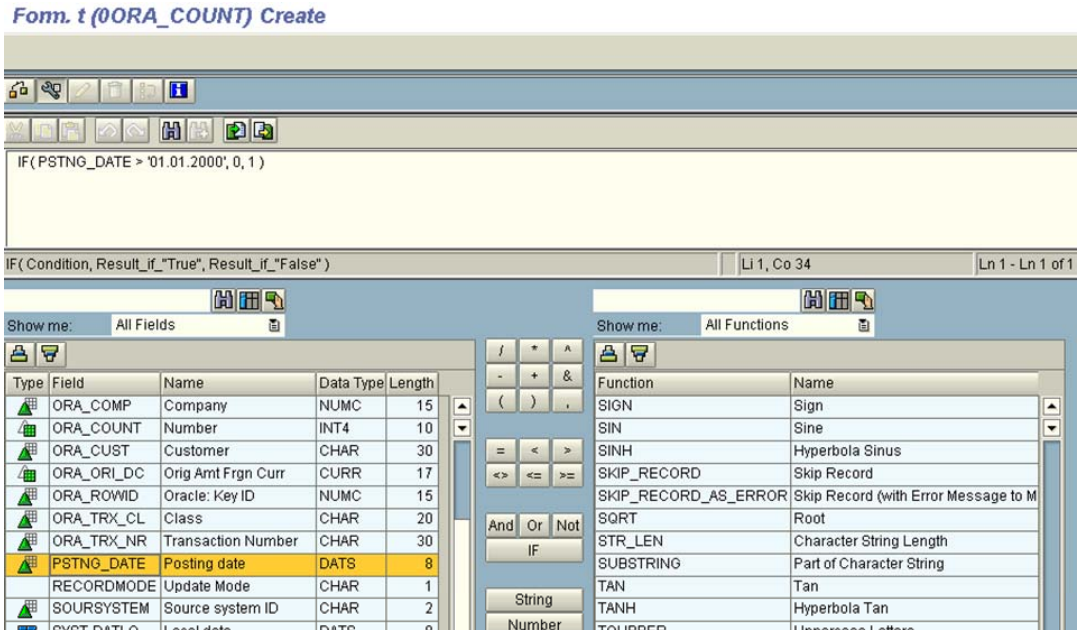

Figure 3: On-the-fly transformation – Formula

Formula versus Routine

Formulas are very easy to read for customers and other SAP colleagues. The transformation process is more transparent to the power user.

2.2 Scenario 2: Data Enrichment – Cross References

Sometimes not all of the values needed for a data update to an InfoProvider are provided by the communication structure, but the values do already exist in the BI system.

If so, it is possible to enrich the values coming from the communication structure by reading the necessary values from other InfoProviders or master data tables.

