S14 - BW Modeling - DSO & Transformations (AMDP) - Part 1

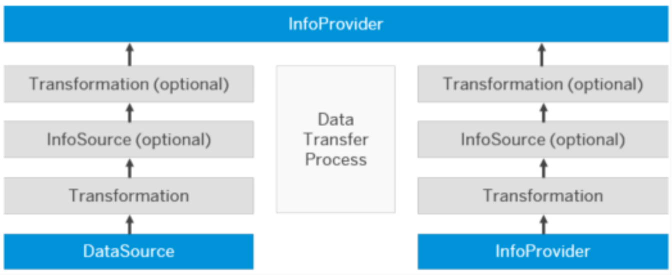
ADSO & Transformations (AMDP)	1. Transformations in BW4HANA. 2. DTP in BW4HANA 3. Creating LSA++ flow for Header and Item. Header from stnd DS and Item from CDS View DS. 4. Enhance CDS view for item for material type. 5. Crate Infosource for Header and Item 6. Create transformation (1st level) 7. Create transformation (2nd level) 8. Create lookups 9. Create formulae 10. Create AMDP based Start routine for removing special characters. 11. Create AMDP based field routine for quantity conversion from sales unit to base unit (0QUANT_B) 12. Create ABAP based end routine to populate update date.(0UPD_DATE) 13. Debug AMDP transformation. 14. Test the overall flow. 15. Post Q & A	19th Sep: 8:30 AM - 10:30 AM

Q & A - Pre Session

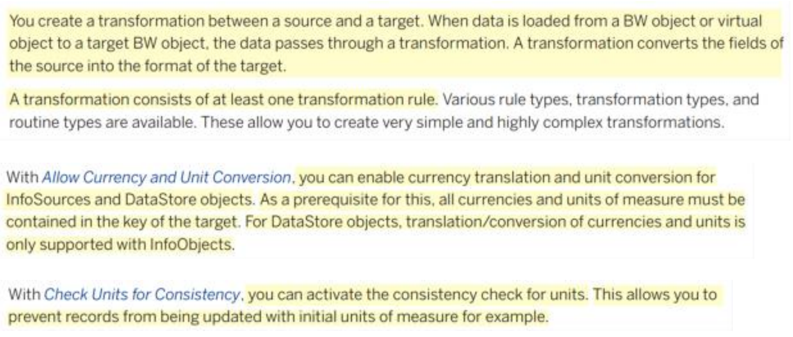
Shakthi	How field based ADSO are faster in loading compared to Info object based ADSO How and when the external SAP HANA views generated using ADSO creation are used? How to enable analytical privileges for SAP HANA view What are SPOs. How are we replacing them in BW4HANA. Any real time example? Can we see one example of Direct Update ADSO with API and HAP each. Any real time scenario? What are data slices and how do we define them? Any example Write interface enabled - any real time example?
Srinu	- Should I create two DTP's or only one, if one which one I need to create? (please see screenshot below) - Which DTP's are needed in Process Chain? - Why InfoSource is needed between CM & Std ADSO? - Look up logic in Integration ADSO written from IS to Std ADSO or CM ADSO to Std ADSO?

Transformations in B4H

Positioning



General Tab



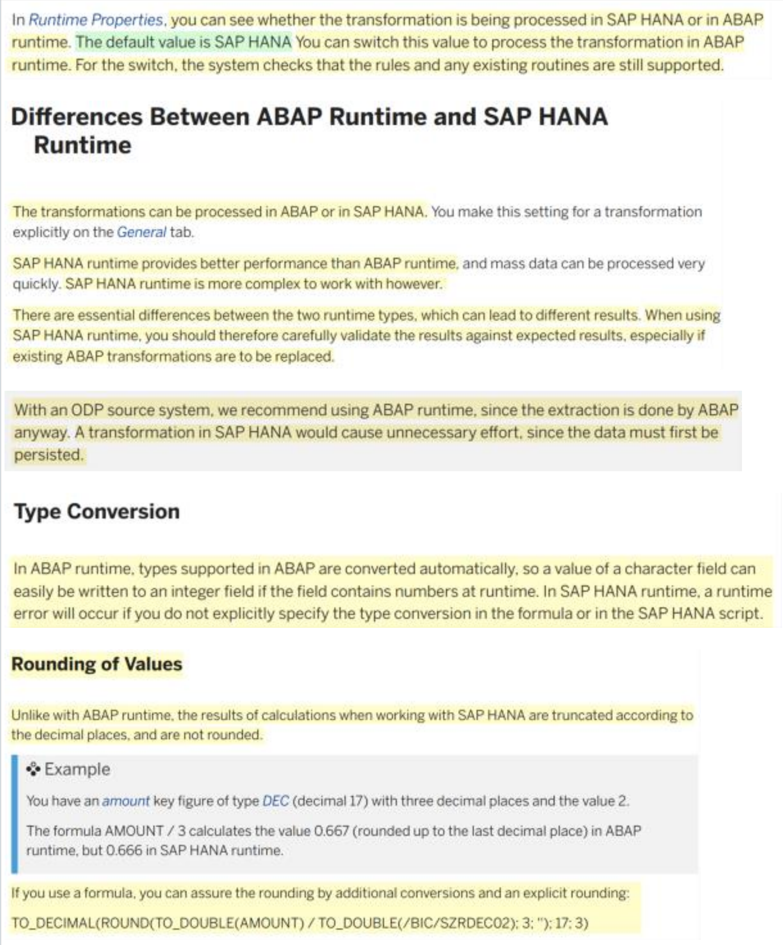
You create a transformation between a source and a target. When data is loaded from a BW object or virtual object to a target BW object, the data passes through a transformation. A transformation converts the fields of the source into the format of the target.

A transformation consists of at least one transformation rule. Various rule types, transformation types, and routine types are available. These allow you to create very simple and highly complex transformations.

With *Allow Currency and Unit Conversion*, you can enable currency translation and unit conversion for InfoSources and DataStore objects. As a prerequisite for this, all currencies and units of measure must be contained in the key of the target. For DataStore objects, translation/conversion of currencies and units is only supported with InfoObjects.

With *Check Units for Consistency*, you can activate the consistency check for units. This allows you to prevent records from being updated with initial units of measure for example.

Runtime



In *Runtime Properties*, you can see whether the transformation is being processed in SAP HANA or in ABAP runtime. The default value is SAP HANA You can switch this value to process the transformation in ABAP runtime. For the switch, the system checks that the rules and any existing routines are still supported.

Differences Between ABAP Runtime and SAP HANA Runtime

The transformations can be processed in ABAP or in SAP HANA. You make this setting for a transformation explicitly on the *General* tab.

SAP HANA runtime provides better performance than ABAP runtime, and mass data can be processed very quickly. SAP HANA runtime is more complex to work with however.

There are essential differences between the two runtime types, which can lead to different results. When using SAP HANA runtime, you should therefore carefully validate the results against expected results, especially if existing ABAP transformations are to be replaced.

With an ODP source system, we recommend using ABAP runtime, since the extraction is done by ABAP anyway. A transformation in SAP HANA would cause unnecessary effort, since the data must first be persisted.

Type Conversion

In ABAP runtime, types supported in ABAP are converted automatically, so a value of a character field can easily be written to an integer field if the field contains numbers at runtime. In SAP HANA runtime, a runtime error will occur if you do not explicitly specify the type conversion in the formula or in the SAP HANA script.

Rounding of Values

Unlike with ABAP runtime, the results of calculations when working with SAP HANA are truncated according to the decimal places, and are not rounded.

Example

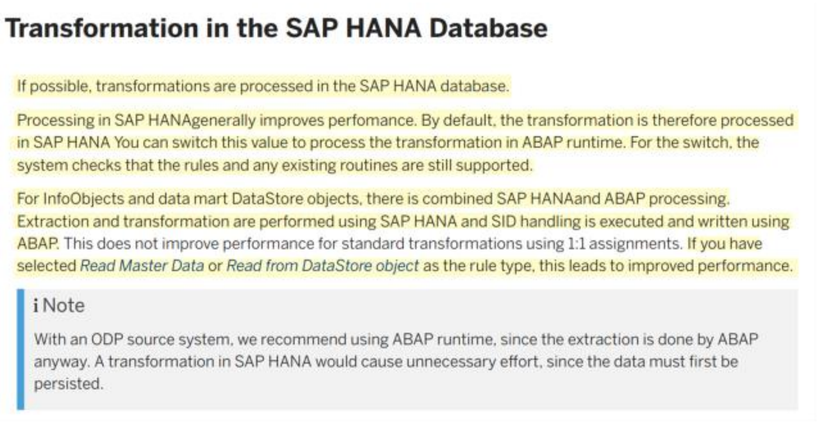
You have an *amount* key figure of type *DEC* (decimal 17) with three decimal places and the value 2.

The formula AMOUNT / 3 calculates the value 0.667 (rounded up to the last decimal place) in ABAP runtime, but 0.666 in SAP HANA runtime.

If you use a formula, you can assure the rounding by additional conversions and an explicit rounding:

TO_DECIMAL(ROUND(TO_DOUBLE(AMOUNT) / TO_DOUBLE(/BIC/SZRDEC02); 3; ''); 17; 3)

HANA Runtime



Transformation in the SAP HANA Database

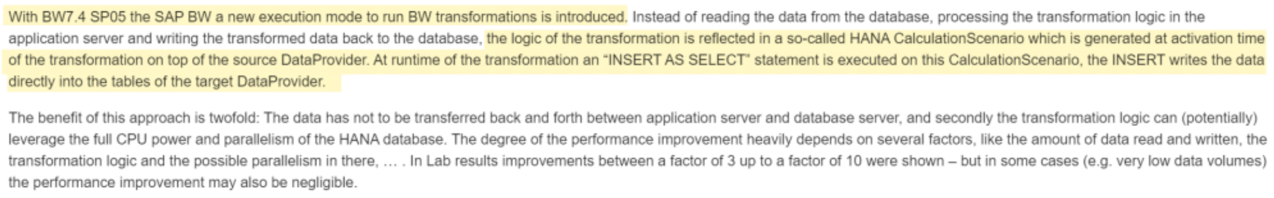
If possible, transformations are processed in the SAP HANA database.

Processing in SAP HANAgenerally improves perfomance. By default, the transformation is therefore processed in SAP HANA You can switch this value to process the transformation in ABAP runtime. For the switch, the system checks that the rules and any existing routines are still supported.

For InfoObjects and data mart DataStore objects, there is combined SAP HANAand ABAP processing. Extraction and transformation are performed using SAP HANA and SID handling is executed and written using ABAP. This does not improve performance for standard transformations using 1:1 assignments. If you have selected *Read Master Data* or *Read from DataStore object* as the rule type, this leads to improved performance.

