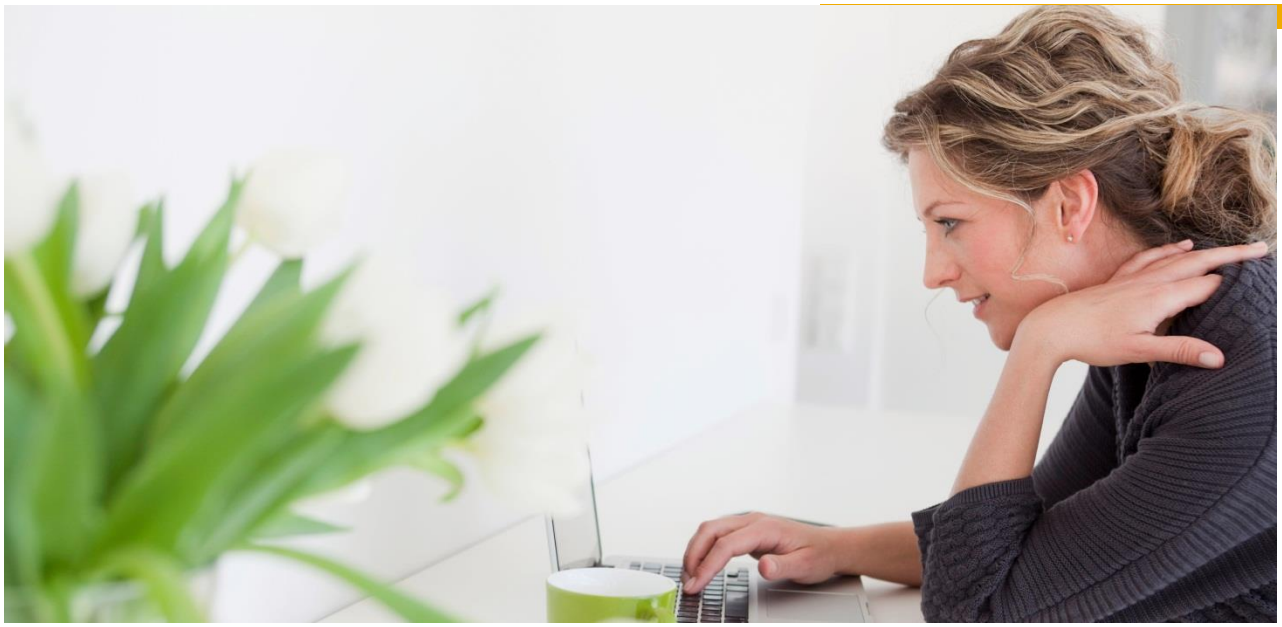


DTMT User Manual

© 2014 SAP SE or an SAP affiliate company. All rights reserved.
No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.
SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. Please see http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark for additional trademark information and notices. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors.
National product specifications may vary.
These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP SE or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP SE or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.
In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, which speak only as of their dates, and they should not be relied upon in making purchasing decisions.

1 GENERAL ... 3
1.1 Preliminary Notes ... 3
1.2 Data Migrations with DTMT ... 3
1.2.1 Adaptations and Enhancements ... 3
1.2.2 Summary of Data Migration Process ... 3
1.2.3 Testing Migration Results ... 3
1.3 Data Migration Versus Data Transport ... 4
1.4 DTMT Architecture ... 4
1.4.1 Generation Approach ... 4
1.4.2 Migration Object ... 5
1.4.3 Migration Scope ... 5
1.4.4 Summary of Data Migration Process ... 5

2 SOURCE CONTENT DEVELOPMENT ... 6
2.1.1 Start DTMT Source Content Design Time ... 6
2.1.2 Source Scope ... 6
2.1.3 Migration Object ... 7

3 SPECIAL FEATURES ON SOURCE SIDE ... 36
3.1 Object ... 36
3.1.1 Execute Parallel Extraction ... 36
3.1.2 File Encryption ... 36
3.1.3 Create an Object Automatically ... 37
3.2 Code Comparison ... 40
3.2.1 Code Comparison on Scope Level ... 40
3.2.2 Code Comparison on Object Level ... 44

4 TARGET CONTENT DEVELOPMENT ... 50
4.1 Start DTMT Target Content Design Time ... 50
4.1.1 Target Scope ... 50
4.1.2 Create Migration Object ... 51
4.1.3 Create Domain Rules ... 69
4.1.4 Create Central Routines ... 73
4.1.5 Create Control Parameters ... 74
4.1.6 Create Fixed Values ... 77
4.1.7 Create Translation Objects ... 79
4.1.8 Scope Level “Preload Tables” ... 82

5 SPECIAL FEATURES ON TARGET SIDE ... 83
5.1 Object ... 83
5.1.1 DTMT Import Launcher ... 83
5.1.2 Update Mode ... 84
5.2 Code Comparison ... 84
5.2.1 Code Comparison on Scope Level ... 84
5.2.2 Code Comparison on Object Level ... 87

6 MIGRATION CUSTOMIZING ... 89
6.1 Domain Rules ... 89
6.2 Central Routines ... 90
6.3 Control Parameters ... 90
6.4 Fixed Values ... 90
6.5 Translation Objects ... 90

1 GENERAL

This manual is intended for users who want to develop, adapt, or enhance migration content within DTMT. It describes the architecture and the principles of DTMT, as well as all its features and functions, which are required to solve common migration tasks. For a better understanding of this, the manual runs through the end-to-end straightforward development process of an example object. Other more specific migration tasks, for example the migration of multiple source tables into a single target or vice versa, will be covered in an additional “How-To” section.

1.1 Preliminary Notes

DTMT stands for **D**ata **T**ransport and **M**igration **T**ool. It is a new ETL tool specifically developed to migrate business object data between ABAP-based SAP systems. The focus on ABAP-based SAP applications and the close integration with the source and target systems make several aspects much easier than with generic ETL tools. Direct access to the respective metadata (dictionary) and detailed knowledge of the system architecture allows SAP to provide DTMT users with convenient solutions to common migration problems.
The reason for SAP to develop a new migration tool was the necessity to support a very specific use case, namely the data migration from AFS (Apparel and Footware Solution) to FMS (Fashion Management Solution). In this scenario, AFS customers need to migrate a large portion of their SAP ERP data to FMS. From a technical point of view, this requires a tool that is able to deal with very complex structural mapping, to cover the many architectural differences between AFS and FMS, and to handle a huge data volume within an acceptable timeframe. Therefore, SAP decided to build DTMT based on the principles and methods of the extremely successful R/2–R/3 migration, since this was a comparable scenario. Most of the old capabilities and some new ones that are now state of the art in modern ETL tools have been implemented in a new SAP GUI-based application.

1.2 Data Migrations with DTMT

Unlike generic ETL tools, DTMT does not depend on the availability of standard data import interfaces (IDocs, BAPIs, and so on) since DTMT allows you to bypass the application logic, and write data directly to the target tables if necessary. This expands the scope of the data transfer beyond only master data and open business transactions, unlike normal legacy data migrations. However, in some cases it might be necessary to use function modules, BAPIs, or IDocs for the final data import for example, for audit reasons.

1.2.1 Adaptations and Enhancements

DTMT is an open tool that can be used to change the provided content, expand the content, or add your own migration content to cover custom objects. The fact that DTMT does not have to be very generic, since its use is restricted to SAP systems, means that several convenience functions are integrated into the tool, which makes content development easier. This would not be possible in heterogeneous scenarios using external ETL tools. Since DTMT acts on table level, users who want to change or develop content should have in-depth knowledge of the actual data model.

1.2.2 Summary of Data Migration Process

Simply put, data migrations carried out with DTMT read data directly from the relevant source tables, and write it to a CSV file. From here, it is read by the corresponding migration and import program on the target side and transferred to the migration rule set. Once the conversion has been completed, the data is either written to the relevant tables in the target system or handed over to BAPIs or function modules that perform the final data insert (see figure 1-2).

1.2.3 Testing Migration Results

It should also be noted that bypassing the business logic and writing data directly to the target tables is not without risk. You risk losing data or changing the data content to such an extent that it can no longer be processed by the application at a later time. Although the risk can be minimized by using the content that SAP provides, the results of the test migrations have to be tested thoroughly. The success of these migrations can be judged only from a business perspective. This means that detailed analyses must be available before and after a migration in both the source and target, to assess the migration.

1.3 Data Migration Versus Data Transport

A data migration is any process whereby data from different structures is transferred from A to B, and the source data is adjusted to the target structures. If the source and target structures are identical (that is, if there is a 1:1 structural relationship between the source and the target), this is referred to as a data transport since no transformation takes place. As far as DTMT is concerned, these two cases differ only in terms of mapping. Mapping can be highly complex or, in the case of a transport, consist of simple assignments between fields of the same name. However, even in the case of a transport, the process is referred to as an ETL process, where “E” stands for Extract, “T” for Transform, and “L” for Load. DTMT allows modeling and execution of these processes.

1.4 DTMT Architecture

Even though DTMT is architecturally separated in a source and a target part, both are bundled together and can only be installed as one package. This allows you to perform data migrations within one system, for example between two clients, as well as between different systems. You simply have to install and use DTMT in both locations if the target is not the source system.

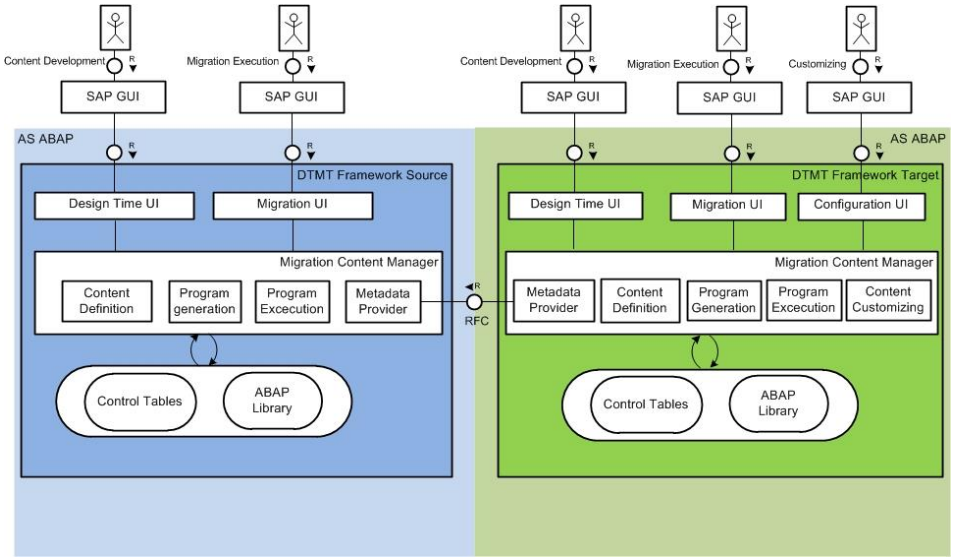


Figure 1-1

1.4.1 Generation Approach

The source part, as well as the target part, of DTMT contains a design studio and a runtime environment. The design studio allows the creation of the source and target content, which is stored in DTMT’s control tables (details will be explained later on). The content serves as input for program generators that are also embedded in DTMT, which ultimately generate the actual export and import programs within the customer environment. The runtime environment of DTMT takes care of the actual program execution and everything that is related to it, such as logs or file management.

The generation approach introduces several advantages that are described below:

- Creating content within the design studio of DTMT does not require a great deal of programming knowledge.
- The direct integration of DTMT in the source and target system allows the program generators to consider the customers’ dictionary during the generation process. The generated data export programs, for example, will automatically extract the field contents of custom fields in migration-relevant tables. As soon as the metadata of the customer-specific fields has been entered in the target system, DTMT recognizes that the target system contains fields for which a transfer rule has to be defined, and requests this rule.
- Since all generated programs are structurally identical, support becomes much easier.

1.4.2 Migration Object

The units in which data is summarized and then migrated are subject to definition. In an ERP environment, they usually contain data that belongs together from a business perspective. Customers, vendors, material masters, purchase orders, or sales orders are just some examples of these types of unit. In DTMT, these units are called migration objects (MO). All data regarding the relevant tables, their key relationships to one another, and all of the required mapping instructions for table and field level are stored within DTMT's control tables.

1.4.3 Migration Scope

In most cases, customers will use DTMT for specific use cases for which SAP provides migration content. One example for such a use case is the previously mentioned **AFS2FMS** migration. Therefore, multiple MOs can be bundled together in a "migration scope". Simply put, a migration scope is an umbrella above all migration objects that belong to a specific migration scenario. Multiple migration scopes can exist side by side in one system.

1.4.4 Summary of Data Migration Process

The figure below shows the simplified data migration process for a migration object.

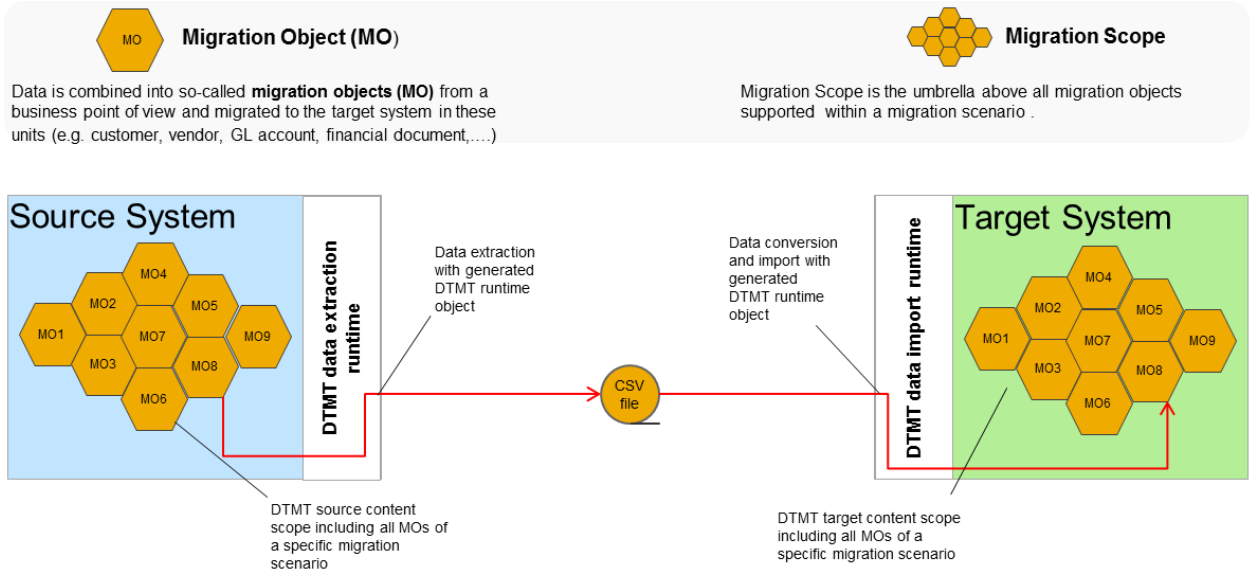


Figure 1-2

- Data of an MO is read from relevant source tables and written to a CSV file by the DTMT extraction runtime object that has been generated based on the extraction content (developed in DTMT design time for source content).
- On the other side, the generated DTMT import runtime object reads data from the CSV file, transforms it to NFS requirements, and fills the relevant target tables (developed in DTMT design time for target content).

2 SOURCE CONTENT DEVELOPMENT

This section describes how to implement source content for a given business object within DTMT, starting with the creation of a source scope and ending with the execution of the data extraction.

2.1.1 Start DTMT Source Content Design Time

DTMT design time for source content can be reached via transaction **DTMT_DTS**.

Note: Together with DTMT, the authorization role **SAP_DTMT_DTS** is imported into the customer system. It has to be assigned to those users who need to work with DTMT.

2.1.2 Source Scope

DTMT supports source content development for multiple migration scenarios in parallel. Therefore, migration scenarios are bundled together in scopes. Technically, a scope is nothing more than a bracket around a number of migration objects. When calling transaction **DTMT_DTS** the first screen you will see is the scope level screen, with the two tabs *Properties* and *Migration Objects*. There you have the option to either select an existing scope or to create a new one. For this user manual, a source scope named **TR02** was created, by pressing the *Create Source Scope* button. The fields on the scope properties screen are explained in figure 2-1 below.

Note: Scope names in customer systems always have to start with a “Z”.

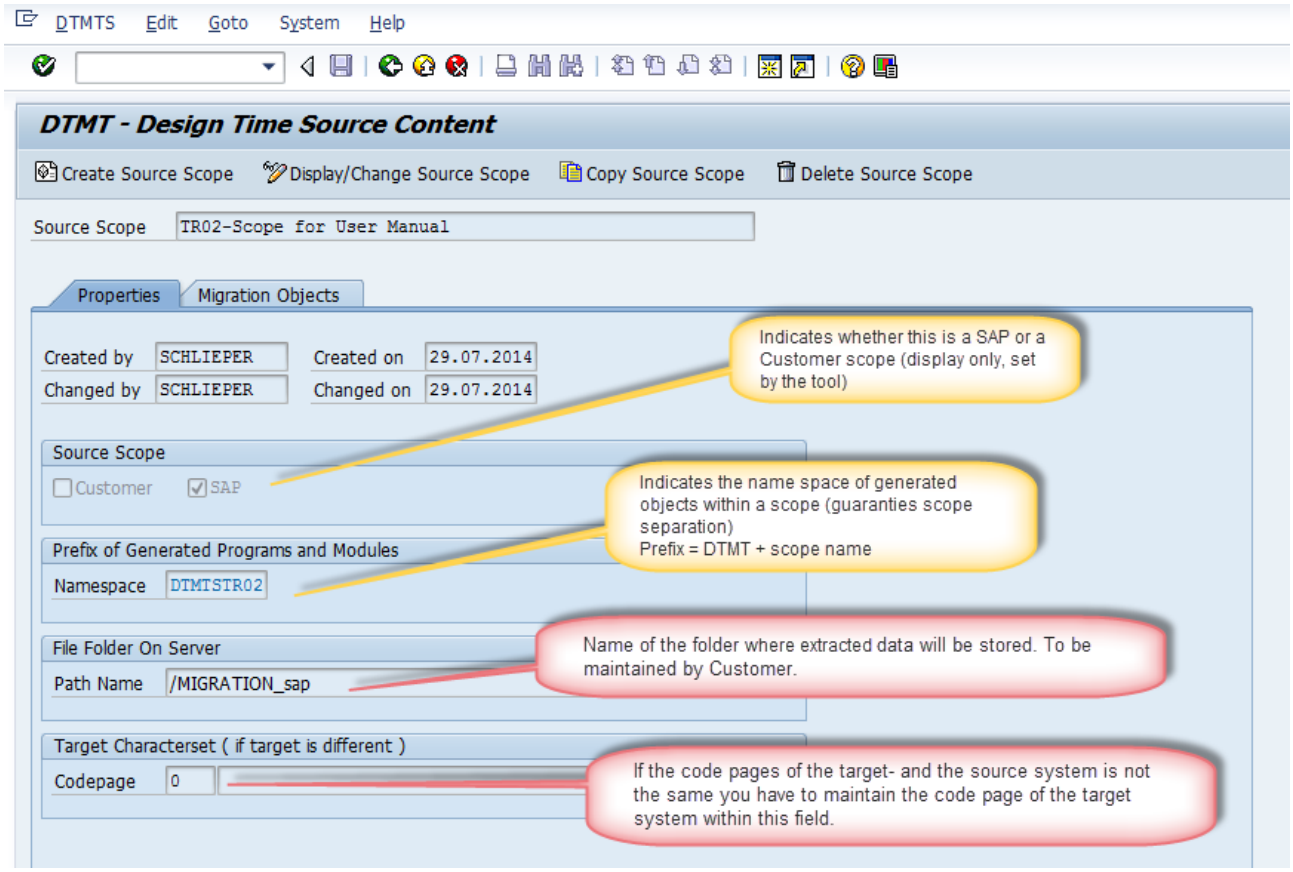


Figure 2-1

2.1.2.1 Copy Source Code

Content provided by SAP is always protected against changes within customer systems. If a customer wants to change or enhance SAP content, he or she needs to copy the relevant SAP source scope to a customer version in advance. The whole DTMT functionality will then be available within the customer scope. The copy

process of a scope is supported by a wizard with which you, and others, can determine which migration objects should be copied together with the scope.

2.1.3 Migration Object

The business object, which serves as an example to demonstrate the content implementation process, is a simplified version of the ERP vendor (the real vendor is more extensive).
The figure below shows the data model of this example business object on a table level.

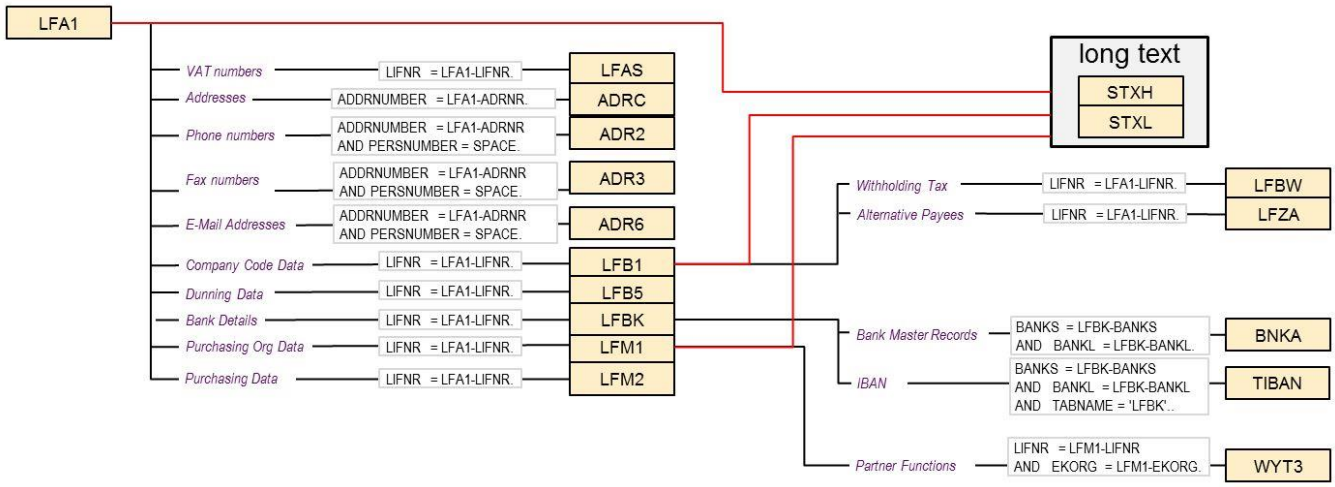


Figure 2-2

<u>Note:</u> This is what a content developer needs to be familiar with if he or she wants to implement source content within DTMT.

2.1.3.1 Create Migration Object

Clicking the *Migration Objects* tab on the scope level screen displays a list of all migration objects of a specific scope. In this case the list is empty, because a new source scope was just created.
After pressing *Create Object,* you will be asked to enter a migration object name and a description. In this example the migration object is called **VEN_TR02**.

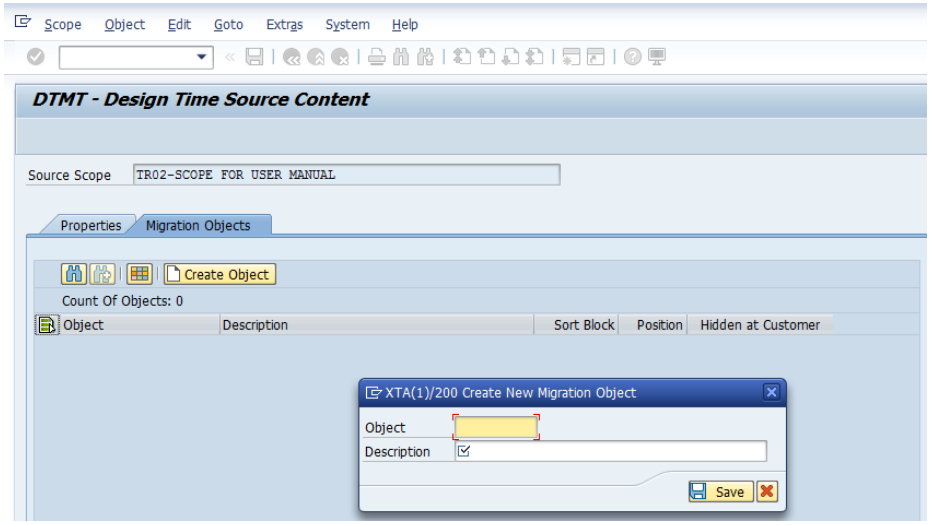


Figure 2-3

2.1.3.2 Migration Object Level Screen

The figure below shows the main screen on migration object level, which appears after you save the example object **VEN_TR02**. To get a first overview of the functionality behind the eight tabs, they will now be explained briefly. Later in this manual you will learn more about them when they are used during the content implementation process for the example object.

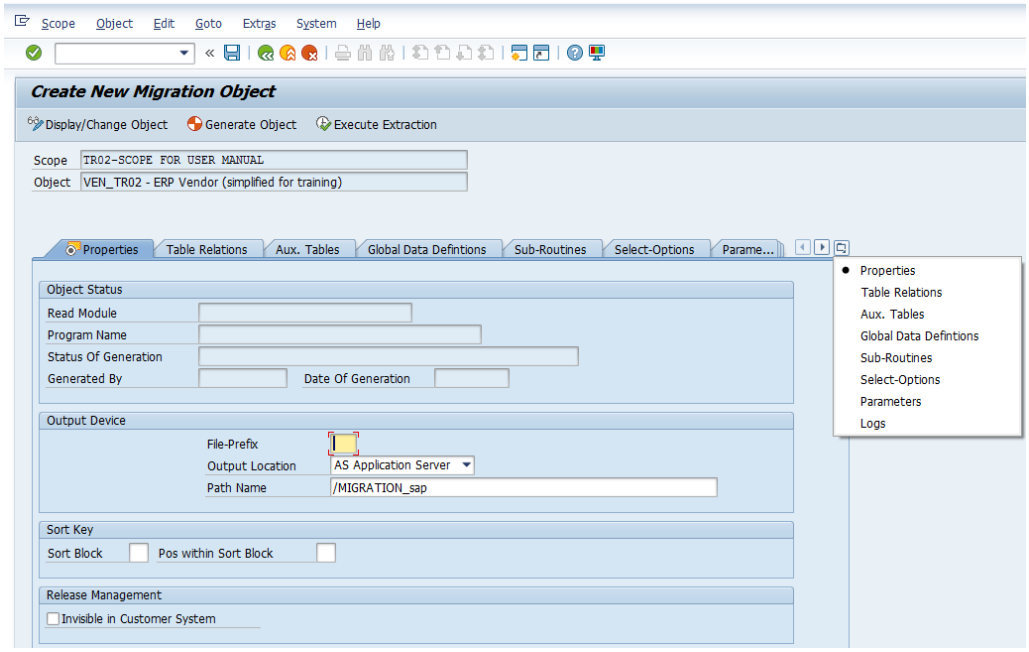


Figure 2-4

- *Properties*
  - Display and define the properties of a migration scope
- *Table Relations*
  - Display and define the data model of the business object on table level
  - Additional functions on table level (long text, pre- and post-processing)
- *Aux. Tables*
  - Display and define auxiliary tables (customizing only) that should be attached to the migration object
- *Global Data Definitions*
  - Data definitions available on object level
- *Sub-Routines*
  - Form routines accessible on object level
- *Select-Options*
  - Definition of select options for the data extraction program
- *Parameters*
  - Definition of parameters for the data extraction program
- *Logs*
  - Display log files and extraction files

After saving the new object, the migration object's properties screen is displayed. As on the source scope level, you can make default settings on migration object level. Note that the fields on this screen are not mandatory because the tool will use defaults if required.
The most important fields on this screen are explained in figure 2-5 below.

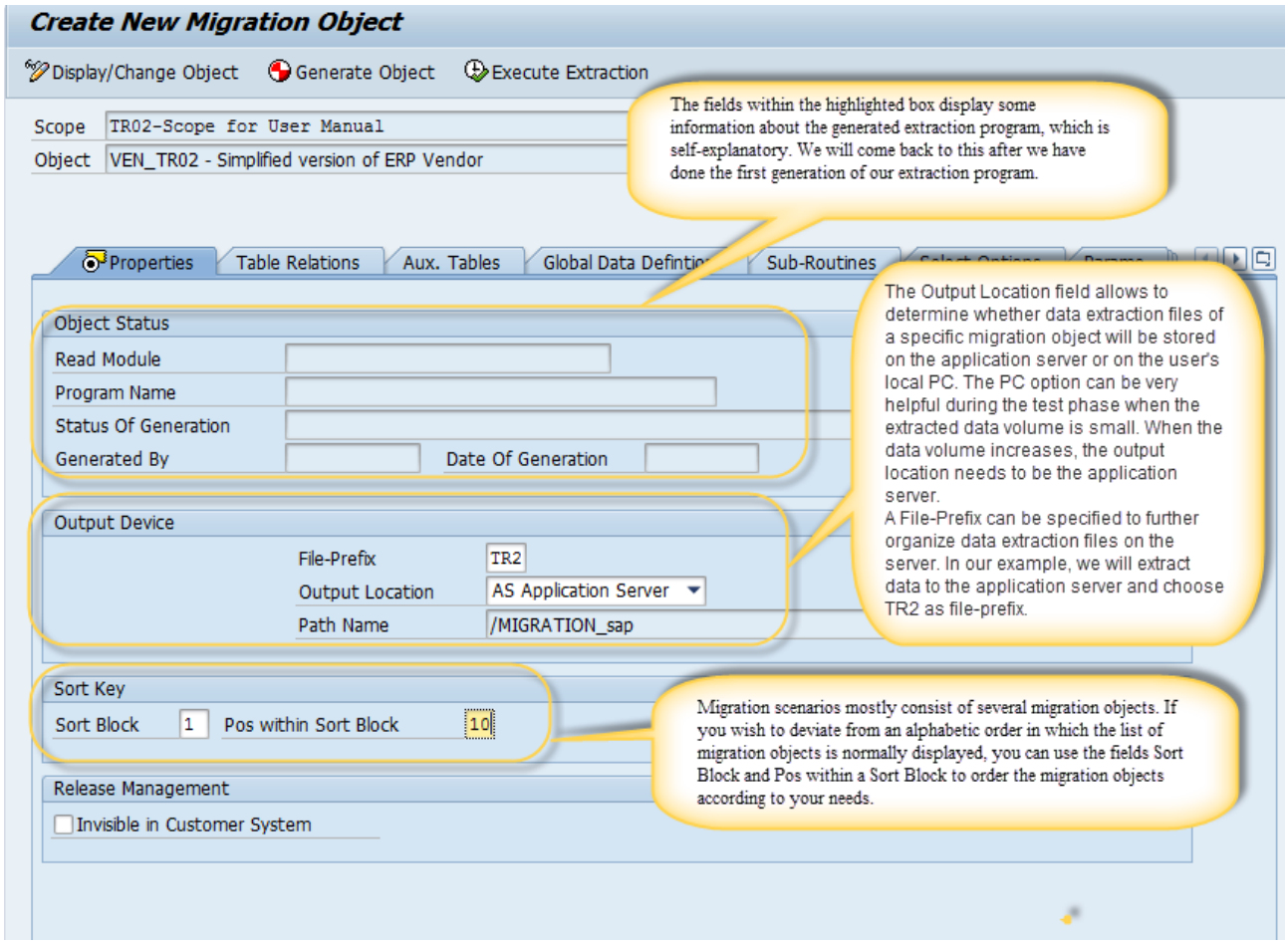


Figure 2-5

2.1.3.3 Migration Object Level - Table Relations Tab

DTMT follows a generation approach; therefore, the data model of a particular business object must be defined in DTMT design time. The program generator will later use those definitions to generate the final extraction program while taking into account customer enhancements to tables that are involved in the migration object.
Clicking the *Table Relations* tab opens a screen where you can enter all involved tables and the foreign key relations between them, as well as other settings on table level that might be required to solve more specific migration requirements. Even though you need to implement the complete data model of the example object **VEN_ TR02**, the focus here is on the first 4 tables, since this is sufficient to demonstrate the functionality.
The following series of screenshots leads you through the implementation process.