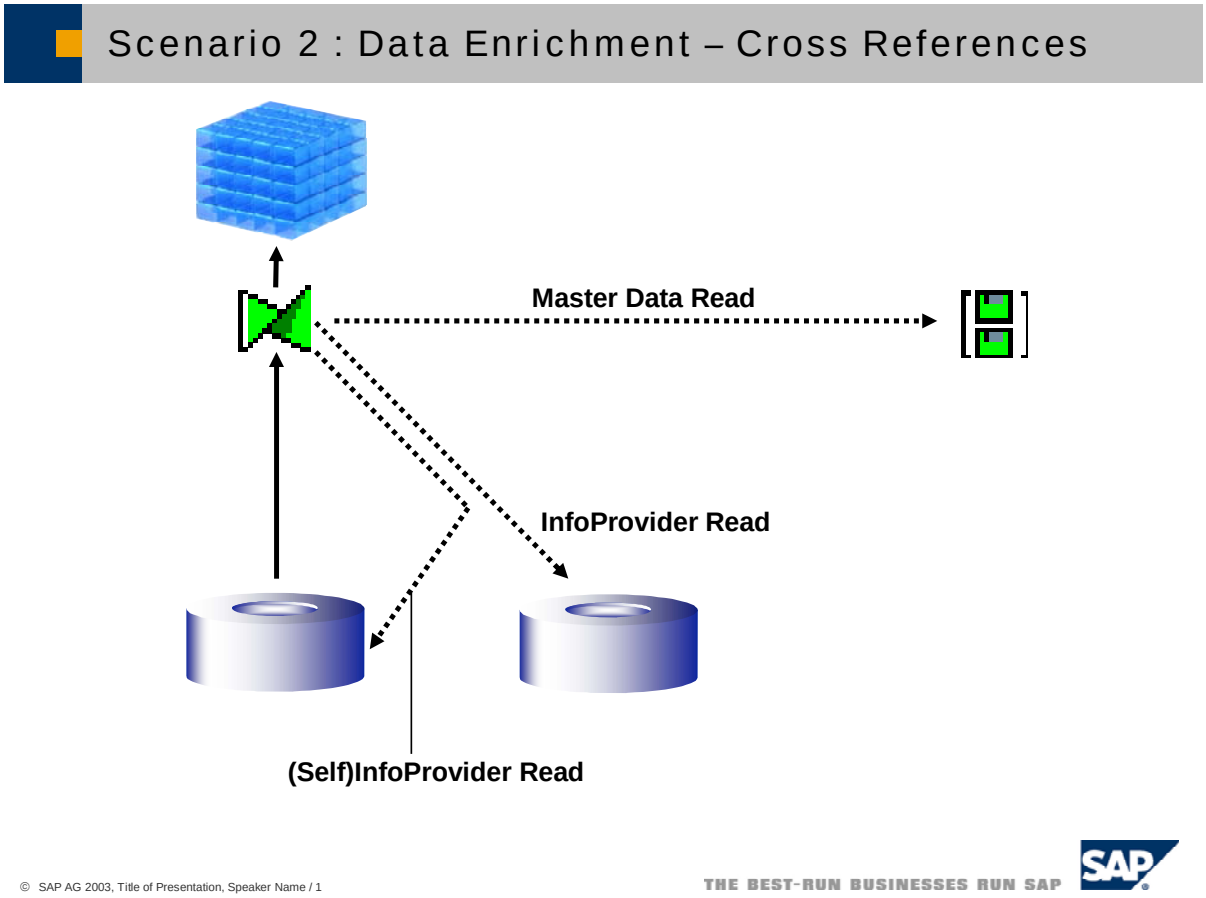

Figure 4: Data Enrichment - Cross References

Performance Issues

You should remember that data load performance will decrease if you cross reference data from other InfoProviders during the upload process.

For example, if a simple database read such as:

```
Select single FIELD from DB_TABLE into WA where DB_FIELD1="XYZ"
```

takes 10 milliseconds, then the data upload time will increase by 16 minutes for a full upload of 100.000 records and it goes from bad to worse if you must upload 1 million records. See the table below.

db read time for single record in ms	count of records	extra upload time in minutes

10	50000	8,33
10	100000	16,67
10	500000	83,33
10	1000000	166,67

This of course is a rough example that does not take buffering, indexes, or other considerations into account. Still, you can see the potential impact that cross-references can have on upload performance.

2.2.1 Master Data Read

A simple master data read should be done using generic BI functionality in the update rules. Choose the option “Master Data Attribute of” in the update rule details for the selected characteristic and select an attribute by using the F4 help. With this setting, the value of the characteristic to be updated is read from the master data table of the specified characteristic.

Example:

- Characteristic *Material* is provided by the InfoSource.
- Characteristic *Material Group* is not provided by the InfoSource.
- Characteristic *Material Group* is an attribute of characteristic *Material*.

Therefore you can fill *Material Group* from the master data table of *Material*: Select the method “Master Data Attribute from” and *Material* as source.

The generic master data read only works for characteristics that are available in the communication structure.

For a more complex master data read or if the characteristic is not available in the communication structure, you have to implement the master data read yourself in a routine for the specific characteristic. If so, the function module RSAU_READ_MASTER_DATA should be used to read the master data.

2.2.2 InfoProvider Read

There is no generic BI functionality for a read from an InfoProvider. Therefore you have to implement it yourself in the update rules in the routine of the specific InfoObject. Normally the function module RSDRI_INFOPROV_READ should be used to read data from InfoProviders.

It is often quicker and easier to implement a select to the corresponding DataStore object table instead of using the function module, especially when reading from DataStore objects (formerly known as ODS objects). The DataStore object table is generated if the DataStore object is created. The name of the DataStore object table follows a naming convention. For DataStore object *0SRPO_D1* the name of the corresponding table is */BI0/ASRPO_D100*.

/BI0/A = Prefix for generated DataStore object tables
SRPO_D1 = Name of the DataStore object without leading zero
00 = Suffix for active records (suffix 40 and 50 inactive and change log are also available)

Note:

It is not recommended to program a select statement directly with regard to the generated DataStore object tables, because these DDIC objects exist only after BI Content activation. The select must be implemented with dynamic coding. Please see Section 4.3 for details.

Please make sure that you read the active data of the InfoProvider and master data with each read. Otherwise you will run into data inconsistency problems.

2.3 Scenario 3: Conditional Update

You program a conditional update if you want to filter out data records that fulfill certain conditions before you perform the upload into the final InfoProvider. This filter must be implemented manually in the start routine of the update rule.