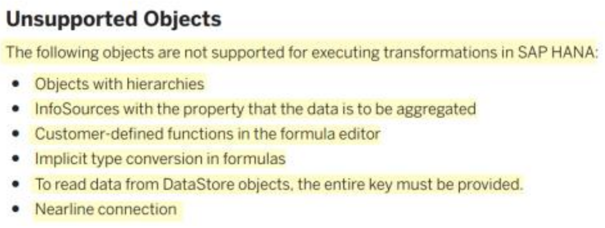
i Note

With an ODP source system, we recommend using ABAP runtime, since the extraction is done by ABAP anyway. A transformation in SAP HANA would cause unnecessary effort, since the data must first be persisted.



With BW7.4 SP05 the SAP BW a new execution mode to run BW transformations is introduced. Instead of reading the data from the database, processing the transformation logic in the application server and writing the transformed data back to the database, the logic of the transformation is reflected in a so-called HANA CalculationScenario which is generated at activation time of the transformation on top of the source DataProvider. At runtime of the transformation an "INSERT AS SELECT" statement is executed on this CalculationScenario, the INSERT writes the data directly into the tables of the target DataProvider.

The benefit of this approach is twofold: The data has not to be transferred back and forth between application server and database server, and secondly the transformation logic can (potentially) leverage the full CPU power and parallelism of the HANA database. The degree of the performance improvement heavily depends on several factors, like the amount of data read and written, the transformation logic and the possible parallelism in there, … . In Lab results improvements between a factor of 3 up to a factor of 10 were shown – but in some cases (e.g. very low data volumes) the performance improvement may also be negligible.

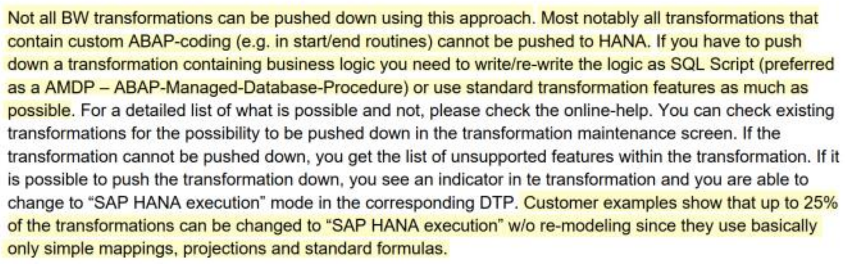


Unsupported Objects

The following objects are not supported for executing transformations in SAP HANA:

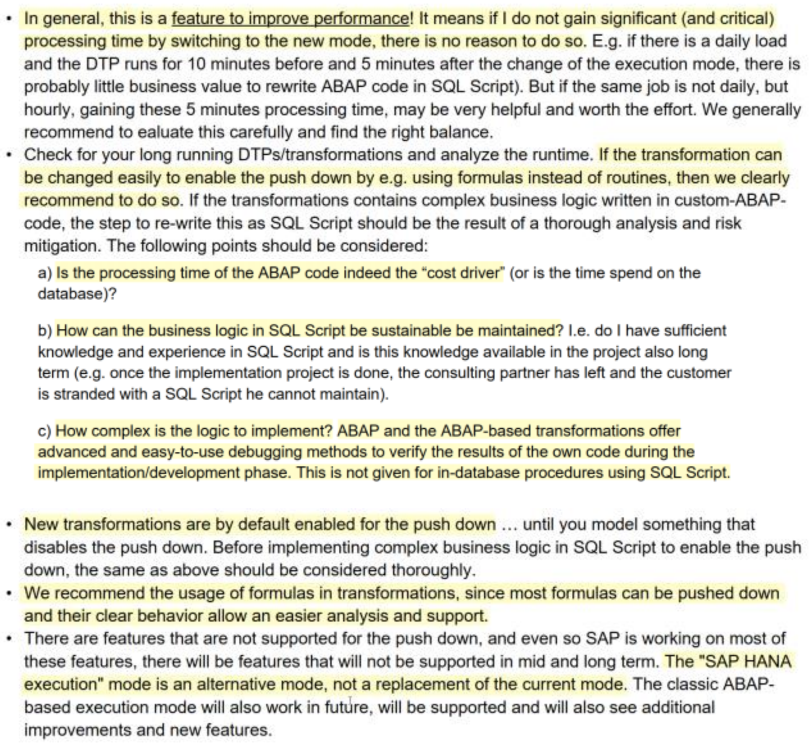
- Objects with hierarchies
- InfoSources with the property that the data is to be aggregated
- Customer-defined functions in the formula editor
- Implicit type conversion in formulas
- To read data from DataStore objects, the entire key must be provided.
- Nearline connection

Recommendations for SAP HANA runtime:



Not all BW transformations can be pushed down using this approach. Most notably all transformations that contain custom ABAP-coding (e.g. in start/end routines) cannot be pushed to HANA. If you have to push down a transformation containing business logic you need to write/re-write the logic as SQL Script (preferred as a AMDP – ABAP-Managed-Database-Procedure) or use standard transformation features as much as possible. For a detailed list of what is possible and not, please check the online-help. You can check existing transformations for the possibility to be pushed down in the transformation maintenance screen. If the transformation cannot be pushed down, you get the list of unsupported features within the transformation. If it is possible to push the transformation down, you see an indicator in te transformation and you are able to change to "SAP HANA execution" mode in the corresponding DTP. Customer examples show that up to 25% of the transformations can be changed to "SAP HANA execution" w/o re-modeling since they use basically only simple mappings, projections and standard formulas.

What should be considered before switching to HANA runtime?



- In general, this is a feature to improve performance! It means if I do not gain significant (and critical) processing time by switching to the new mode, there is no reason to do so. E.g. if there is a daily load and the DTP runs for 10 minutes before and 5 minutes after the change of the execution mode, there is probably little business value to rewrite ABAP code in SQL Script). But if the same job is not daily, but hourly, gaining these 5 minutes processing time, may be very helpful and worth the effort. We generally recommend to ealuate this carefully and find the right balance.
- Check for your long running DTPs/transformations and analyze the runtime. If the transformation can be changed easily to enable the push down by e.g. using formulas instead of routines, then we clearly recommend to do so. If the transformations contains complex business logic written in custom-ABAP-code, the step to re-write this as SQL Script should be the result of a thorough analysis and risk mitigation. The following points should be considered:

  a) Is the processing time of the ABAP code indeed the "cost driver" (or is the time spend on the database)?

  b) How can the business logic in SQL Script be sustainable be maintained? I.e. do I have sufficient knowledge and experience in SQL Script and is this knowledge available in the project also long term (e.g. once the implementation project is done, the consulting partner has left and the customer is stranded with a SQL Script he cannot maintain).

  c) How complex is the logic to implement? ABAP and the ABAP-based transformations offer advanced and easy-to-use debugging methods to verify the results of the own code during the implementation/development phase. This is not given for in-database procedures using SQL Script.

- New transformations are by default enabled for the push down … until you model something that disables the push down. Before implementing complex business logic in SQL Script to enable the push down, the same as above should be considered thoroughly.
- We recommend the usage of formulas in transformations, since most formulas can be pushed down and their clear behavior allow an easier analysis and support.
- There are features that are not supported for the push down, and even so SAP is working on most of these features, there will be features that will not be supported in mid and long term. The "SAP HANA execution" mode is an alternative mode, not a replacement of the current mode. The classic ABAP-based execution mode will also work in future, will be supported and will also see additional improvements and new features.

ABAP Routines

ABAP Routines in Transformations

You use routines to define complex transformation rules.

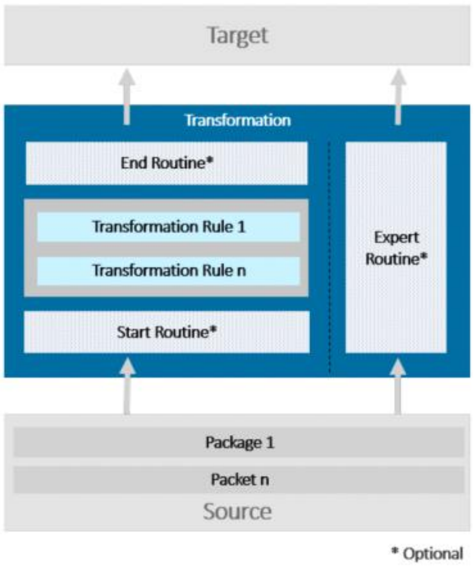
i Note

If you want to create ABAP routines, you must select the ABAP runtime under *Runtime Properties*.

ABAP routines are methods of a local ABAP class that consist of a predefined definition area and an implementation area. The TYPES for the inbound and outbound parameters and the signature of the routine (ABAP method) are stored in the definition area. The routine itself is created in the implementation area. ABAP object statements are available in the coding of the routine. Upon generation, the coding is embedded in the local class of the transformation program as the method.

i Note

You should not change this class, and you should only call it using the transformation.



Features

The routine has a global part and a local part. In the global part, you define global data declarations 'CLASS DATA'. These are available in all routines.

i Note

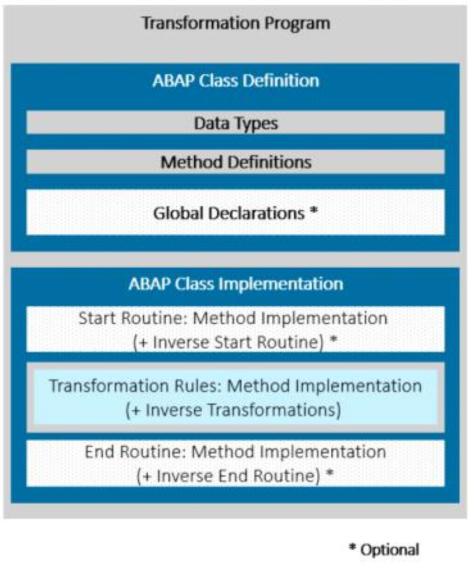
Do not create any global implementations. These cannot be processed. Changes or enhancements outside of the methods cannot be applied and are therefore lost.

i Note

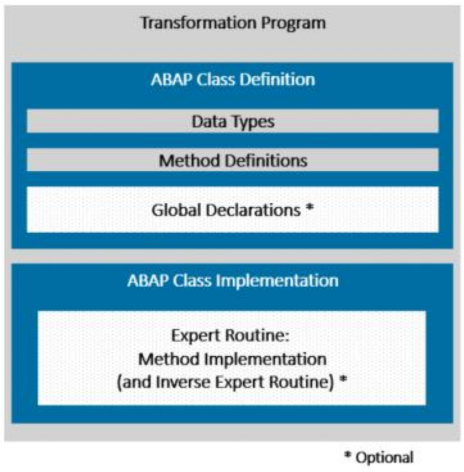
In earlier releases you could create declarations in a second global area. The second global area is no longer supported. You have to copy these declarations from the second global area into the first global area. See SAP Note 2654109 for more details.