2.1.3.3.1 Maintain Table Relations for Level 1

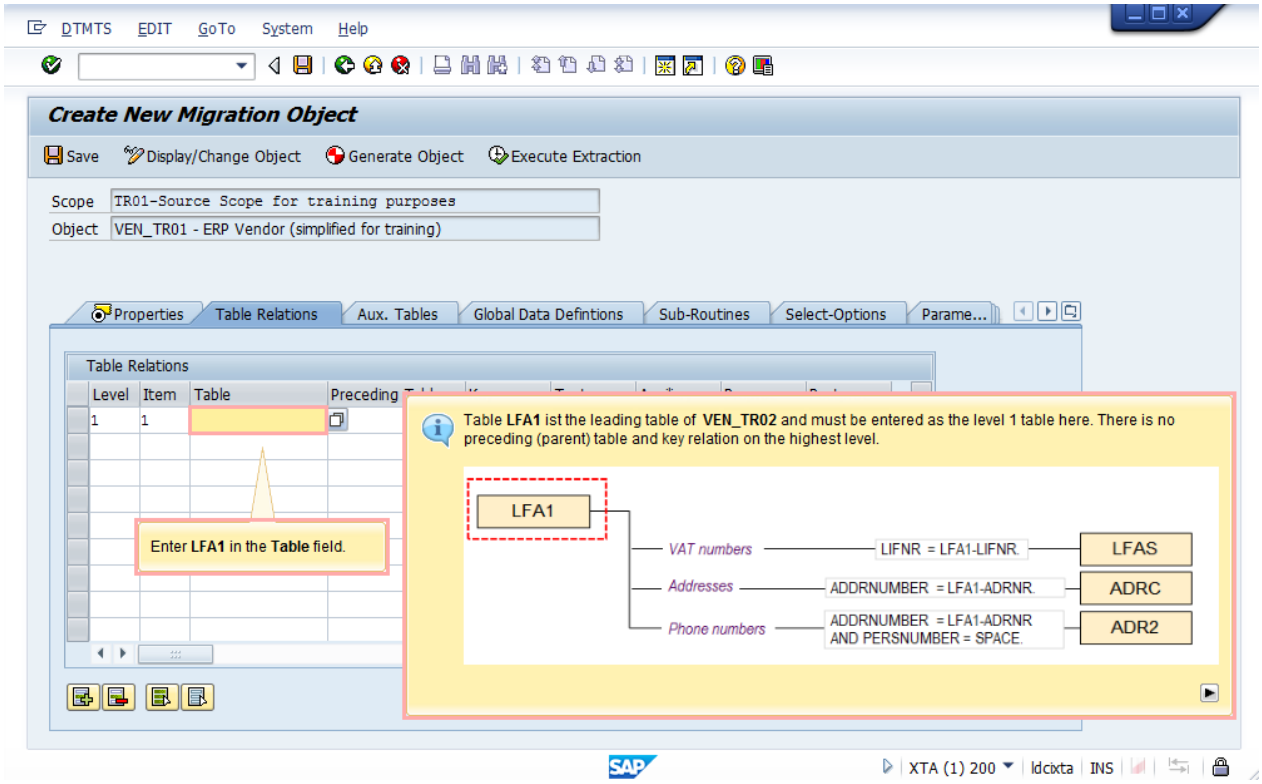


Figure 2-6

2.1.3.3.2 Maintain Table Relations for Table LFAS on Level 2

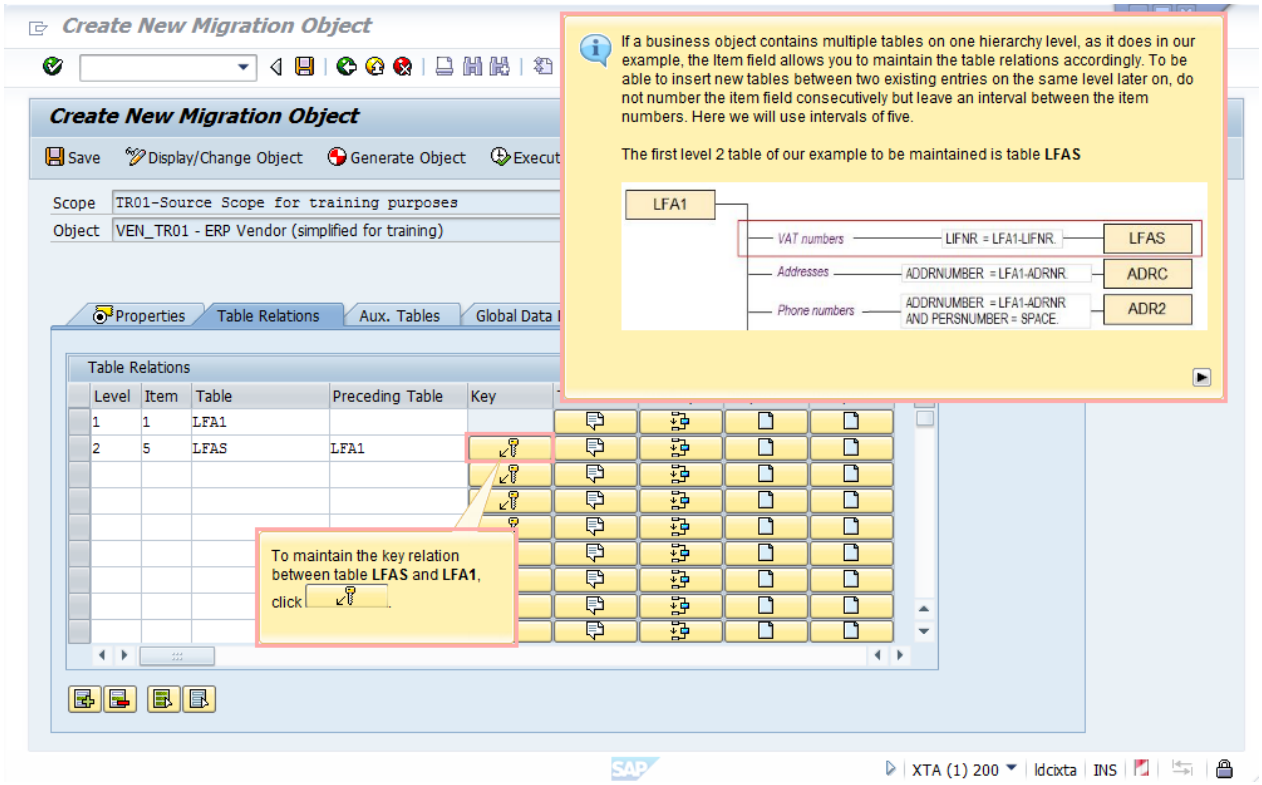


Figure 2-7

Maintain *Foreign Key Relations* using *Pattern* functionality (LFA1/LFAS)

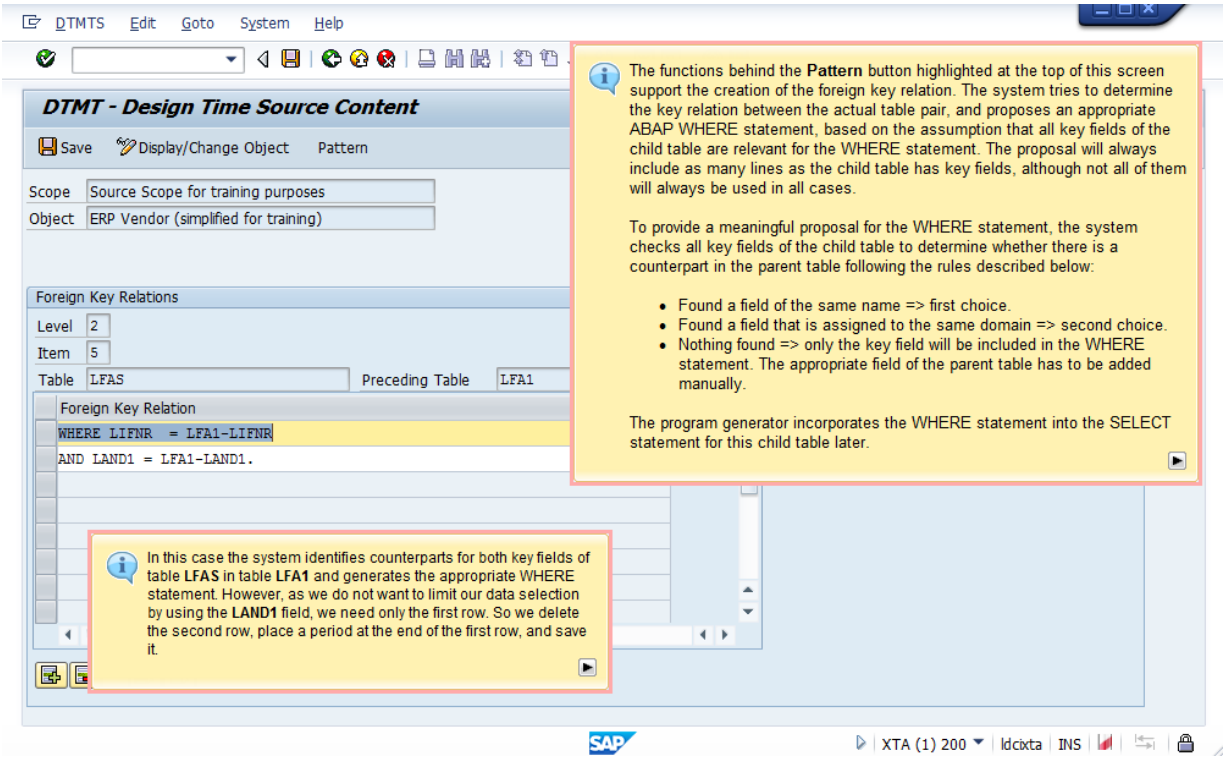


Figure 2-8

2.1.3.3.3 Maintain Table Relations for Table ADRC on Level 2

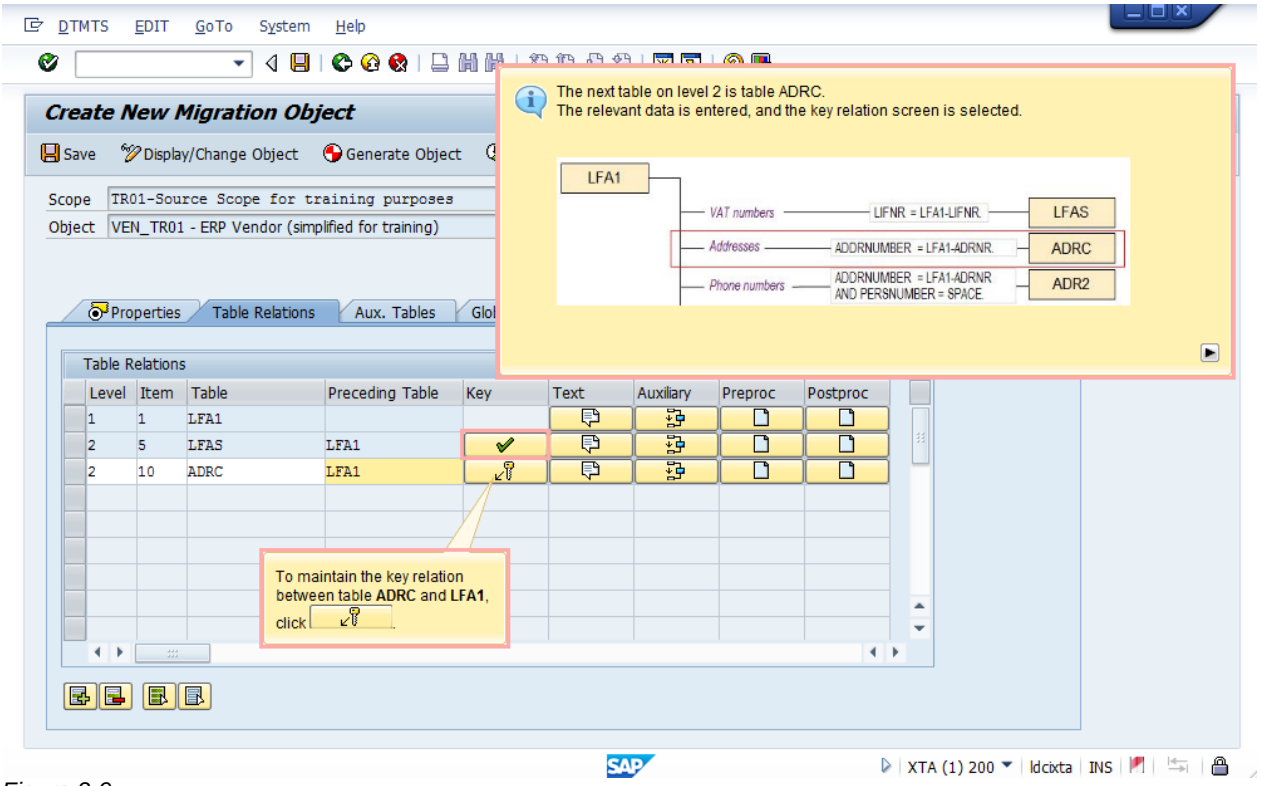


Figure 2-9

2.1.3.3.4 *Maintain Foreign Key Relations Using Pattern Functionality (LFA1/ADRC)*

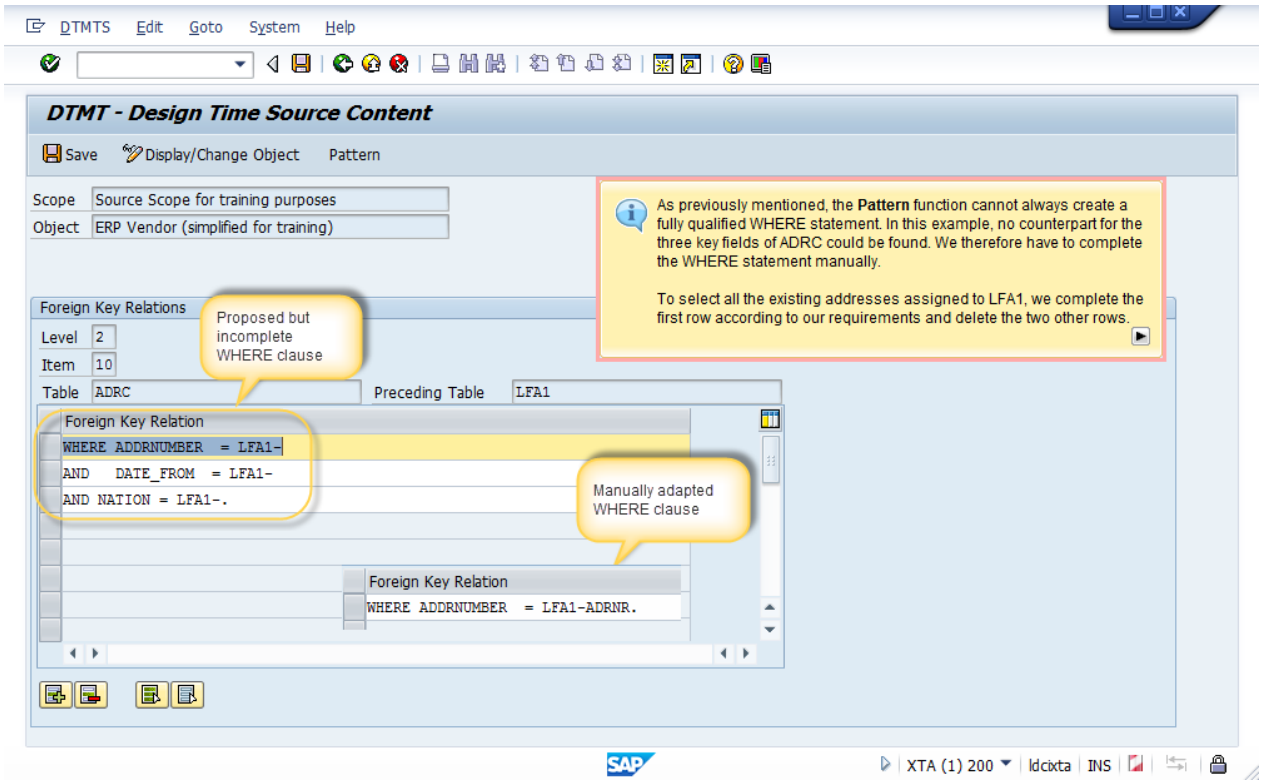


Figure 2-10

2.1.3.3.5 *Maintain Table Relations for Table ADR2 on Level 2*

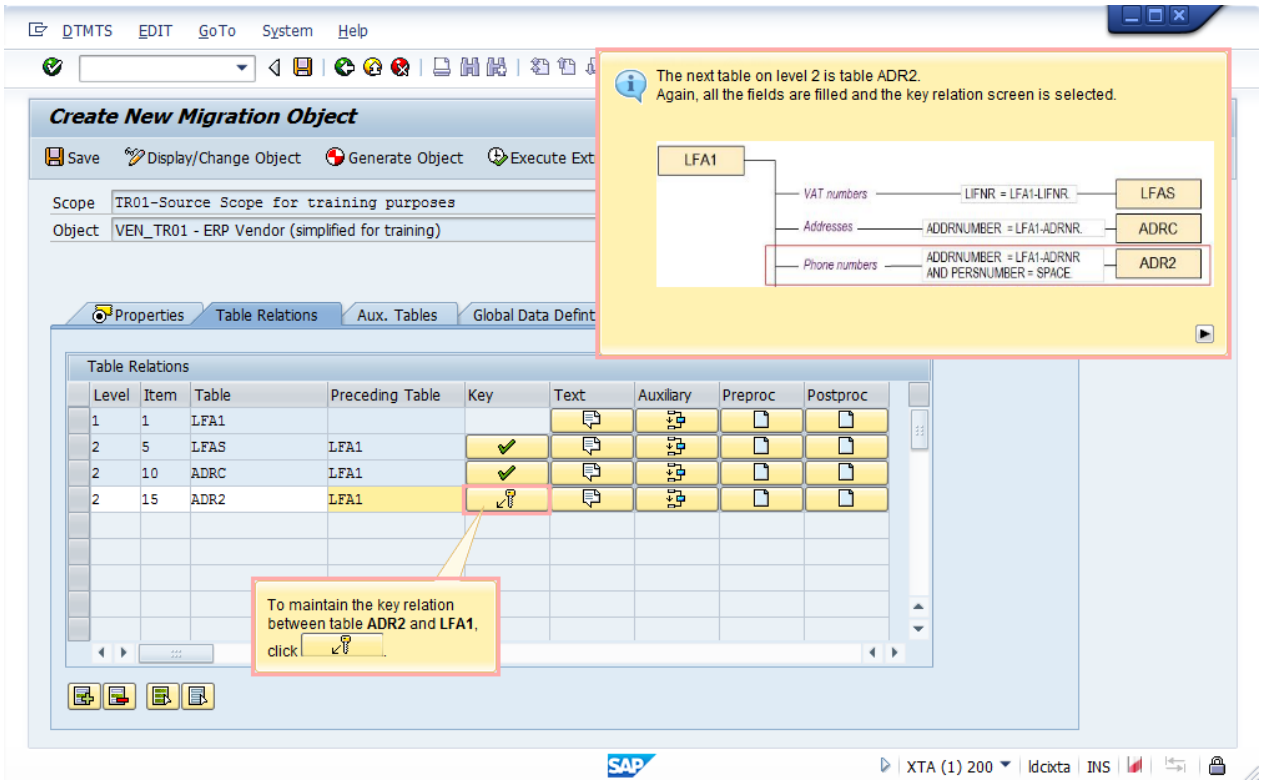


Figure 2-11

2.1.3.3.6 Maintain Foreign Key Relations Using Pattern Functionality (LFA1/ADR2)

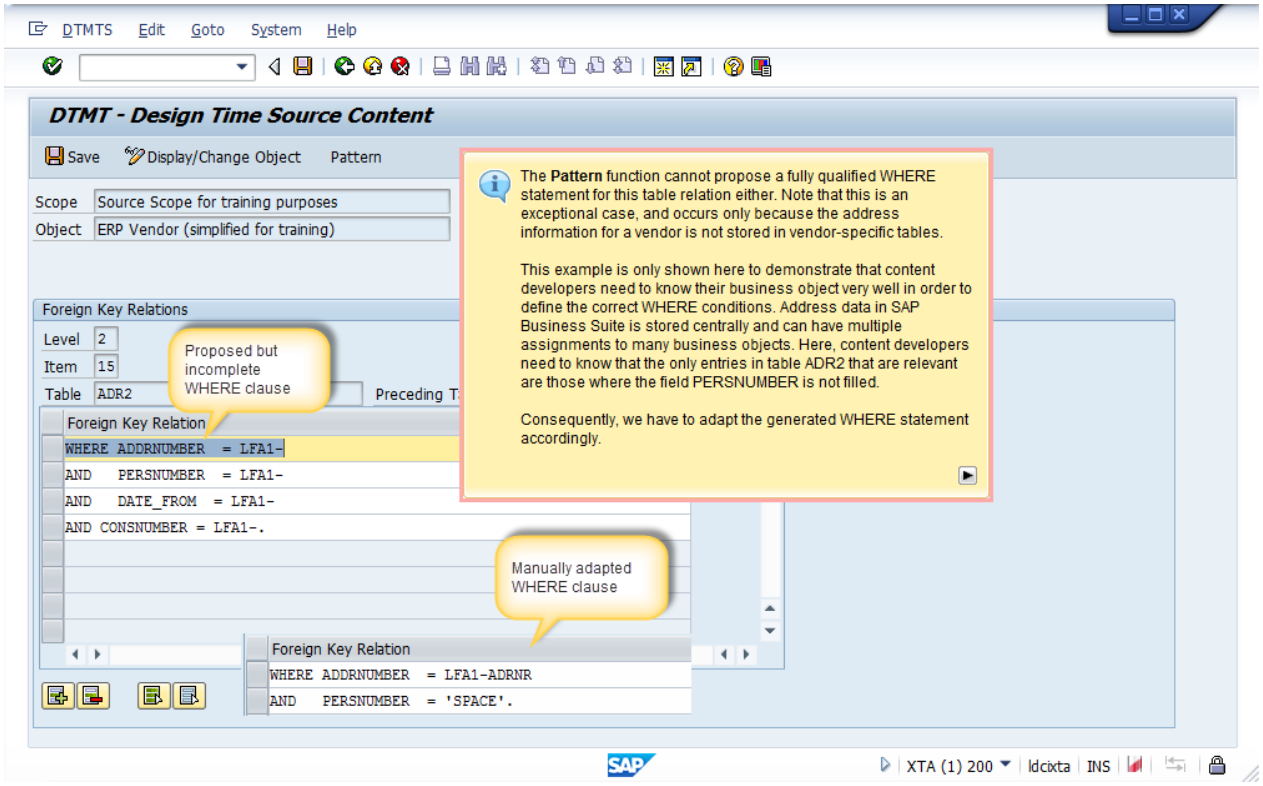


Figure 2-12

Implementing the other table relations for the example object does not differ greatly from what has been explained so far. Therefore, the figure below shows the point where all the table relations have been entered.

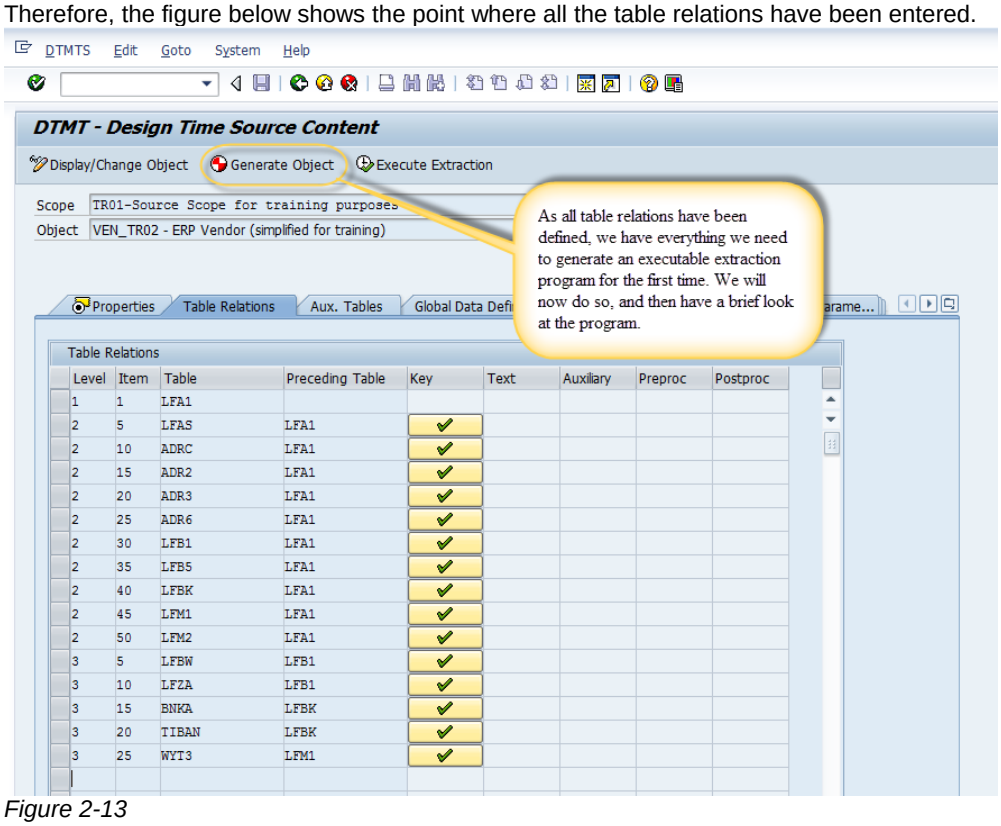


Figure 2-13

Since the main part of the data model is now implemented in DTMT, a first generation of the extraction program should work. However, before you generate the program and look at the coding, you need to understand what happens during the generation process and what the mode of operation of a DTMT extraction program is.

2.1.3.3.7 Generation Process of a DTMT Extraction Program

To understand the generated program, you need to know exactly what happens during object generation. The figure below gives an overview of the generation process.

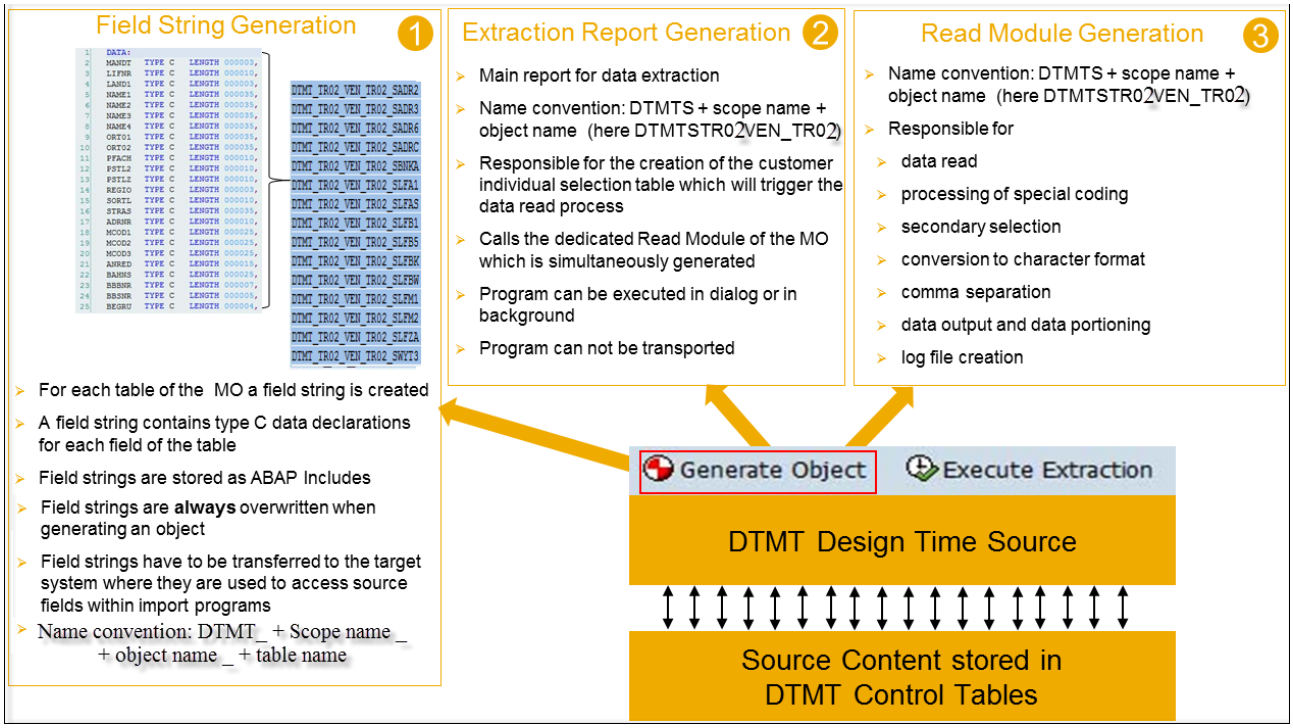


Figure 2-14

- Field String Generation
  - Whenever a migration object is generated, a field string is created for each table of the object. A field string is an ABAP-include program that contains all the data declarations of the actual table in its character format, that is, with data instructions of type C.
  - After the extraction program has read the database tables, data is moved into those field strings, and converted to character format at the same time. Conversion to character format ensures code-page independence.
  - All field strings have to be transferred to the target system, where they are used to access source fields within the import programs, for example, in the mapping part of the program.
  - Field string transportation via RFC is embedded in DTMT.
- Extraction Report Generation
  - The actual extraction report is created during the object generation process. This is the main program that is executed when you start the data extraction.
  - From a functional point of view, the main program is responsible for identifying all the keys that are relevant for data extraction based on the selection screen input of a user. All the keys are written to an internal selection table, which is then handed over to the actual read module where the data selects are executed.
- Read Module Generation
  - The generated read module is called by the extraction report and works as follows:
    1. Loop at selection table handed over from the extraction report
    2. Execute the selected statements that have been generated based on the definitions on the *Table Relations* tab in transaction **DTMT_DTS**
    3. Execute special coding if applicable
    4. Convert all data into character format
    5. Write the relevant metadata to the CSV file
    6. Write all data to the CSV file

2.1.3.3.8 Data Extraction Process

When you start a data extraction program, you first have to fill the selection screen of the main program, where you specify which data is to be extracted. You will learn exactly how this works later on in this manual.

For now, it is sufficient to understand that the main program creates an internal table that contains all the keys of the leading table of the migration object that have to be extracted. In this case, this internal selection table includes the respective **LFA1** vendor numbers (**LFA1-LIFNR**). The table is handed over to the read module, and triggers the actual data selection process.
To keep memory consumption within acceptable limits, the read process is interrupted after a certain number (100) of level 1 table records are read (that is, 100 complete LUWs). In this case, the internal data record table is emptied to the flat file, and allocated memory is released. The process is repeated until all the data has been read. The data write routine also covers file management, which is responsible for splitting files after a certain threshold value in megabytes is reached. This means that a data extraction will sometimes create more than one flat file. This feature allows the parallelization of import runs in the target system, which is a very good option for optimizing performance on the import side.

The following flowchart shows the actual data extraction process:

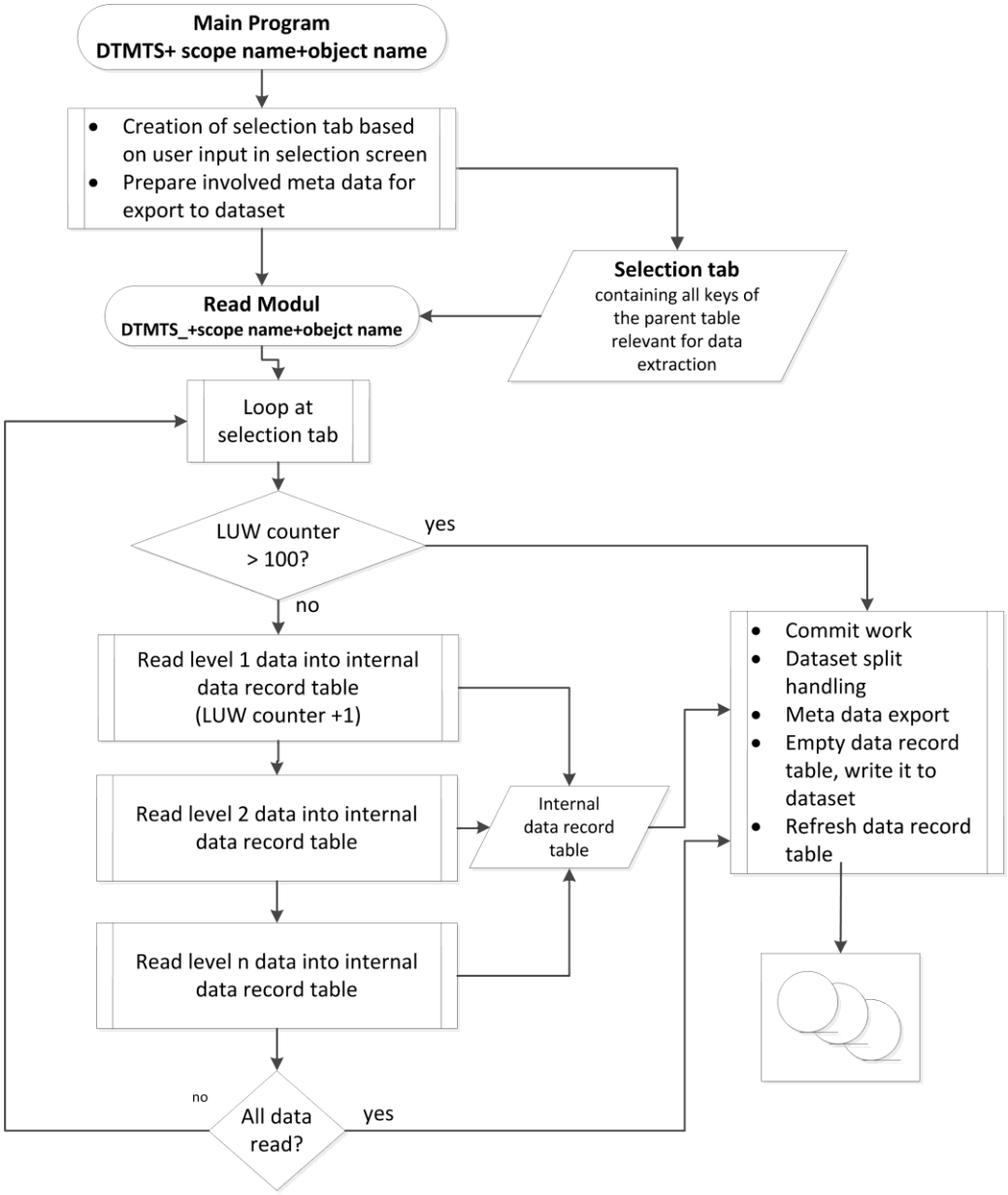


Figure 2-15

2.1.3.3.9 *The Data Extraction Program (Based on Table Relation Settings Only)*

After the extraction program is generated, the first field of the migration object's *Properties* tab is filled. Double-clicking the read module name displays the program.

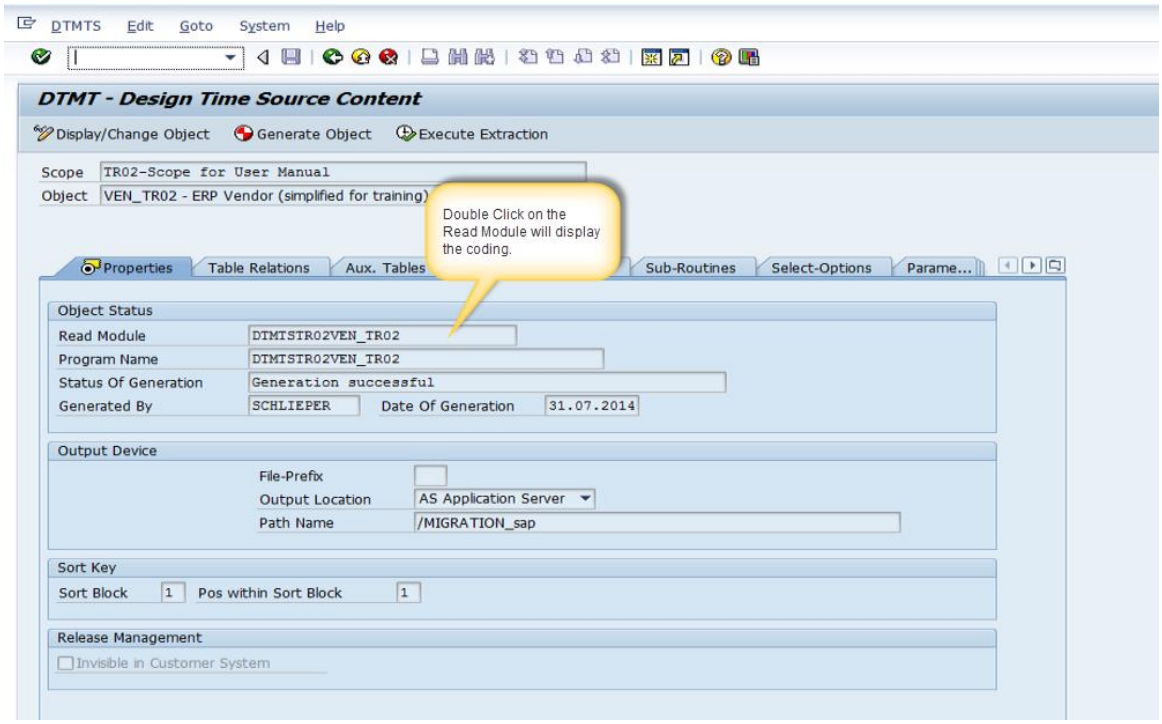


Figure 2-16

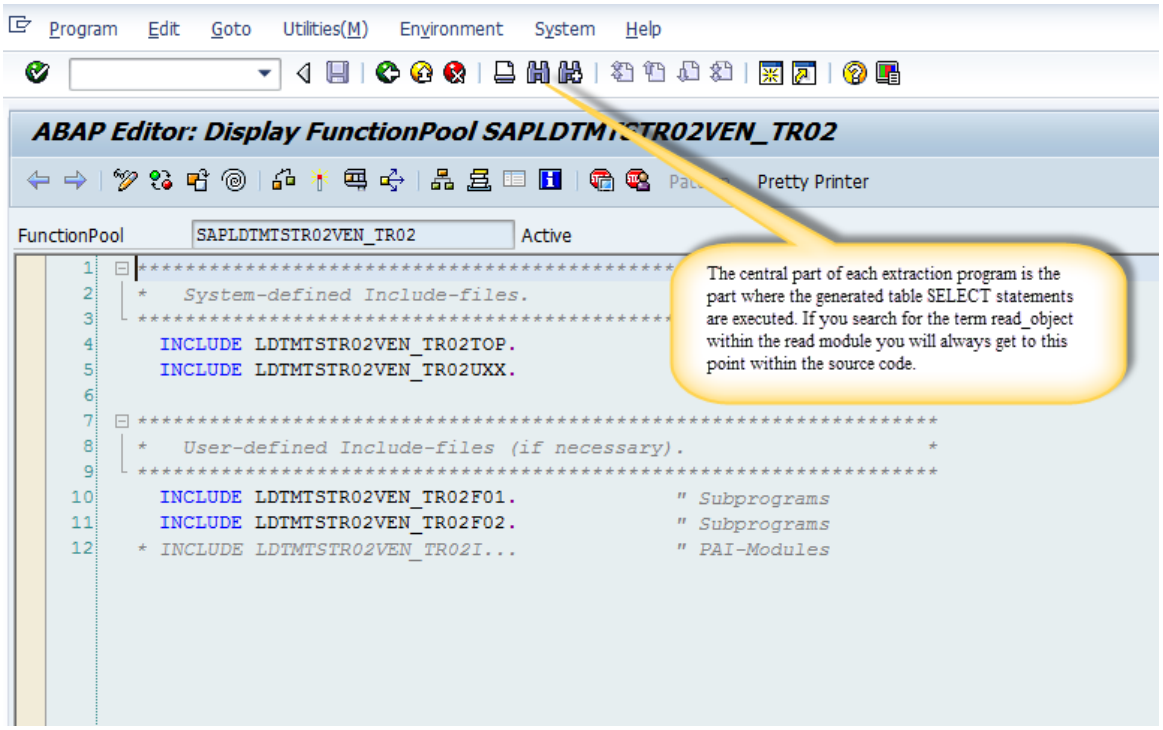


Figure 2-17

Below you see parts of the coding of the form routine **READ_OBJECT_STR02VEN_TR02**, where the data read for the example object takes place. The program generator creates nested select statements according to the settings on the *Table Relations* tab of **DTMT_DTS**. This coding passage shows only the first two levels but the other levels are equal to those you see below. As already shown in the flowchart, you can also see when data is written to the CSV file.