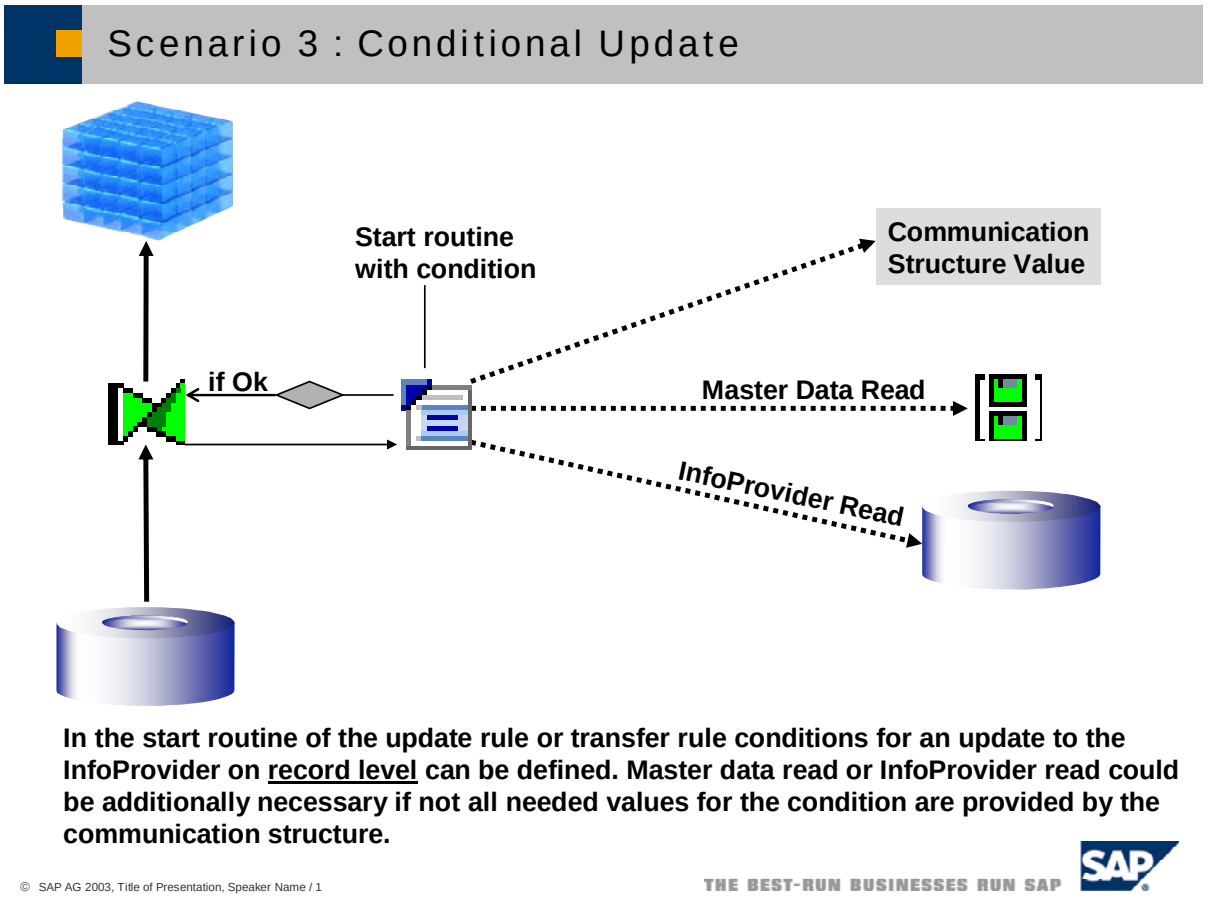

Figure 5: Conditional Update

2.3.1 Delete Data Package

If records fulfilling certain conditions should not be updated to the final InfoProvider at all, the records can be deleted in the update rule. The deletion on record level must be done in the start routine of the update rule. A conditional update by deleting records could look like this:

```
Loop at DATA_PACKAGE.

  l_days_open = sy-datum - DATA_PACKAGE-PLDDELDATE.

    If DATA_PACKAGE-STSPP_COMP = 'X' or l_days_open <= 0.
      delete DATA_PACKAGE.
    Endif.

Endloop.
```

All records of the current data package are stored in the internal table DATA_PACKAGE. This is done automatically by the update rule framework. The table structure is equal to the communication structure. During the loop through all records of the current data package the condition can be defined. If the condition is fulfilled the current record will be deleted out of the internal table DATA_PACKAGE.

An update to the InfoProvider will be performed for the records in the internal table DATA_PACKAGE only. This is an easy example for a conditional update. Please have a look at Section 4 for more complex examples.

2.3.2 Record mode

Apart from deleting records in the update rules it could make sense to manipulate the value of the characteristic 0RECORDMODE. The record mode will be taken into account during the data activation in a DataStore object. Depending on the value of 0RECORDMODE, values will be added, deleted, or cumulated in the DataStore object.

By using the different recordmode types you can influence the update behavior as follows.

' ': The record delivers an after image.

> The status is transferred after something is changed or added. You can update the record into an InfoCube only if the corresponding before image exists in the request.

'**X**': The record delivers a before image

> The record reflects the status before the data change.
> All record attributes that can be aggregated have to be transferred with a reverse +/- sign. The reversal of the sign is carried out either by the extractor (default) or the Service API. In this case, the indicator 'Field is inverted in the cancellation field' must be set for the relevant extraction structure field in the DataSource.
> These records are ignored if the update is a non-additive update of a DataStore object.
> The before image is complementary to the after image.

'**D**': The record has to be deleted.

> Only the key is transferred. This record (and its DataSource) can only be updated into a DataStore object.

'**R**': The record delivers a reverse image.

> The content of this record is the same as the content of a before image. The only difference is with a DataStore object update: Existing records with the same key are deleted.

A conditional update by using the Recordmode could look like this.

```
LOOP AT DATA_PACKAGE.
  IF DATA_PACKAGE-lw_gists EQ 'C' OR DATA_PACKAGE-sched_del EQ 'X'.
    MOVE 'R' TO DATA_PACKAGE-recordmode.
    MODIFY DATA_PACKAGE.
  ENDIF.
ENDLOOP.
```

3 Conversion of Currencies and Units of Measure

3.1 Currency Conversion

In SAP systems, every amount key figure is stored with respect to a currency key field. There are several types of currency keys on OLTP data. The two most important currency types are the document (or transaction) currency (BI InfoObject 0DOC_CURRCY) and the local (or company code) currency (BI InfoObject 0LOC_CURRCY. The document currency is entered for each posting transaction of an OLTP document-type record (for example financial documents), whereas the local currency is uniquely assigned to a company code when the OLTP system is customized.