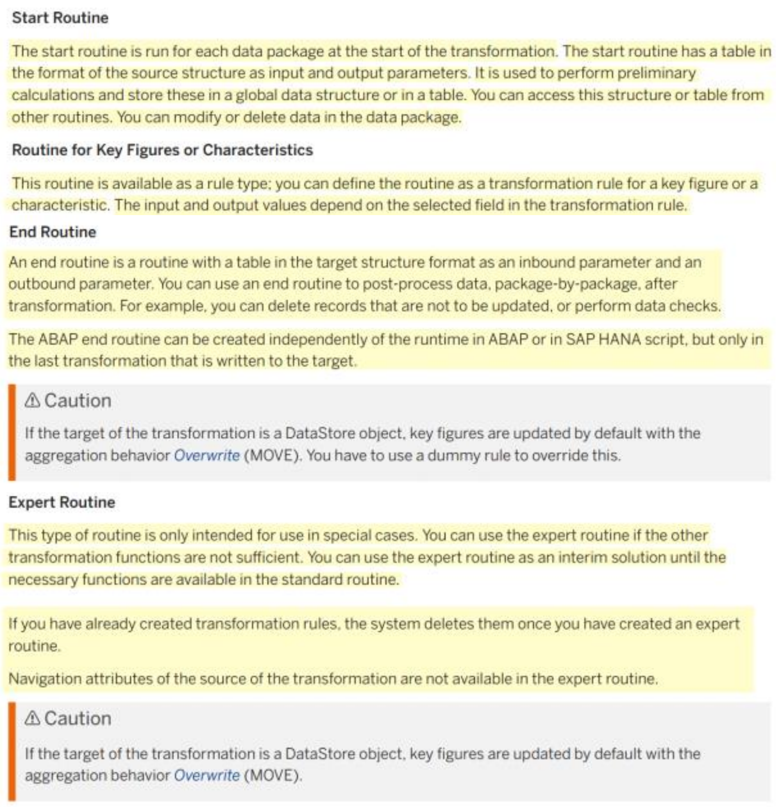
You can create function modules, methods or external subprograms in the ABAP Workbench if you want to reuse source code in routines. You can call these in the local part of the routine. The source code of the method is copied into the metadata of the transformation and transported with the transformation. In the target system, a new class is created from the metadata.

There are various types of routine in the transformation: start routine, routine for key figures or characteristics, end routine and expert routine.


Transformation Program
ABAP Class Definition
Data Types
Method Definitions
Global Declarations *
ABAP Class Implementation
Start Routine: Method Implementation (+ Inverse Start Routine) *
Transformation Rules: Method Implementation (+ Inverse Transformations)
End Routine: Method Implementation (+ Inverse End Routine) *
* Optional

For Expert Routine:





Start Routine

The start routine is run for each data package at the start of the transformation. The start routine has a table in the format of the source structure as input and output parameters. It is used to perform preliminary calculations and store these in a global data structure or in a table. You can access this structure or table from other routines. You can modify or delete data in the data package.

Routine for Key Figures or Characteristics

This routine is available as a rule type; you can define the routine as a transformation rule for a key figure or a characteristic. The input and output values depend on the selected field in the transformation rule.

End Routine

An end routine is a routine with a table in the target structure format as an inbound parameter and an outbound parameter. You can use an end routine to post-process data, package-by-package, after transformation. For example, you can delete records that are not to be updated, or perform data checks.

The ABAP end routine can be created independently of the runtime in ABAP or in SAP HANA script, but only in the last transformation that is written to the target.

⚠ Caution

If the target of the transformation is a DataStore object, key figures are updated by default with the aggregation behavior *Overwrite* (MOVE). You have to use a dummy rule to override this.

Expert Routine

This type of routine is only intended for use in special cases. You can use the expert routine if the other transformation functions are not sufficient. You can use the expert routine as an interim solution until the necessary functions are available in the standard routine.

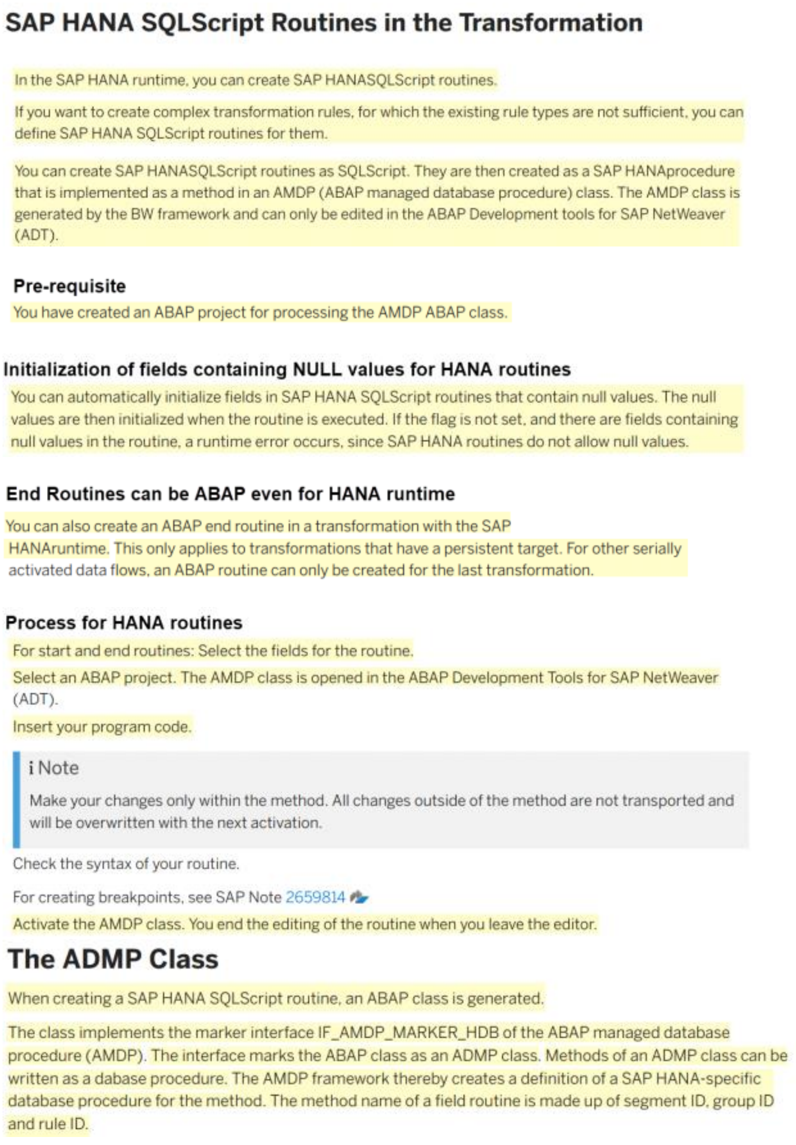
If you have already created transformation rules, the system deletes them once you have created an expert routine.

Navigation attributes of the source of the transformation are not available in the expert routine.

⚠ Caution

If the target of the transformation is a DataStore object, key figures are updated by default with the aggregation behavior *Overwrite* (MOVE).

SAP HANA SQL Script Routines	BW 7.50 SP04 you are able to use AMDP scripts for Start-, Field-, End-, and Expert-routines.



SAP HANA SQLScript Routines in the Transformation

In the SAP HANA runtime, you can create SAP HANASQLScript routines.

If you want to create complex transformation rules, for which the existing rule types are not sufficient, you can define SAP HANA SQLScript routines for them.

You can create SAP HANASQLScript routines as SQLScript. They are then created as a SAP HANAprocedure that is implemented as a method in an AMDP (ABAP managed database procedure) class. The AMDP class is generated by the BW framework and can only be edited in the ABAP Development tools for SAP NetWeaver (ADT).

Pre-requisite

You have created an ABAP project for processing the AMDP ABAP class.

Initialization of fields containing NULL values for HANA routines

You can automatically initialize fields in SAP HANA SQLScript routines that contain null values. The null values are then initialized when the routine is executed. If the flag is not set, and there are fields containing null values in the routine, a runtime error occurs, since SAP HANA routines do not allow null values.

End Routines can be ABAP even for HANA runtime

You can also create an ABAP end routine in a transformation with the SAP HANAruntime. This only applies to transformations that have a persistent target. For other serially activated data flows, an ABAP routine can only be created for the last transformation.

Process for HANA routines

For start and end routines: Select the fields for the routine.

Select an ABAP project. The AMDP class is opened in the ABAP Development Tools for SAP NetWeaver (ADT).

Insert your program code.

> **i Note**
>
> Make your changes only within the method. All changes outside of the method are not transported and will be overwritten with the next activation.

Check the syntax of your routine.

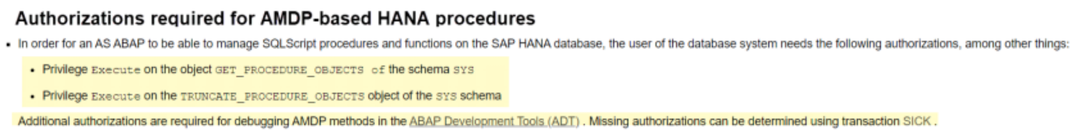
For creating breakpoints, see SAP Note 2659814

Activate the AMDP class. You end the editing of the routine when you leave the editor.

The ADMP Class

When creating a SAP HANA SQLScript routine, an ABAP class is generated.

The class implements the marker interface IF_AMDP_MARKER_HDB of the ABAP managed database procedure (AMDP). The interface marks the ABAP class as an ADMP class. Methods of an ADMP class can be written as a dabase procedure. The AMDP framework thereby creates a definition of a SAP HANA-specific database procedure for the method. The method name of a field routine is made up of segment ID, group ID and rule ID.