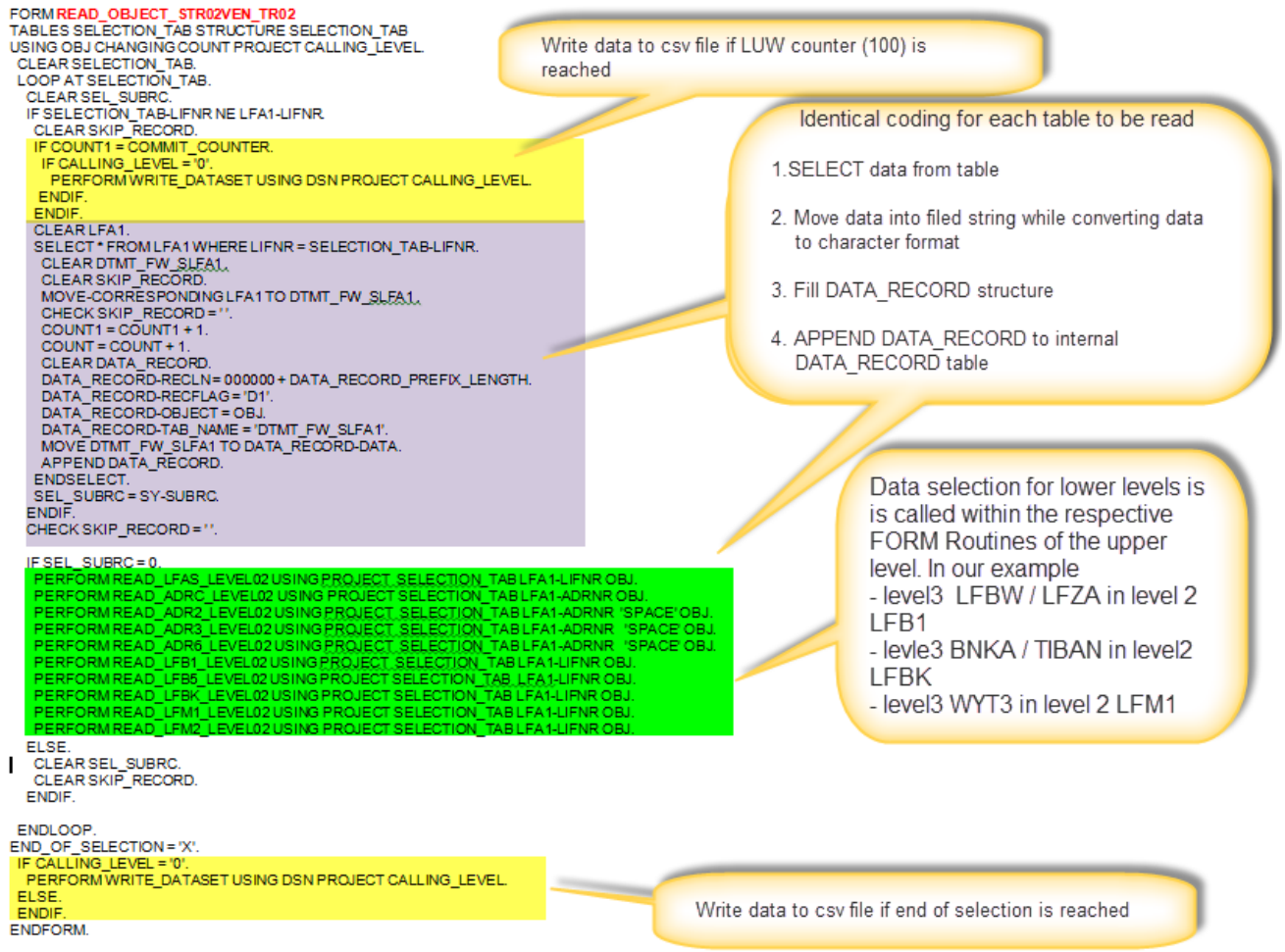


Figure 2-18

Note: The nested select structure of the DTMT extraction program is generated based on the settings made on the *Table Relations* tab that actually define the LUW (logical unit of work) of the respective migration object. It is important to know that data is always extracted in LUW format.

2.1.3.3.10 Executing Data Extraction (First Time)

As long as you are navigating on migration object level, you are able to start the extraction process by pressing the button *Execute Extraction*. The screen below appears after you do this for the migration object **VEN_TR02**. In the figure below, you can see how the selection screen is filled before the execution is started.

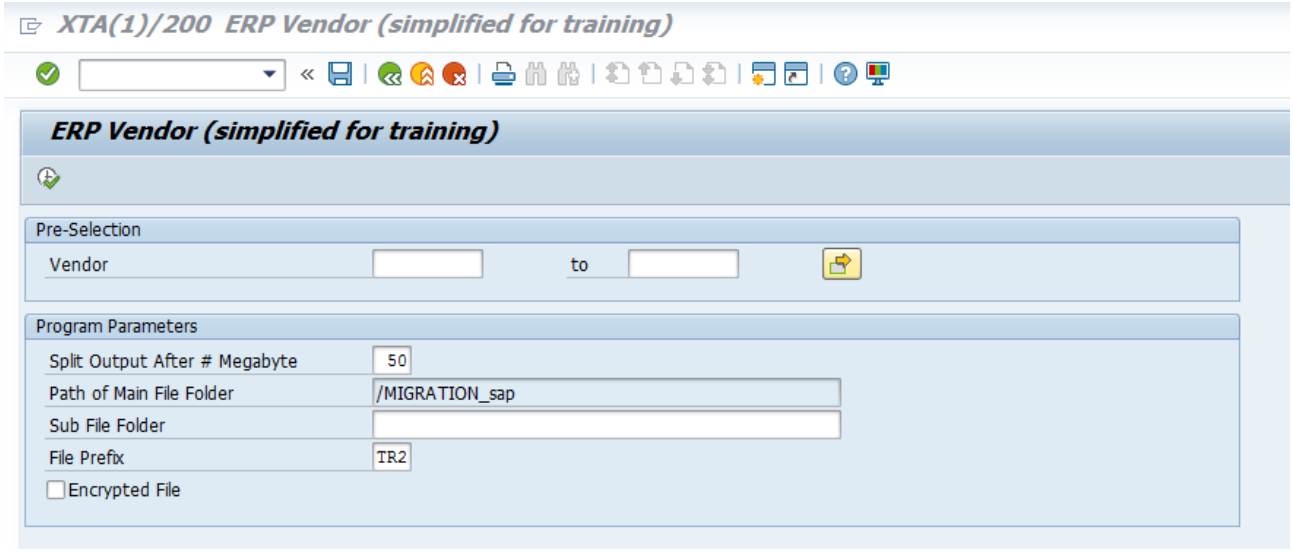


Figure 2-19

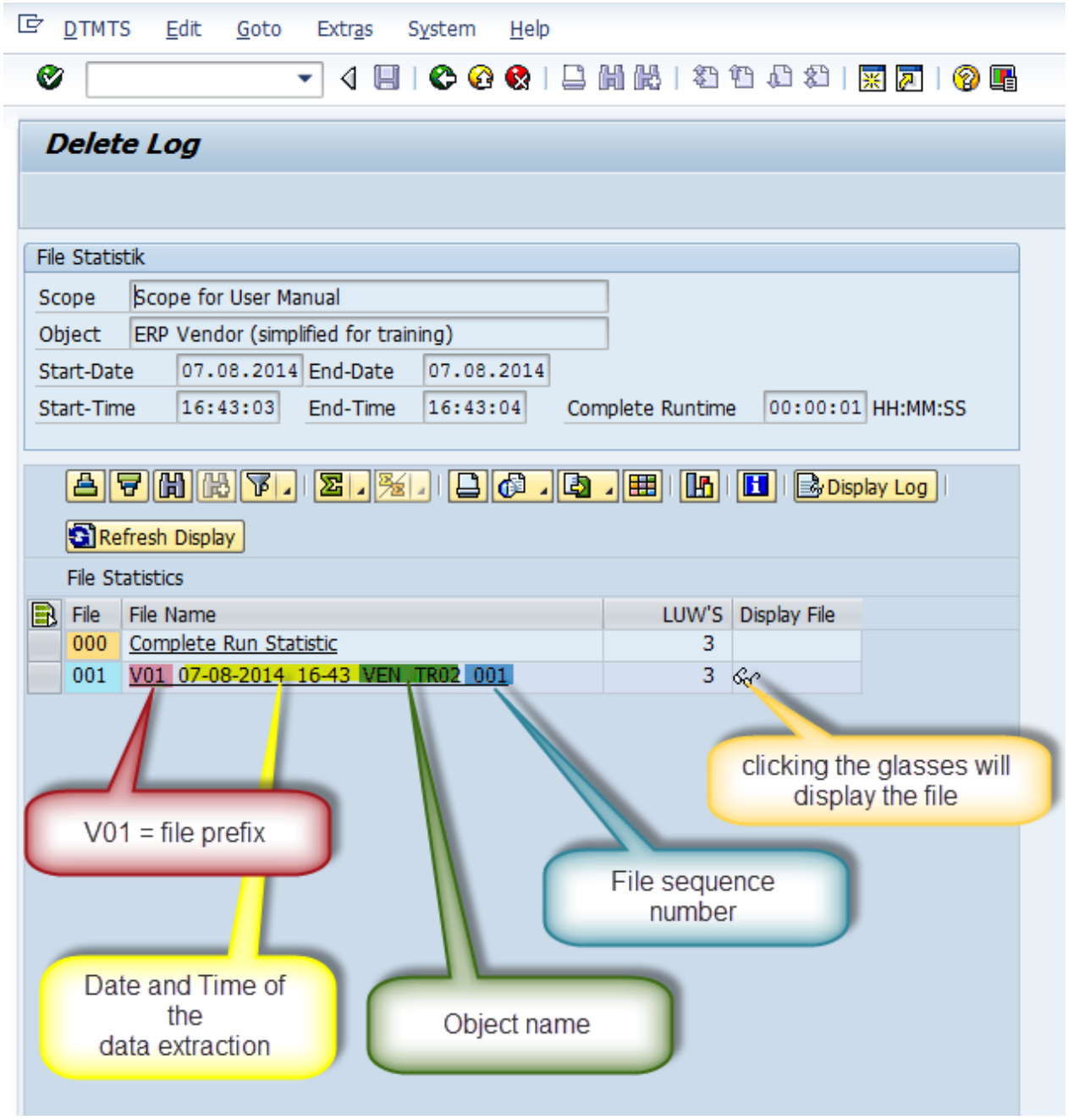
- In the *Pre-Selection* screen area, you can make a selection on vendor number level, as is usual within SAP systems. The definition of this select option is carried out automatically by the program generator. By default, all key fields of the leading table defined in the *Table Relations* of an object are defined as select options and stored within DTMT. You will later see how additional select options can easily be added.
  In this example, you want to extract only the three vendors **DE_VEND01**, **DE_VEND02**, and **DE_VEND03** so you fill the pre-selection field with **DE_VEN*.**
- In the *Program Parameters* screen area, which is the same in all extraction programs, there are four fields which are explained below:
  - *Split Output After # Megabyte*
    DTMT is able to distribute the extracted data to multiple files. The size of one file can be defined here. This feature can be very helpful when it comes to performance optimizations.
    The tool will ensure that each of the extracted files is consistent in itself. Therefore, the real file size will always slightly exceed the limit because all data belonging to an LUW has to be completely extracted before the file can be closed.
    Here, the default value of 50 MB is kept.
  - *Path of Main File Folder*
    The name of the path where the file is stored (taken over from the scope properties)
  - *Sub File Folder*
    It might be very helpful to subdivide the main folder into subfolders for example, to differentiate between test migration runs or to collect files of a content developer. Note that the subfolders are not created if they do not exist. They have to be created manually in advance.
    Enter the value **/TR02**. Note that you **always** have to maintain the leading slash yourself.
  - *File Prefix*
  - The file prefix is an additional option to administrate extraction files on the server. It is always placed in front of the tool-generated file name.
    The value **V01** is entered, since this is the first data extraction
  - *Encrypted File*
    The file will be created in an unreadable format, with base64 encoding

The program ends with a message showing the number of LUWs exported.

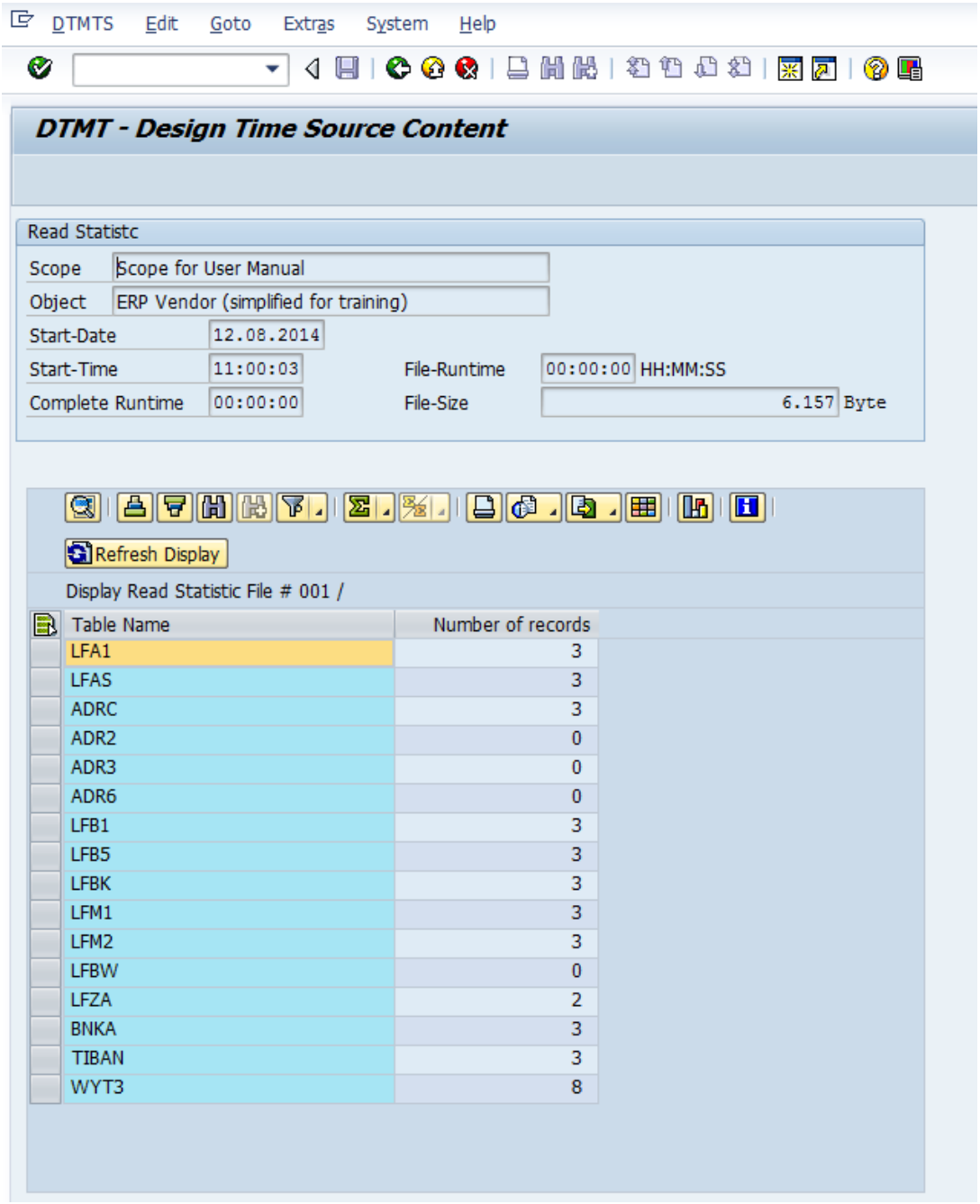


Data Extraction Log

You now navigate to the *Logs* tab on migration object level, and display the log of data extraction that has just finished. The figure below shows the log screen, and describes the name structure of the extracted file or files.



You always get a log for each file that has been created (remembering the file split functionality) during an extraction run, and a summary statistic for the complete run. In this case, because there is only one file, the complete run statistic is equal to the file statistic. You also see the file name that has been generated by the tool. The glasses icon on the right side allows you to display the file, which is sometimes very helpful. Clicking the file name displays the log.



The figure above shows the file log. It just shows the number of records extracted per table.

2.1.3.3.11 Data Extraction File

Click the file name on the file statistic screen to display the file.

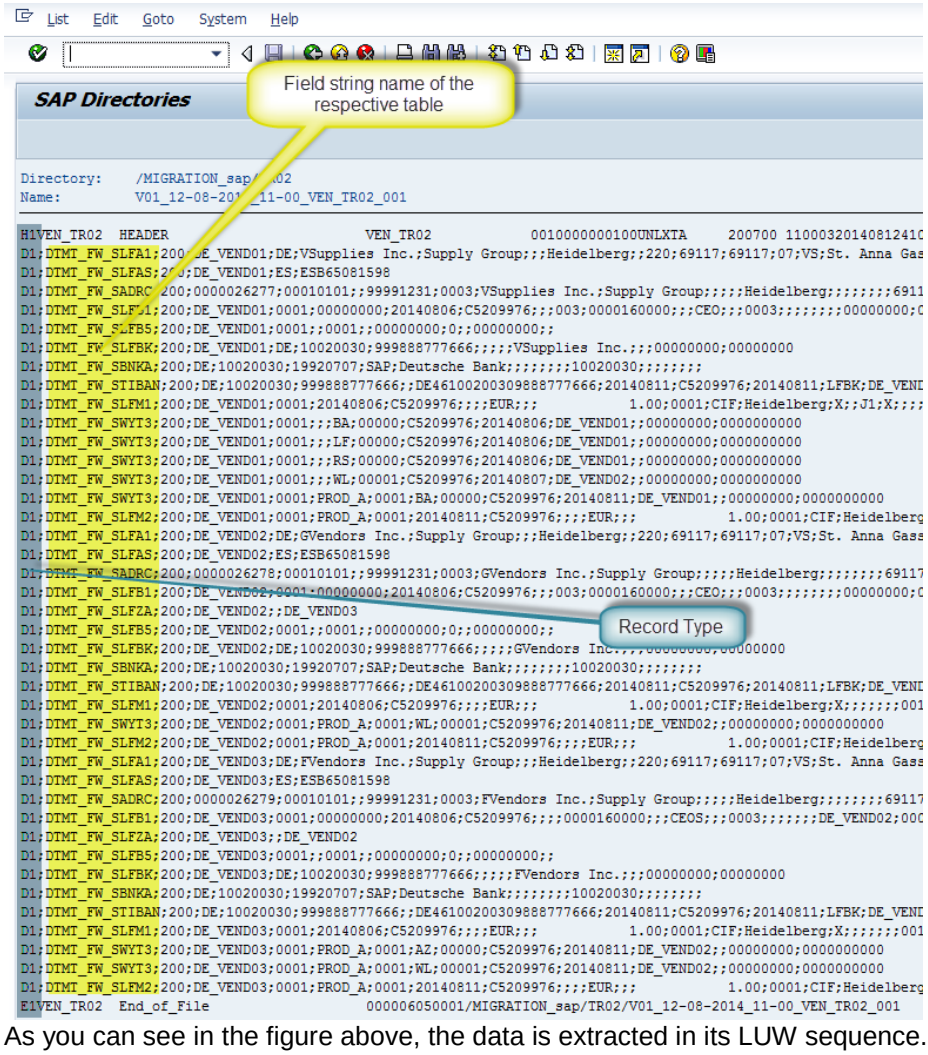
Besides the pure data records, the file includes other information that is needed for administration purposes. The different records are separated by a two-digit record type at the beginning of each record, as follows:

Rec. Type	Description
H1	File header record at the beginning of each file, containing general information about object name, file number, commit counter, code page, and so on.
HT	Data records of auxiliary tables (help tables)
D1	The actual table data (each D1 record has a record header in front, containing the field string name of the respective table)

M1	Collection of included material numbers in the current files
	needs additional coding in post-processing:
	`DATA_RECORD_MATNR-RECFLAG = 'M1'.`
	`DATA_RECORD_MATNR-OBJECT = 'MATERIALS'.`
	`DATA_RECORD_MATNR-TAB_NAME = 'MATNR'.`
	`MOVE LS_SOURCE_"TABLENAME"-MATNR(18) TO DATA_RECORD_MATNR-DATA.`
	`APPEND DATA_RECORD_MATNR.`

E1	End of file record

Displayed file:



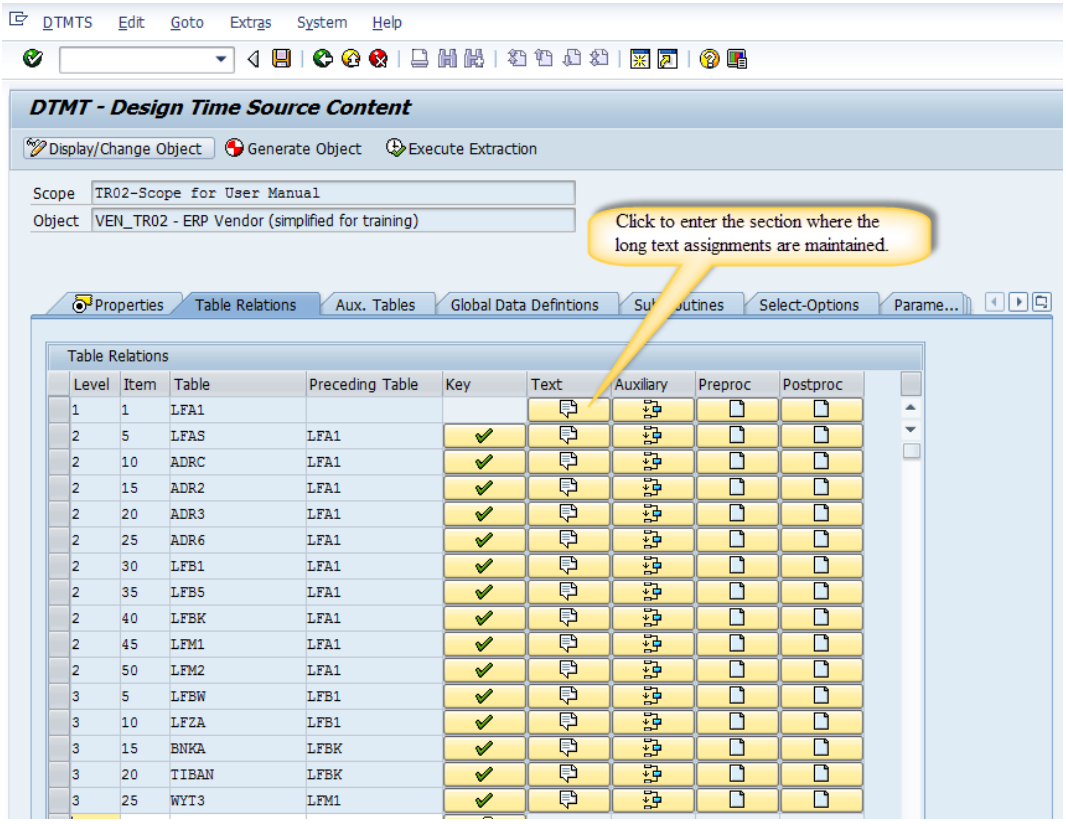
As you can see in the figure above, the data is extracted in its LUW sequence.

2.1.3.3.12 Long Text Extraction

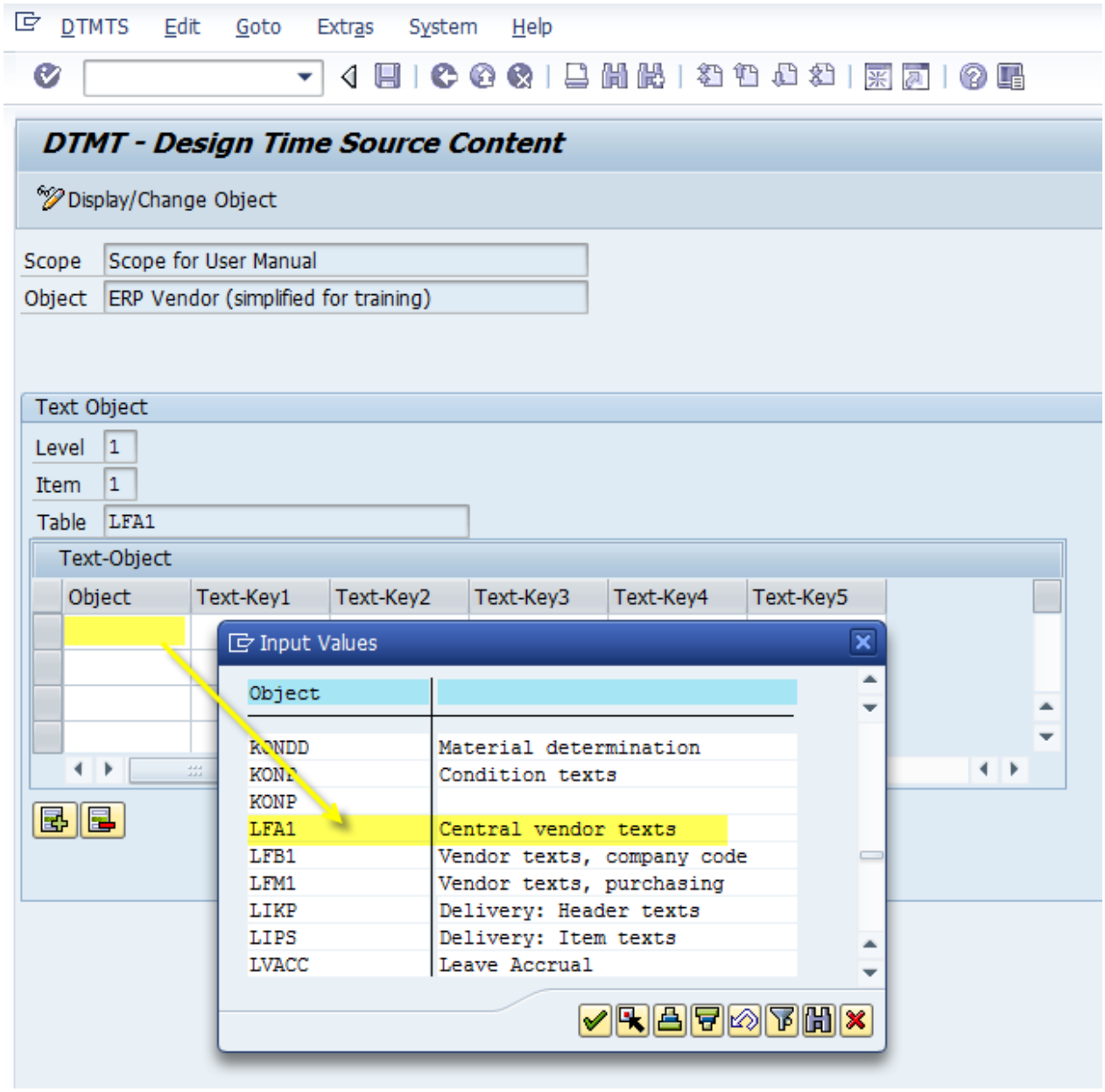
Looking at the data model of the example object, there are long texts assigned to table **LFA1** and **LFB1**. In a similar way to addresses, long texts are not stored within the tables of the actual business object but centrally, in two tables named **STXTH** (text header) and **STXTL** (text line). For this example, only table **STXTH** is of importance, since the association between the hosting table of the business object and the associated text is achieved via the key of table **STXTH**, the key structure of which is displayed below.

Field	Key	Data Type	Length	Short Description
MANDT	☑	CLNT	3	Client
TDOBJECT	☑	CHAR	10	Texts: Application Object
TDNAME	☑	CHAR	70	Name
TDID	☑	CHAR	4	Text ID
TDSPRAS	☑	LANG	1	Language Key

Within DTMT, only **TDOBJECT** (which stores the name of the application text object) and **TDNAME** (which is filled with the key fields of the hosting table, to ensure the unique association of a single application table) are relevant. Since there are no naming conventions for text objects, a content developer needs to know which text objects belong to a migration object. Very often, text objects are named according to the name of the corresponding hosting table, but sometimes they are given a meaningful name such as **MATERIAL**. The next steps demonstrate how this can be implemented in DTMT.
On the *Table Relations* tab, switch to change mode and click the *Text* field of the **LFA1** row.

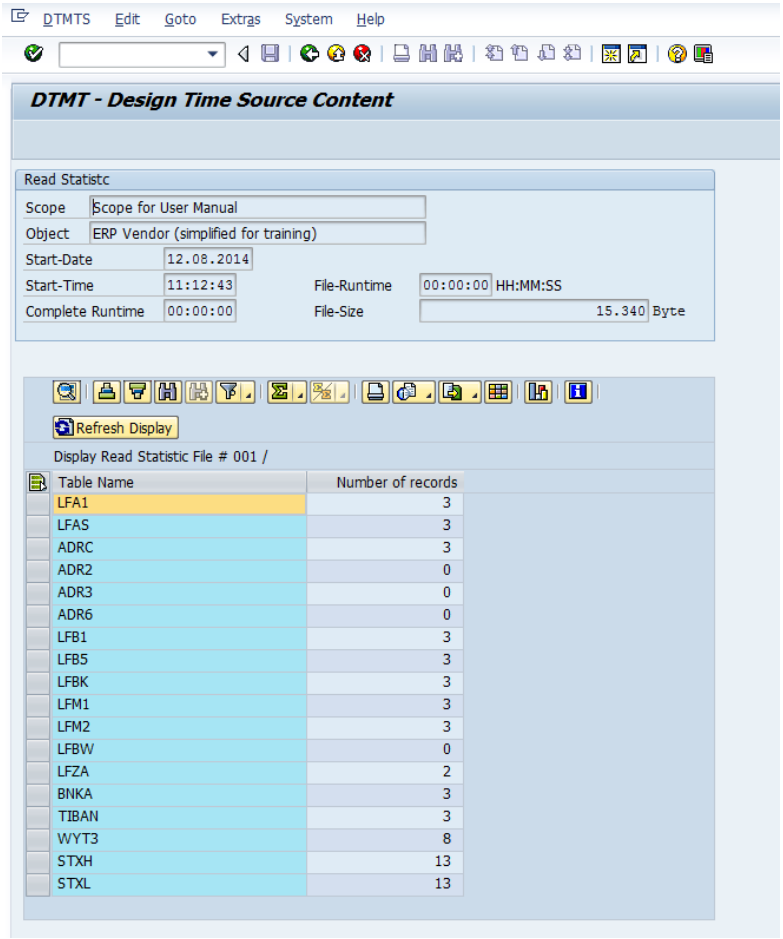


There may be more than one long text assigned to a hosting table. Therefore, the UI allows maintenance of multiple text objects. Customers, for example, would have to add their own text objects within their migration project.



In this example, you need to fill the name of the text object and the key fields of **STXTH** for the long text assigned to **LFA1**. If the name of the text object is not known, there is a value help available. You use this value help, search for the string “vendor”, and get the result shown above. After the *Object* field (**TDOBJECT=LFA1**) is filled, the key fields (concatenated with DTMT) and the **TDNMAE** key field of **STXH** are built.
In this case, since table **LFA1** has only the vendor number **LIFNR** as its key, only *Text-Key1* has to be filled. The text assignment on **LFB1** level works in a similar way. Since the key of table **LFB1** consists of the vendor number and company code, *Text-Key1* and *Text-Key2* have to be maintained.

After the long text assignment for **LFA1** and **LFB1** is done, you generate and run the program again. Below you see the statistic of this run, where the **STXH** and **STXL** records are now extracted.

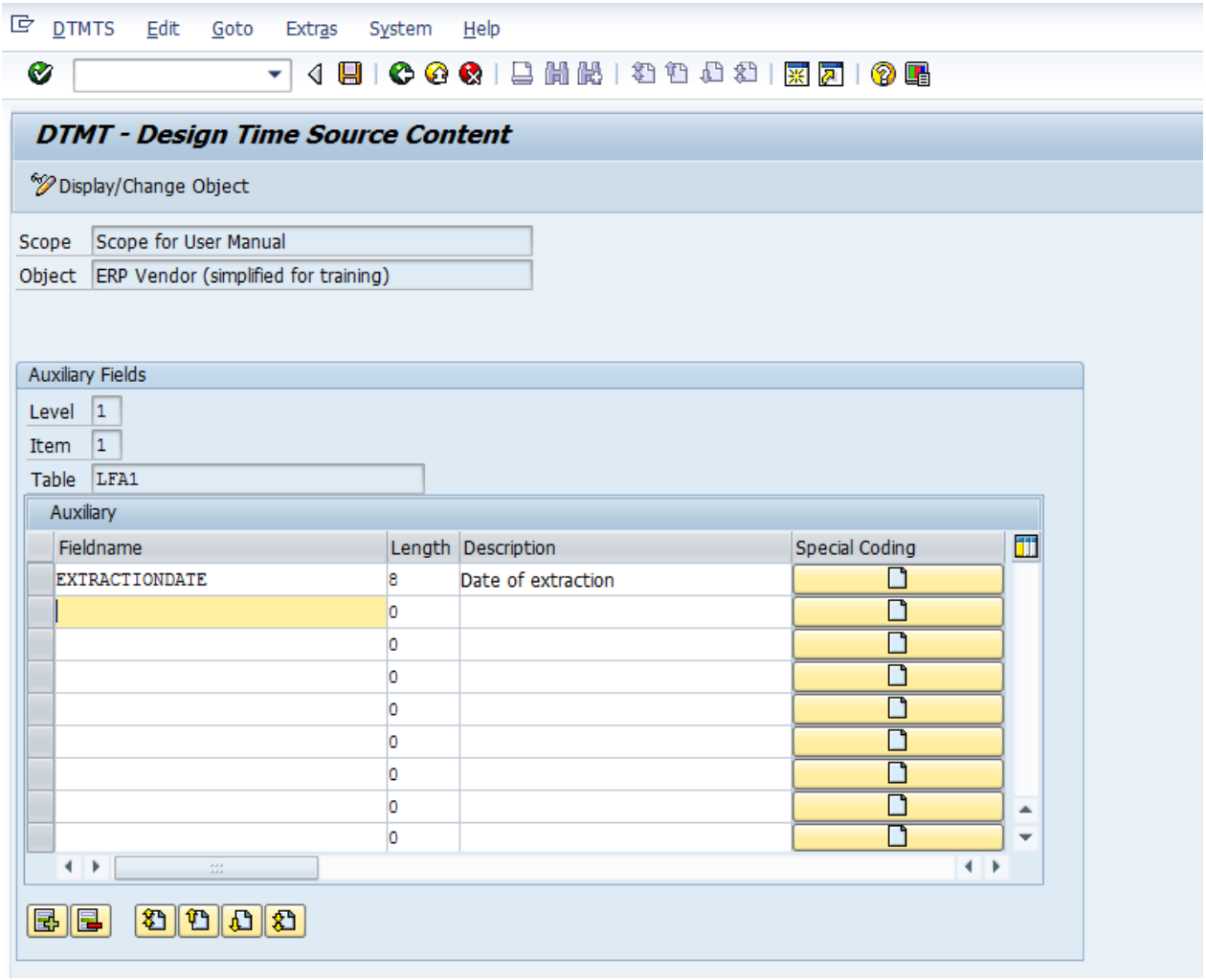


The following figure shows the first LUW for demonstration purposes:

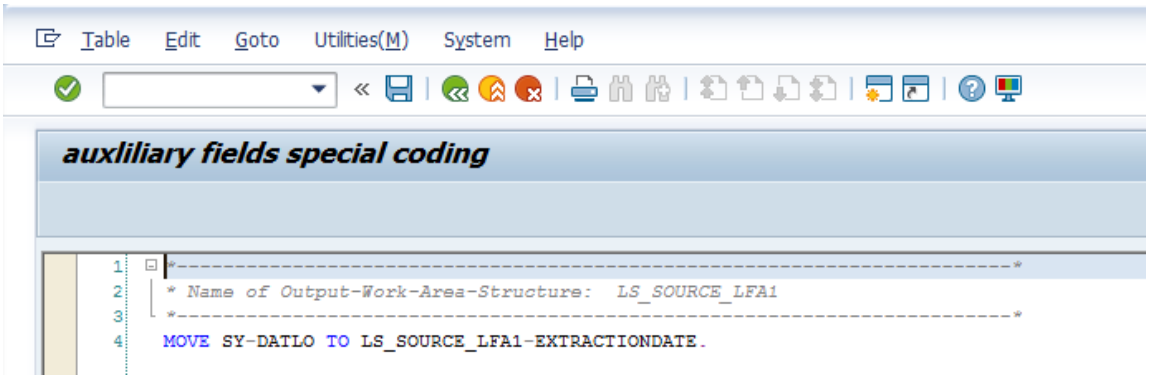
```
H1VEN_TR02  HEADER                      VEN_TR02           0010000000100UNLXTA     200700 111243201408124103
D1;DTMT_FW_SLFA1;200;DE_VEND01;DE;VSupplies Inc.;Supply Group;;;Heidelberg;;220;69117;69117;07;VS;St. Anna Gass
D1TEXTS     STXH                        200LFA1      DE_VEND01
D1TEXTS     STXL                        200TXLFA1      DE_VEND01
D1TEXTS     STXH                        200LFA1      DE_VEND01
D1TEXTS     STXL                        200TXLFA1      DE_VEND01
D1TEXTS     STXH                        200LFA1      DE_VEND01
D1TEXTS     STXL                        200TXLFA1      DE_VEND01
D1TEXTS     STXH                        200LFA1      DE_VEND01
D1TEXTS     STXL                        200TXLFA1      DE_VEND01
D1;DTMT_FW_SLFAS;200;DE_VEND01;ES;ESB65081598
D1;DTMT_FW_SADRC;200;0000026277;00010101;;99991231;0003;VSupplies Inc.;Supply Group;;;;Heidelberg;;;;;;;;69117
D1;DTMT_FW_SLFB1;200;DE_VEND01;0001;00000000;20140806;C5209976;;;003;0000160000;;;CEO;;;0003;;;;;;;;00000000;00
D1TEXTS     STXH                        200LFB1      DE_VEND01 0001
D1TEXTS     STXL                        200TXLFB1      DE_VEND01 0001
D1;DTMT_FW_SLFB5;200;DE_VEND01;0001;;0001;;00000000;0;;00000000;;
D1;DTMT_FW_SLFBK;200;DE_VEND01;DE;10020030;999888777666;;;;;VSupplies Inc.;;;00000000;00000000
D1;DTMT_FW_SBNKA;200;DE;10020030;19920707;SAP;Deutsche Bank;;;;;;;;10020030;;;;;;;;
D1;DTMT_FW_STIBAN;200;DE;10020030;999888777666;;DE46100200309888777666;20140811;C5209976;20140811;LFBK;DE_VEND0
D1;DTMT_FW_SLFM1;200;DE_VEND01;0001;20140806;C5209976;;;;EUR;;;           1.00;0001;CIF;Heidelberg;X;;J1;X;;;;0
D1;DTMT_FW_SWYT3;200;DE_VEND01;0001;;;BA;00000;C5209976;20140806;DE_VEND01;;00000000;0000000000
D1;DTMT_FW_SWYT3;200;DE_VEND01;0001;;;LF;00000;C5209976;20140806;DE_VEND01;;00000000;0000000000
D1;DTMT_FW_SWYT3;200;DE_VEND01;0001;;;RS;00000;C5209976;20140806;DE_VEND01;;00000000;0000000000
D1;DTMT_FW_SWYT3;200;DE_VEND01;0001;;;WL;00001;C5209976;20140807;DE_VEND02;;00000000;0000000000
D1;DTMT_FW_SWYT3;200;DE_VEND01;0001;PROD_A;0001;BA;00000;C5209976;20140811;DE_VEND01;;00000000;0000000000
D1;DTMT_FW_SLFM2;200;DE_VEND01;0001;PROD_A;0001;20140811;C5209976;;;;EUR;;;           1.00;0001;CIF;Heidelberg;
D1;DTMT_FW_SLFA1;200;DE_VEND02;DE;GVendors Inc.;Supply Group;;;Heidelberg;;220;69117;69117;07;VS;St. Anna Gasse
```

2.1.3.3.13 Auxiliary Fields

Auxiliary fields are fields that can be added on table level without changing the data dictionary. They will be visible only in DTMT's own structures, known as field strings (see also section 2.1.3.3.7). An auxiliary field is now added to table **LFA1** to demonstrate the functionality. Within the *Table Relations* tab, you switch to change mode and click the auxiliary symbol in the **LFA1** row. In the screen below, an eight-byte-long auxiliary field is defined, named **EXTRACTIONDATE**.