So, the local currency can be used as a common currency for cross-application BI reporting (that is a common reporting currency). Besides the **document currency**, every record within the final InfoProviders of the BI system should contain the **local currency** and the corresponding amounts with respect to the local currency.

There are different scenarios for the currency conversion of source system data:

1. **No conversion necessary**

   The source system tables contain amounts with respect to both currency types (document and local currency). The currency conversion was already done at the time when the document was posted into the database of the OLTP system. In this case, the extractor should read local and document currencies (amounts and currency keys) and transfer them to the extract structure of the DataSource. No further currency conversion to local currency should take place neither in the OLTP extractor nor in the BI update rules.

   This is the case for FI / CO documents.

2. **Currency conversion in BI update rules**

   If the local currency (amounts and currency key) cannot be provided by the DataSource from OLTP tables, the currency conversion has to take place at data upload time. Currency conversion in update rules provides the option of translating data records from the source currency (document currency) of the InfoSource into a target currency (local currency) in the InfoProviders.

   The currency conversion in a BI update rule uses a pre-defined currency conversion type of the BI Content or it uses hard-coded routines, if necessary.

   For DataStore objects it is not possible to execute the currency conversion by using predefined currency conversion types. Instead, you have to revert to routines.

   Example for update rule

   Source InfoObjects:
   - 0DEB_CRE_DC Amount in Document Currency
   - 0DOC_CURRCY Document Currency

   Target InfoObjects:
   - 0DEB_CRE_LC Amount in Local Currency
   - 0LOC_CURRCY Local Currency

   Currency translation type: Average Rate for Current Date

Currency Conversion in Update Rules (1)

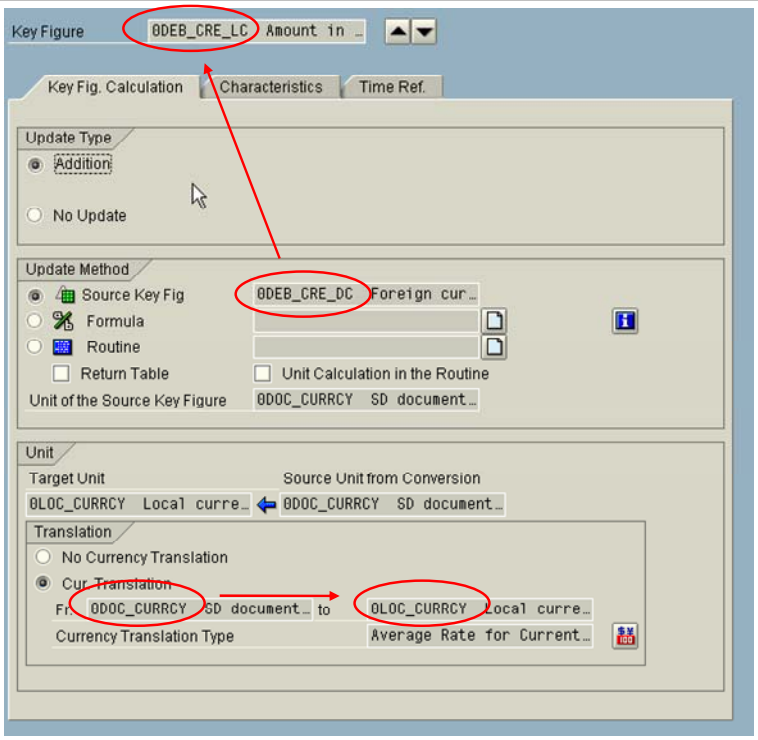

Figure 6: Currency conversion (1)

The currency conversion to the target currency type *Local Currency* in BI update rules can only be performed if the communication structure of the source data (InfoSource or export DataSource of an InfoProvider) contains the local currency (InfoObject 0LOC_CURRCY). This InfoObject 0LOC_CURRCY has to be filled in a previous data staging step (for example the transfer rules of an InfoSource, update rules of an inbound ODS object) by reading the local currency information of the corresponding company code. That means the attribute *Local Currency* of the InfoObject *Company Code* has to be read for each InfoSource record at the previous staging step. This master data loop-up is a standard update method for BI update rules.

Example

InfoObject *Company Code*:	0ORA_COMP
Attribute *Local Currency* of 0ORA_COMP:	0LOC_CURRCY

Currency Conversion in Update Rules (2)

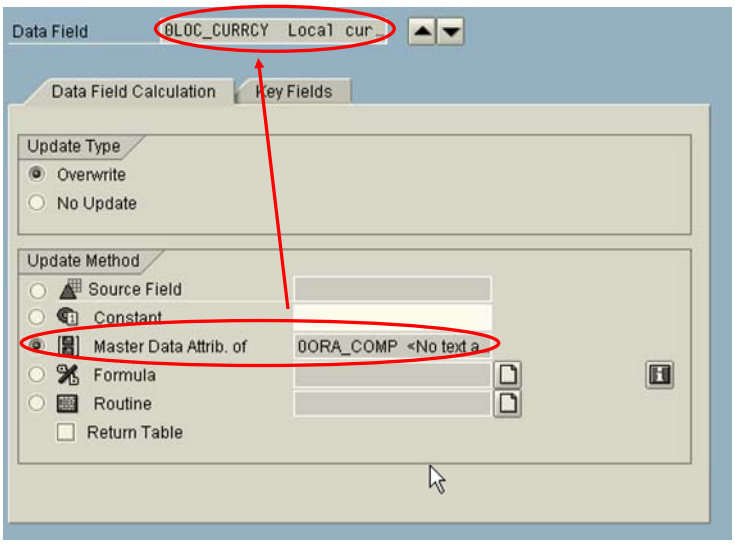

Figure 7: Currency conversion

In conclusion, the currency conversion from document currency to local currency (amounts and currency key) in BI update rules is a two-step procedure:

Currency Conversion in Update Rules (3)

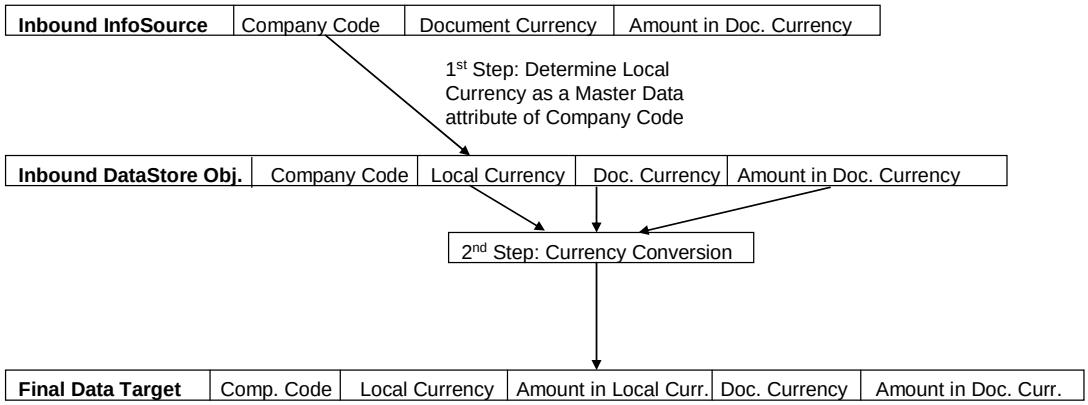

Figure 8: Currency conversion (3)

Currency translation types

If the currency conversion from document currency to local currency is performed at data upload time, it is important to choose an appropriate currency translation type. Especially, the time reference should be set variable with respect to the date, when the document was posted into the database of the OLTP system. For this purpose, use the InfoObjects *Posting Date* ore *Transaction Date*, which are common InfoObjects in document-type InfoSources.

You can create/change/display Currency Translation Types in your BI system using the transaction RRC1/RRC2/RRC3.

Example: Conversion Type 0CPPURPODA: *Conversion to 0PSTNG_DATE*

- Exchange Rate Type: M (Standard translation at average rate)
- Source Currency from Data Record
- Selection of Target Currency within Translation
- Time Reference

The following currency translation types are delivered with SAP standard BI Content:

- 0MEANTODAY Average rate using actual date (0SY-DATUM)
- 0MEANDAILY Average rate using 0CALDAY of the data record
- 0TRA_DATE Average rate using 0TRANS_DATE of the data record
- 0PSTNG_DAT Average rate using 0PSTNG_DATE of the data record
- 0DOC_DATE Average rate using 0DOC_DATE of the data record

Currency Conversion in Update Rules (4)

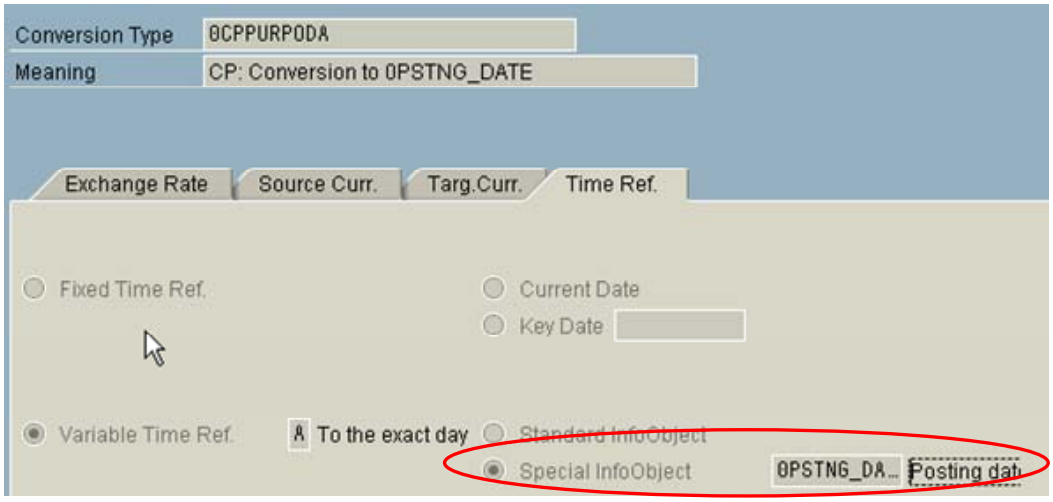

Figure 9: Currency conversion (4)

3. Currency conversion at query runtime

If you want to convert the currency of key figures at BI query runtime, you can determine the currency conversion type at two points of time:

- **Currency conversion defined at BI query design time**
  Complex currency conversion is available in the query definition. A currency translation type and a target currency can be entered per query element. Depending on the elements and the selected translation, the values for the query can be available in different currencies.