Authorizations required for AMDP-based HANA procedures

- In order for an AS ABAP to be able to manage SQLScript procedures and functions on the SAP HANA database, the user of the database system needs the following authorizations, among other things:
  - Privilege `Execute` on the object `GET_PROCEDURE_OBJECTS` of the schema `SYS`
  - Privilege `Execute` on the `TRUNCATE_PROCEDURE_OBJECTS` object of the `SYS` schema

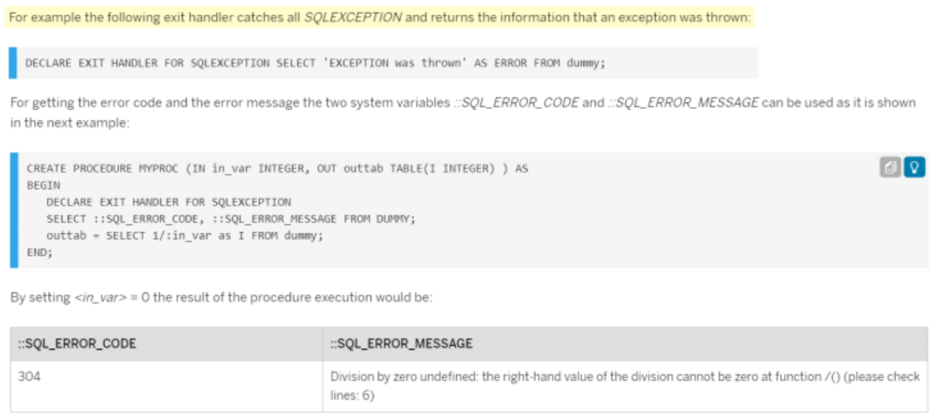
  Additional authorizations are required for debugging AMDP methods in the ABAP Development Tools (ADT) . Missing authorizations can be determined using transaction SICK .

The biggest difference when switching from ABAP to AMDP is that Field routines take the whole columns instead of 1 field value as in ABAP stack. This means you can process the whole datapackage field in 1 go instead of 1 by 1 p rocessing in the old ABAP world. This is great as many developers wrote their code in Start- or End- routines to improve performance compared to Field routines though sometimes they wanted to change only 1 field. With AMDP it is now possible to process only that field in 1 go.

Important variables

inTab - Internal table with incoming data (source_package)
outTab - Internal table with output data (result_package)
errorTab - Table containing erroneous records

Exit Handling in HANA Transformations using AMDP



For example the following exit handler catches all *SQLEXCEPTION* and returns the information that an exception was thrown:

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'EXCEPTION was thrown' AS ERROR FROM dummy;
```

For getting the error code and the error message the two system variables *::SQL_ERROR_CODE* and *::SQL_ERROR_MESSAGE* can be used as it is shown in the next example:

```
CREATE PROCEDURE MYPROC (IN in_var INTEGER, OUT outtab TABLE(I INTEGER) ) AS
BEGIN
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   SELECT ::SQL_ERROR_CODE, ::SQL_ERROR_MESSAGE FROM DUMMY;
   outtab = SELECT 1/:in_var as I FROM dummy;
END;
```

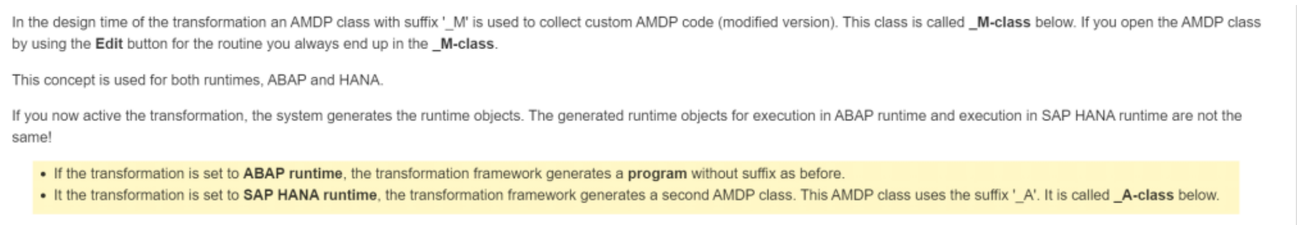
By setting *<in_var>* = 0 the result of the procedure execution would be:

::SQL_ERROR_CODE	::SQL_ERROR_MESSAGE
304	Division by zero undefined: the right-hand value of the division cannot be zero at function /() (please check lines: 6)

Debugging AMDP based

Concept:

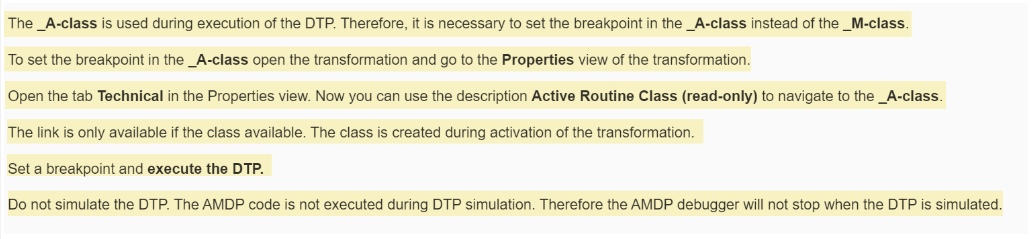
TRFN code


In the design time of the transformation an AMDP class with suffix '_M' is used to collect custom AMDP code (modified version). This class is called **_M-class** below. If you open the AMDP class by using the **Edit** button for the routine you always end up in the **_M-class**.

This concept is used for both runtimes, ABAP and HANA.

If you now active the transformation, the system generates the runtime objects. The generated runtime objects for execution in ABAP runtime and execution in SAP HANA runtime are not the same!

- If the transformation is set to **ABAP runtime**, the transformation framework generates a **program** without suffix as before.
- It the transformation is set to **SAP HANA runtime**, the transformation framework generates a second AMDP class. This AMDP class uses the suffix '_A'. It is called **_A-class** below.


The **_A-class** is used during execution of the DTP. Therefore, it is necessary to set the breakpoint in the **_A-class** instead of the **_M-class**.

To set the breakpoint in the **_A-class** open the transformation and go to the **Properties** view of the transformation.

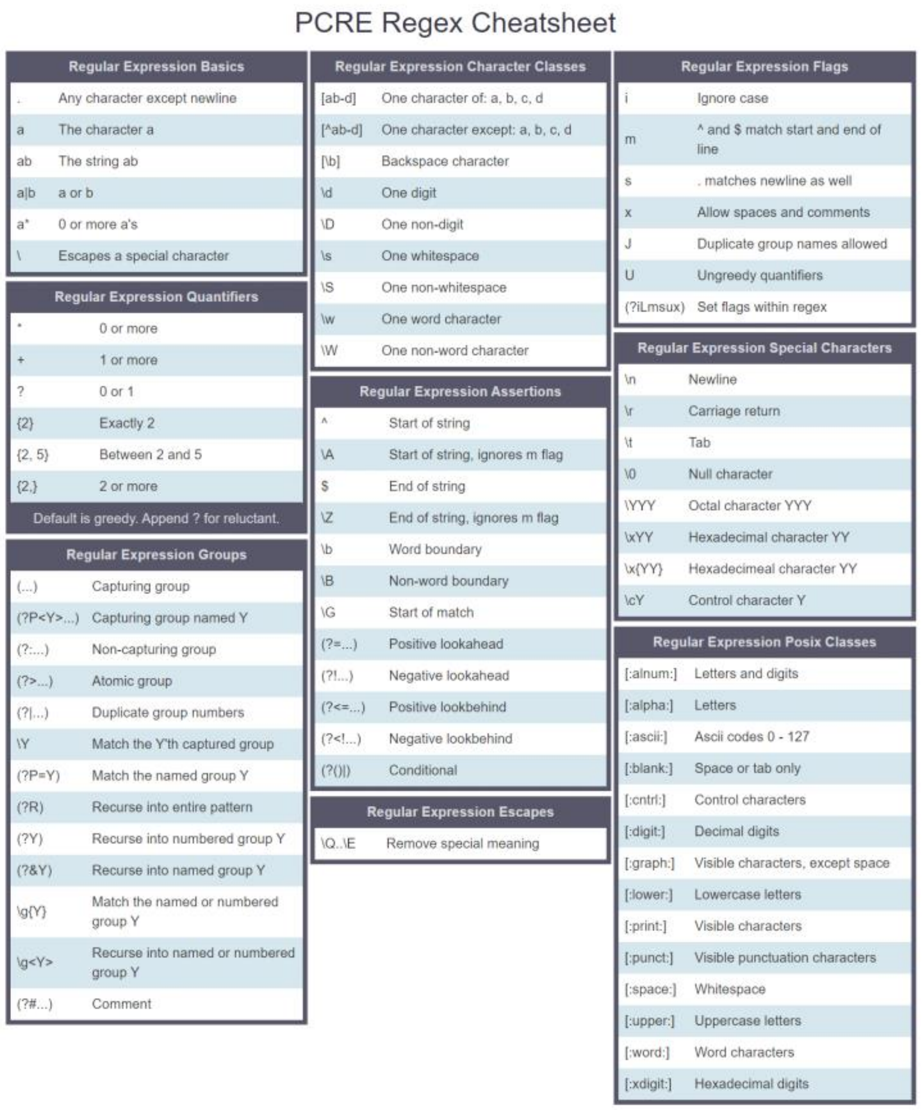
Open the tab **Technical** in the Properties view. Now you can use the description **Active Routine Class (read-only)** to navigate to the **_A-class**.

The link is only available if the class available. The class is created during activation of the transformation.

Set a breakpoint and **execute the DTP.**

Do not simulate the DTP. The AMDP code is not executed during DTP simulation. Therefore the AMDP debugger will not stop when the DTP is simulated.

PCRE (Perl Compatible Regular Expression)


PCRE Regex Cheatsheet

Regular Expression Basics	
.	Any character except newline
a	The character a
ab	The string ab
a|b	a or b
a*	0 or more a's
\	Escapes a special character

Regular Expression Quantifiers	
*	0 or more
+	1 or more
?	0 or 1
{2}	Exactly 2
{2, 5}	Between 2 and 5
{2,}	2 or more
Default is greedy. Append ? for reluctant.	

Regular Expression Groups	
(...)	Capturing group
(?P<Y>...)	Capturing group named Y
(?:...)	Non-capturing group
(?>...)	Atomic group
(?|...)	Duplicate group numbers
\Y	Match the Y'th captured group
(?P=Y)	Match the named group Y
(?R)	Recurse into entire pattern
(?Y)	Recurse into numbered group Y
(?&Y)	Recurse into named group Y
\g{Y}	Match the named or numbered group Y
\g<Y>	Recurse into named or numbered group Y
(?#...)	Comment

Regular Expression Character Classes	
[ab-d]	One character of: a, b, c, d
[^ab-d]	One character except: a, b, c, d
[\b]	Backspace character
\d	One digit
\D	One non-digit
\s	One whitespace
\S	One non-whitespace
\w	One word character
\W	One non-word character

Regular Expression Assertions	
^	Start of string
\A	Start of string, ignores m flag
$	End of string
\Z	End of string, ignores m flag
\b	Word boundary
\B	Non-word boundary
\G	Start of match
(?=...)	Positive lookahead
(?!...)	Negative lookahead
(?<=...)	Positive lookbehind
(?<!...)	Negative lookbehind
(?()|)	Conditional

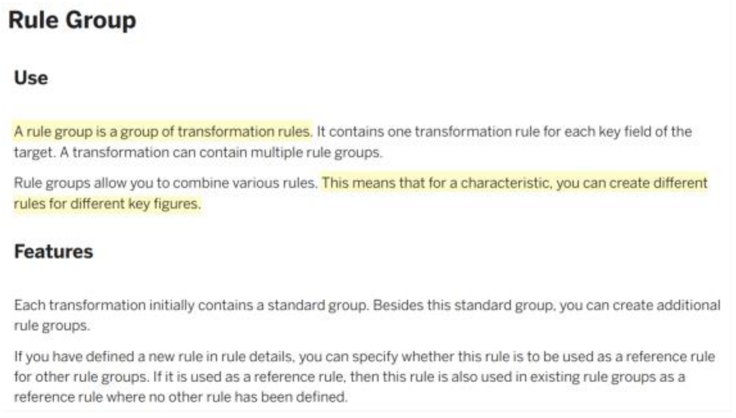
Regular Expression Escapes	
\Q..\E	Remove special meaning

Regular Expression Flags	
i	Ignore case
m	^ and $ match start and end of line
s	. matches newline as well
x	Allow spaces and comments
J	Duplicate group names allowed
U	Ungreedy quantifiers
(?iLmsux)	Set flags within regex

Regular Expression Special Characters	
\n	Newline
\r	Carriage return
\t	Tab
\0	Null character
\YYY	Octal character YYY
\xYY	Hexadecimal character YY
\x{YY}	Hexadecimeal character YY
\cY	Control character Y

Regular Expression Posix Classes	
[:alnum:]	Letters and digits
[:alpha:]	Letters
[:ascii:]	Ascii codes 0 - 127
[:blank:]	Space or tab only
[:cntrl:]	Control characters
[:digit:]	Decimal digits
[:graph:]	Visible characters, except space
[:lower:]	Lowercase letters
[:print:]	Visible characters
[:punct:]	Visible punctuation characters
[:space:]	Whitespace
[:upper:]	Uppercase letters
[:word:]	Word characters
[:xdigit:]	Hexadecimal digits

Rule Groups


Rule Group

Use

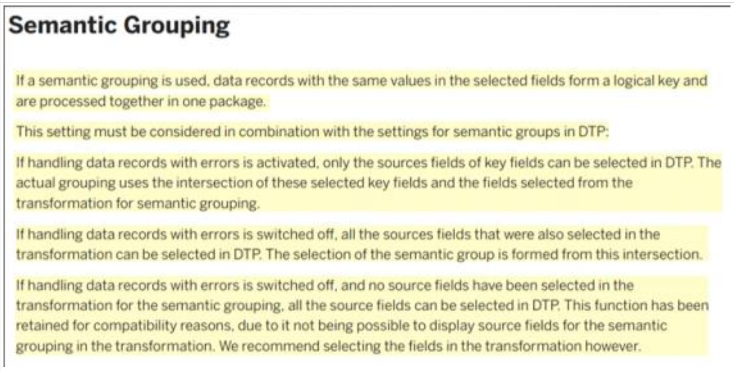
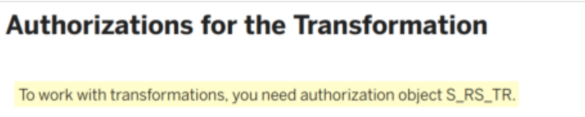
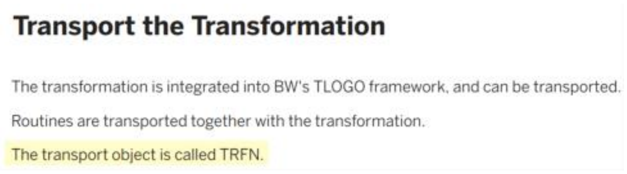
A rule group is a group of transformation rules. It contains one transformation rule for each key field of the target. A transformation can contain multiple rule groups.

Rule groups allow you to combine various rules. This means that for a characteristic, you can create different rules for different key figures.

Features

Each transformation initially contains a standard group. Besides this standard group, you can create additional rule groups.

If you have defined a new rule in rule details, you can specify whether this rule is to be used as a reference rule for other rule groups. If it is used as a reference rule, then this rule is also used in existing rule groups as a reference rule where no other rule has been defined.

Semantic Group	 **Semantic Grouping** If a semantic grouping is used, data records with the same values in the selected fields form a logical key and are processed together in one package. This setting must be considered in combination with the settings for semantic groups in DTP: If handling data records with errors is activated, only the sources fields of key fields can be selected in DTP. The actual grouping uses the intersection of these selected key fields and the fields selected from the transformation for semantic grouping. If handling data records with errors is switched off, all the sources fields that were also selected in the transformation can be selected in DTP. The selection of the semantic group is formed from this intersection. If handling data records with errors is switched off, and no source fields have been selected in the transformation for the semantic grouping, all the source fields can be selected in DTP. This function has been retained for compatibility reasons, due to it not being possible to display source fields for the semantic grouping in the transformation. We recommend selecting the fields in the transformation however.
Authorization & Transport	 **Authorizations for the Transformation** To work with transformations, you need authorization object S_RS_TR.  **Transport the Transformation** The transformation is integrated into BW's TLOGO framework, and can be transported. Routines are transported together with the transformation. The transport object is called TRFN.

Transformations in B4H

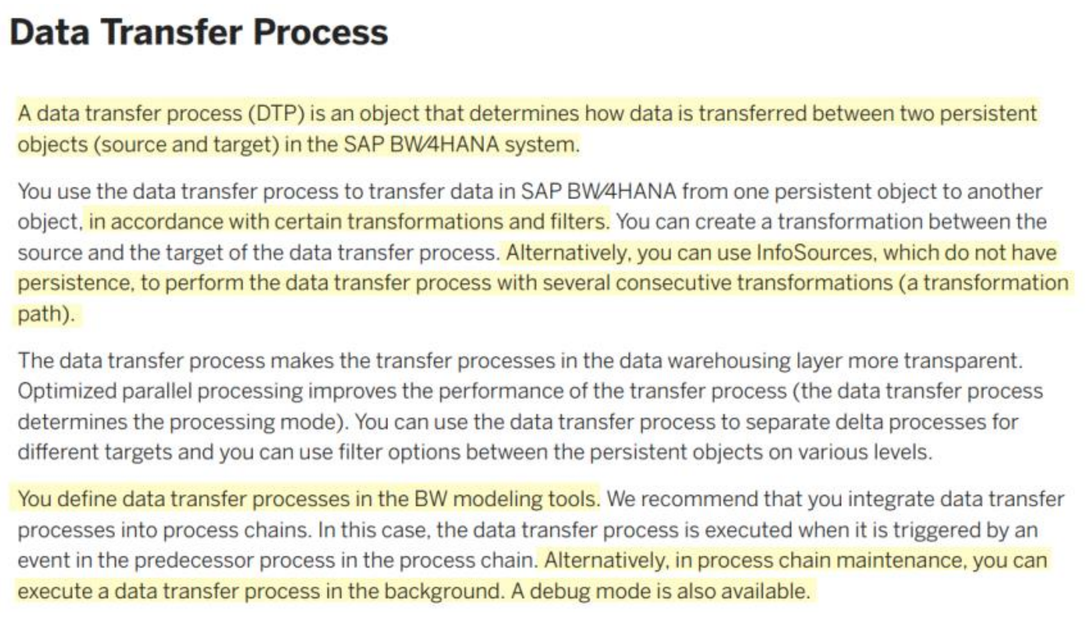
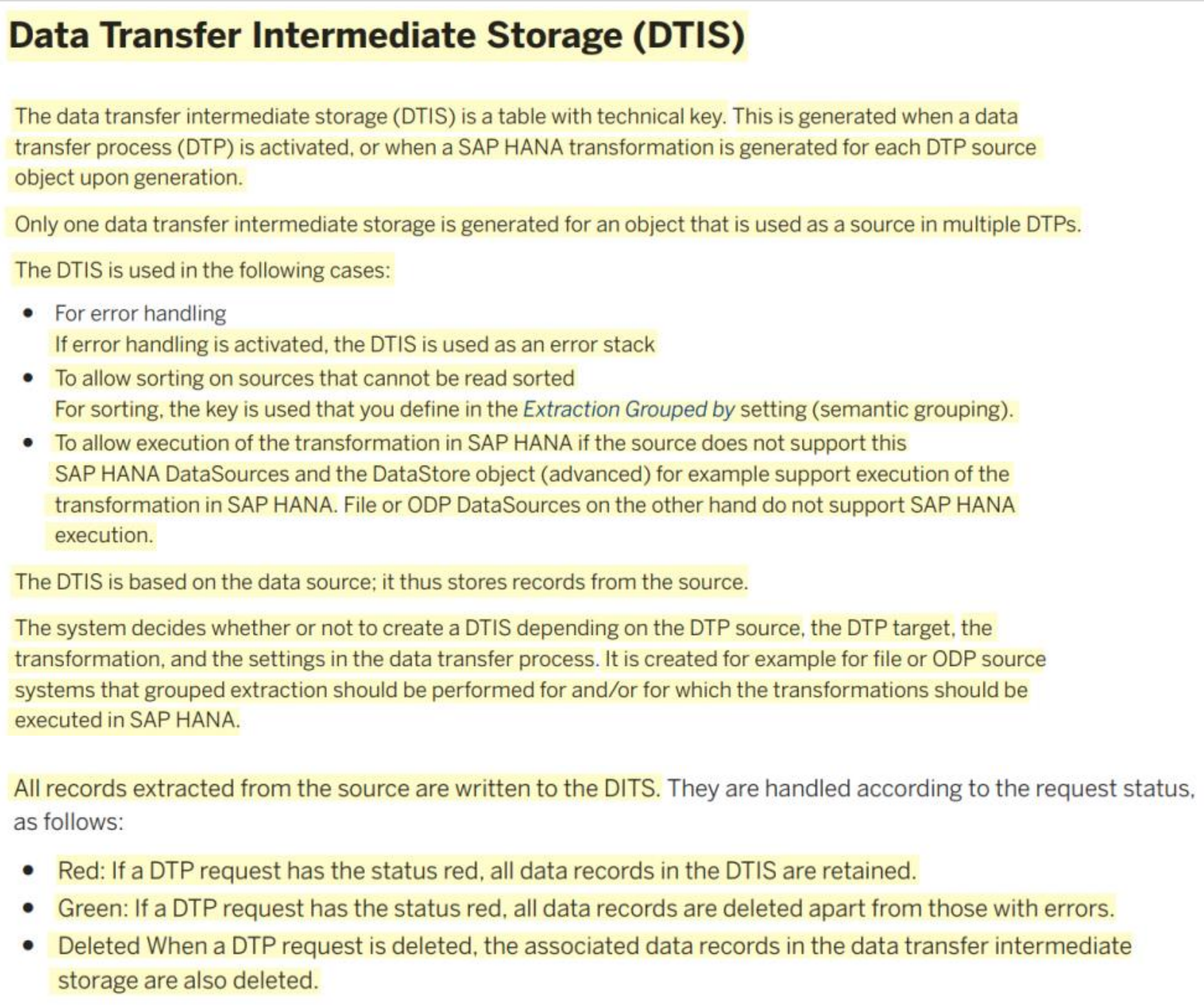
- Everything is now Class and method.
- All routines (Start, Field, End) - same class different methods.
- They are regular class in case of ABAP runtime & AMDP Class in case of HANA runtime.
- The way to identify the class whether it's AMDP or not is via 2 things:
  - Tag interface is there or not (IF_AMDP_MARKER_HDB)
  - Keywords :
    BY DATABASE PROCEURE FOR HDB
        LANGUAGE SQLSCRIPT
        READ ONLY

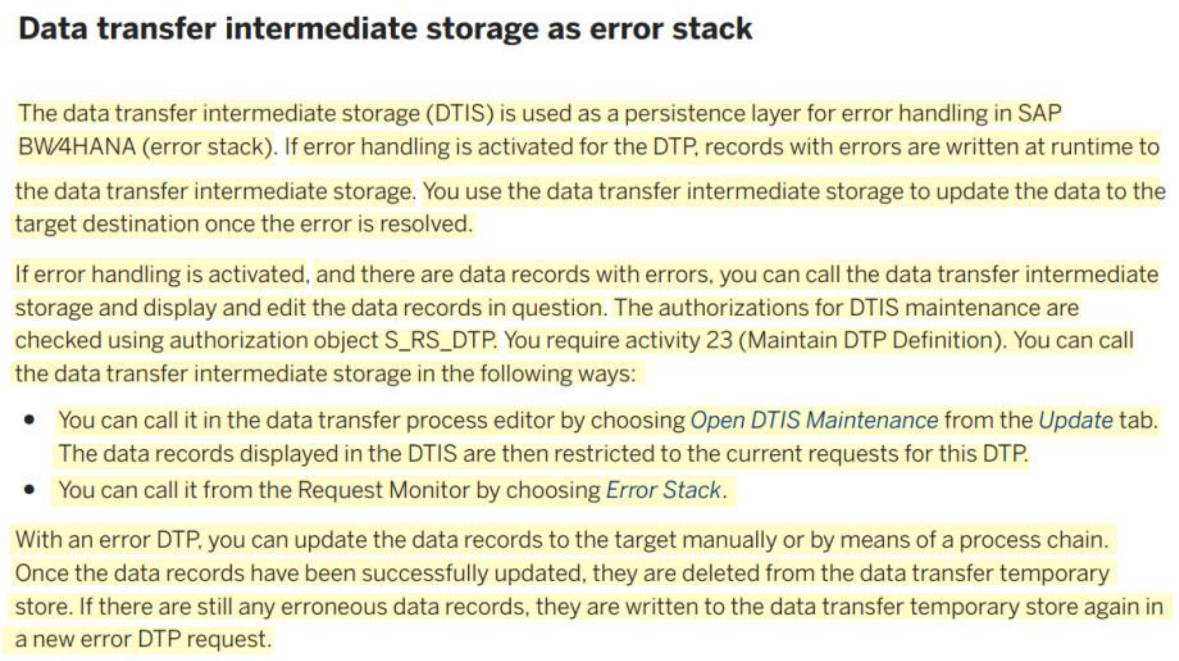
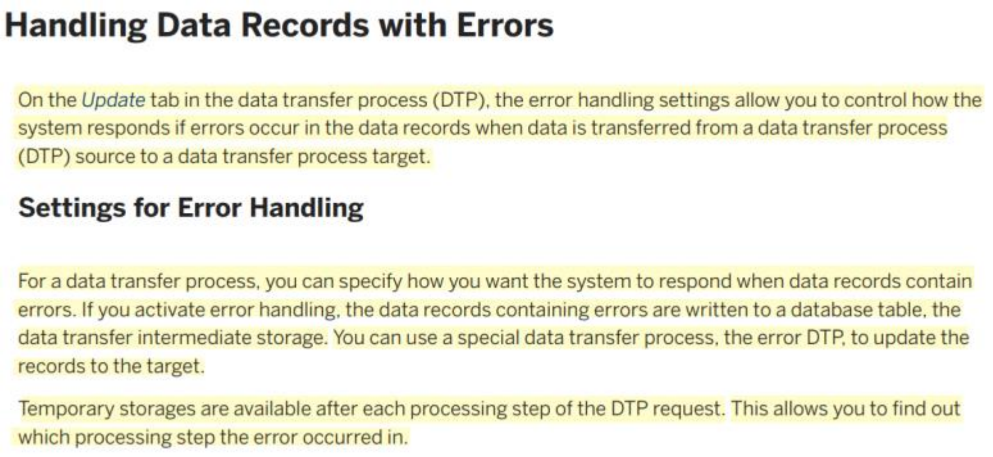
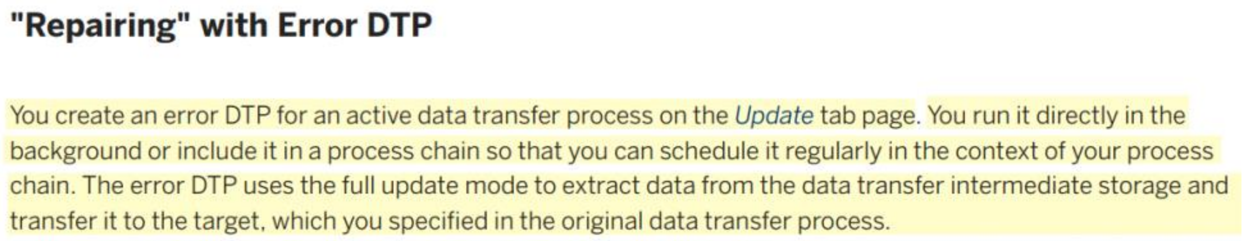
- Field routine logic for qty conversion:
  Sales Unit: KAR
  Base Unit: EA
  Numerator (for conversion) 5
  Denominatory (for conversion) 1
  Actual Qty: 1

  Qty in BUOM = (Qty in SUOM * Numerator) / Denominator

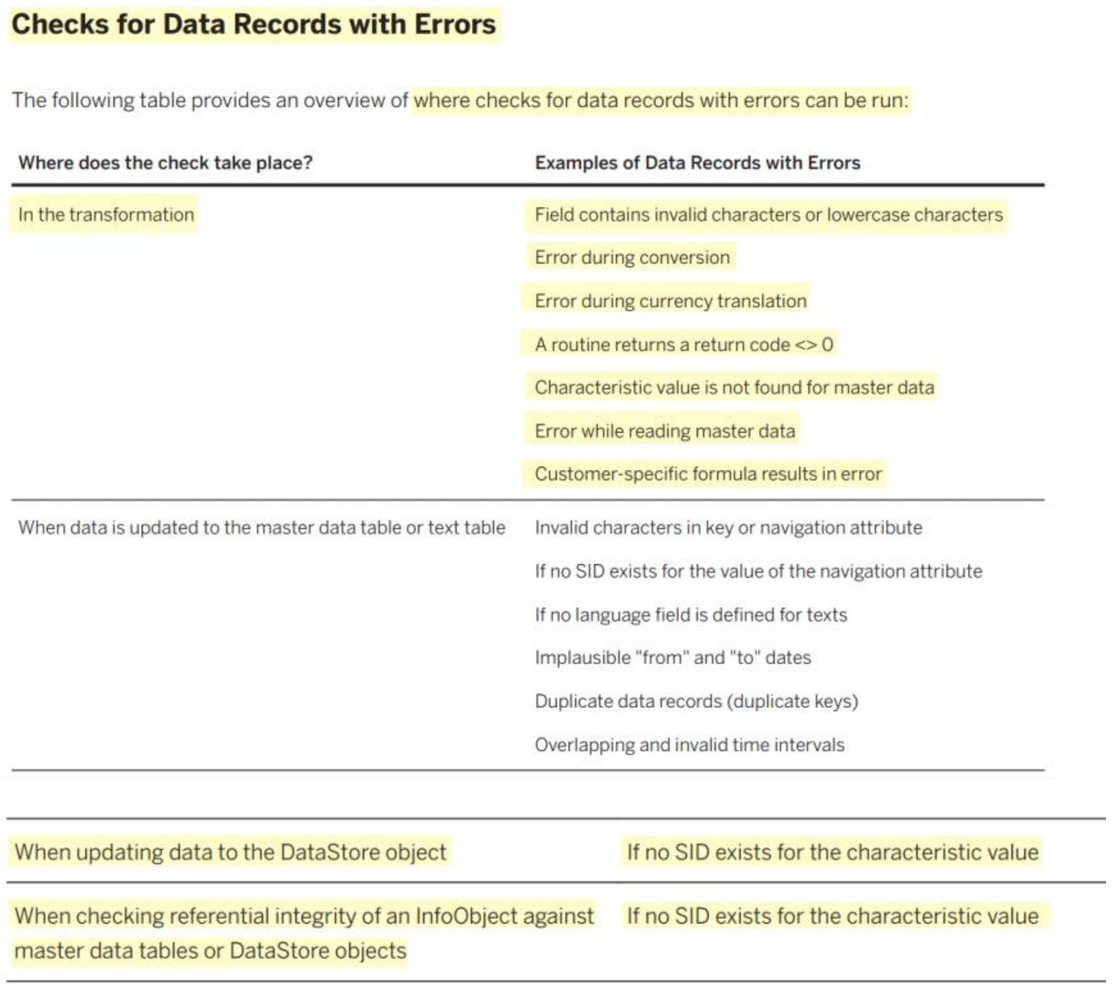
  If doc category is either 'H' or 'K' then put negative value else positive

DTP in B4H