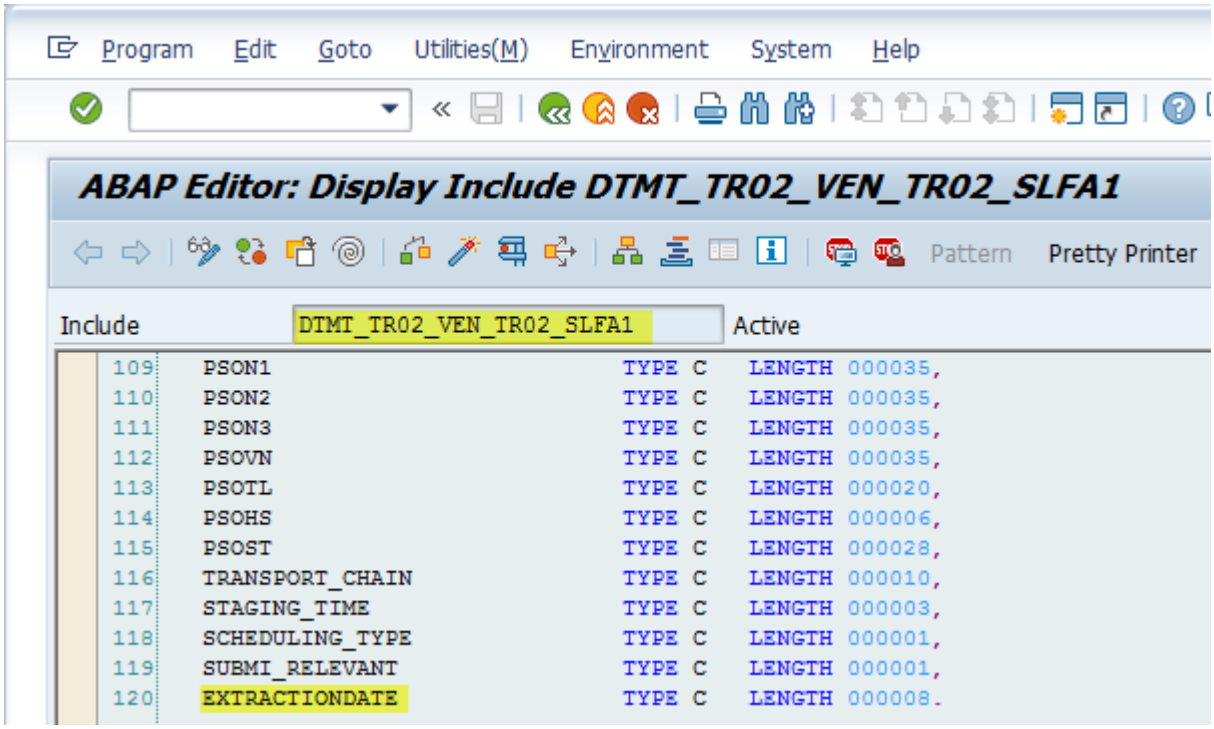


By pressing the *Special Coding* button, the ABAP editor will open and allow you to implement coding to fill the new field. In this case, you just move **SY_DATLO** to **EXTRACTIONDATE** and press *Save*.

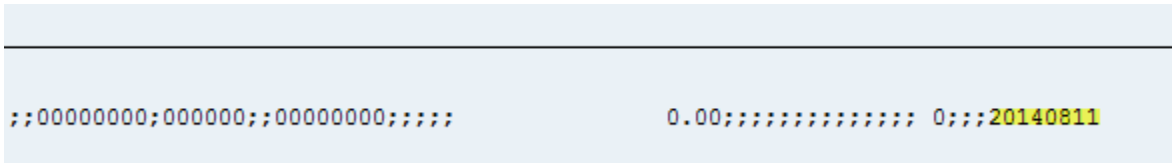


Again the program is generated; look at the field string of **LFA1** and see whether the auxiliary field is filled and written to the file.

In the figure below, you see that the bottom of the field string **DTMT_TR02_VEN_TR02_SLFA1** now includes the auxiliary field that you just defined.



The figure below shows the end of the first **LFA1** record in the extraction file, with the filled auxiliary field.



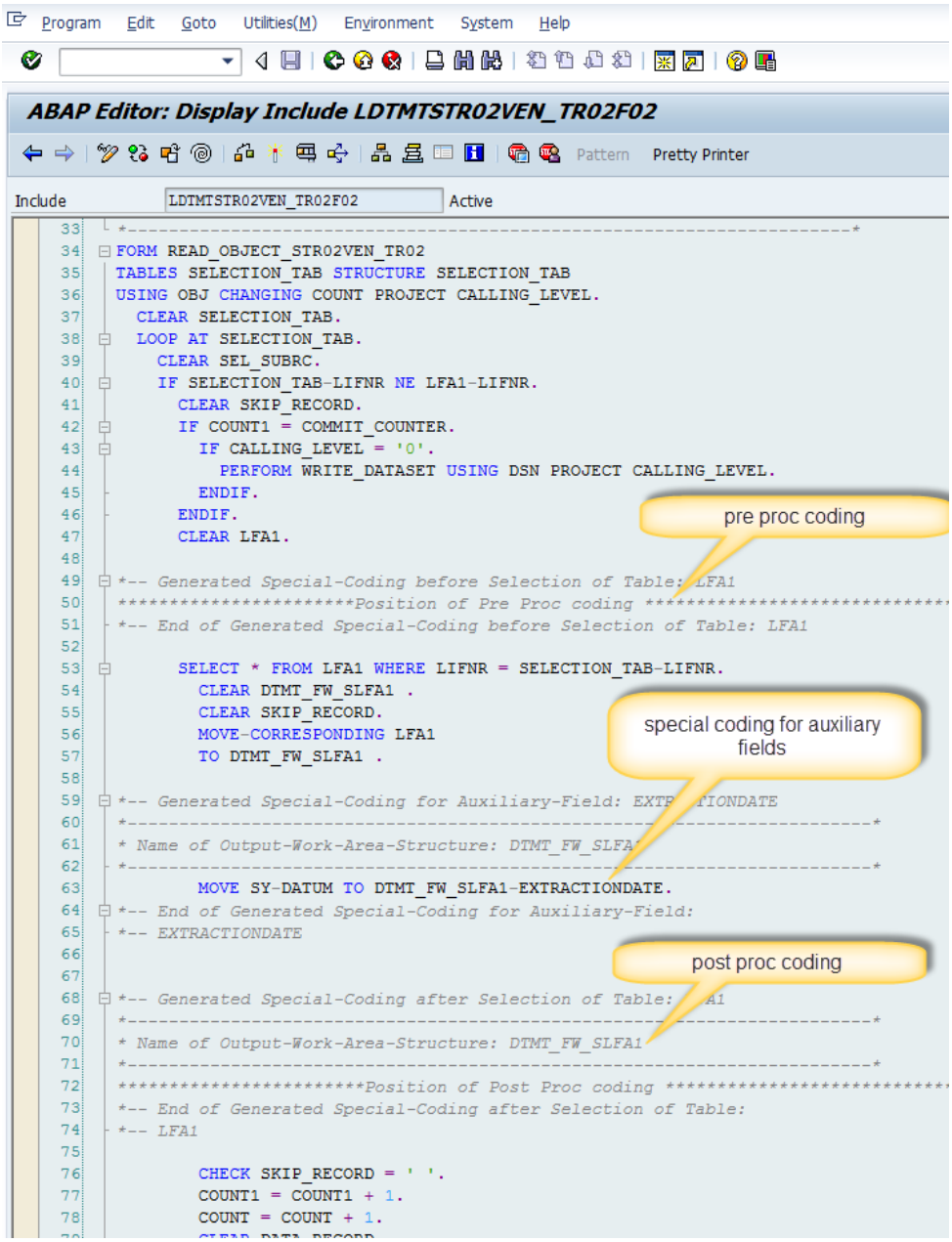
Note: All field strings need to be transported to the target system, to be able to access the source tables and fields within the import programs of DTMT. A transport function (via RFC) is embedded in DTMT's design time target.

2.1.3.3.14 Pre- and Post-Processing

The last two columns on the *Table Relations* screen allow the creation of special coding, which will be incorporated in defined places within the read module when generating the extraction program.

- Coding developed under *Pre-Proc* will be positioned within the read module before the *SELECT* statement for the respective table.
- Coding developed under *Post-Proc* will be positioned within the read module after the *SELECT* statement.

The customer will very likely not use this function in their custom objects, but they might encounter it in SAP content. Therefore, a comment line was added for table **LFA1** for both pre-processing and post-processing, and the program was generated again. The figure below shows the coding area around the *SELECT* statement of **LFA1** where the special coding was placed within the read module.



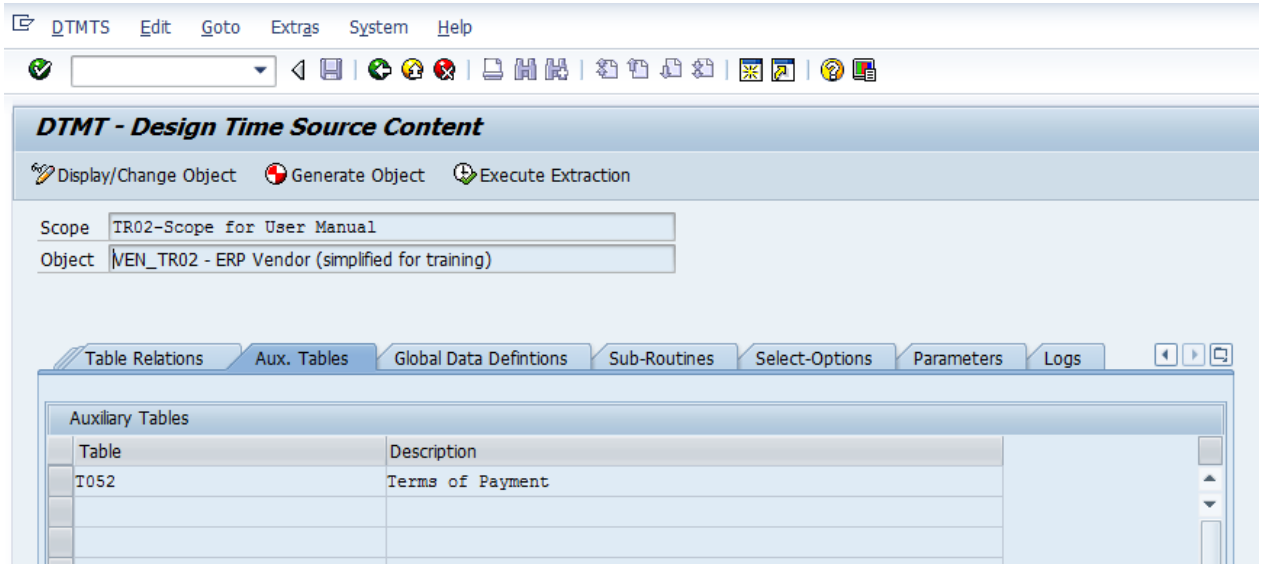

```
*---------------------------------------------------------------------*
FORM READ_OBJECT_STR02VEN_TR02
TABLES SELECTION_TAB STRUCTURE SELECTION_TAB
USING OBJ CHANGING COUNT PROJECT CALLING_LEVEL.
  CLEAR SELECTION_TAB.
  LOOP AT SELECTION_TAB.
    CLEAR SEL_SUBRC.
    IF SELECTION_TAB-LIFNR NE LFA1-LIFNR.
      CLEAR SKIP_RECORD.
      IF COUNT1 = COMMIT_COUNTER.
        IF CALLING_LEVEL = '0'.
          PERFORM WRITE_DATASET USING DSN PROJECT CALLING_LEVEL.
        ENDIF.
      ENDIF.
      CLEAR LFA1.

*-- Generated Special-Coding before Selection of Table: LFA1
**********************Position of Pre Proc coding ****************************
*-- End of Generated Special-Coding before Selection of Table: LFA1

      SELECT * FROM LFA1 WHERE LIFNR = SELECTION_TAB-LIFNR.
        CLEAR DTMT_FW_SLFA1 .
        CLEAR SKIP_RECORD.
        MOVE-CORRESPONDING LFA1
        TO DTMT_FW_SLFA1 .

*-- Generated Special-Coding for Auxiliary-Field: EXTRACTIONDATE
*---------------------------------------------------------------------*
* Name of Output-Work-Area-Structure: DTMT_FW_SLFA1
*---------------------------------------------------------------------*
        MOVE SY-DATUM TO DTMT_FW_SLFA1-EXTRACTIONDATE.
*-- End of Generated Special-Coding for Auxiliary-Field:
*-- EXTRACTIONDATE


*-- Generated Special-Coding after Selection of Table: LFA1
*---------------------------------------------------------------------*
* Name of Output-Work-Area-Structure: DTMT_FW_SLFA1
*---------------------------------------------------------------------*
**********************Position of Post Proc coding ***********************
*-- End of Generated Special-Coding after Selection of Table:
*-- LFA1

        CHECK SKIP_RECORD = ' '.
        COUNT1 = COUNT1 + 1.
        COUNT = COUNT + 1.
```

2.1.3.4 Migration Object Level – Aux. Tables Tab

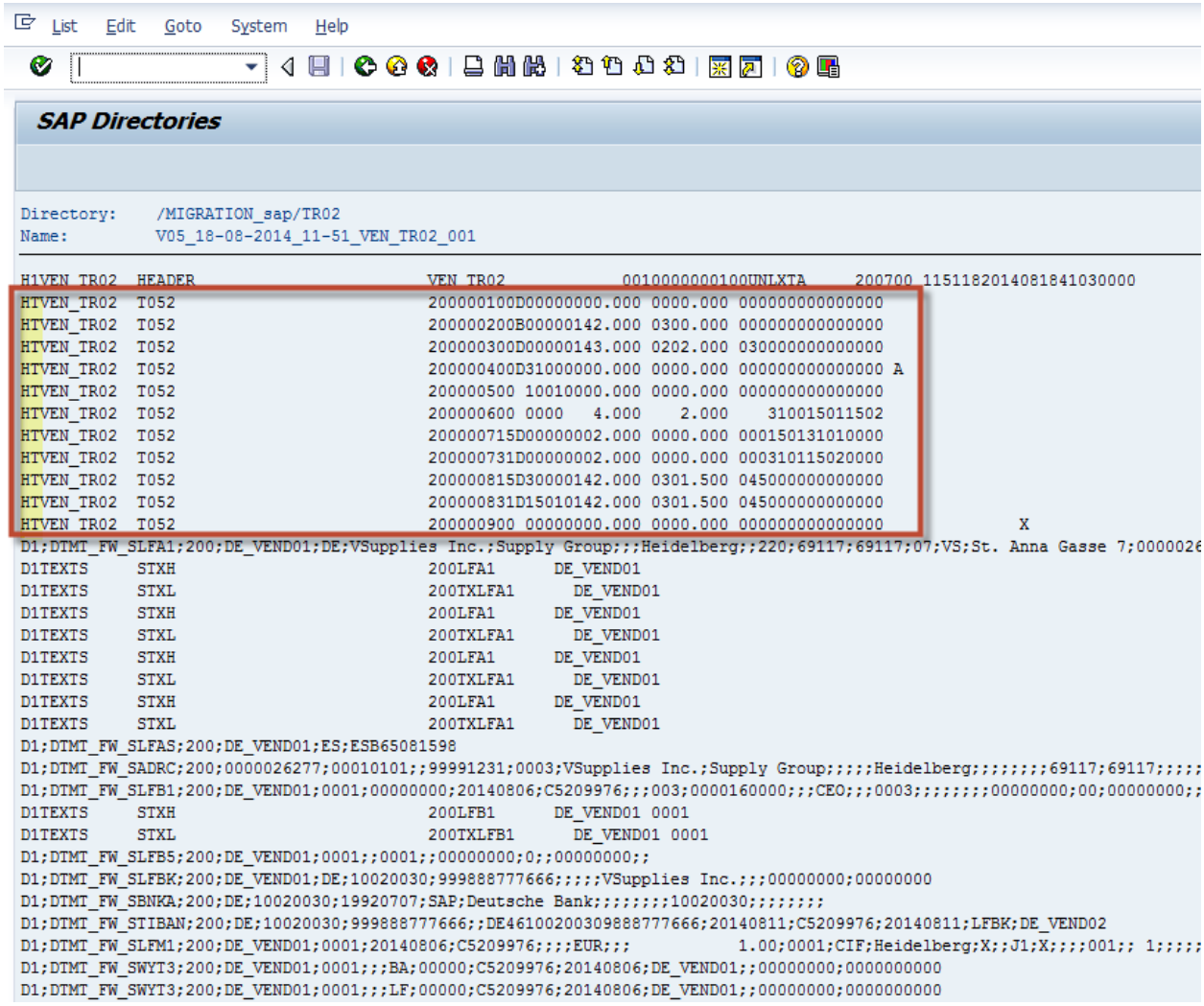
To correctly interpret application data directly read from a table, it is sometimes necessary to read customizing settings as well. Since the interpretation of source data takes place in the target system, because the complete mapping happens there, you need to ensure that the required source customizing is

available in the target system as well. Therefore, DTMT provides the *Auxiliary Tables* functionality. You only need to register those customizing tables that you want to use on the *Aux. Tables* tab.

For demonstration purposes, table **T001** is registered, and the program is generated and run again, to see how it looks in the file.

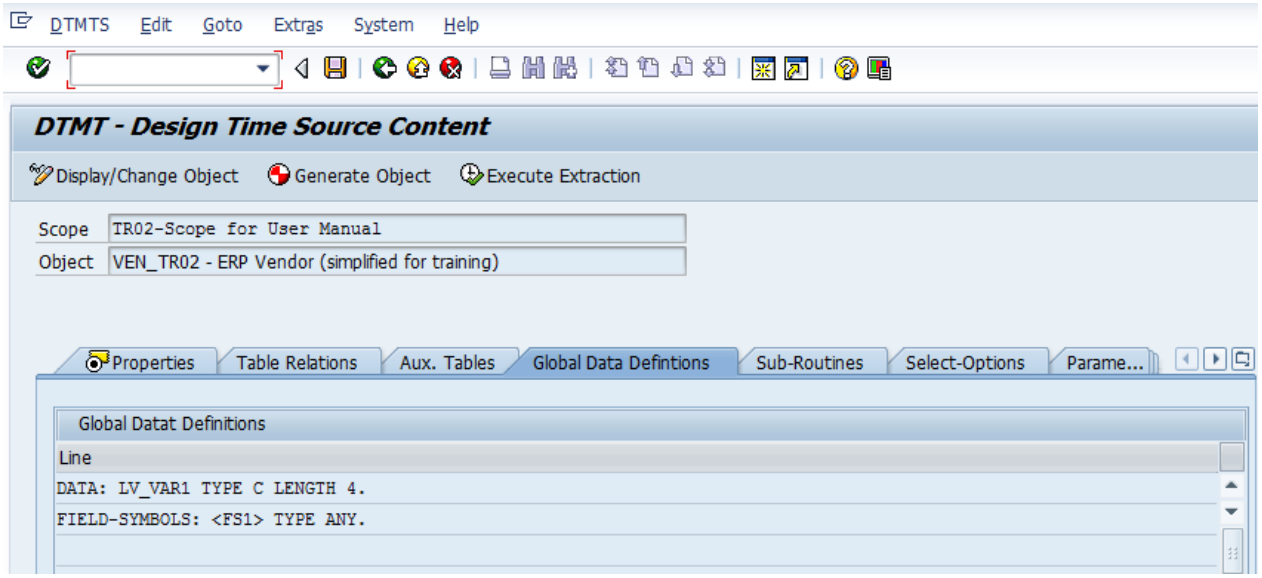


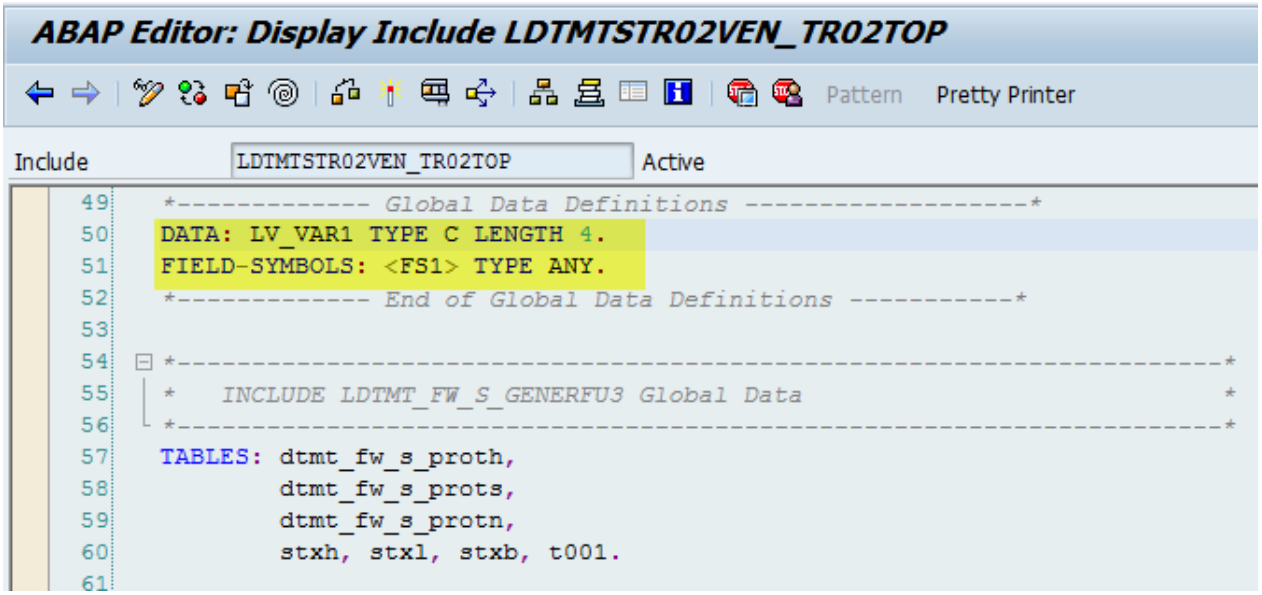
Note: The system accepts only tables that are defined as customizing tables.


List Edit Goto System Help
SAP Directories
Directory: /MIGRATION_sap/TR02
Name: V05_18-08-2014_11-51_VEN_TR02_001
H1VEN_TR02 HEADER VEN_TR02 001000000010OUNLXTA 200700 1151182014081841030000
HTVEN_TR02 T052 200000100D00000000.000 0000.000 000000000000000
HTVEN_TR02 T052 200000200B00000142.000 0300.000 000000000000000
HTVEN_TR02 T052 200000300D00000143.000 0202.000 030000000000000
HTVEN_TR02 T052 200000400D31000000.000 0000.000 000000000000000 A
HTVEN_TR02 T052 200000500 10010000.000 0000.000 000000000000000
HTVEN_TR02 T052 200000600 0000 4.000 2.000 310015011502
HTVEN_TR02 T052 200000715D00000002.000 0000.000 000150131010000
HTVEN_TR02 T052 200000731D00000002.000 0000.000 000310115020000
HTVEN_TR02 T052 200000815D30000142.000 0301.500 045000000000000
HTVEN_TR02 T052 200000831D15010142.000 0301.500 045000000000000
HTVEN_TR02 T052 200000900 00000000.000 0000.000 000000000000000 X
D1;DTMT_FW_SLFA1;200;DE_VEND01;DE;VSupplies Inc.;Supply Group;;;Heidelberg;;220;69117;69117;07;VS;St. Anna Gasse 7;0000026
D1TEXTS STXH 200LFA1 DE_VEND01
D1TEXTS STXL 200TXLFA1 DE_VEND01
D1TEXTS STXH 200LFA1 DE_VEND01
D1TEXTS STXL 200TXLFA1 DE_VEND01
D1TEXTS STXH 200LFA1 DE_VEND01
D1TEXTS STXL 200TXLFA1 DE_VEND01
D1TEXTS STXH 200LFA1 DE_VEND01
D1TEXTS STXL 200TXLFA1 DE_VEND01
D1;DTMT_FW_SLFAS;200;DE_VEND01;ES;ESB65081598
D1;DTMT_FW_SADRC;200;0000026277;00010101;;99991231;0003;VSupplies Inc.;Supply Group;;;;;Heidelberg;;;;;;;;69117;69117;;;;;
D1;DTMT_FW_SLFB1;200;DE_VEND01;0001;00000000;20140806;C5209976;;;003;0000160000;;;CEO;;;0003;;;;;;;;00000000;00;00000000;;
D1TEXTS STXH 200LFB1 DE_VEND01 0001
D1TEXTS STXL 200TXLFB1 DE_VEND01 0001
D1;DTMT_FW_SLFB5;200;DE_VEND01;0001;;0001;;00000000;0;;00000000;;
D1;DTMT_FW_SLFBK;200;DE_VEND01;DE;10020030;999888777666;;;;;VSupplies Inc.;;;00000000;00000000
D1;DTMT_FW_SBNKA;200;DE;10020030;19920707;SAP;Deutsche Bank;;;;;;;;10020030;;;;;;;;
D1;DTMT_FW_STIBAN;200;DE;10020030;999888777666;;DE46100200309888777666;20140811;C5209976;20140811;LFBK;DE_VEND02
D1;DTMT_FW_SLFM1;200;DE_VEND01;0001;20140806;C5209976;;;;EUR;;; 1.00;0001;CIF;Heidelberg;X;;J1;X;;;;001;; 1;;;;;
D1;DTMT_FW_SWYT3;200;DE_VEND01;0001;;;BA;00000;C5209976;20140806;DE_VEND01;;00000000;0000000000
D1;DTMT_FW_SWYT3;200;DE_VEND01;0001;;;LF;00000;C5209976;20140806;DE_VEND01;;00000000;0000000000

2.1.3.5 Migration Object Level - Global Data Definitions Tab

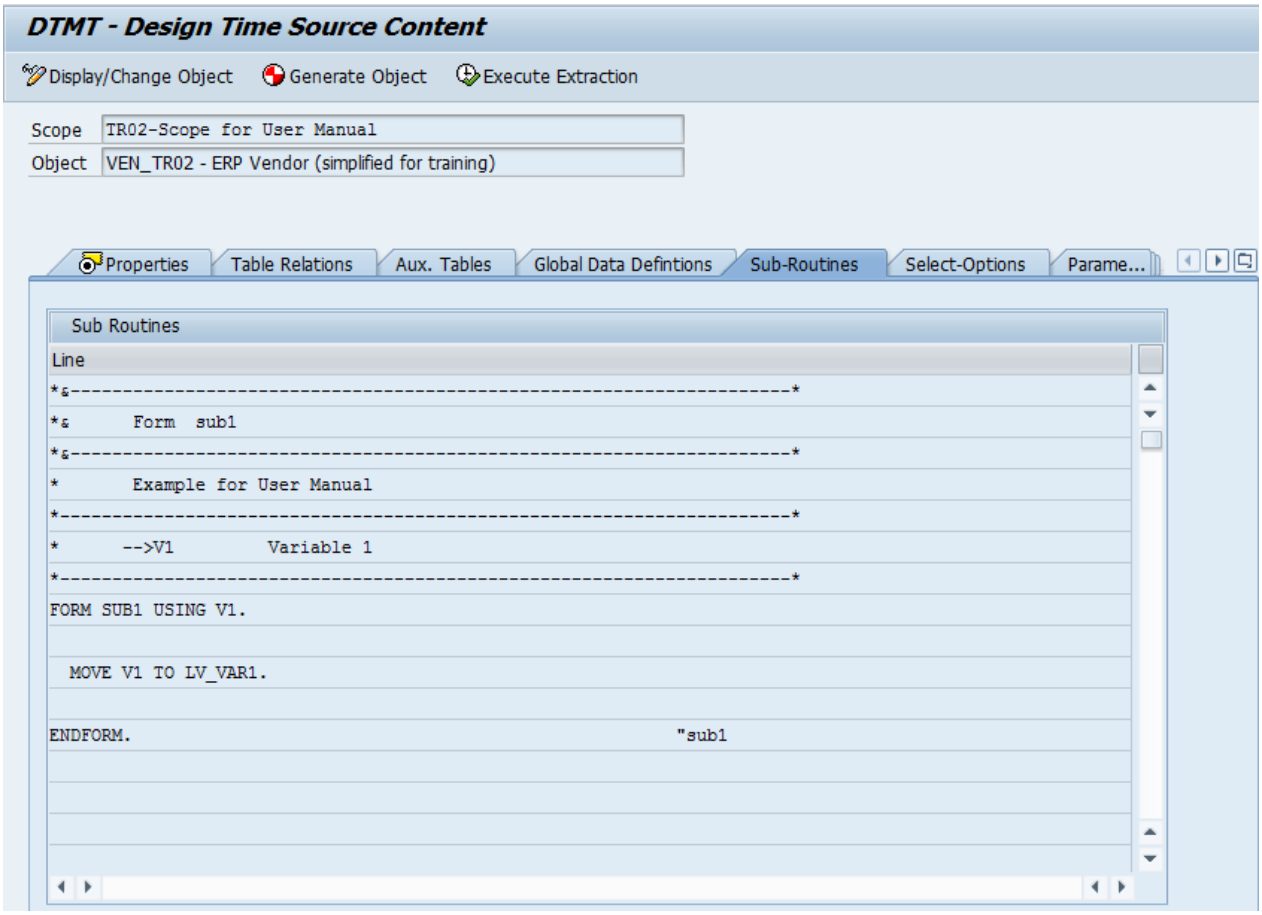
Additional field variables, field symbols, or any type of data definitions that are available in each section of the read module can be defined on this tab.
For demonstration purposes, field symbols and field variables were defined, to show how it looks in the generated read module:



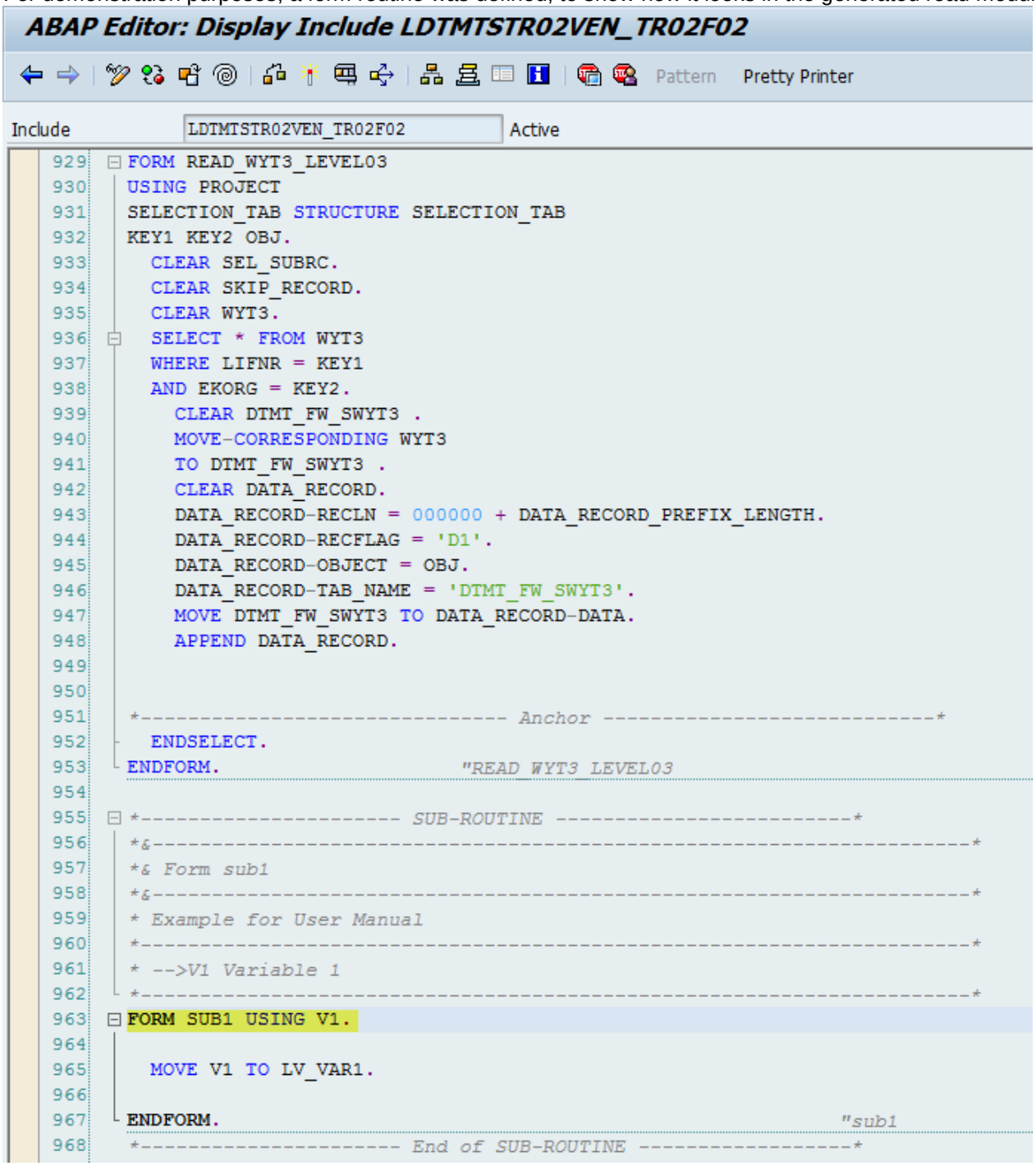


Migration Object Level - Sub-Routines Tab

Additional form (sub) routines that are available in each section of the read module can be defined on this tab.