  You can also define the currency conversion with a **variable**. When you execute the query, the system gets the variables for the target currency.

- **Ad-hoc currency conversion executed at BI query runtime**
  Simple ad-hoc currency conversions are also available that do not have to be defined in the query designer. You can access it via the context menu in the BEx Analyzer. Using a pre-defined currency conversion type, all values for a query are translated into a common target currency**.** The exchange rate is automatically created using the selected currency conversion type.

Recommendations

- The currency conversion at query runtime can only serve for display information. Data integration with respect to the currency should be done during BI data staging / modeling using the common local currency derived from the company code.
- For each record uploaded into a final InfoProvider of the BI system, provide both currency types (amounts and currency key): **document currency and local currency**.

  The document currency displays the amount, which was originally posted into the database of the OLTP system, whereas the local currency (with respect to the company code) can be used to integrate the data from different applications.

- If possible, extract both currency types (amounts and currency key) directly from the corresponding fields of the OLTP tables.
- The second best way is to convert from document to local currency within the BI update rules. Use an appropriate BI currency translation type with variable time reference. The referenced InfoObject should contain the posting date or transaction date of each record.

3.2 Unit of measure conversion

Much like amount values, quantity key figures in the SAP system are stored with respect to a unit of measure (UoM). There are several types of UoM on the OLTP data. The most important UoM is the **base unit of measure** (BI InfoObject 0BASE_UOM). In the OLTP system, the base UoM is used to **manage the stock of a material**. In Inventory Management, the base unit of measure is the same as the stock keeping unit.

The OLTP system converts all quantities entered in other units of measure (alternative units of measure) to the base UoM. The base unit of measure is the unit satisfying the highest necessary requirement for precision.

So, the base unit of measure can be used as a common unit of measure for cross-application BI reporting. Besides alternative units of measure, every record within the data targets of the BI system should contain the **base UoM** and the corresponding quantities with respect to the base UoM.

Note

In contrast to currency conversion, it is not possible to define UoM translation types in the BI system that could be used in update rules or at query runtime.

Recommendations

- Material quantities should be extracted by the DataSource with respect to the base UoM (quantity and UoM key).
- If the source system tables contain only alternative UoM, the unit conversion to the base UoM should be performed by the extraction function module using the base UoM of the material master data.
- The SAP standard function modules for UoM conversions are bundled in function group SCV0.

4 Programming Routines

Routines should be used if the standard BI transformation opportunities are not sufficient.

The ABAP coding of the routines is integrated into the generated report that is responsible for the data upload to the InfoProviders. The routine itself is a BI Content object stored in tables (RSAABAP). The routine has A, M and D version like other BI Content objects as well. Routines can be assigned to update rules, transfer rules, and InfoObjects. Moreover you have to differentiate between start routines and standard routines.

Figure 10 presents the data model for routines in BI. The relationship between update rules, transfer rules and routines is represented by database tables for metadata and mapping information.