DTP Definition	 **Data Transfer Process** A data transfer process (DTP) is an object that determines how data is transferred between two persistent objects (source and target) in the SAP BW∕4HANA system. You use the data transfer process to transfer data in SAP BW∕4HANA from one persistent object to another object, in accordance with certain transformations and filters. You can create a transformation between the source and the target of the data transfer process. Alternatively, you can use InfoSources, which do not have persistence, to perform the data transfer process with several consecutive transformations (a transformation path). The data transfer process makes the transfer processes in the data warehousing layer more transparent. Optimized parallel processing improves the performance of the transfer process (the data transfer process determines the processing mode). You can use the data transfer process to separate delta processes for different targets and you can use filter options between the persistent objects on various levels. You define data transfer processes in the BW modeling tools. We recommend that you integrate data transfer processes into process chains. In this case, the data transfer process is executed when it is triggered by an event in the predecessor process in the process chain. Alternatively, in process chain maintenance, you can execute a data transfer process in the background. A debug mode is also available.
DTIS	 **Data Transfer Intermediate Storage (DTIS)** The data transfer intermediate storage (DTIS) is a table with technical key. This is generated when a data transfer process (DTP) is activated, or when a SAP HANA transformation is generated for each DTP source object upon generation. Only one data transfer intermediate storage is generated for an object that is used as a source in multiple DTPs. The DTIS is used in the following cases: • For error handling If error handling is activated, the DTIS is used as an error stack • To allow sorting on sources that cannot be read sorted For sorting, the key is used that you define in the *Extraction Grouped by* setting (semantic grouping). • To allow execution of the transformation in SAP HANA if the source does not support this SAP HANA DataSources and the DataStore object (advanced) for example support execution of the transformation in SAP HANA. File or ODP DataSources on the other hand do not support SAP HANA execution. The DTIS is based on the data source; it thus stores records from the source. The system decides whether or not to create a DTIS depending on the DTP source, the DTP target, the transformation, and the settings in the data transfer process. It is created for example for file or ODP source systems that grouped extraction should be performed for and/or for which the transformations should be executed in SAP HANA. All records extracted from the source are written to the DITS. They are handled according to the request status, as follows: • Red: If a DTP request has the status red, all data records in the DTIS are retained. • Green: If a DTP request has the status red, all data records are deleted apart from those with errors. • Deleted When a DTP request is deleted, the associated data records in the data transfer intermediate storage are also deleted.

DTIS as Error stack	 **Data transfer intermediate storage as error stack** The data transfer intermediate storage (DTIS) is used as a persistence layer for error handling in SAP BW∕4HANA (error stack). If error handling is activated for the DTP, records with errors are written at runtime to the data transfer intermediate storage. You use the data transfer intermediate storage to update the data to the target destination once the error is resolved. If error handling is activated, and there are data records with errors, you can call the data transfer intermediate storage and display and edit the data records in question. The authorizations for DTIS maintenance are checked using authorization object S_RS_DTP. You require activity 23 (Maintain DTP Definition). You can call the data transfer intermediate storage in the following ways: • You can call it in the data transfer process editor by choosing *Open DTIS Maintenance* from the *Update* tab. The data records displayed in the DTIS are then restricted to the current requests for this DTP. • You can call it from the Request Monitor by choosing *Error Stack*. With an error DTP, you can update the data records to the target manually or by means of a process chain. Once the data records have been successfully updated, they are deleted from the data transfer temporary store. If there are still any erroneous data records, they are written to the data transfer temporary store again in a new error DTP request.
Error Handling	 **Handling Data Records with Errors** On the *Update* tab in the data transfer process (DTP), the error handling settings allow you to control how the system responds if errors occur in the data records when data is transferred from a data transfer process (DTP) source to a data transfer process target. **Settings for Error Handling** For a data transfer process, you can specify how you want the system to respond when data records contain errors. If you activate error handling, the data records containing errors are written to a database table, the data transfer intermediate storage. You can use a special data transfer process, the error DTP, to update the records to the target. Temporary storages are available after each processing step of the DTP request. This allows you to find out which processing step the error occurred in.  **"Repairing" with Error DTP** You create an error DTP for an active data transfer process on the *Update* tab page. You run it directly in the background or include it in a process chain so that you can schedule it regularly in the context of your process chain. The error DTP uses the full update mode to extract data from the data transfer intermediate storage and transfer it to the target, which you specified in the original data transfer process.

Scenarios resulting in errors

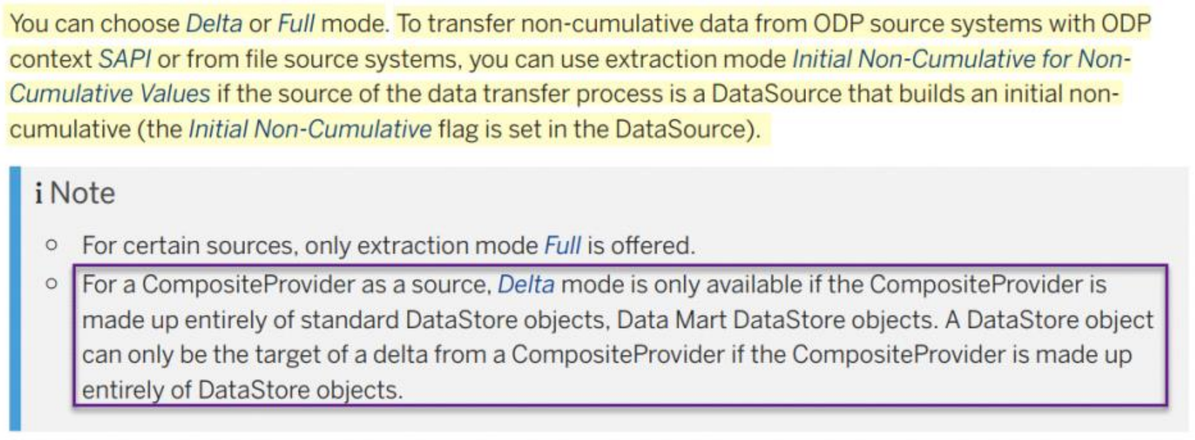


Checks for Data Records with Errors

The following table provides an overview of where checks for data records with errors can be run:

Where does the check take place?	Examples of Data Records with Errors
In the transformation	Field contains invalid characters or lowercase characters Error during conversion Error during currency translation A routine returns a return code <> 0 Characteristic value is not found for master data Error while reading master data Customer-specific formula results in error
When data is updated to the master data table or text table	Invalid characters in key or navigation attribute If no SID exists for the value of the navigation attribute If no language field is defined for texts Implausible "from" and "to" dates Duplicate data records (duplicate keys) Overlapping and invalid time intervals
When updating data to the DataStore object	If no SID exists for the characteristic value
When checking referential integrity of an InfoObject against master data tables or DataStore objects	If no SID exists for the characteristic value

General Settings Extraction

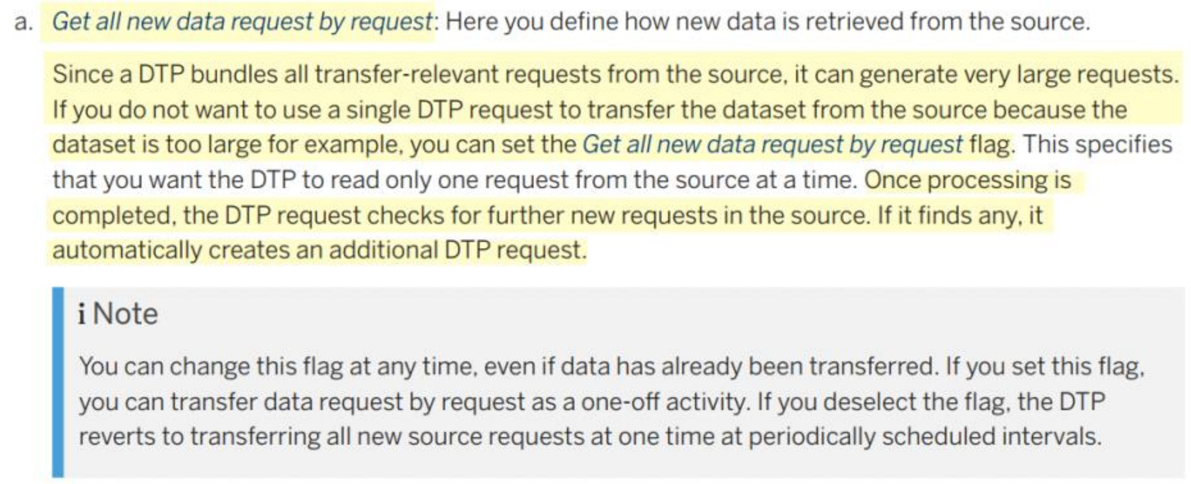


You can choose *Delta* or *Full* mode. To transfer non-cumulative data from ODP source systems with ODP context *SAPI* or from file source systems, you can use extraction mode *Initial Non-Cumulative for Non-Cumulative Values* if the source of the data transfer process is a DataSource that builds an initial non-cumulative (the *Initial Non-Cumulative* flag is set in the DataSource).

i Note

- For certain sources, only extraction mode *Full* is offered.
- For a CompositeProvider as a source, *Delta* mode is only available if the CompositeProvider is made up entirely of standard DataStore objects, Data Mart DataStore objects. A DataStore object can only be the target of a delta from a CompositeProvider if the CompositeProvider is made up entirely of DataStore objects.