For demonstration purposes, a form routine was defined, to show how it looks in the generated read module.

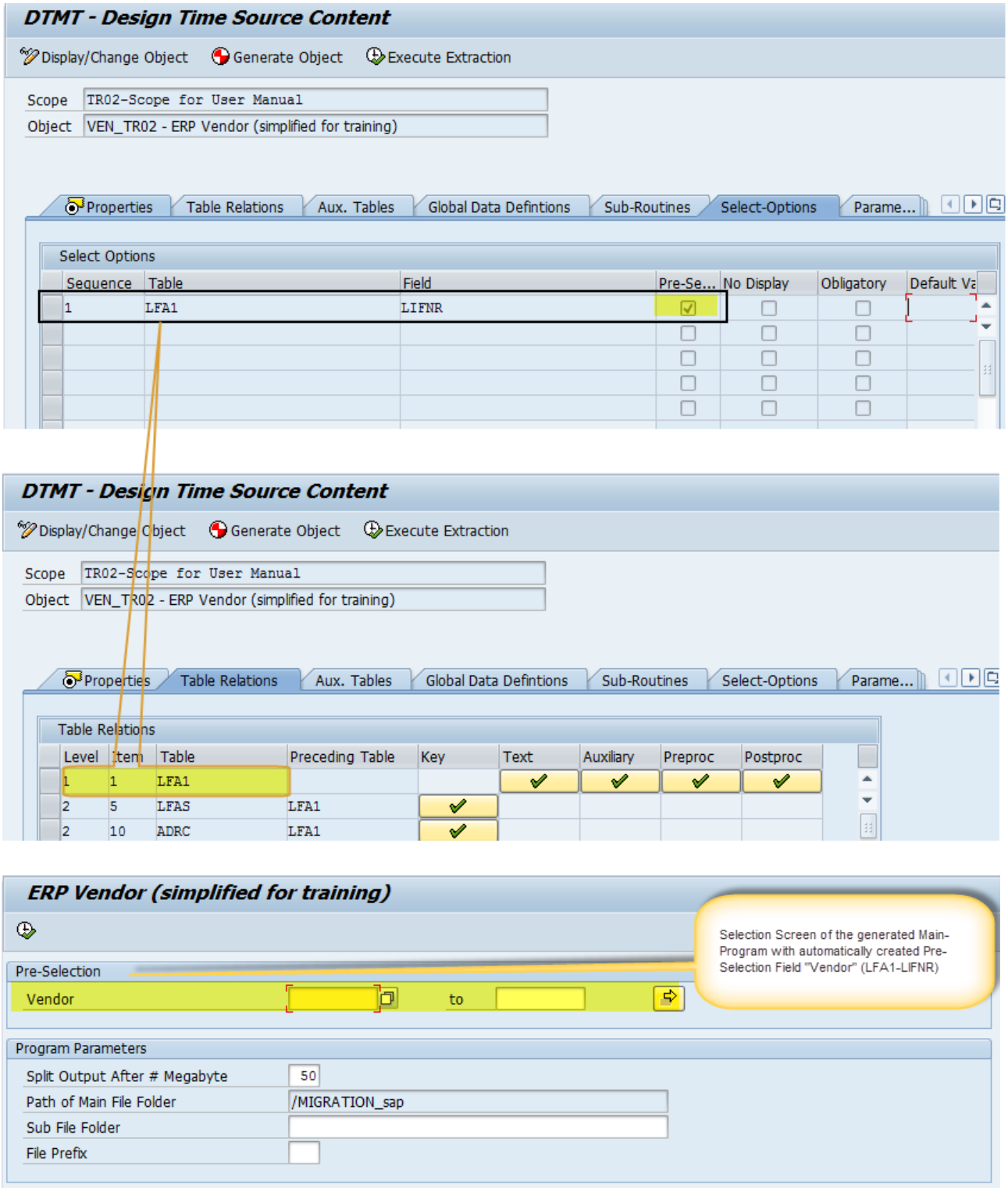



```
929  FORM READ_WYT3_LEVEL03
930  USING PROJECT
931  SELECTION_TAB STRUCTURE SELECTION_TAB
932  KEY1 KEY2 OBJ.
933    CLEAR SEL_SUBRC.
934    CLEAR SKIP_RECORD.
935    CLEAR WYT3.
936    SELECT * FROM WYT3
937    WHERE LIFNR = KEY1
938    AND EKORG = KEY2.
939      CLEAR DTMT_FW_SWYT3 .
940      MOVE-CORRESPONDING WYT3
941      TO DTMT_FW_SWYT3 .
942      CLEAR DATA_RECORD.
943      DATA_RECORD-RECLN = 000000 + DATA_RECORD_PREFIX_LENGTH.
944      DATA_RECORD-RECFLAG = 'D1'.
945      DATA_RECORD-OBJECT = OBJ.
946      DATA_RECORD-TAB_NAME = 'DTMT_FW_SWYT3'.
947      MOVE DTMT_FW_SWYT3 TO DATA_RECORD-DATA.
948      APPEND DATA_RECORD.
949
950
951  *------------------------------ Anchor ----------------------------*
952    ENDSELECT.
953  ENDFORM.                      "READ_WYT3_LEVEL03
954
955  *---------------------- SUB-ROUTINE -------------------------*
956  *&---------------------------------------------------------------------*
957  *& Form sub1
958  *&---------------------------------------------------------------------*
959  * Example for User Manual
960  *---------------------------------------------------------------------*
961  * -->V1 Variable 1
962  *---------------------------------------------------------------------*
963  FORM SUB1 USING V1.
964
965    MOVE V1 TO LV_VAR1.
966
967  ENDFORM.                                                    "sub1
968   *---------------------- End of SUB-ROUTINE ------------------*
```

2.1.3.6 *Migration Object Level - Select-Options Tab*

2.1.3.6.1 *Automatically Set Select Options*

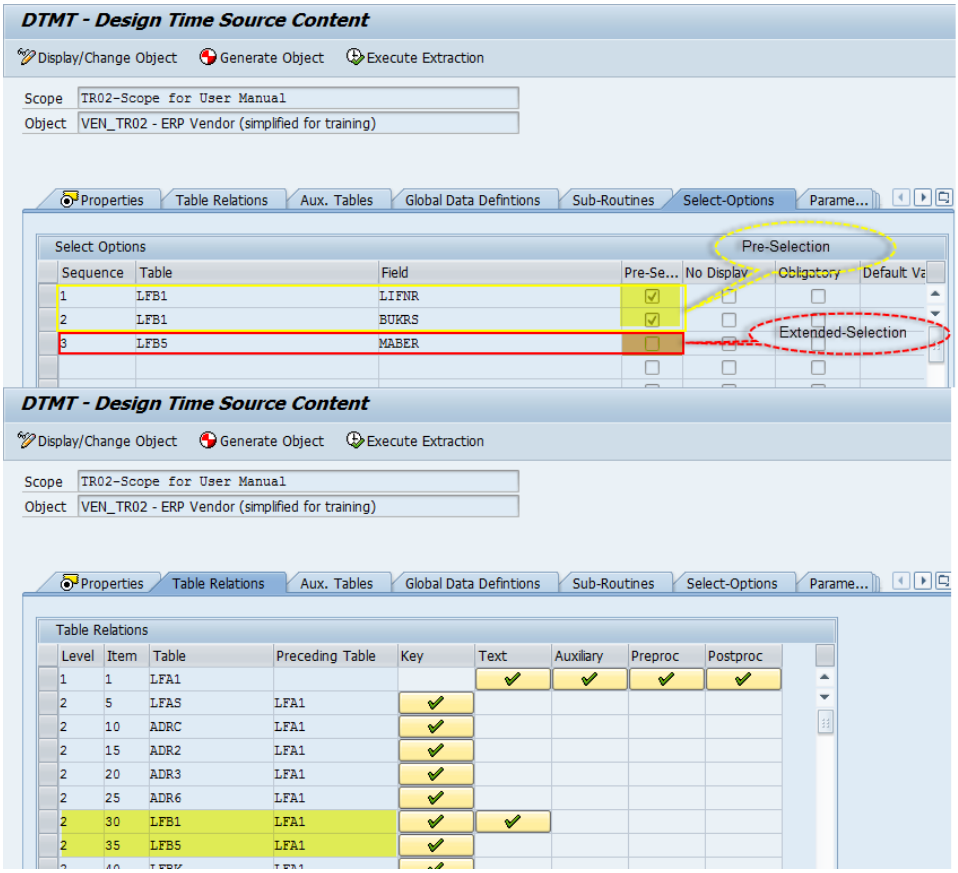
It is not necessary to define select options manually. The tool automatically determines select options of type *Pre-Selection*. All key fields of the relevant table will be defined on level 1.

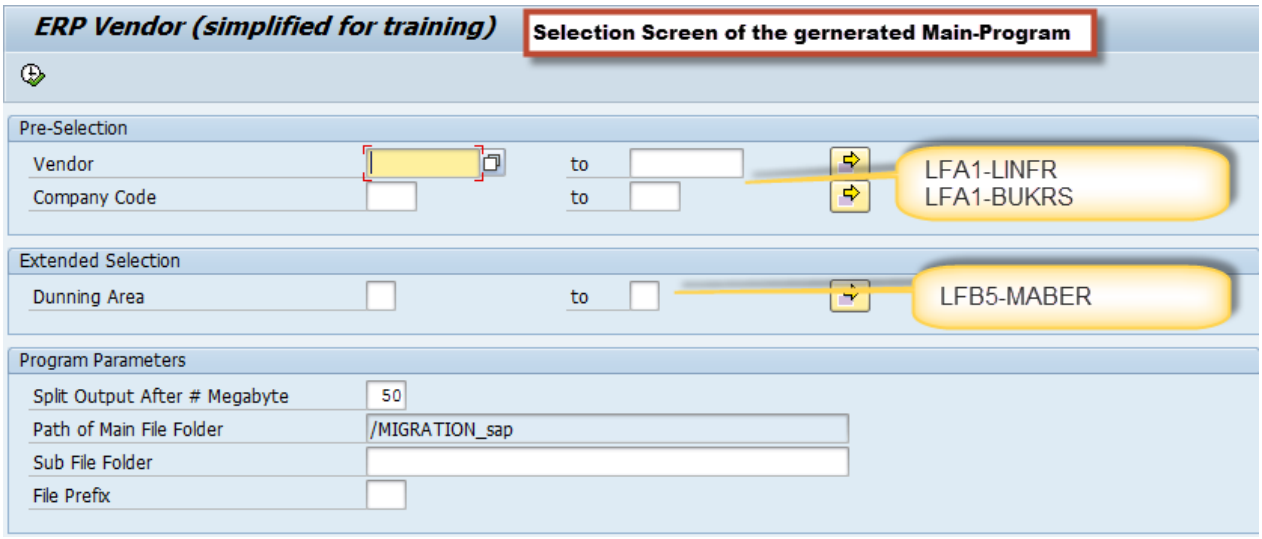


2.1.3.6.2 *Manually Set Select Options*

When setting select options manually, it is always necessary to define pre-selection for only one table. In this case, one table of any level can be set as the *Pre-Selection* type.

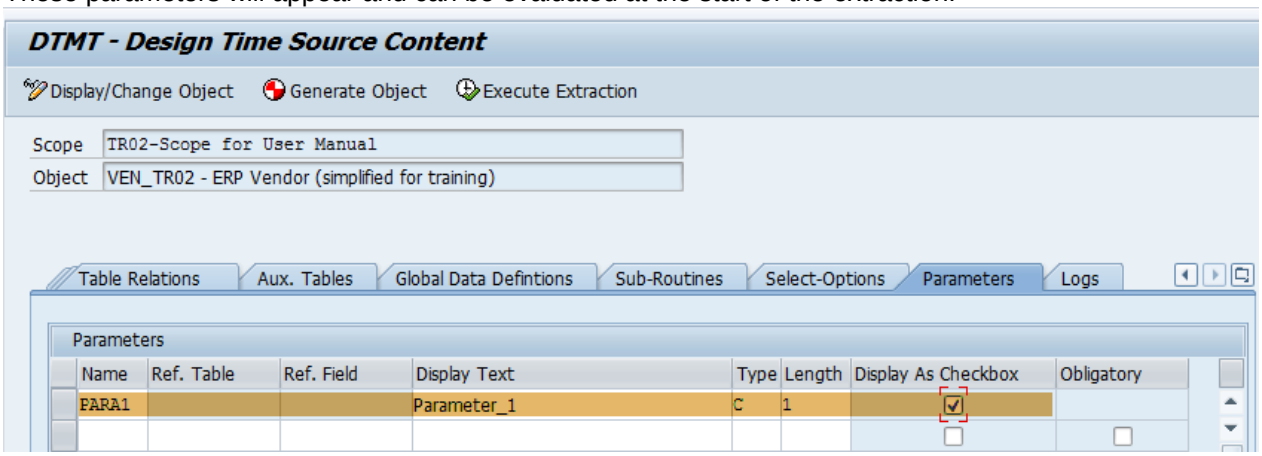
In addition, extended selection of different tables can be set by deselecting the checkbox in the column *Pre-Selection*.

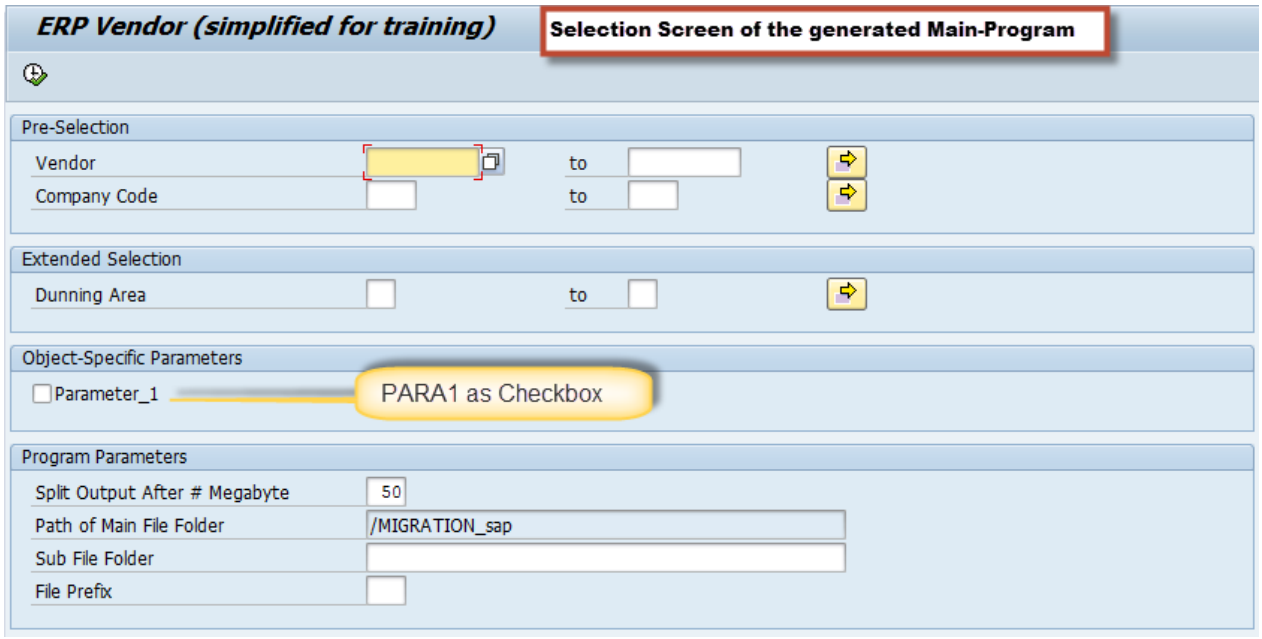




2.1.3.7 Migration Object Level - Parameters Tab

Definitions of parameters can be reached at each level of the read module and main program.
These parameters will appear and can be evaluated at the start of the extraction.

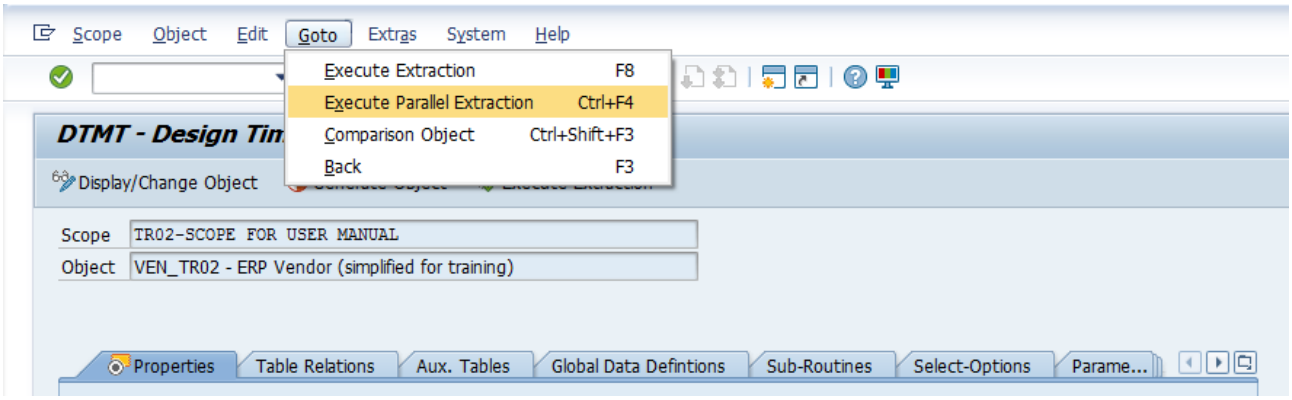




3 SPECIAL FEATURES ON SOURCE SIDE

3.1 Object

3.1.1 Execute Parallel Extraction

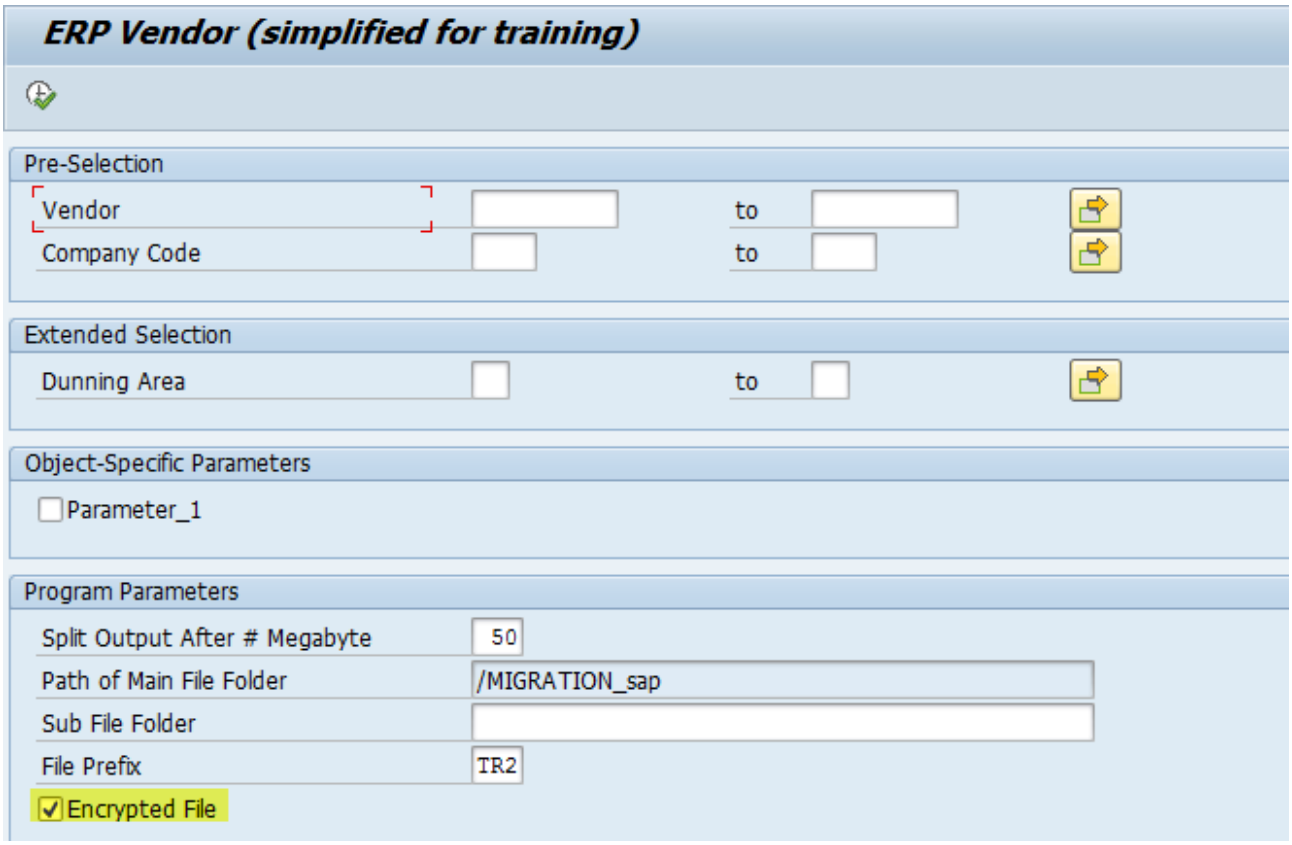


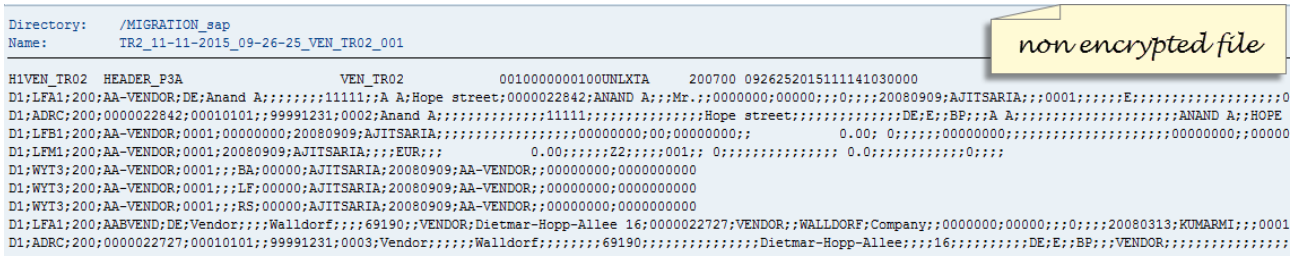
Choose *Goto -> Execute Parallel Extraction*.

In the case of a huge amount of data, it is possible to execute the extraction in parallel tasks as background jobs.
The number of parallelization runs can be set manually or automatically. In the case of an automatic setting, the system will count the number of parallelization runs depending on the amount of free batch process types.

3.1.2 File Encryption

To ensure the security of the data, it is possible to encrypt the extract when creating it. To do so, select the checkbox *Encrypted File*.

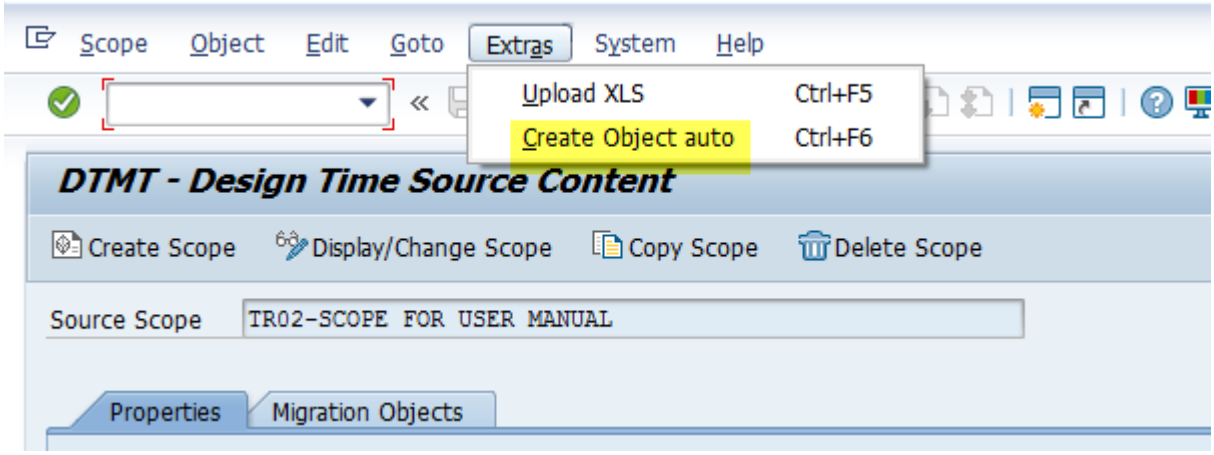






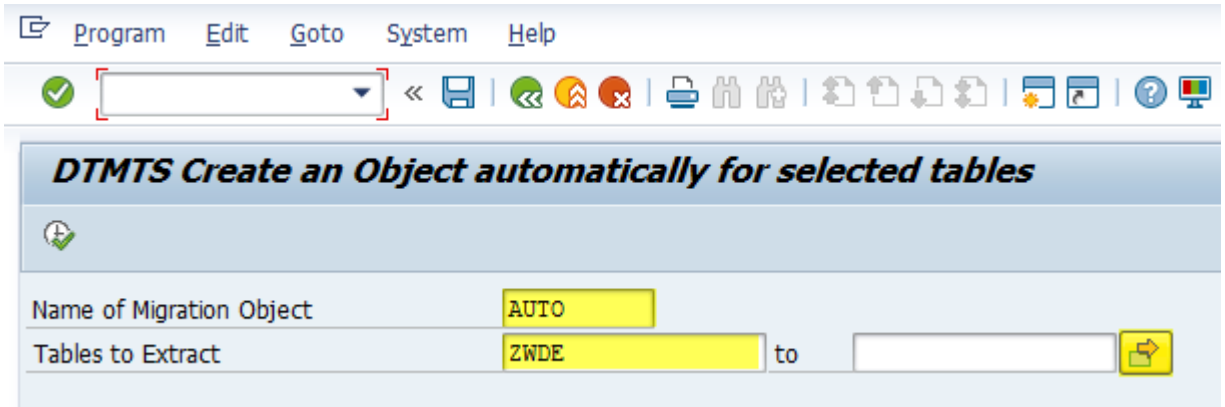
3.1.3 Create an Object Automatically

This feature automatically creates a migration object. All the content of the selected tables is extracted and then loaded again on the target side.

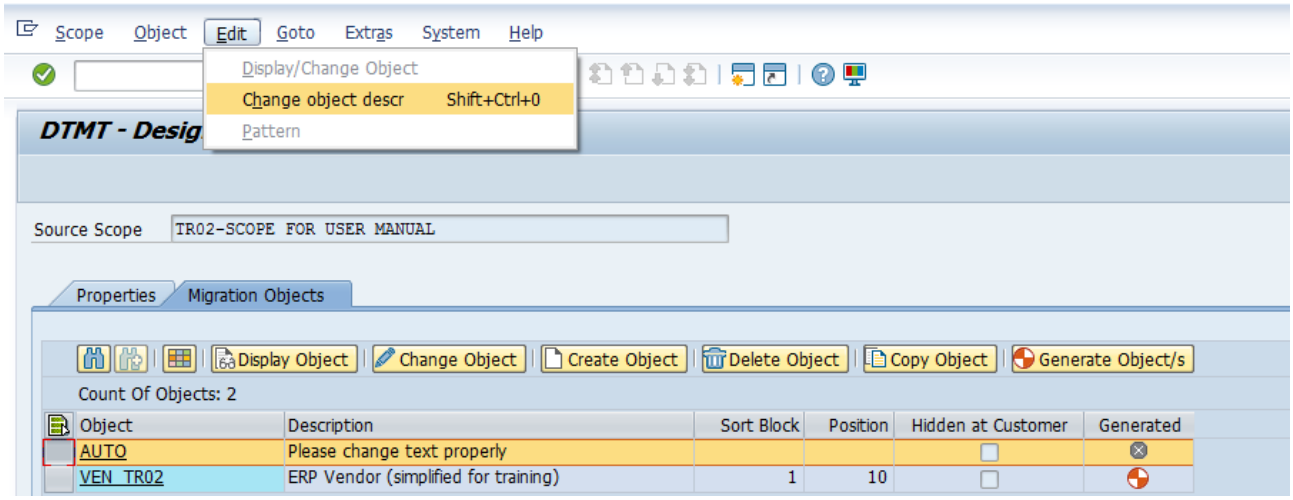


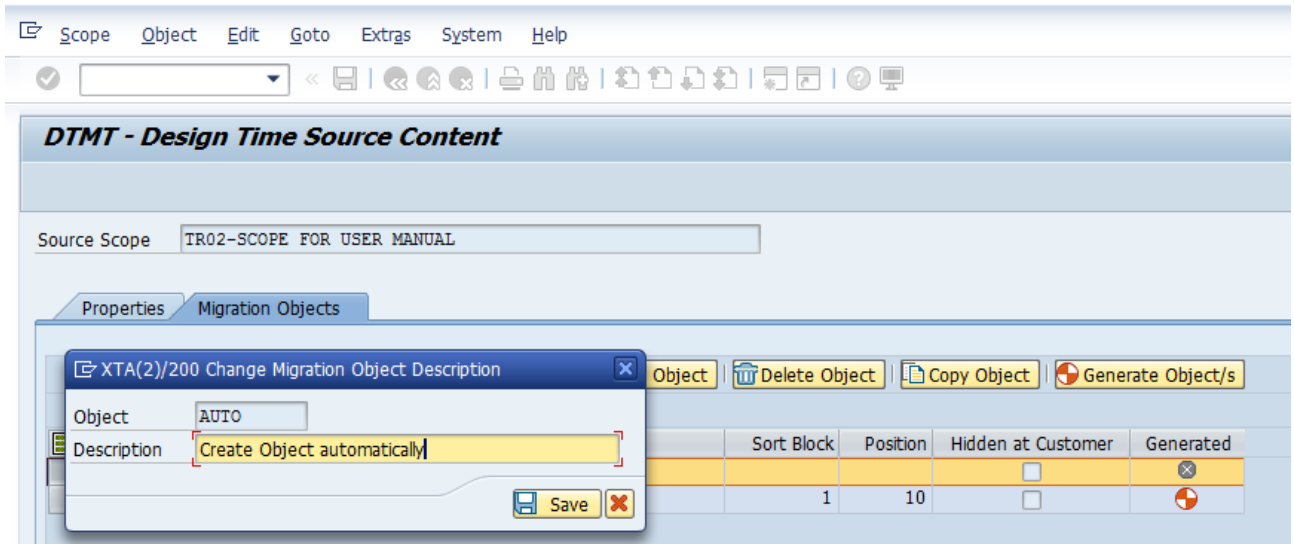
3.1.3.1 Select Tables

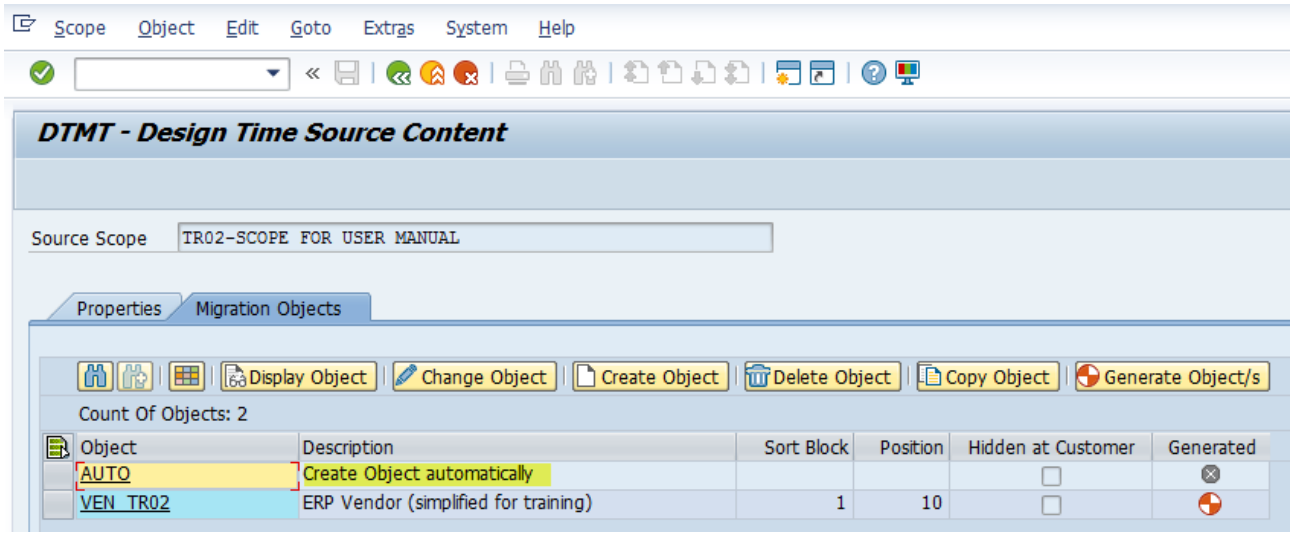
Content from all types of transparent tables can be selected and extracted.



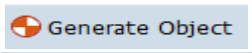
3.1.3.2 Adjust Object Description

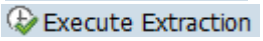






3.1.3.3 Generate Object and Execute Extraction

- To open the object, click on the object-name.
- Press the Generate button 

- Press the Execute button 

The object will be extracted the selected table (s).

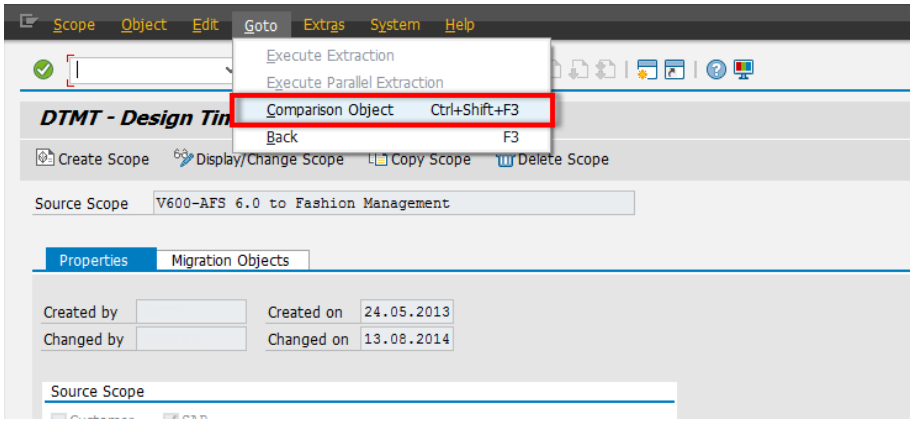
3.2 Code Comparison

SAP delivers standard content (scope: V600) that you can copy to the customer namespace as a template for your migration. Corrections and enhancements in the standard scope are delivered with each support package. There is now a function that you can use to recognize what has changed in the standard scope, in comparison with your custom scope: the coding comparison.

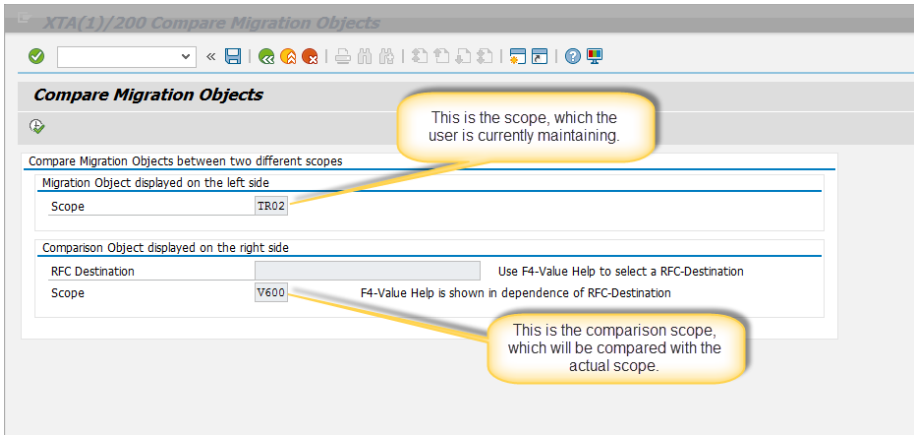
You can carry out the coding comparison in two different ways; on scope level and on object level.

3.2.1 Code Comparison on Scope Level

Call transaction DTMT_DTS. Choose *Goto -> Comparison Object*.