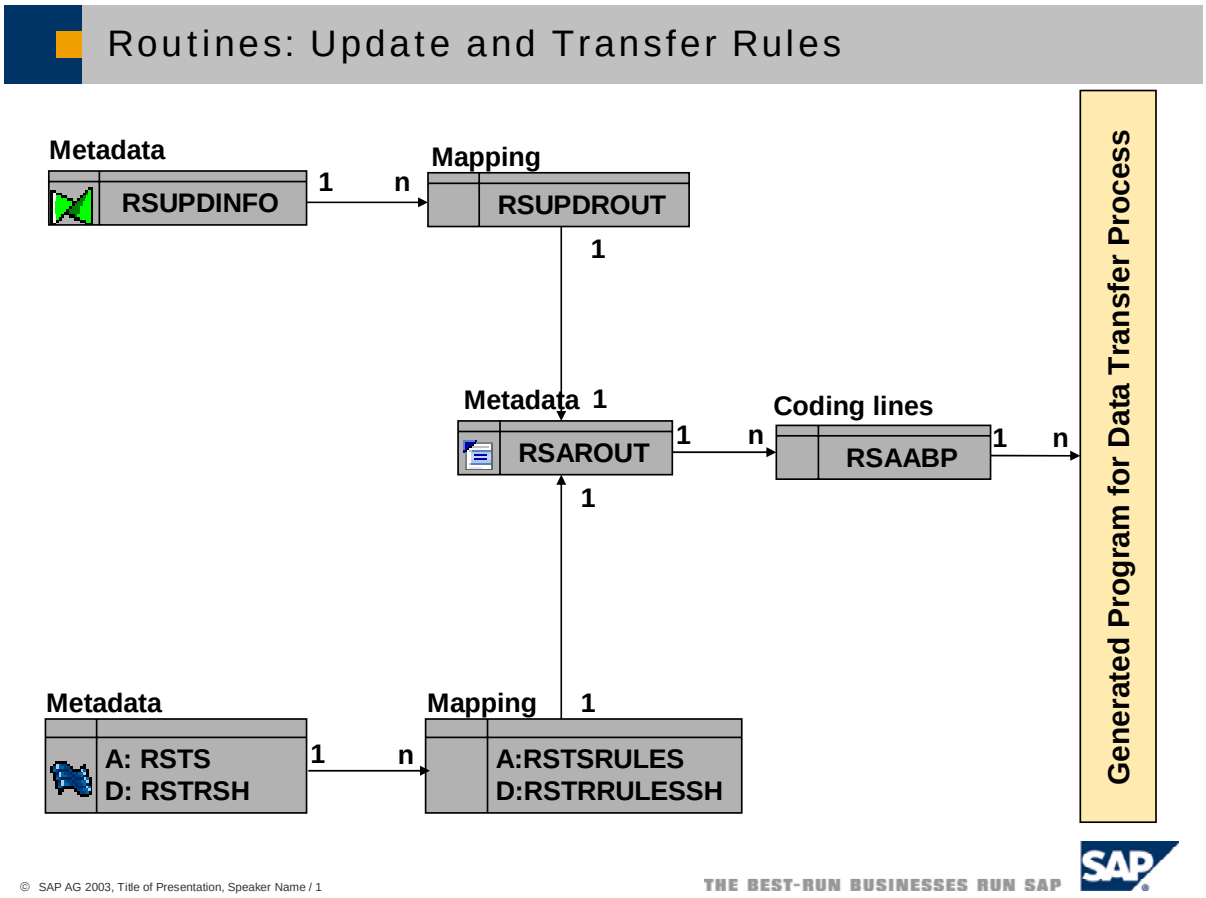

Figure 10: Routines

4.1 Start Routines

Start Routines can be created for update rules and transfer rules. During the data transfer process the coding in the start routine will be called for each data package at the beginning of the update or transfer rule processing. In the start routine you have access to all records of an entire data package.

The data package size will be determined at runtime and can not be influenced. Therefore the actual number of records in a data package and the number of data packages itself varies. Therefore it is not possible to calculate new key figure values using aggregation/summation of key figures in the data package, because you never know if other records for the same aggregation level will be uploaded by one of the following data packages.

4.2 Standard Routines

In standard routines you have access to a single, currently processed record which has the work area `COMM_STRUCTURE`. This routine will be processed for each record after the start routine and is assigned to a single key figure or characteristic. You can assign new values to the key figure or characteristic by changing the value of `RESULT`, at the end of the routine.

4.3 Programming DDIC references

References to generated DDIC structures of BI Objects (DataStore objects, InfoCubes, or InfoObjects) are very often required by BI Content developers. Unfortunately it is not advisable to refer to the generated DDIC structure directly, like

```
DATA: WA_DATA_STORE type /BI0/ACRM_CASE00.
```

In this example the structure `/BI0/ACRM_CASE00` will be generated during the content activation of the DataStore object 0CRM_CASE. If you would make a reference to the generated structure and the content installation of your routine will be done before the installation of 0CRM_CASE, the DDIC structure is missing and the content installation of your routine fails due to syntax checks.

There are several possibilities to bypass this problem. Some of them will be explained in the next subsections.

4.3.1 Generic Structures and Tables

There are already some generic structures and tables which can be used in routines.

- **`COMM_STRUCTURE`** is a work area of the type of the communication structure and can be used in the routines for update rules. `It` is filled with the values of the currently uploaded record.
- **`DATA_PACKAGE`** is an internal table of the type of the communication structure and includes all records of the current data package. The internal table can be used in start routines of update rules in order to loop through all records in the package.

  Example:

  ```
  LOOP AT DATA_PACKAGE.
        DATA_PACKAGE-Field = 'A'.
  ENDLOOP
  ```
- **`DATAPAK`**: Same as `DATA_PACKAGE`, but in the transfer rules
- **`TRANSFER_STRUCTURE`** represents the transfer structures in the routines for transfer rules.
- **`MONITOR`** is a table of the type `RSMONITOR` and should be used to add messages to the loading monitor.

4.3.2 Includes

The usage of includes is quite common and recommended. You add the include program to the global declaration part of your routine.

```
PROGRAM UPDATE_ROUTINE.
*$*$ begin of global - insert your declaration only below this line  *-*
INCLUDE rsbctsrm_ve.
```

In the implementation part of the routine you are now able to refer to subroutines of the include.

```
*$*$ begin of routine - insert your code only below this line        *-*
* fill the internal tables "MONITOR" and/or "MONITOR_RECNO",
* to make monitor entries
  l_bustyp = 'BUS2206'.

  PERFORM ve_determine_survey_vl
  TABLES   MONITOR
           DATA_PACKAGE
  USING    l_bustyp
  CHANGING ABORT.
```

There are three main advantages of using includes.

1. Includes can be reused in other routines as well. Therefore the coding must be maintained only once.
2. Subroutines which are used in the same context can be grouped together in the same include.
3. Includes will not be taken into account during the syntax check if there is no reference to other reports. Therefore references to DDIC structures like

   ```
   DATA: WA_DATA_STORE type /BI0/ACRM_CASE00.
   ```

   are allowed in includes.

   If you use DDIC references to generated objects directly in routines you have to be sure that the DDIC objects are already generated during the content activation/installation. Otherwise you will get an activation error (that is a syntax check error). On the other hand, if you use DDIC references in an include program you have only assure that the DDIC object is generated later at data loading time.

4.3.3 Type Pools

Type pools should be used if there are references to the same DDIC structure in different routines.

The idea is to define the name of the structure in the type pool as a constant. Even where-clauses for select statements can be stored here as a constant and can be reused for several routines.