General Settings Request Selection



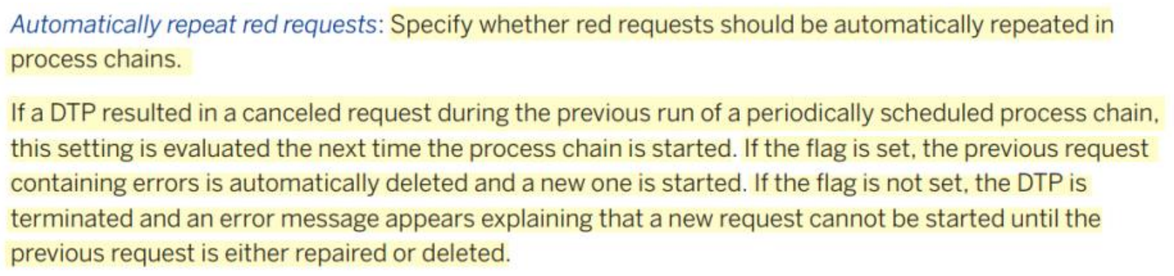
a. *Get all new data request by request*: Here you define how new data is retrieved from the source.

Since a DTP bundles all transfer-relevant requests from the source, it can generate very large requests. If you do not want to use a single DTP request to transfer the dataset from the source because the dataset is too large for example, you can set the *Get all new data request by request* flag. This specifies that you want the DTP to read only one request from the source at a time. Once processing is completed, the DTP request checks for further new requests in the source. If it finds any, it automatically creates an additional DTP request.

i Note

You can change this flag at any time, even if data has already been transferred. If you set this flag, you can transfer data request by request as a one-off activity. If you deselect the flag, the DTP reverts to transferring all new source requests at one time at periodically scheduled intervals.

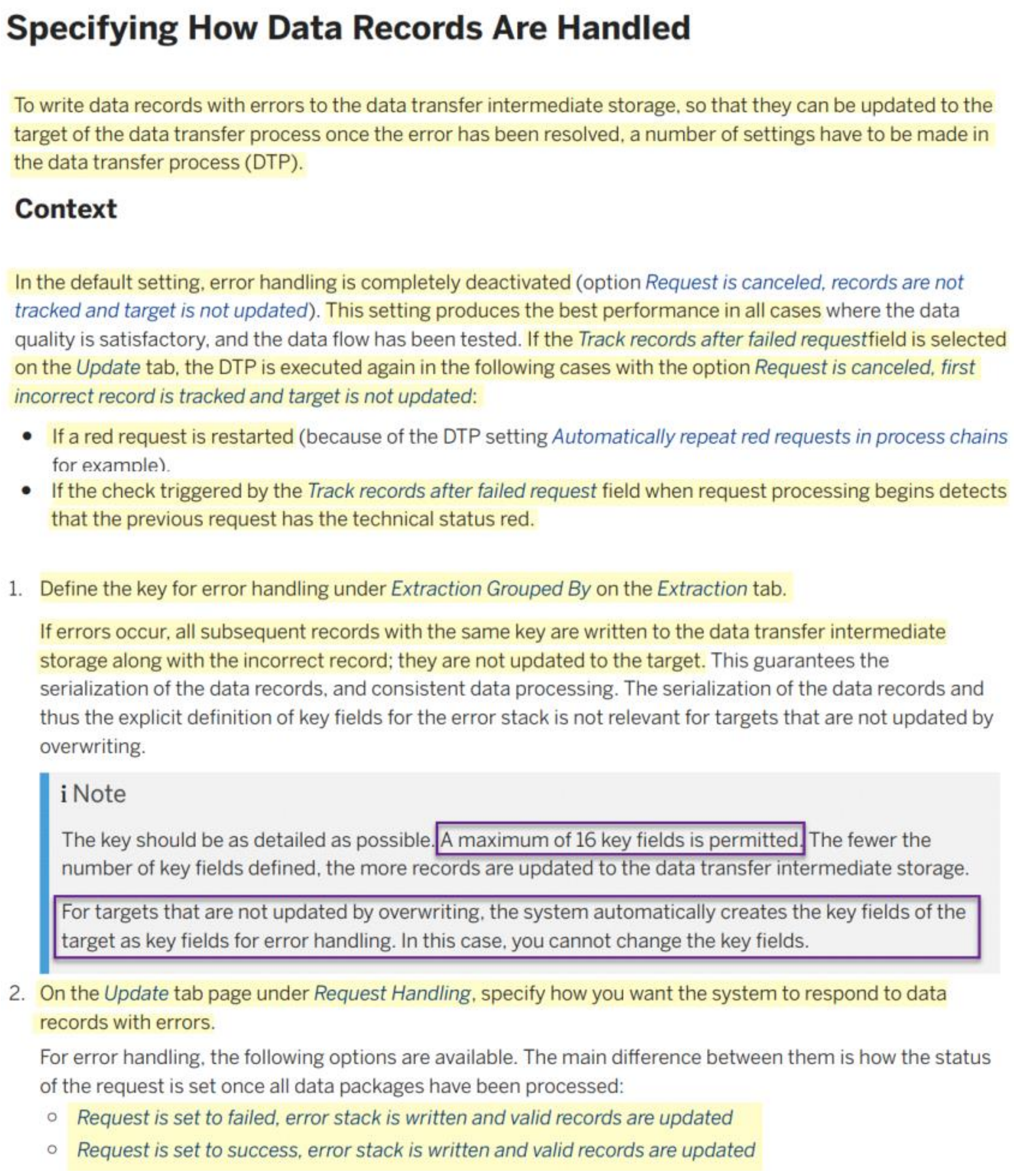
	b. *Only get delta once*: Here you specify whether the source requests should only be transferred once. Setting this flag ensures that the content of the InfoProvider is an exact representation of the source data. A scenario of this type might be required if you always want an InfoProvider to contain the most up-to-date data set for a source, but technical reasons prevent the DataSource on which it is based from delivering a delta (new, changed, or deleted data records). For this type of DataSource, the current data set for the required selection can only be transferred using a Full Update. In this case, a DataStore object cannot usually be used to determine the missing delta information (overwrite and creation of delta). If this is not logically possible because data is deleted in the source without delivering reverse records for example, you can set this flag and perform a *snapshot scenario*. Only the most up-to-date request for this DataSource is retained in the InfoProvider. Earlier requests for the DataSource are deleted from the (target) InfoProvider before a new one is requested (this is done by a process in a process chain, for example). They are not transferred again during the DTP delta process. When the system determines the delta when a new DTP request is generated, these earlier (source) requests are seen as *already fetched*. c. *Perform delta initialization without data*: Here you specify whether the first request of a delta DTP should be transferred without any data. If this flag is set, the source data is flagged as fetched, but is not actually transferred to the target. The data flagged as fetched is not read by the next request. Instead only the new source data accrued in the meantime is read. Use this setting in scenarios where the DTP is automatically implemented within a process chain to rebuild a data target. **i Note** This setting is a transport-relevant change to the metadata of the DTP. If you want to flag source data as fetched just for test purposes, use the *Set delta status to 'Fetched'* option. You thus prevent a transport-relevant change being made to the metadata of the DTP. d. *Extract all green requests*: This flag is displayed if the source of the data transfer process is a staging DataStore object that has only one inbound table. If you set this flag, all green requests are extracted, regardless of whether there are yellow or red requests between them. If the flag is not set, only those green requests are extracted that are older than the first yellow or red request. *Only retrieve last request*: For a data transfer process that loads data in Full mode from a DataSource or an InfoPackage, you can stipulate here that the request of the data transfer process only retrieves the last request in the queue.
General Settings Execution	Execution 6. *Processing Mode*: The system normally defines the processing mode for the background processing of the respective data transfer process automatically. 7. *Technical Request Status*: Specify which technical status the request should be given if there are warnings in the log. 8. *Overall Status of Request*: Specify how the overall status of the request is defined. Once the technical processing stage for a DTP request has been completed, the overall status of the request can be set automatically (based on the technical status of the request) or set manually (by the user). If the overall status is set manually, the status initially remains unchanged, if the technical processing stage was completed with a red or green status. In particular, this means that data for a green request is not released for reporting or further processing. The overall status has to be set manually by the user or by a process in a process chain.



Automatically repeat red requests: Specify whether red requests should be automatically repeated in process chains.

If a DTP resulted in a canceled request during the previous run of a periodically scheduled process chain, this setting is evaluated the next time the process chain is started. If the flag is set, the previous request containing errors is automatically deleted and a new one is started. If the flag is not set, the DTP is terminated and an error message appears explaining that a new request cannot be started until the previous request is either repaired or deleted.

Settings Update



Specifying How Data Records Are Handled

To write data records with errors to the data transfer intermediate storage, so that they can be updated to the target of the data transfer process once the error has been resolved, a number of settings have to be made in the data transfer process (DTP).

Context

In the default setting, error handling is completely deactivated (option *Request is canceled, records are not tracked and target is not updated*). This setting produces the best performance in all cases where the data quality is satisfactory, and the data flow has been tested. If the *Track records after failed request*field is selected on the *Update* tab, the DTP is executed again in the following cases with the option *Request is canceled, first incorrect record is tracked and target is not updated*:

- If a red request is restarted (because of the DTP setting *Automatically repeat red requests in process chains* for example).
- If the check triggered by the *Track records after failed request* field when request processing begins detects that the previous request has the technical status red.

1. Define the key for error handling under *Extraction Grouped By* on the *Extraction* tab.

   If errors occur, all subsequent records with the same key are written to the data transfer intermediate storage along with the incorrect record; they are not updated to the target. This guarantees the serialization of the data records, and consistent data processing. The serialization of the data records and thus the explicit definition of key fields for the error stack is not relevant for targets that are not updated by overwriting.

   > i Note
   >
   > The key should be as detailed as possible. A maximum of 16 key fields is permitted. The fewer the number of key fields defined, the more records are updated to the data transfer intermediate storage.
   >
   > For targets that are not updated by overwriting, the system automatically creates the key fields of the target as key fields for error handling. In this case, you cannot change the key fields.

2. On the *Update* tab page under *Request Handling*, specify how you want the system to respond to data records with errors.

   For error handling, the following options are available. The main difference between them is how the status of the request is set once all data packages have been processed:
   - *Request is set to failed, error stack is written and valid records are updated*
   - *Request is set to success, error stack is written and valid records are updated*

Q & A - Post Session

Q	
A	

Code Blocks

CDS View Enhancement for Item