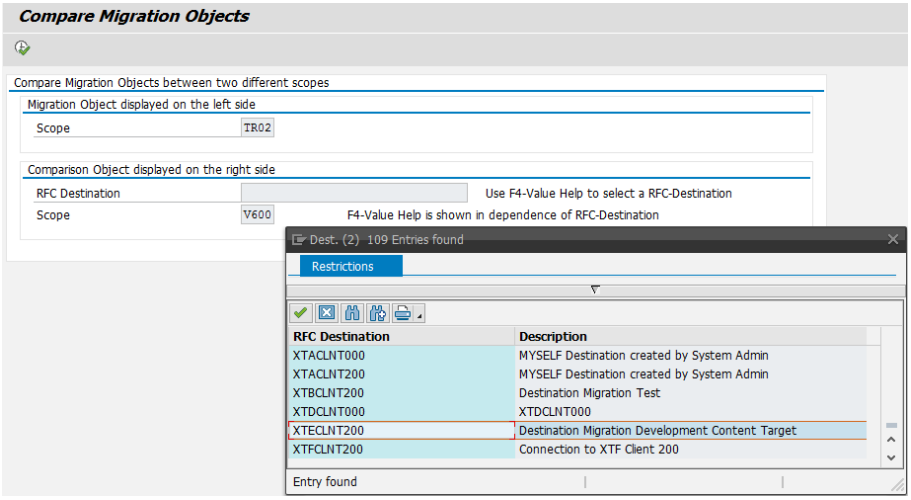


The scope that you are currently in as a user (in this example, TR02) is shown in the upper part of the screen, and the scope delivered by SAP (V600) is shown as the comparison scope in the lower part.

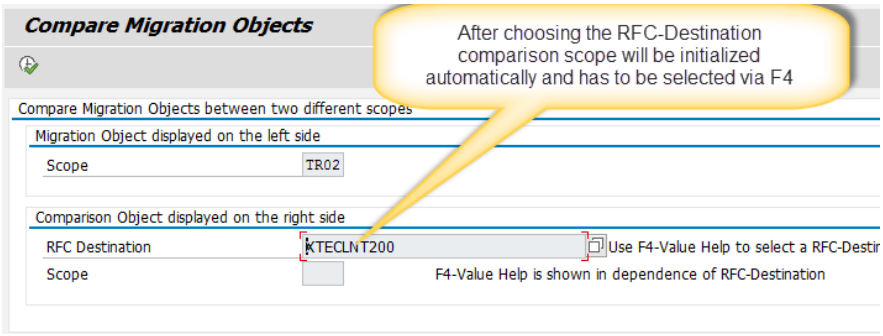


You can use the F4 help to change both the current scope and the comparison scope. You can also compare scopes from different systems.

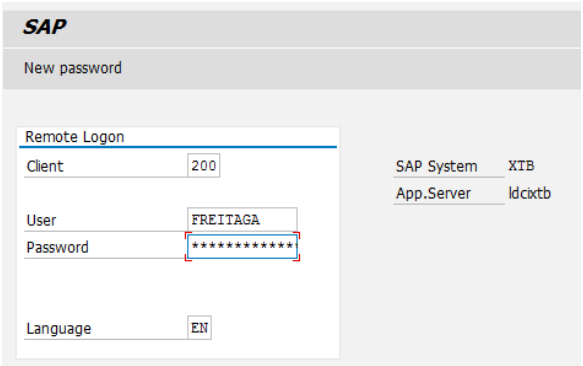
To do this, choose the corresponding RFC connection from the F4 help.



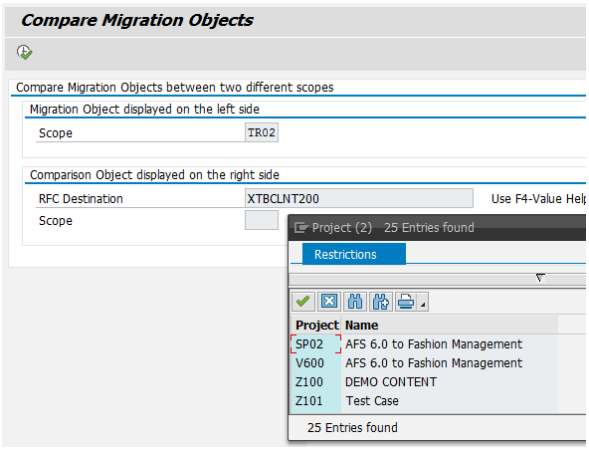
After you have selected the RFC connection and thereby the system that should be used for the comparison, the associated scope is automatically initialized. You then need to choose it again by using the F4 help. This is necessary because the preset scope for the RFC connection may no longer be valid.



When you try to choose the scope, a logon screen appears. Log on.

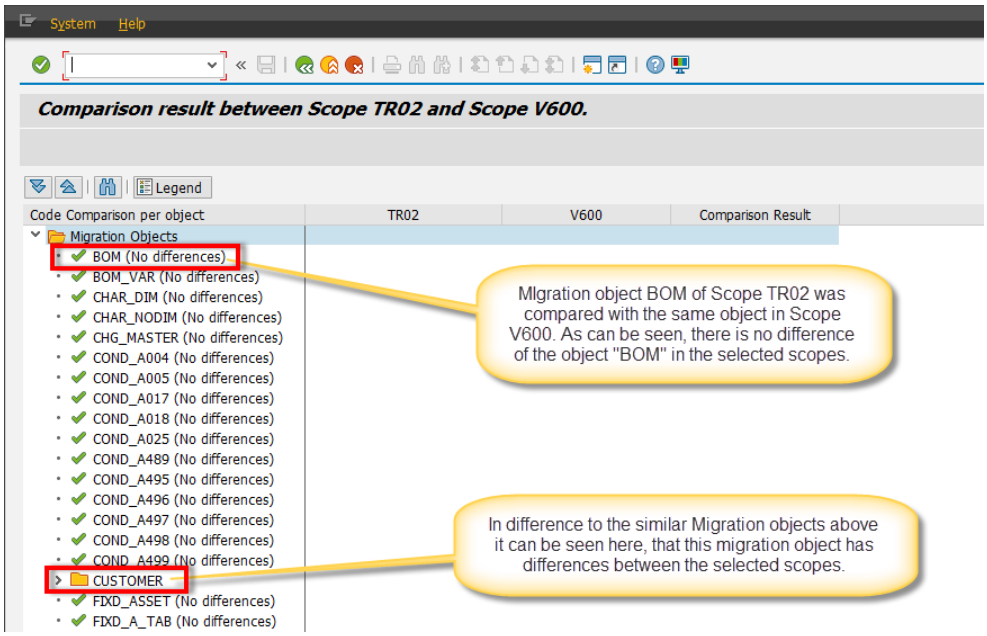


After the logon, you can choose the comparison scope by using F4.

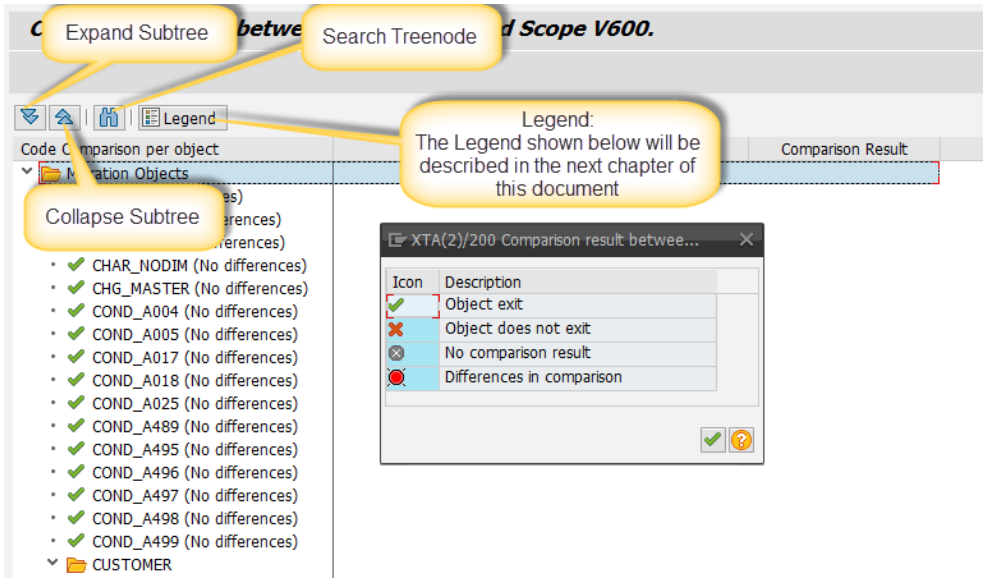


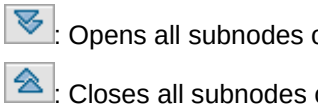
You do not have to choose the original scope (V600). It is also possible to compare two custom scopes.

Choose *Execute* .
The system now carries out a coding comparison on scope level. Objects with the same names are always compared with each other.
The result of the comparison is displayed for each object, in the analysis monitor. Each object has its own subnode in the tree structure. Objects that are flagged with ✔ contain no differences, unlike the objects that are shown with 🗀. Section 3.2.2 contains a detailed description of the differences.



Function description for the individual buttons:

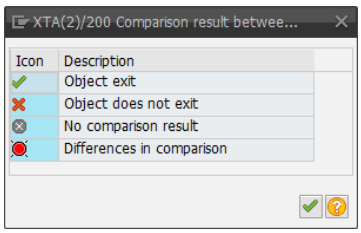


: Opens all subnodes of the selected node

: Closes all subnodes of the selected node

: Searches for a node (for example, CUSTOMER). The cursor is then automatically positioned on the search string that is found.

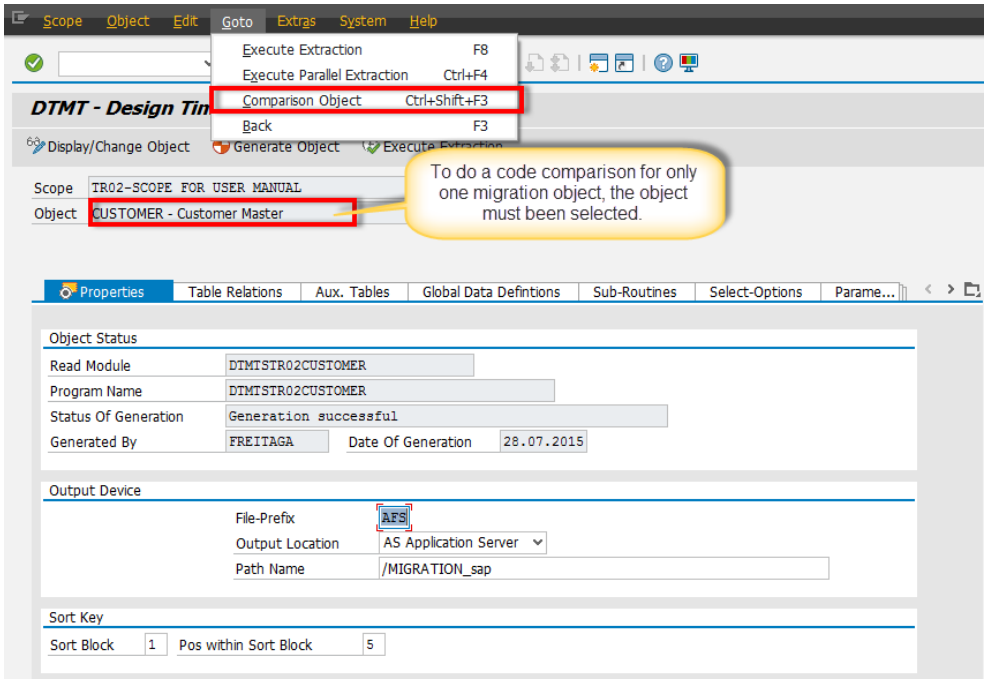
: Displays the legend for the icons that are used, with an explanation. Section 3.2.2 contains more information about this.



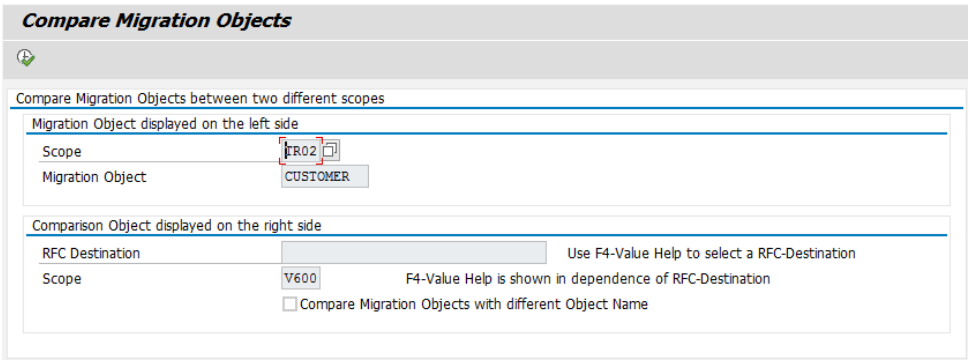
3.2.2 Code Comparison on Object Level

To carry out an analysis on object level, you need to have selected a migration object in the migration tool. You can then call the analysis.

Call transaction DTMT_DTS. Choose *Goto -> Comparison Object*.

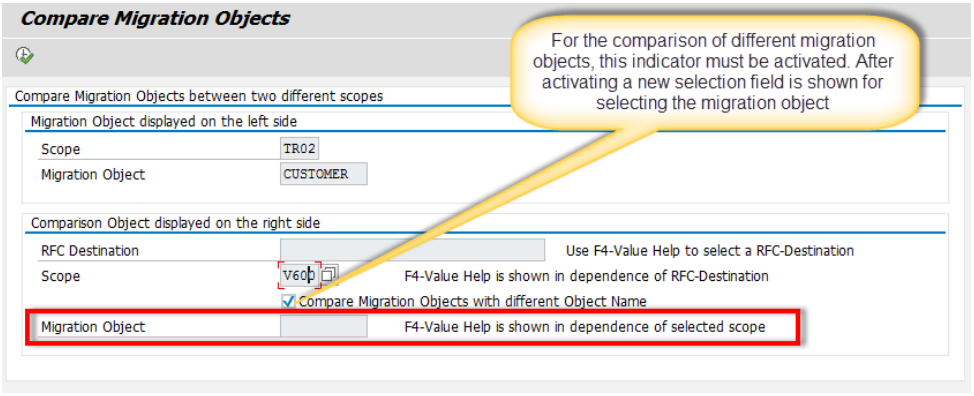


The scope that you are currently in as a user (in this example, TR02) is shown in the upper part of the screen, and the scope delivered by SAP (V600) is shown as the comparison scope in the lower part.



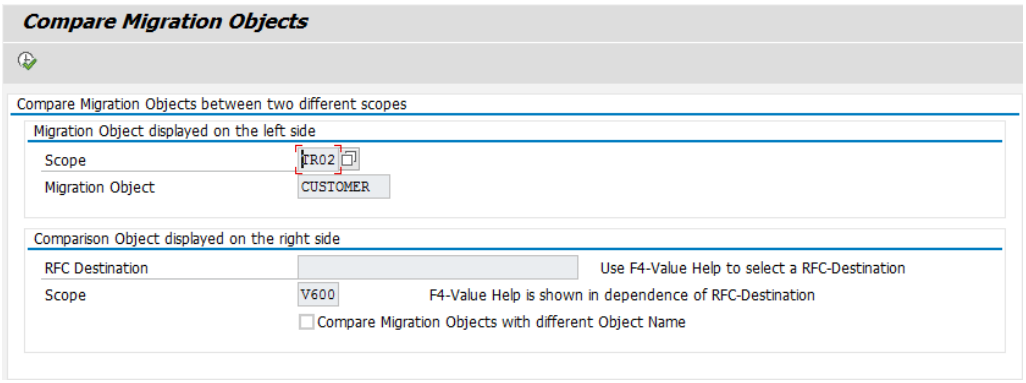
Here, it is also possible to choose a comparison scope other than V600. You can use the F4 help to choose any other scope, even from another system (see also section 3.2.2.2).

To choose a different migration object for comparison, select the checkbox *Compare Migration Object with Different Object Name*. This may be necessary if, for example, the customer used an object name other than the proposed one when copying the object.



The following sections describe the differences in more detail.

3.2.2.1 Migration Objects with the Same Name



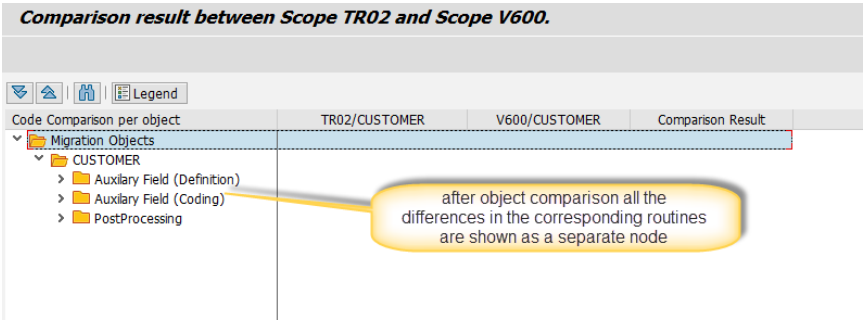
In this comparison, the migration object CUSTOMER is selected in scope TR02, and a migration object with the same name is selected in scope V600. If the migration object does not exist in the comparison scope, a corresponding warning is displayed, but the function can still be executed.

Caution:

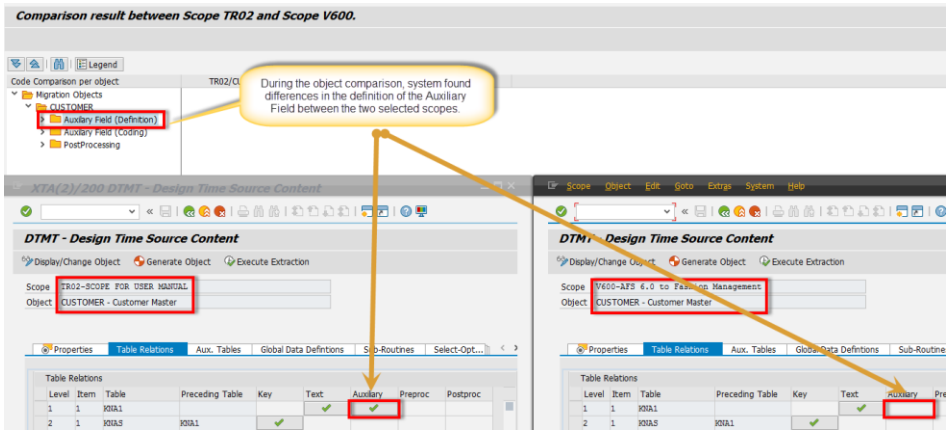
When the system compares two objects, the changes are shown only in relation to the compared object, not the other way around.

Example: Assume that you have changed the object CUSTOMER in scope TR02 so that an additional table is added to the table relations. In this case, you could see the change on the screen; but if the comparison were made in the other direction, the change would not be apparent.

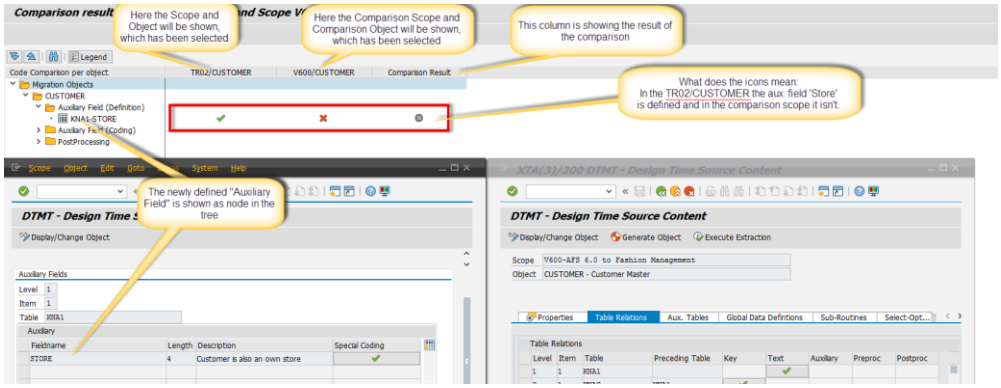
After the comparison, all changes within the corresponding routines are collected and displayed as separate nodes in the tree structure.



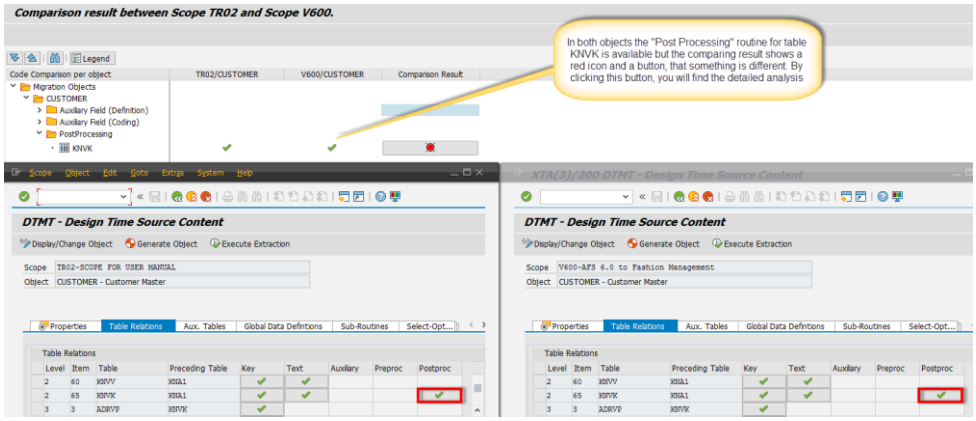
If you open the nodes, the result of the corresponding analysis is displayed.

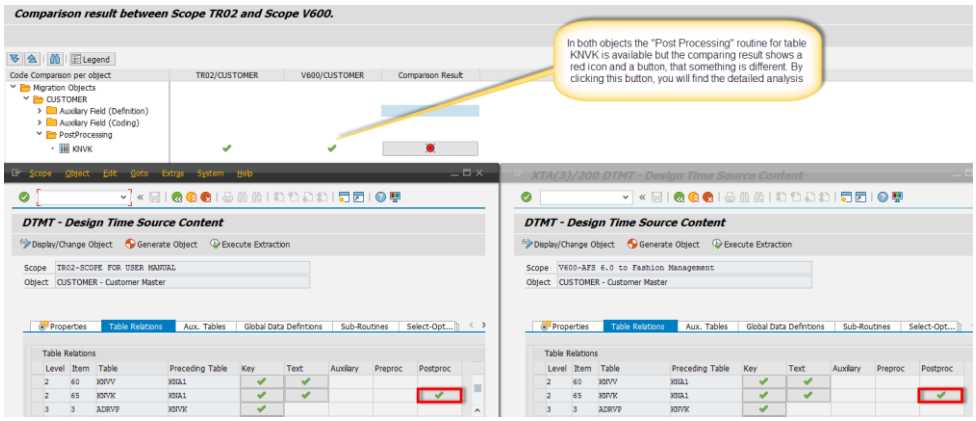


For table KNA1 in scope TR02, an auxiliary field was added that does not exist in scope V600.

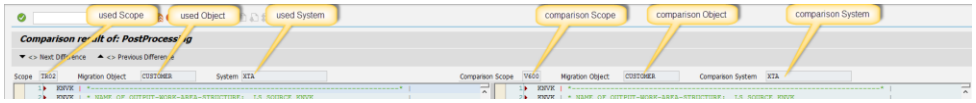


The second screen shows that both routines are implemented, but the content of the definitions differs.



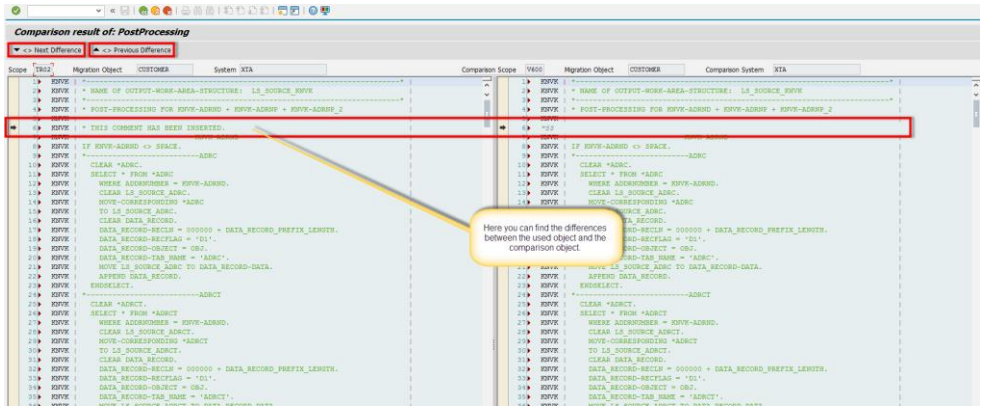


The button in column *Comparison Result* navigates to a detailed analysis.



On the split screen, the left side shows information about the scope used, and the right side shows information about the comparison scope.

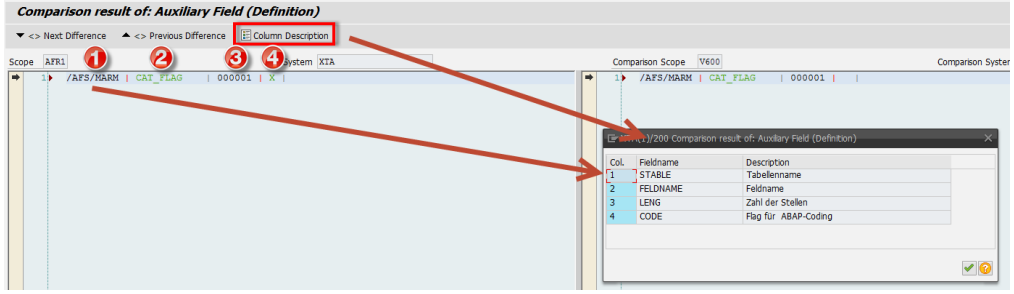
The header line contains information about the scope, objects, and systems. You can now analyze the differences in the detailed analysis:



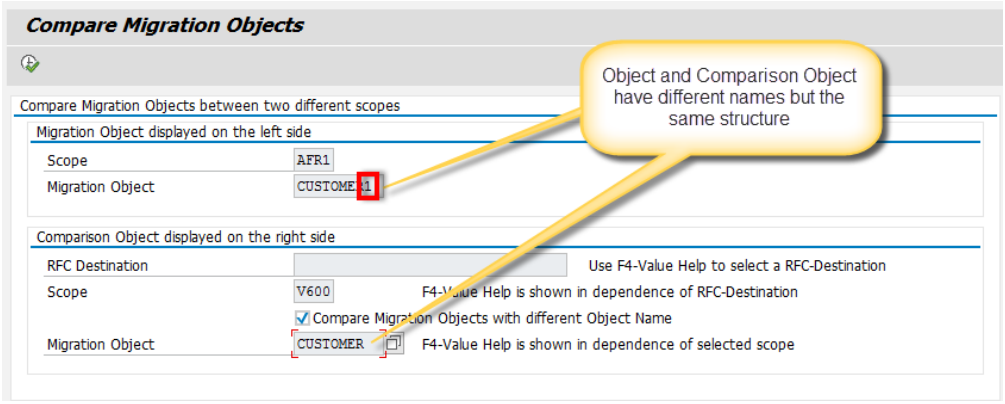
The lines marked with ➡ contain the differences. You can use button ▼ <> Next Difference to jump to the next difference, and button ▲ <> Previous Difference to jump back to the previous difference.

⚠ Caution:

To display the description of the individual columns, choose *Column Description*.



3.2.2.2 Migration Objects with Different Names



You can also compare objects that have different names. When a migration object is copied from the standard scope (V600), the user may rename the object for some reason, or the same migration object may exist multiple times in the customer scope (with different versions).

Caution:

The user must decide whether the objects are suitable for comparison. There is no pre-validation in the system. This would display all differences between the two objects.
After the comparison, all changes within the corresponding routines are collected and displayed as separate nodes in the tree structure (see also section 3.2.2.1).

4 TARGET CONTENT DEVELOPMENT

4.1 Start DTMT Target Content Design Time

DTMT design time for target content can be reached via transaction **DTMT_DTT**

Note: Together with DTMT, the authorization role **SAP_DTMT_DTT** is imported into the customer's system. It has to be assigned to those users who need to work with DTMT.

4.1.1 Target Scope

DTMT supports source content development for multiple migration scenarios in parallel. Therefore, migration scenarios are bundled together in scopes. Technically, a scope is nothing more than a bracket around a number of migration objects.

The target scope must have the same technical name as the source scope. In the example, this is TR02. To create a new scope, choose the pushbutton *Create Target Scope*.

Note: This figure shows the properties of the new scope, and is very similar to the source scope in section 2.1.1.

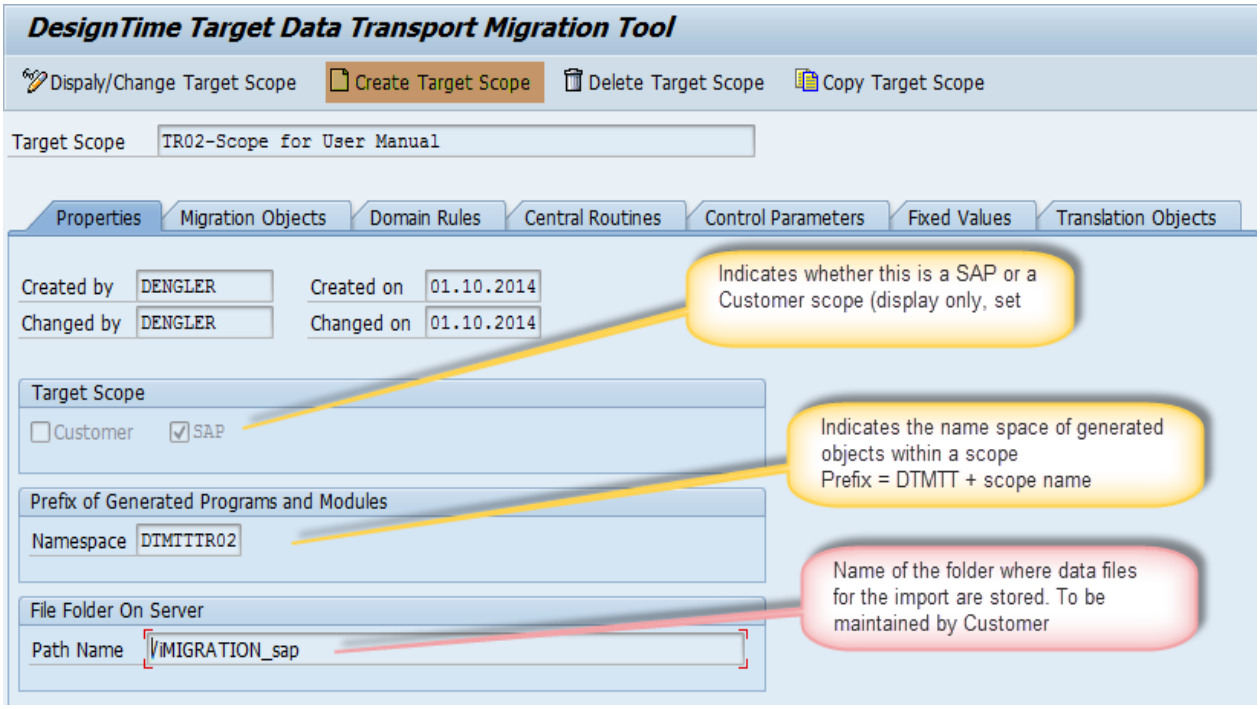


Figure 3-1

4.1.1.1 Copy Target Scope

Content provided by SAP is always protected against changes within customer systems. If a customer wants to change or enhance SAP content, he or she needs to copy the relevant SAP source scope to a customer version in advance. The whole DTMT functionality will then be available within the customer scope. The copy process of a scope is supported by a wizard with which you, and others, can determine which migration objects should be copied together with the scope.

4.1.2 Create Migration Object

Clicking the *Migration Objects* tab on the scope level screen displays a list of all migration objects of a specific scope. In this case, the list is empty because a new source scope was just created.
After pressing *Create Object* you will be asked to enter a migration object name and a description. In this example the migration object **VEN_TR02** is given the same name as in the source scope.

Note: It is very important to give the objects the same name in the source and target.

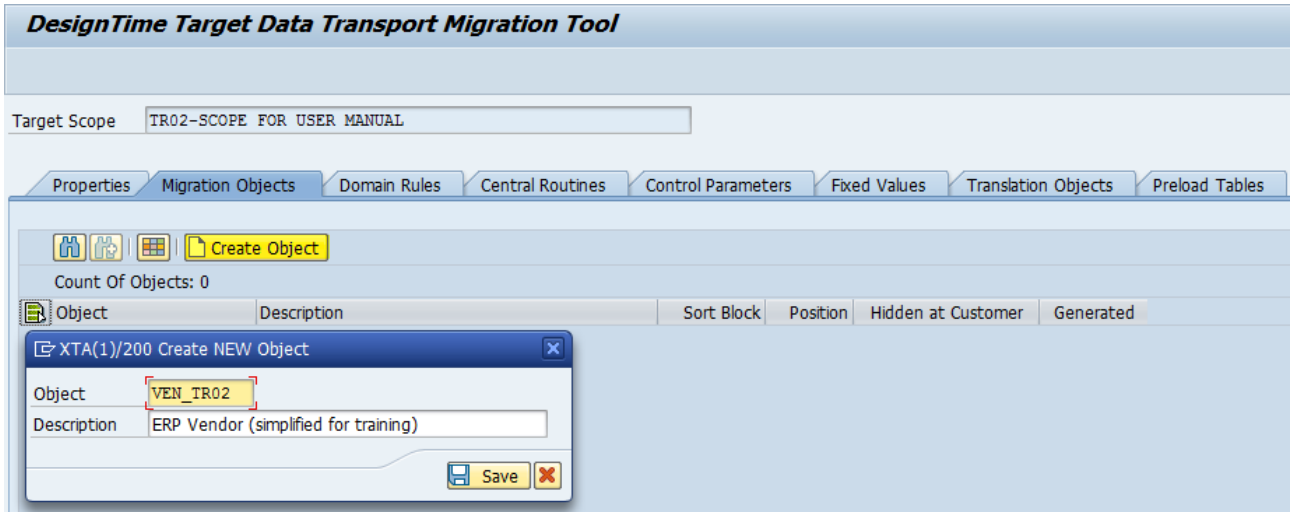


Figure 3-2

4.1.2.1 Migration Object Level Screen

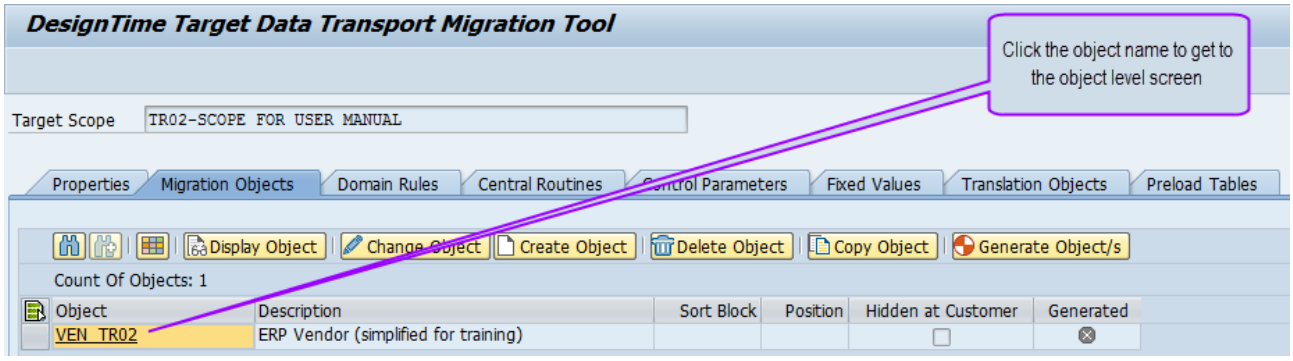


Figure 3-3

The figure below shows the main screen on migration object level that appears when you save the example object **VEN_TR02**. To get a first overview of the functionality behind the ten tabs, they will now be briefly explained. Later in this manual you will see more details for most of them when they are used during the content implementation process for the example object.