Example

- Definition of constants in the type pool:

```
TYPE-POOL ZMTES.

CONSTANTS:
* Name of the DataStore object table with active data
ZMTES_C_TABLE_/BI0/ACRMBPKPI00 TYPE SOBJ_NAME VALUE ‘BI0/ACRMBKPI00’,

* where-clause for the select statement in update rule
ZMTES_C_WHERE_CLAUSE TYPE STRING
           VALUE ‘BPARTNER = DATA_PACKAGE-BPARTNER.
```

- Usage of constants in routines

```
PROGRAM UPDATE_ROUTINE.
*$*$ begin of global - insert your declaration only below this line  *-*
* TABLES: ...

TYPE-POOLS: ZMTES
DATA:  l_s_CRM_LST_DT type d.

*$*$ end of global - insert your declaration only before this line   *-*


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

Loop at DATA_PACKAGE

* get the value of CRM_LST_DT for the business partner value of the
* currently processed record.
Select single CRM_LST_DT from
(ZMTES_C_TABLE_/BI0/ACRMBPKPI00) into l_s_CRM_LST_DT
    where (ZMTES_C_WHERE_CLAUSE).

* Do something with the result in l_s_CRM_LST_DT
...
```

Definition in the global part of the routine

Implementation part of the routine

```
...
ENDLOOP.

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

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

4.3.4 Dynamic Code

An alternative for using includes is to program dynamic references to the generated DDIC structures and tables. You may use the following example as a template for various tasks that you need to perform.

Example: generated DDIC objects with regarding the SAP standard InfoCube 0FIGL_C01

```
*.. data definitions for internal table and workarea
data: lt_data type ref to data,
      ls_data type ref to data.
field-symbols: <t_data> type standard table,
               <s_data> type any.

*.. store name of structure (infocube 0FIGL_C01) as string
constants: c_tabname type rstlogotab value '/BI0/V0FIGL_C012'.
*.. store components of structure as string
constants: c_gl_account type rsd_iobjnm value '0GL_ACCOUNT',
           c_chrt_accts type rsd_iobjnm value '0CHRT_ACCTS',
           c_fiscper    type rsd_iobjnm value '0FISCPER'.
*.. field symbols for each component of structure
field-symbols: <gl_account>,
               <chrt_accts>,
               <fiscper>.

*.. create internal table and assign it to <t_data>
create data lt_data type table of (c_tabname).
assign lt_data->* to <t_data>.
*.. create workarea and assign it to <s_data>
create data ls_data type          (c_tabname).
assign ls_data->* to <s_data>.
```

4.4 Monitor Messages

The data load monitor of the BI administrator workbench displays the current status of data staging into the BI system. The monitor should also get information from the transformation routines. For that to happen, you have to write messages to the data loading monitor from your transformation routines.

In update rules you can use the table MONITOR to add messages to data load monitor. This table is of type RSMONITOR. Messages should be added to the monitor in case of errors or to give a status update in case of long data loading time.

Do not overload the user with needless messages.

4.5 Error Handling

Errors can be reported by the update rules using the return code and abort variables.

Value	**Result when used in characteristic (target: InfoCube) or key field (target: DataStore object)**	**Result when used in fey figure (target: InfoCube) or data field (target: DataStore object)**
ReturnCode <> 0	Entire record ignored, next record processed.	Processing of key figures terminated. **Corresponding key figure initialized and returned**.

Abort <> 0	Load process terminated.	Load process terminated.

Note:

Setting ReturnCode to a non-zero value in update routines of key figures / data fields with update mode Maximum, Minimum, or Overwrite is not recommended.

This may result in the incorrect deletion of correct (existing) data. In case of update mode Maximum this problem will only occur if the key figure values can be negative. Updates of type 'Addition' should normally cause no problem, since the initial value (e.g. 0) will be added to the existing value.

4.6 Encapsulate the coding for BI routines

Reusable ABAP coding for data staging is grouped by SAP developers in includes or classes. The include name consists of RS_BCT_ + application + UPDATE + Scenario Name or InfoProvider Name. Global data definition should be done in type pools.

5 Performance Aspects Related to BI Data Transformation

- In general it is preferable to apply transformations as early as possible in order to reuse the data for several targets. It is better to use transfer rules (ONE transformation) if you have to transform data to several InfoProviders (the same transformation for EACH InfoProvider). Technical transformations (check for right domain etc.) could even be done outside BI (e.g., in the extractor of the DataSource).
- Be sure that ABAP-coded routines are designed and implemented very carefully (for example: avoid nested loops, buffer database tables). If you discover a lot of time spent in transformation rules (e.g., via the Data Load Monitor), check and improve the coding.

6 List of documents related to BI Data Transformation

Currency Conversion

- Currency Translation in Update Rules → See Online Documentation
  http://help.sap.com/saphelp_bw32/helpdata/en/80/1a6507e07211d2acb80000e829fbfe/content.htm
- Currency Translation in the Business Explorer (at Query Runtime) → See Online Documentation
  http://help.sap.com/saphelp_bw32/helpdata/en/80/1a682de07211d2acb80000e829fbfe/content.htm

Debugging

- Debugging of Routines → See Online Documentation
  http://help.sap.com/saphelp_nw04/helpdata/en/cd/be623cbc69c613e10000000a114084/frameset.htm