```
@AbapCatalog.sqlViewAppendName: 'ZV_SALES_ITM_EXT'
@EndUserText.label: 'Tran: Sales Item Enhancements'
extend view Zi_Bw_Sales_Item with ZI_BW_SALES_ITEM_EXT
association to mara as _m
            on $projection.matnr = _m.matnr

{
  _m.mtart
}
```

Start Routine

```
outTab =
      SELECT
        recordmode,
        doc_number,
        s_ord_item,
        comp_code,
        sold_to,
        imodoccat,
        salesorg,
        division,
        distr_chan,
        cml_or_qty,
        REPLACE_REGEXPR( '([!|#])' IN "MATERIAL" WITH '' OCCURRENCE ALL ) AS "MATERIAL",
        matl_group,
        matl_type,
        plant,
        base_uom,
        net_value,
        denomintr,
        numerator,
        st_up_dte,
        sales_unit,
        doc_currcy,
        fiscvarnt,
        gn_r3_ssy,
        record,
        sql__procedure__source__record

      FROM :inTab;
```

End Routine

```
LOOP AT result_package ASSIGNING <result_fields>.
    <result_fields>-upd_date = sy-datum.
  ENDLOOP.
```

Field Routine

```
declare EXIT HANDLER FOR sqlexception
  SELECT ::SQL_ERROR_CODE, ::SQL_ERROR_MESSAGE FROM "PUBLIC"."DUMMY";

  outtab =
    select
      CASE WHEN ( i.numerator = i.denomintr )
         THEN i.cml_or_qty
         ELSE
           CASE WHEN (i.imodoccat in ( 'H', 'K' ))
              THEN round((-1) * ((i.cml_or_qty * i.numerator) / i.denomintr), 3)
           ELSE round(((i.cml_or_qty * i.numerator) / i.denomintr), 3)
           END
      END
      AS "QUANT_B",
      i.base_uom,
      record,
      sql__procedure__source__record
    from :intab as i;

  errortab =
```

	select '' as error_text,
	       '' as sql__procedure__source__record
	from "PUBLIC"."DUMMY"
	where dummy <> 'X';