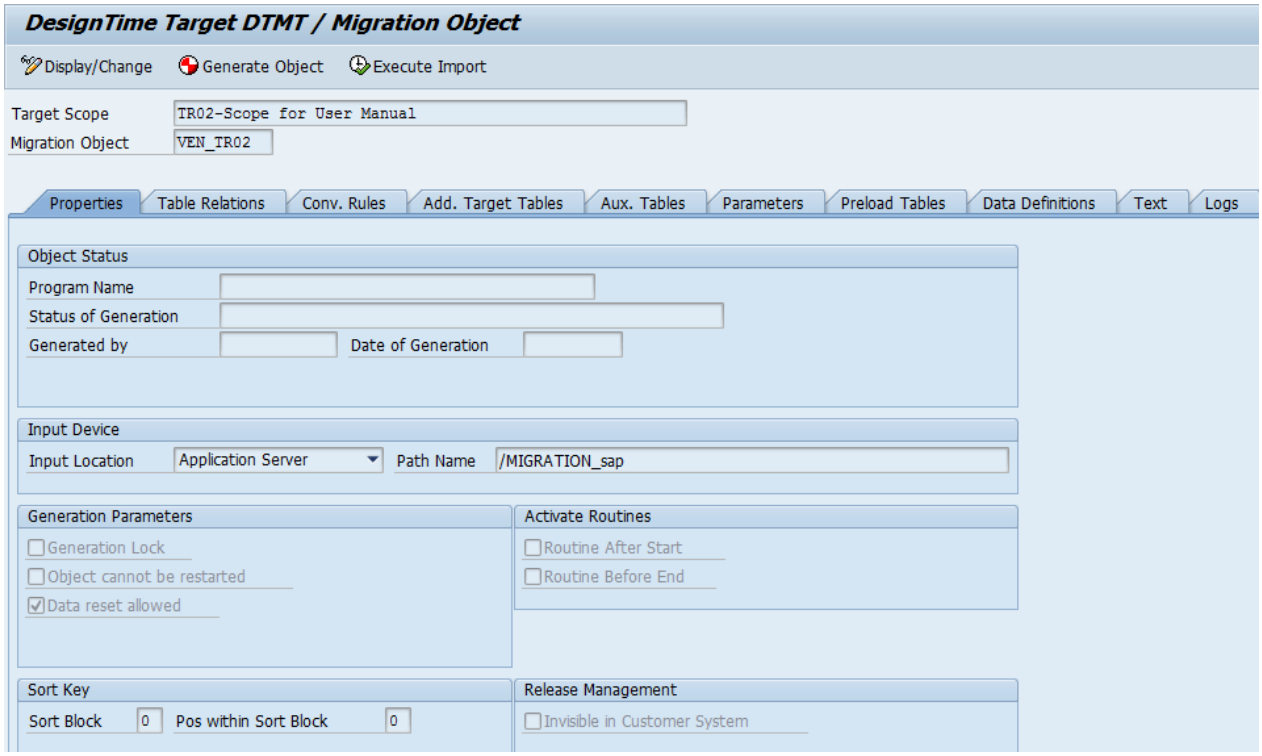


Figure 3-4

- *Properties*
  - Display and define the properties of a migration object
- *Table Relations*
  - Display and define the data model of the business object on table level
  - Additional functions on table level (type of insertion, input check, and so on)
- *Conv. Rules*
  - Field mapping for each table and field with different types of rules
- *Add. Target Tables*
  - Additional target tables without relation to any source table
- *Aux. Tables*
  - Display and define auxiliary tables (customizing only) that are attached to the file
- *Parameters*
  - Definition of parameters for the data import program
- *Preload Tables*
  - Tables that have to be preloaded in the main storage for better performance
- *Data Definitions*
  - Data definitions available on object level
- *Text*
  - Definition of expected texts in the file (corresponding to source)
- *Logs*
  - Display log files and extraction files

4.1.2.2 Migration Object Level "Properties" Tab

The fields on this screen are not mandatory, because the tool will use some defaults if required.
The most important fields on this screen are explained in the figure below.

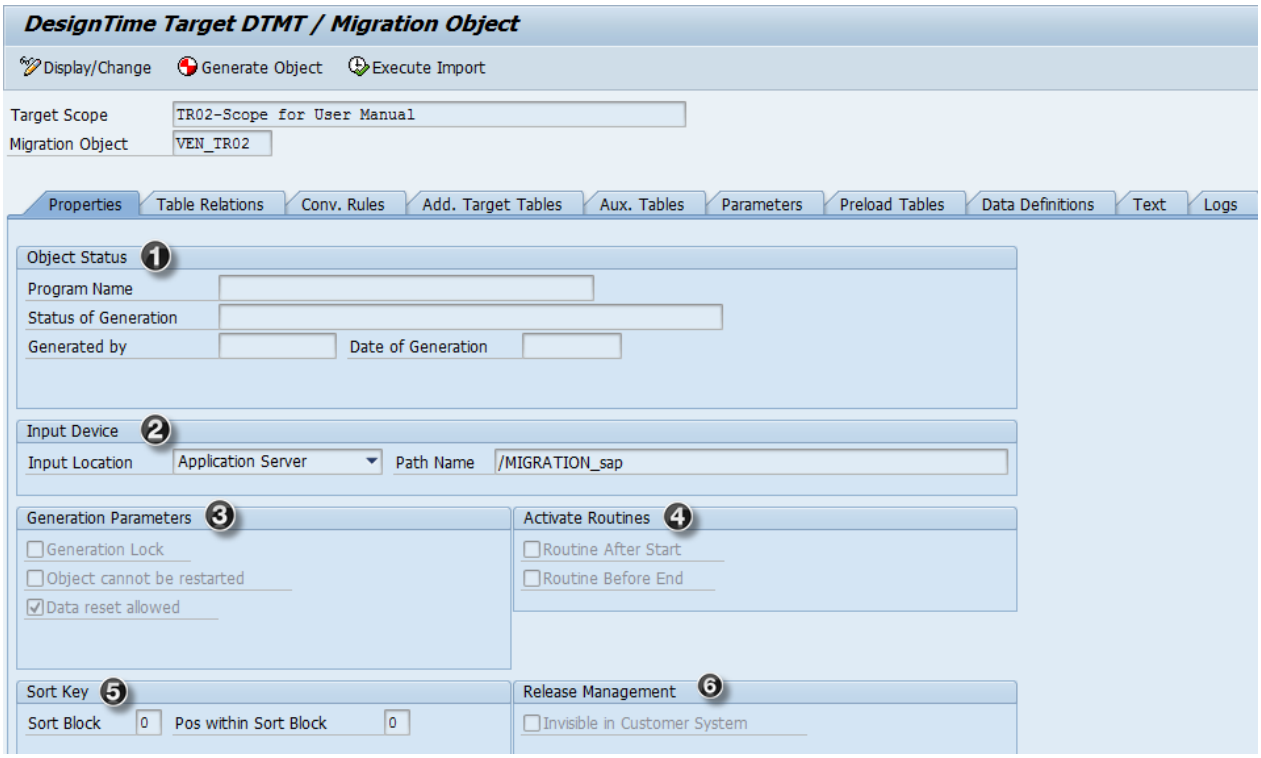


Figure 3-5

(1) Object Status

- *Program Name*
  Name of the main import program
  After you press the button *Generate Object*, the program will be created; it is executable if the generation is successful and without syntax errors.
  All relevant prerequisites have to be fulfilled (for example table relations, conversion rules, and so on).
- *Status of Generation*
  Status of generated main import program
  After you press the button *Generate Object*, the status will be set automatically by the tool *(*for example*, A Generation successful/M Generation no longer current due to object changes/I Generation missing/E Object incomplete/G Generation failed (with errors)*).

(2) Input Device

- *Input Location*
  Input location where the files that will be imported are stored.
  For performance reasons, it is recommended that the files are stored on the application server; "Local PC" makes sense only for the test case (for example, *Application Server/Locally on PC*)
- *Path Name*
  Name of the folder where the files are stored.
  (for example, **Application Server/MIGRATION_sap/Local** on PC **C:\temp)**

(3) Generation Parameters

- *Generation Lock*
  Indicator for generation lock of main import program
  Deselected: Generation of main import program is possible
  Selected: Generation of main import program is locked
- *Object cannot be restarted*
  Deselected: Migration object can be restarted
  Selected: Migration object cannot be restarted
- *Data reset allowed*
  If this parameter is selected, the delete function for the imported data will be activated. The import run creates deletion flags for all imported data of the current object, and allows you to delete all imported data after the import. This will be addressed again in a separate section.

  Selected: The delete function for migration data can be used for the object.
  Deselected: The delete function for migration data cannot be used for the object.

(4) Activate Routines

- *Routine After Start*
  Routine after start of the data import
  Indicator for call of the start routine:
  Deselected: Data import without start routine
  Selected: Call of form routine "**DTMTT_scope_BEGIN_objn**
  scope: Name of the scope
  objn: Name of the migration object
  The routine is called before the highest LOOP for each portion.
- *Routine Before End*
  Routine before end of data import
  Indicator for call of the end routine:
  Deselected: Data import without end routine
  Selected: Call of form routine **DTMTT_scope_END_objn**
  scope: Name of scope
  objn: Name of the migration object
  The routine is called after the last record of the file has been processed.

(5) Sort Key

- Migration scenarios mostly consist of several migration objects. If you wish to deviate from the alphabetic order in which the list of migration objects is normally displayed, you can use the fields *Sort Block* and *Pos* within a sort block to order the migration objects according to your needs.

(6) Release Management

- Only used by SAP Release Management. Checked objects will not be delivered.

4.1.2.3 Migration Object Level "Table Relations" Tab

The definition of the table relations on the target side must correspond exactly to the definition on the source side. This ensures that the data file is correctly read into the expected logical unit of work (LUW.)

To create this definition, choose the pushbutton *Read Source Table Relations* (at the lower edge of the screen). However, you first have to set up an RFC connection. The procedure for this is described in section **2.1.3.3.2**.

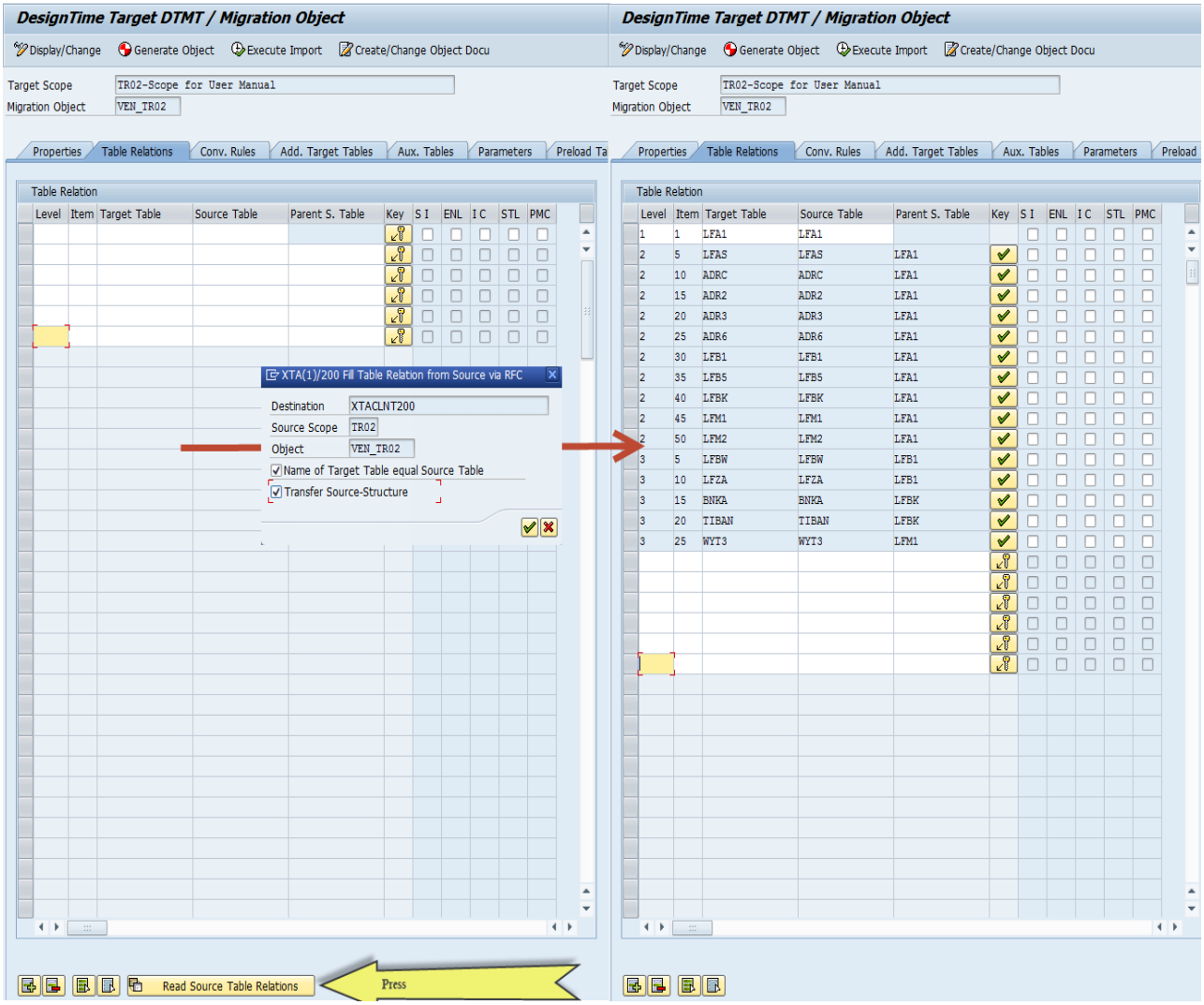
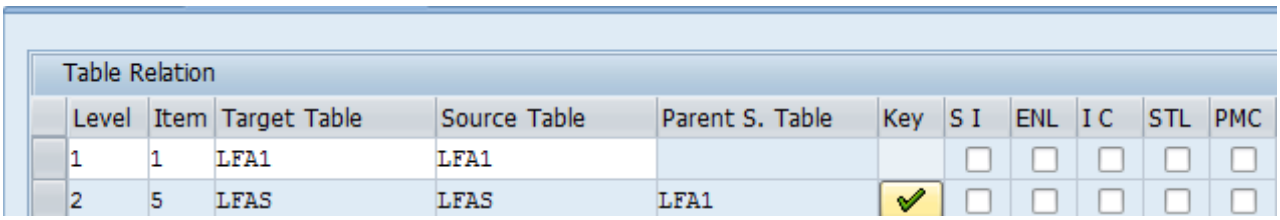


Figure 3-6

All relevant foreign key relations are automatically copied, and saved in the target environment. The column *Key* allows you to display or change the foreign key relations:



If there is no 1:1 relation between the target table and source table, you can make the necessary adjustments here. However, the source table relations (source table – parent source table) must always correspond exactly to those of the source. You can change the target table if necessary.

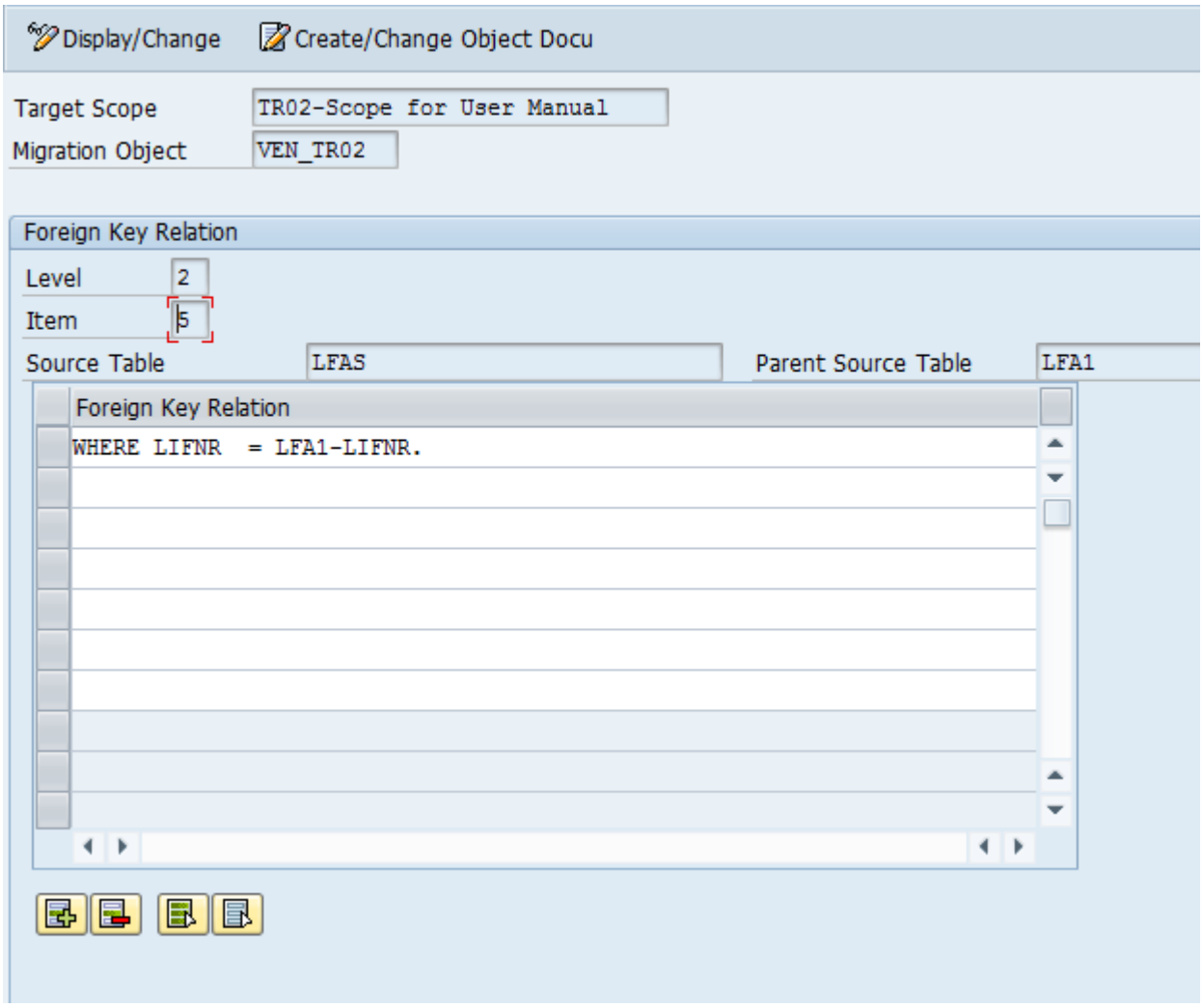

Display/Change
Create/Change Object Docu
Target Scope
TR02-Scope for User Manual
Migration Object
VEN_TR02
Foreign Key Relation
Level
2
Item
5
Source Table
LFAS
Parent Source Table
LFA1
Foreign Key Relation
WHERE LIFNR = LFA1-LIFNR.

Figure 3-7

The most important fields on this screen are explained in the figure below.

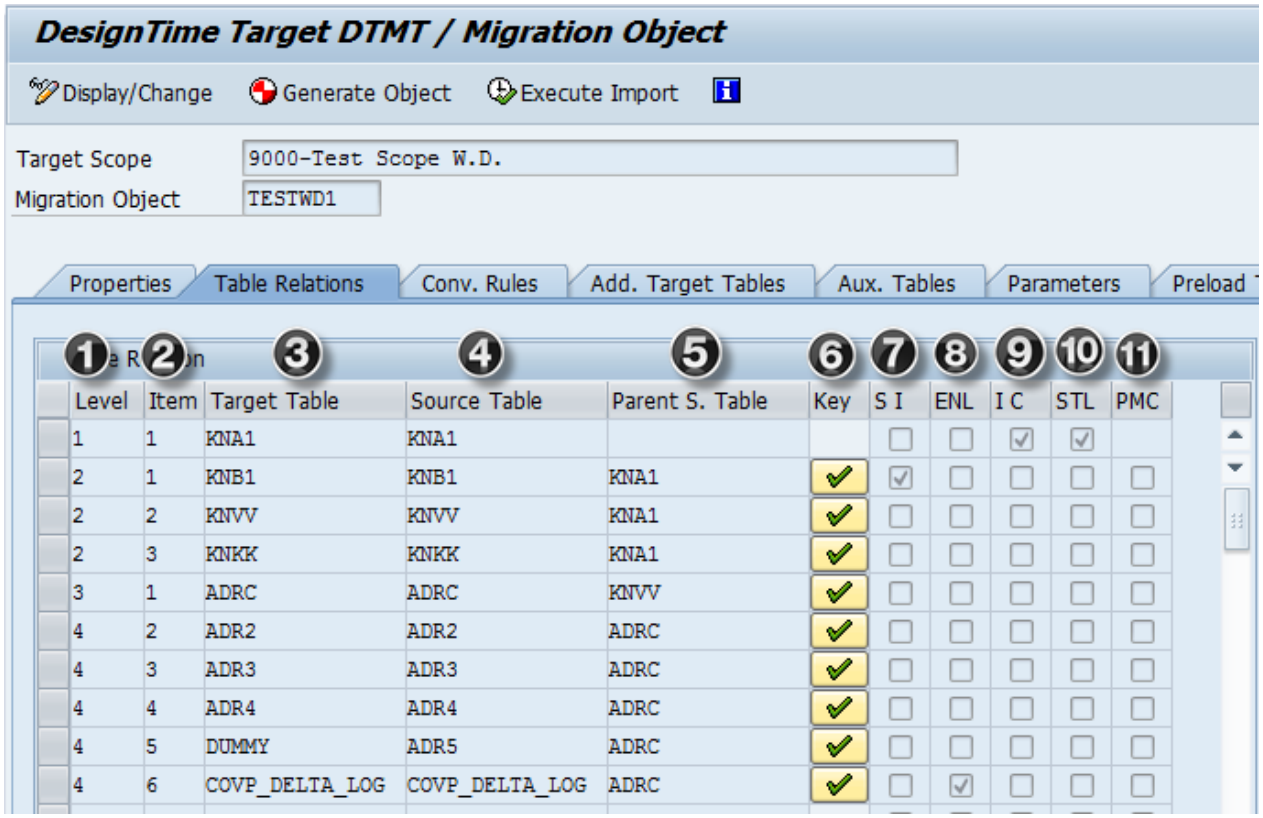


Figure 3-8

(1) Level

- The level of table relations describes the place within a sequence of a "logical unit of work" (LUW)
- For each level, with the exception of level 1, a parent source table is mandatory; this is generally the source table one level above.

(2) Item

- Defines an item within a level or block

(3) Target Table

- *Target Table* is the table that will receive the data of the mapped source table

(4) Source Table

- *Source Table* is the table that will send the data to the mapped target table

(5) Parent Source Table

- For each level, with the exception of level 1, a *Parent Source Table* is mandatory; this is generally the source table one level above.

(6) Key

- The column *Key* allows you to display or change the foreign key relations; see figure 3-7

(7) SI Single Insert

- This flag controls the generation of the import program. If it is selected, the tool will generate a single insert statement instead of a mass insert.

(8) *ENL Endloop Form Routine* **(**Flag for generating Perform to Endloop-Form-Routine)

- The flag controls generation of the import program, and generates a perform in a subroutine into which the database table is written. In display mode, it can be shown via double-click.
- If you want to create a form routine into which the database is written, activate this field.

(9) IC Input Check (Flag for generating Perform to Input-Check-Form-Routine)

- The flag controls generation of the import program, and generates a Perform in a subroutine that is open for coding, to check the source data (the variable **RCODE** will be checked: **RCODE = 'X'**).
- **Selected**: Call form routine **DTMTT_oname_IC_tname**
  (oname = name of the migration object**, tname** = target table name).
- **Deselected**: Check routine not required

(10)STL Startloop Form Routine (Routine at Beginning of Loop Processing)

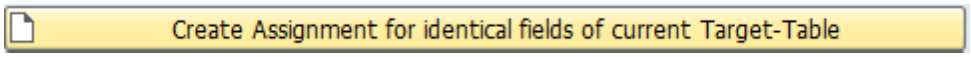
- The flag controls generation of the import program, and generates a Perform in a subroutine that is open for coding, to control the further processing or preparation (the variable **SCONT** will be checked: **SCONT = 'X**').
- **Selected**: Call form routine **DTMTT_oname_STL_tname**
  (**oname** = name of the migration object**, tname** = target table name).
- **Deselected** : Startloop routine not required

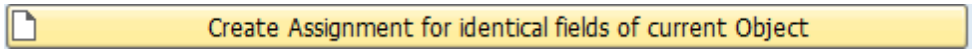
(11)*PMC* Creates a *Move-Corresponding* of the *Parent Source Table* to *Target Table*

- The *Move-Corresponding* will be carried out by the *Parent Source Table* to the *Target Table* and not by the current table, for example **MOVE-CORRESPONDING LS_SOURCE_KNA1 TO LS_TARGET_KNB1**

4.1.2.4 Migration Object Level "Conversion Rules" Tab

The field mapping takes place top-down. You need to define a rule for each target field of a target table. *The Conversion Indicator* describes how the field is filled.
There is often a 1:1 relation between a field of the source table and a field of the target table. The tool offers a function for automatic mapping if the field name is the same in the source table and target table. However, a prerequisite for this function is that the object has already been created and generated in the source system. In addition, the name tabs (structures of the source tables) must be transferred.
You can apply this function per target table or for the entire object. The following pushbuttons are available for this purpose:





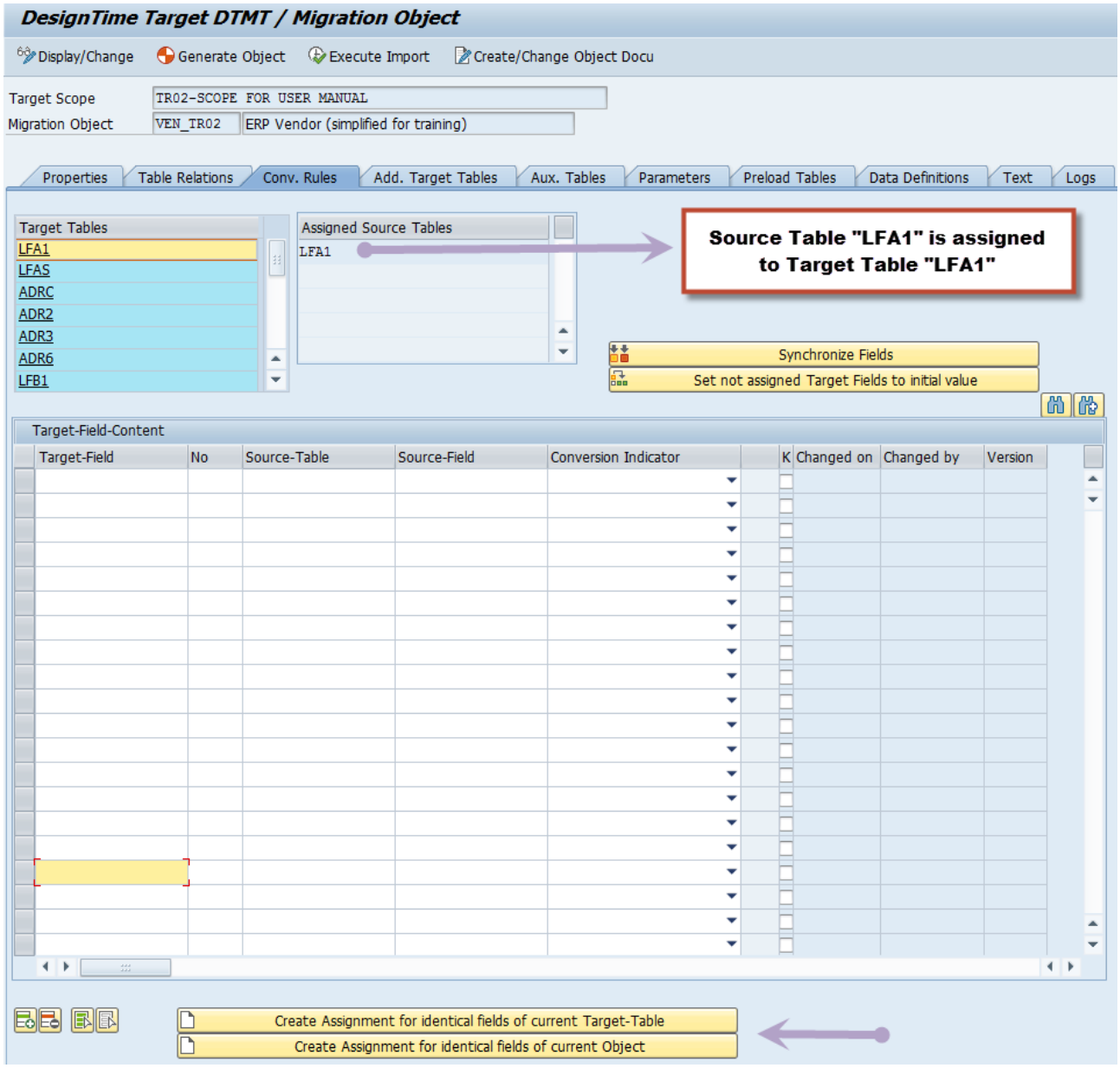
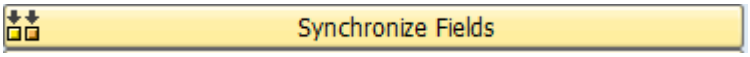


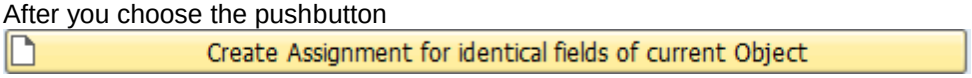
Figure 3-9



To synchronize the mapping that exists for the displayed table with the current structure of this table in the system, choose the pushbutton *Synchronize Fields.* This also determines any fields that are not yet mapped, and deletes non-existent fields from the mapping.



The pushbutton *Set not assigned Target Fields to initial value* sets all fields that do not yet have a rule to conversion indicator *1*, as an initial value.

After you choose the pushbutton


, all fields are mapped as described above. The example of table *WYT3* shows (see figure 3.10) that different *Conversation Indicators* were set.

- ***2 Move:*** A 1:1 relation was determined, and so a transfer of the field was generated.
- ***0 Domain Rule Exists:*** The system determined that a domain rule exists, which is proposed. It needs to be further processed, using the pushbutton  .
- ***9 Open:*** Nothing was determined for this field. It is flagged with *Open* – this means that the field must be processed.

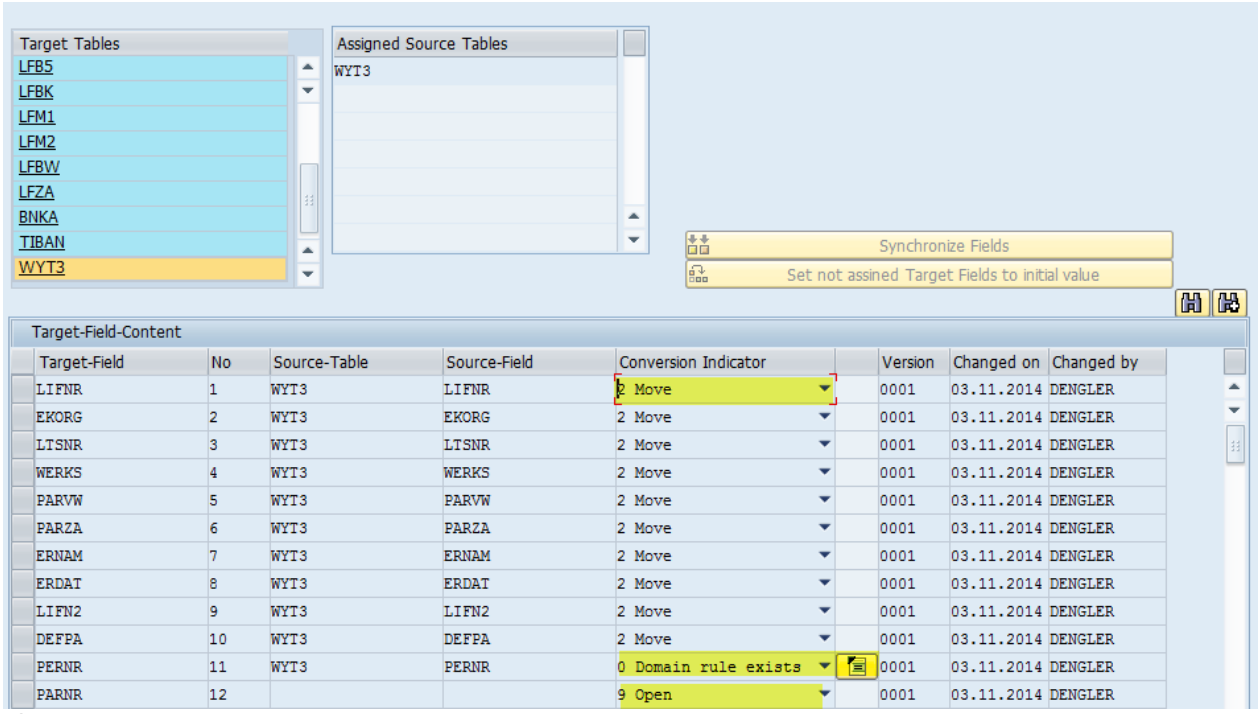
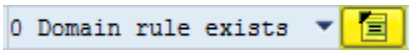
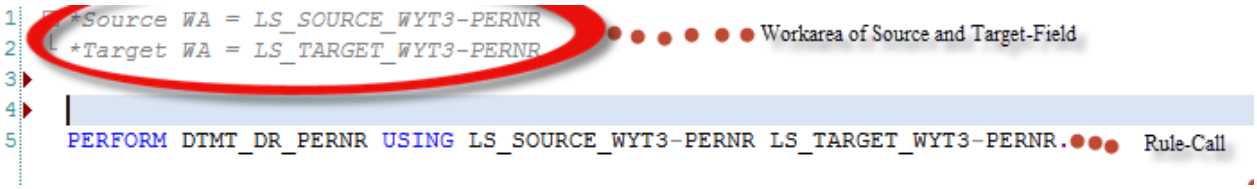


Figure 3.10

Processing the proposal for conversion indicator *0* in change mode: 

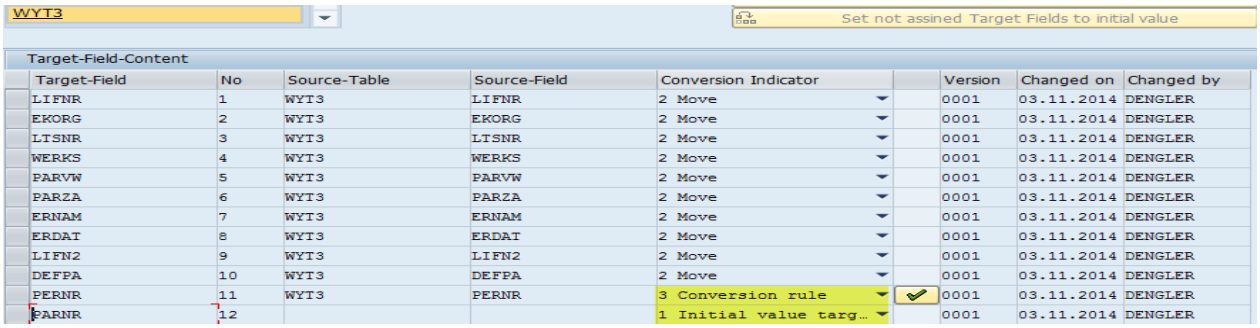
In ABAP editor mode, the call of the determined domain rule is automatically proposed, with reference to the work areas used for the affected fields.



The field marked as *9 Open* must either be supplied with a rule (*3 Conversion Rule*), or set to initial (*1 Initial value target field*).



Figure 3.11 shows the finished result after the processing and mapping of all fields for target table *WYT3.*



Target-Field	No	Source-Table	Source-Field	Conversion Indicator		Version	Changed on	Changed by
LIFNR	1	WYT3	LIFNR	2 Move		0001	03.11.2014	DENGLER
EKORG	2	WYT3	EKORG	2 Move		0001	03.11.2014	DENGLER
LTSNR	3	WYT3	LTSNR	2 Move		0001	03.11.2014	DENGLER
WERKS	4	WYT3	WERKS	2 Move		0001	03.11.2014	DENGLER
PARVW	5	WYT3	PARVW	2 Move		0001	03.11.2014	DENGLER
PARZA	6	WYT3	PARZA	2 Move		0001	03.11.2014	DENGLER
ERNAM	7	WYT3	ERNAM	2 Move		0001	03.11.2014	DENGLER
ERDAT	8	WYT3	ERDAT	2 Move		0001	03.11.2014	DENGLER
LIFN2	9	WYT3	LIFN2	2 Move		0001	03.11.2014	DENGLER
DEFPA	10	WYT3	DEFPA	2 Move		0001	03.11.2014	DENGLER
PERNR	11	WYT3	PERNR	3 Conversion rule	✓	0001	03.11.2014	DENGLER
PARNR	12			1 Initial value targ…		0001	03.11.2014	DENGLER

Figure 3.11

This postprocessing must be carried out for each listed target table. The import program can be generated only once all target fields of each individual target table have a valid *Conversion Indicator.*

4.1.2.5 Migration Object Level "Additional Target Tables" Tab

This tab is used to define additional target tables that cannot be supplied within the normal mapping, or are not to be assigned to a source table. These tables must be supplied by individual coding. The tool automatically generates all necessary work areas and internal tables. They are subject to the same naming conventions as those of the mapped tables.

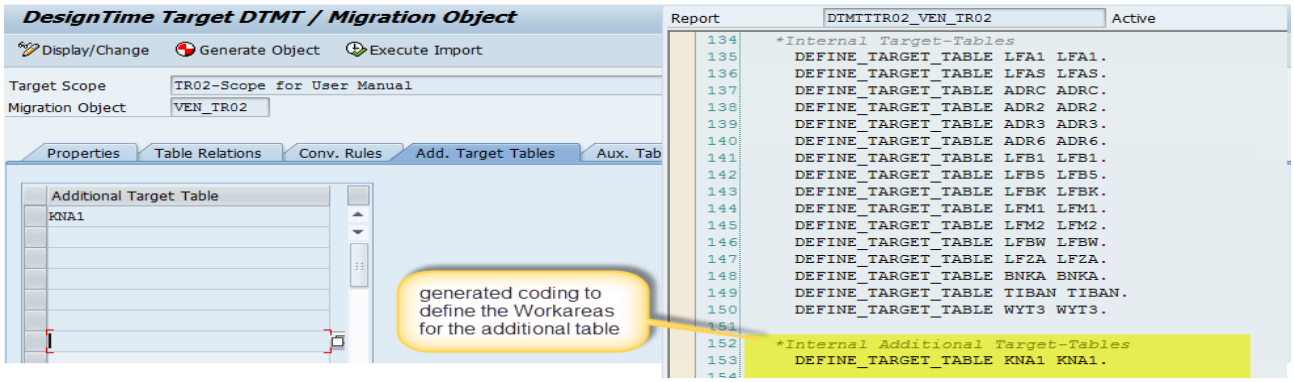


Figure 3.12

The work area and internal table generated in this way have the following names in this example:

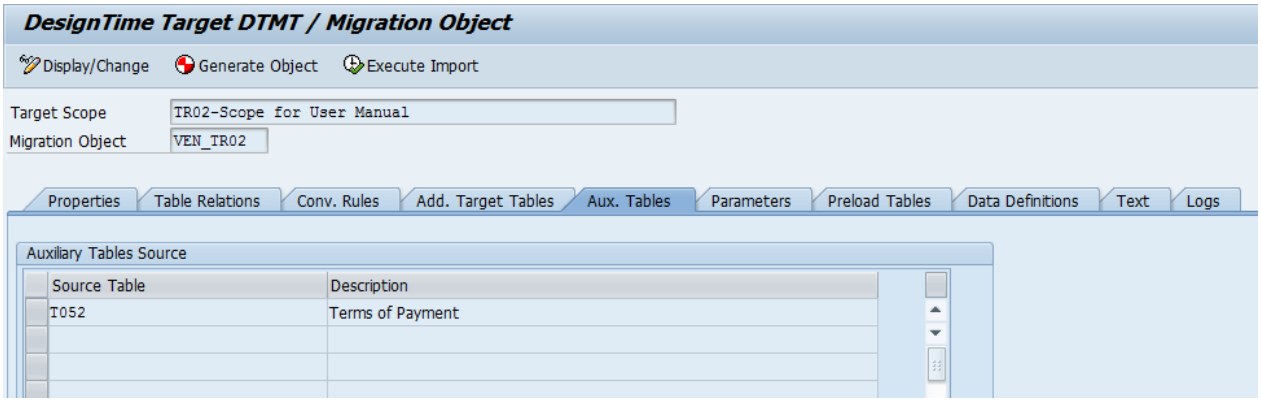
- **LS_TARGET_KNA1**
- **LT_TARGET_KNA1**

The writing to the database takes place via ARRAY-INSERT, and is automatically generated by the tool:

```
612   IF LINES( LT_TARGET_KNA1 ) > 0.
613     IF MG_TESTL NE 'X' AND ERROR_FLAG NE 'X'.
614       TTABLE_KNA1 = TTABLE_KNA1 + LINES( LT_TARGET_KNA1 ).
615       INSERT KNA1 FROM TABLE LT_TARGET_KNA1 ACCEPTING DUPLICATE KEYS.
616       IF SY-SUBRC NE 0.
617         ROLLBACK WORK.
618         PERFORM INSERT_FAIL_TAB TABLES LT_TARGET_KNA1 USING LS_TARGET_KNA1 'KNA1' KNA1.
619       ENDIF.
620       PERFORM FILL_DEL_KEY_TAB TABLES LT_TARGET_KNA1 USING LS_TARGET_KNA1 'KNA1'.
621     ELSE.
622       PERFORM FILL_DEL_KEY_TAB TABLES LT_TARGET_KNA1 USING LS_TARGET_KNA1 'KNA1'.
623     ENDIF.
624   ENDIF.
```

4.1.2.6 Migration Object Level "Auxiliary Tables" Tab

This tab is identical to the *Extraction* tab and is described in section 2.1.2.4.
The tool automatically generates the work area and internal table, and will fill this from the extraction file after starting the import. The content of the customizing source table will be available and can be used for several evaluations by creating any additional coding.

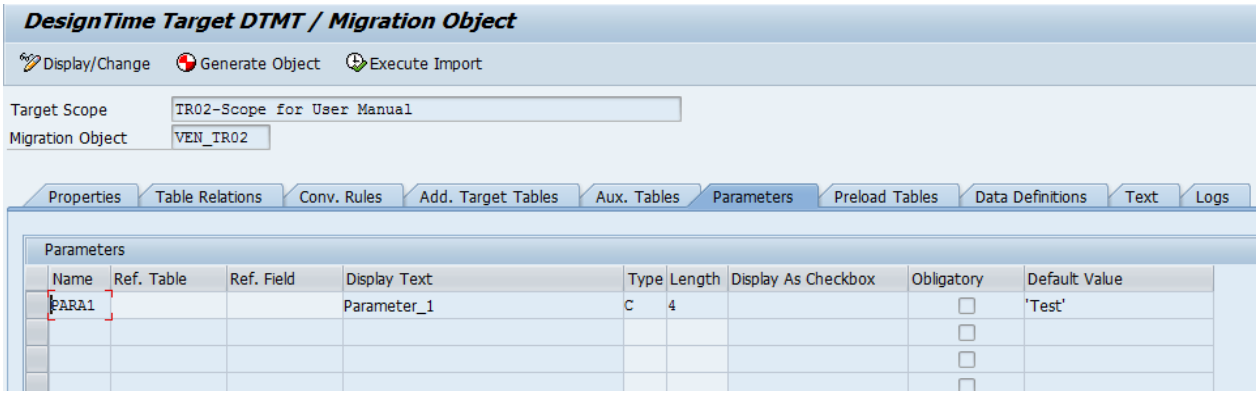



```
155    *Internal Source-Auxiliaries-Tables
156      DEFINE_SOURCE_AUX_TABLE T052 DTMT_FW_STT052.
157
```

The generated work area will be named **LS_SOURCE_AUX_T052**.
The generated internal table will be named **LT_SOURCE_AUX_T052**.

4.1.2.7 Migration Object Level "Parameters"

Definitions of parameters can be reached at each level of the import program.
These parameters will appear and can be appraised at the start of the import.

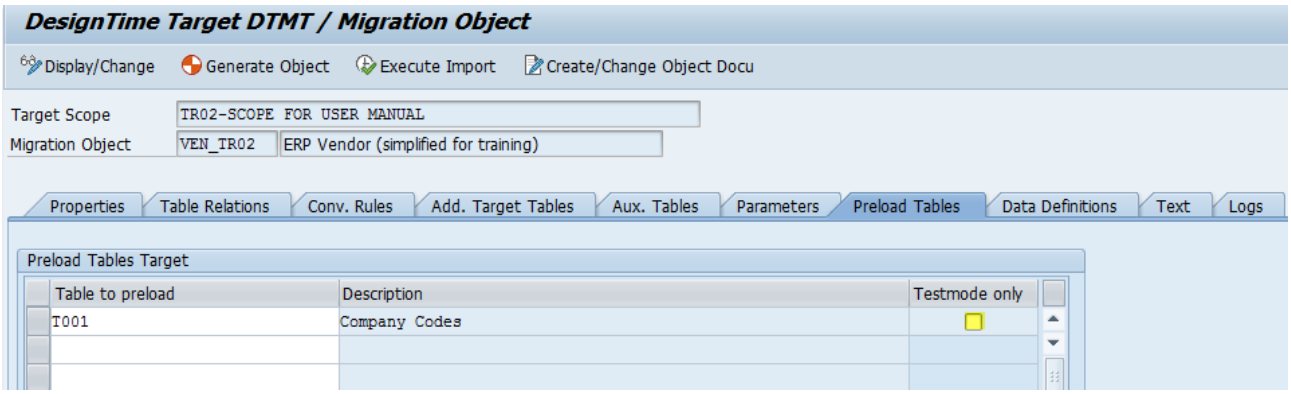



```
27  *&---------------------------------------------------------------------*
28  *&  object parameter
29  *&---------------------------------------------------------------------*
30  SELECTION-SCREEN BEGIN OF BLOCK OBJ WITH FRAME TITLE TEXT-009.
31  PARAMETERS:
32  PARA1 TYPE C LENGTH 000004 DEFAULT 'Test'.
33  SELECTION-SCREEN END OF BLOCK OBJ.
34  *&---------------------------------------------------------------------*
```

4.1.2.8 Migration Object Level "Preload Tables"

Preload tables are customizing tables that are read during the processing of the data import, and loaded into internal tables (for performance reasons), to avoid too many instances of read access to the database.
These internal tables are filled directly after the start of the data import.
Accessibly just for the object!

LS_PRELOAD_Table = Name of work area of the internal tables
LT_PRELOAD_Table = Name of the internal table



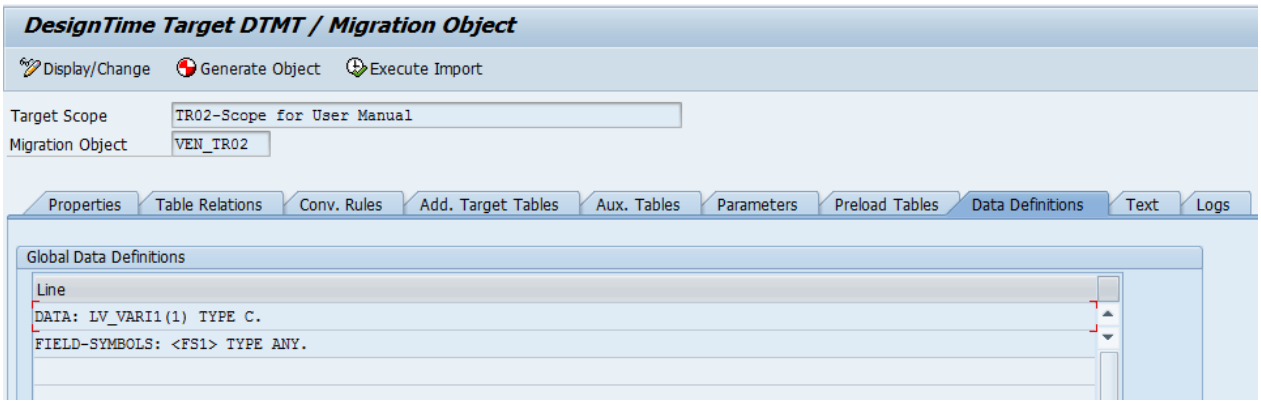
*If **"Testmode only"** is selected, the preload function is just called in mode "Test-Run".*

Report DTMTTR02_VEN_TR02 Active

```
157   *Internal Source-Auxiliaries-Tables
158
159   *Internal Preloaded-Customizing-Tables on Scope Level
160
161   *Internal Preloaded-Customizing-Tables on Object Level
162     DEFINE_PRELOAD_TABLE T001.
163     IF MG_TESTL = 'X'.
164       SELECT * FROM T001 INTO TABLE LT_PRELOAD_T001
165                                 ORDER BY PRIMARY KEY.
166     ENDIF.
```

4.1.2.9 Migration Object Level "Data Definitions"

Additional field variables, field symbols, or any type of data definition that is available in each section of the import program can be defined on this tab.
For demonstration purposes, some field symbols and field variables have been defined, to show how it looks in the generated import program.




```
5
6   REPORT DTMTTR02_VEN_TR02 MESSAGE-ID DTMT_FW_T_CONTENT.
7
8   TABLES: ADR2,
9           ADR3,
10          ADR6,
11          ADRC,
12          BNKA,
13          KNA1,
14          LFA1,
15          LFAS,
16          LFB1,
17          LFB5,
18          LFBK,
19          LFBW,
20          LFM1,
21          LFM2,
22          LFZA,
23          TIBAN,
24          WYT3.
25
26  DATA: LV_VARI1(1) TYPE C.
27  FIELD-SYMBOLS: <FS1> TYPE ANY.
```

4.1.2.10 Migration Object Level "Text" Tab

At this point, the migration object must receive information about whether long texts for the tables to be migrated should be transferred. In the extraction description in section **2.1.2.3.1.2**, these texts were made known to the migration object. Now the system needs to refer to them in the import, so that the texts for the respective extract are recognized and correctly assigned.

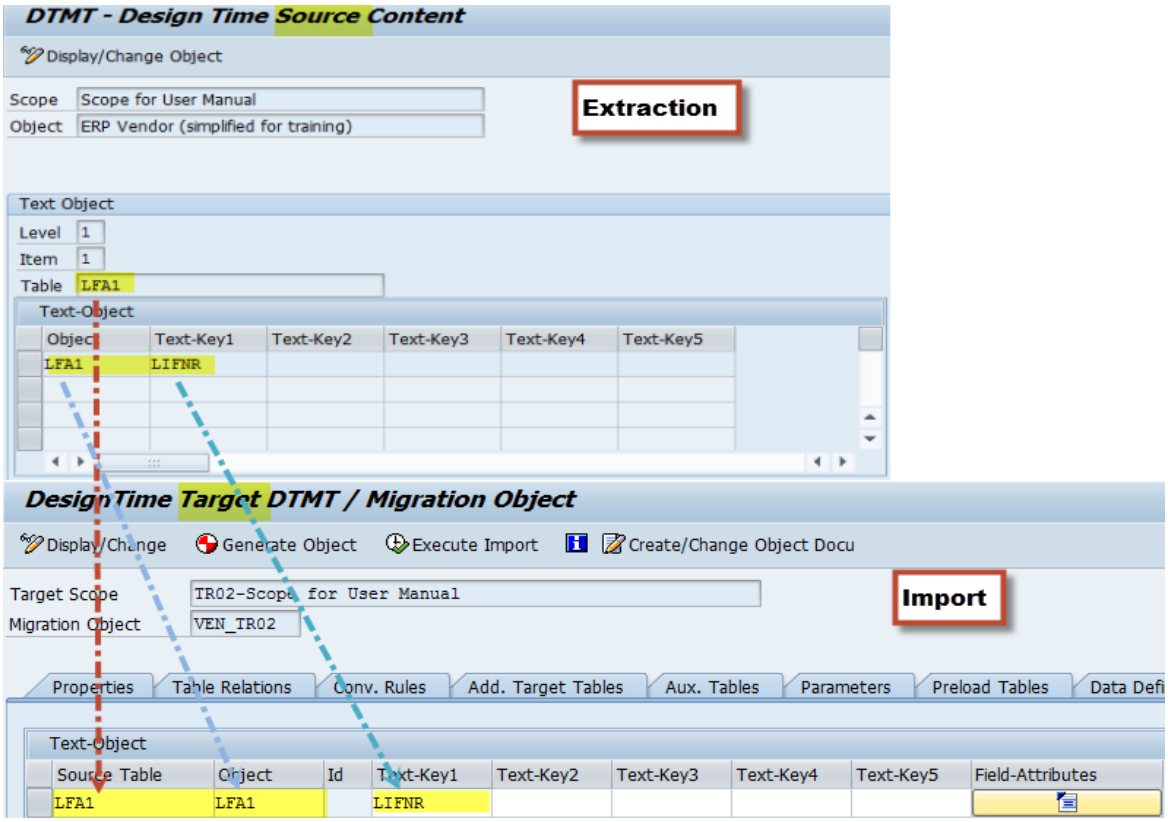
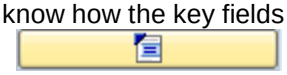


Figure 3.13

The migration object must also know how the key fields (text keys 1 to 5) of the texts are defined. Choose the pushbutton *Field Attributes*. 

Since the structures (name tabs) of the source tables should already have been transferred to the target environment, the tool can propose the attributes of the key fields.

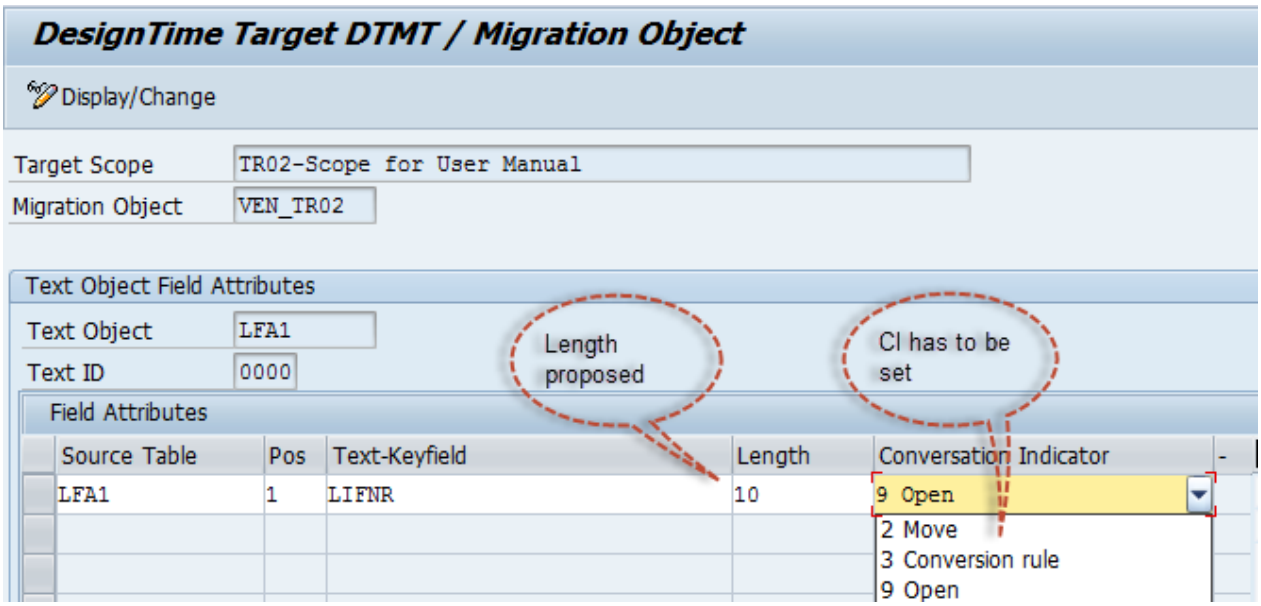


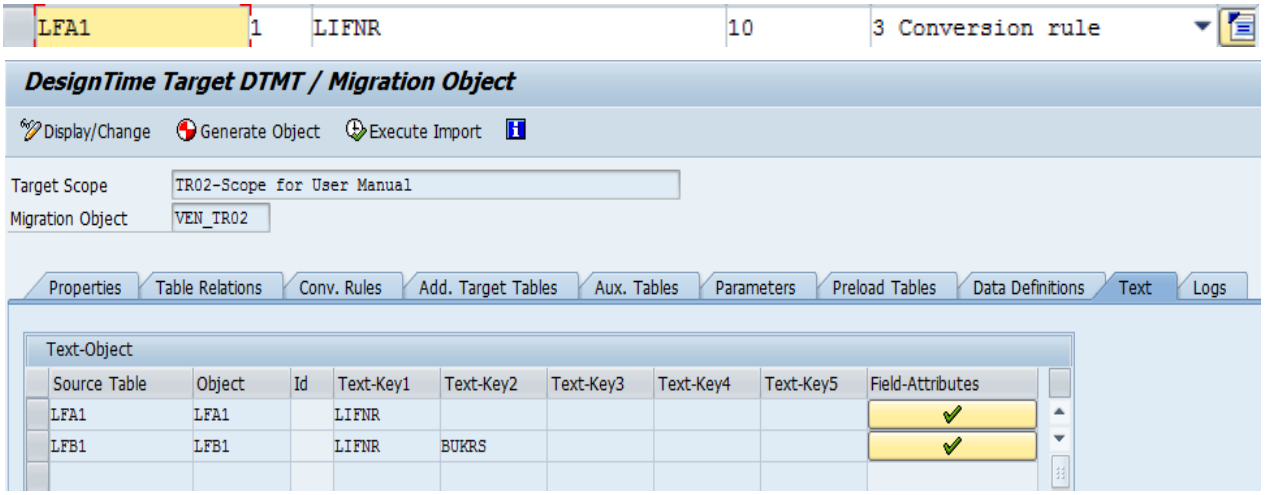
Figure 3.14

The conversion indicator must be set so that the object knows how this field should be migrated. Either 1:1 by setting the indicator *2 Move*, or via conversion rule by setting indicator *3 Conversion rule*.

If you set indicator *3 Conversion rule*, you must then allocate the corresponding conversion rule by choosing button :

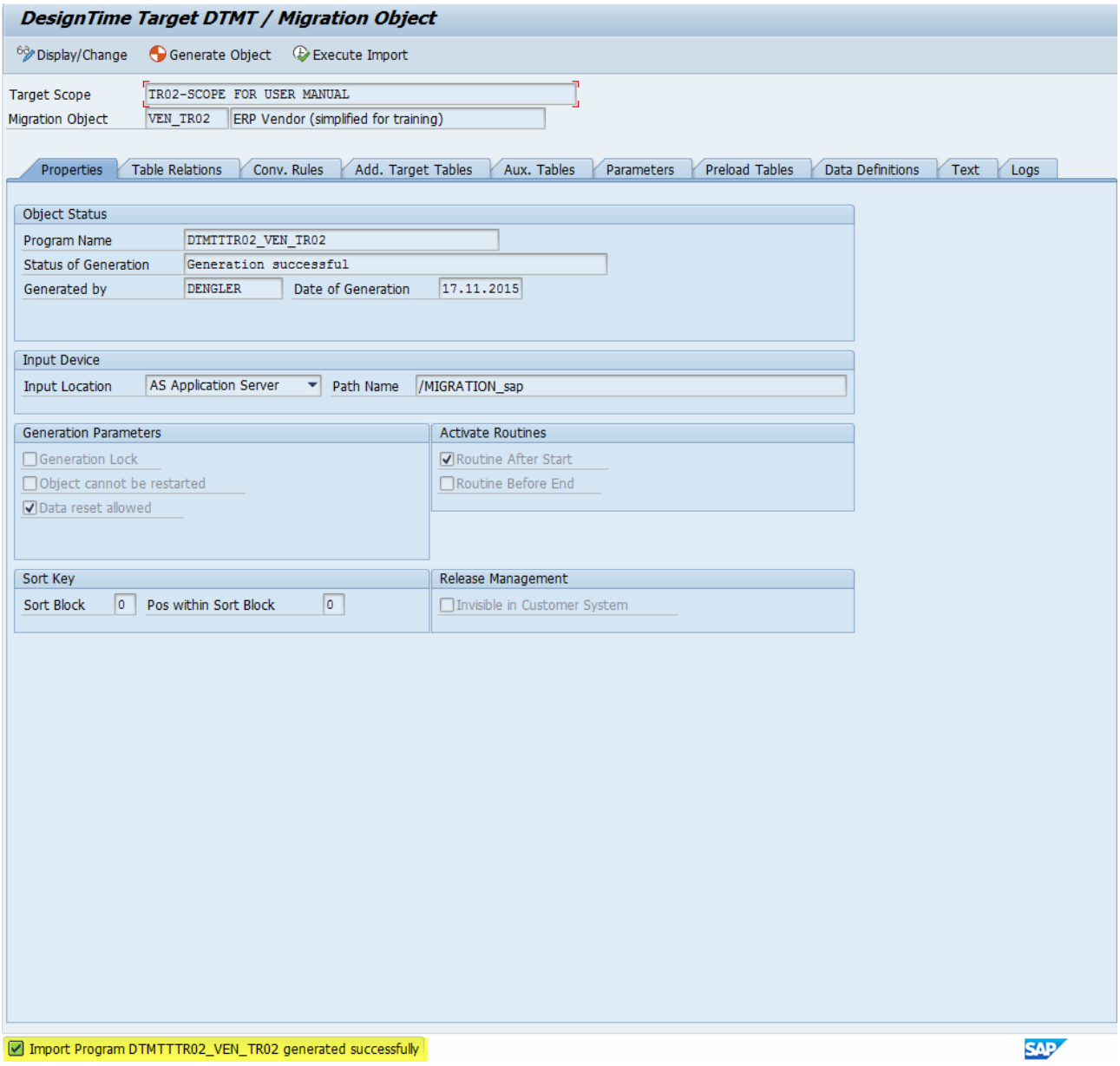
In this example, the indicator *2 Move* is set, so that the field is supplied 1:1.

After the supply of all necessary texts, the example looks like this:



4.1.2.11 Migration Object Level "Generation & Execution Object"

After all necessary settings and information have been supplied to the object, the import program can be generated. You generate the import program by choosing the pushbutton Generate Object. It can then be executed by using Execute Import.



4.1.3 Create Domain Rules

These are central conversion rules that can be used in all MOs of a scope. They are always connected to a domain, and are used to convert fields assigned to the same domain to ensure a consistent conversion across MOs within a scope. Some of the domain rules consist of two or more variants reflecting different types of conversion rules (for example **Move; Translate**). In this example you will see them in the migration customizing/*Domain Rules* section, and you can activate the variant you want in the customizing step. If you do not activate a variant, the standard variant will be used.

Note: Domain rules are provided for most of the domains referring to organizational entities like company codes, plants, and so on. They are also provided for other central fields like customer, vendor, or material numbers because the values of those fields are often the subject of value changes. If you need to consistently change the values of fields where a domain rule exists, you can do this easily. However, it is in your responsibility to ensure that the changes make sense. Renumbering of plants or company codes, for example, becomes very simple. Should the variants of a domain rule designated in the system not be sufficient for your environment, new variants can be created or existing variants can be changed using the DTMT design time. However, domain rule changes should be undertaken only by experienced migration experts with sufficient knowledge of the respective application area.

If you click the *Domain Rules* tab on the scope level screen, a list of all domain rules of a specific scope will be displayed. In this example, the list is empty because a new source scope has just been created.
By pressing *Create Domain*, you will be asked to enter a domain name, a domain suffix, and a description. In this example, the migration domain is called **LIFNR**; the same name as the existing domain in the DDIC.

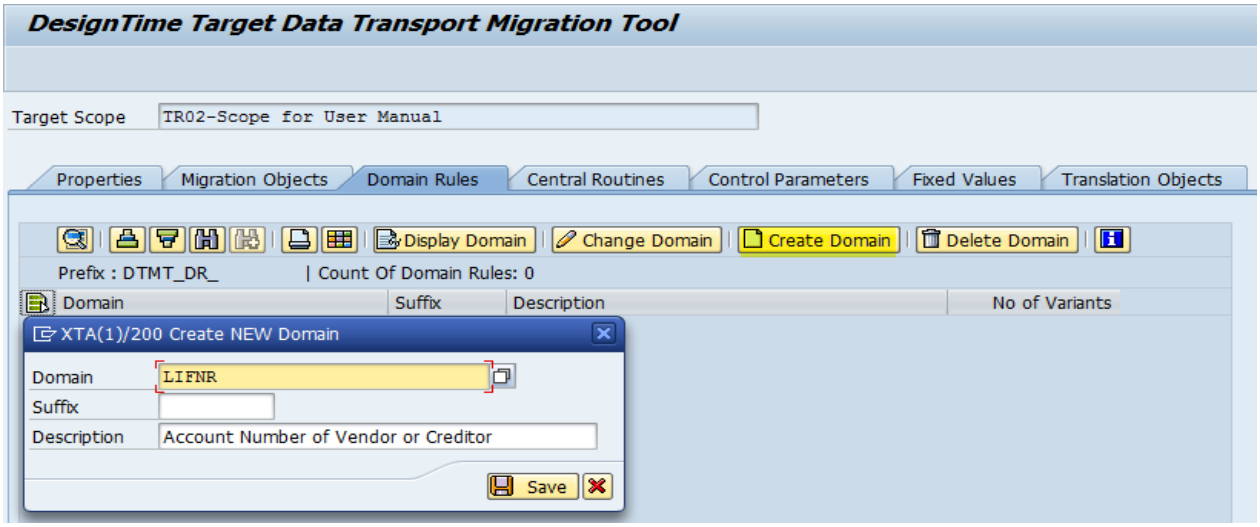


Figure 3.15

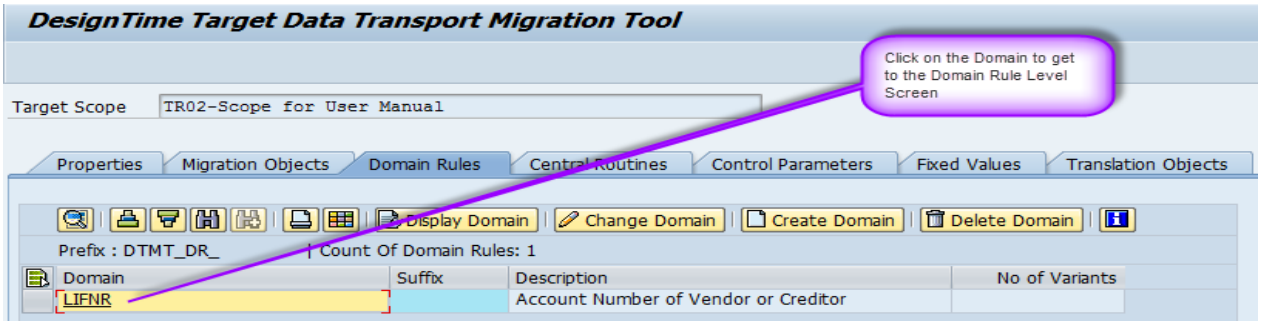


Figure 3.16

4.1.3.1 Domain Rule Screen

Up to four variants can be defined for a domain rule. You use migration customizing to determine which variant is actually used in the import.

1. Variants:1 to 4
2. Values:
    a. Standard Variant
    The standard variant is used if no migration customizing has been carried out.
    b. Translation Variant
    Variant that should be defined as the translation
    c. Automatic Translation allowed
    Allows automatic translation, and automatically generates the entry in the translation table.
    d. No. of Input Parameters
    Number of actual input parameters in the call of the respective subroutine
    e. No. of Output Parameters
    Number of actual output parameters in the call of the respective subroutine

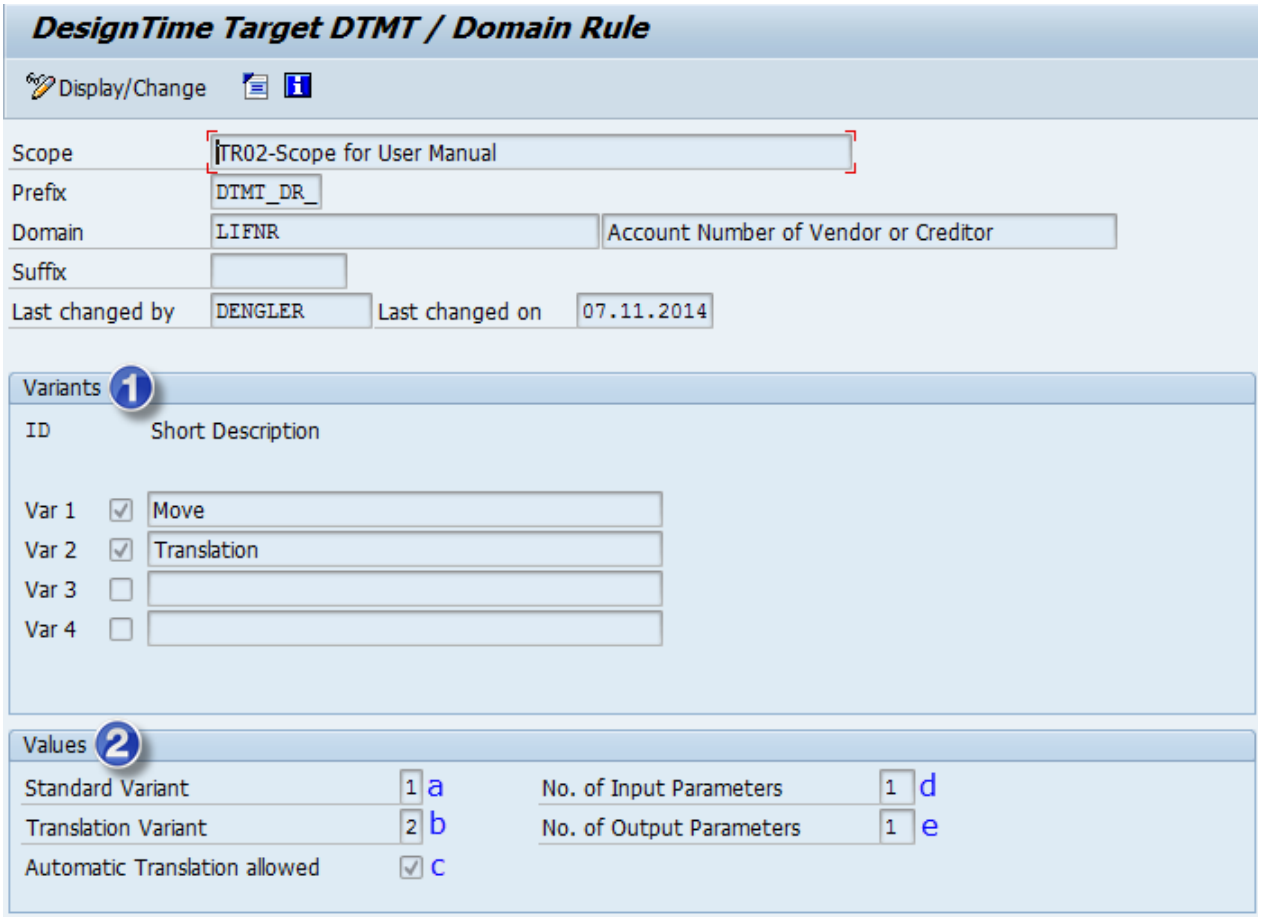
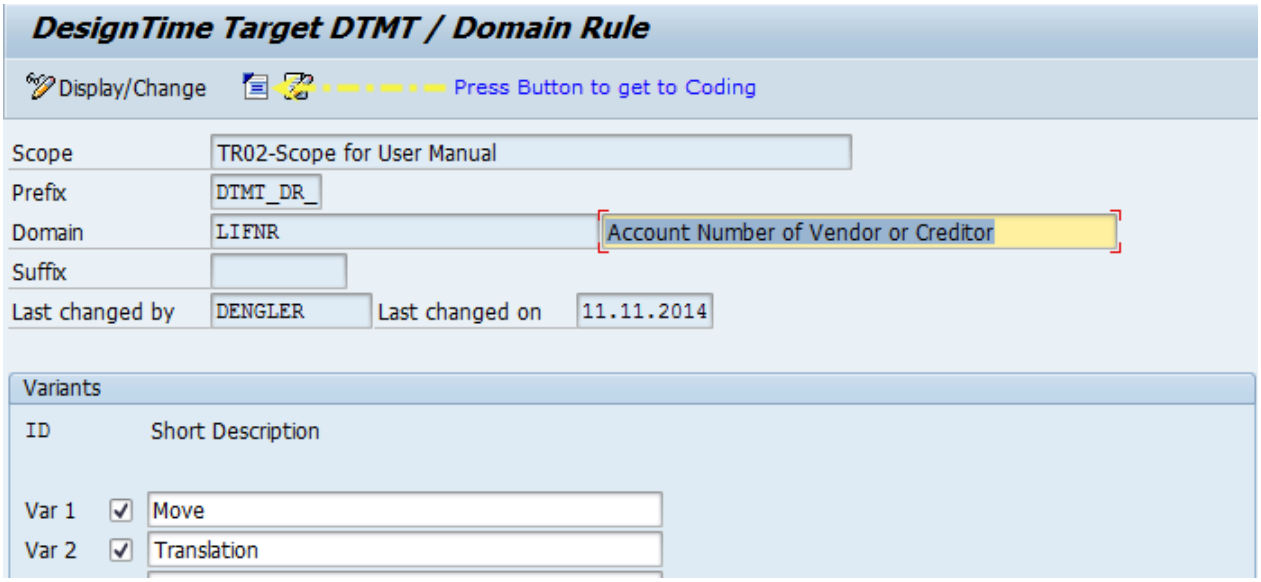
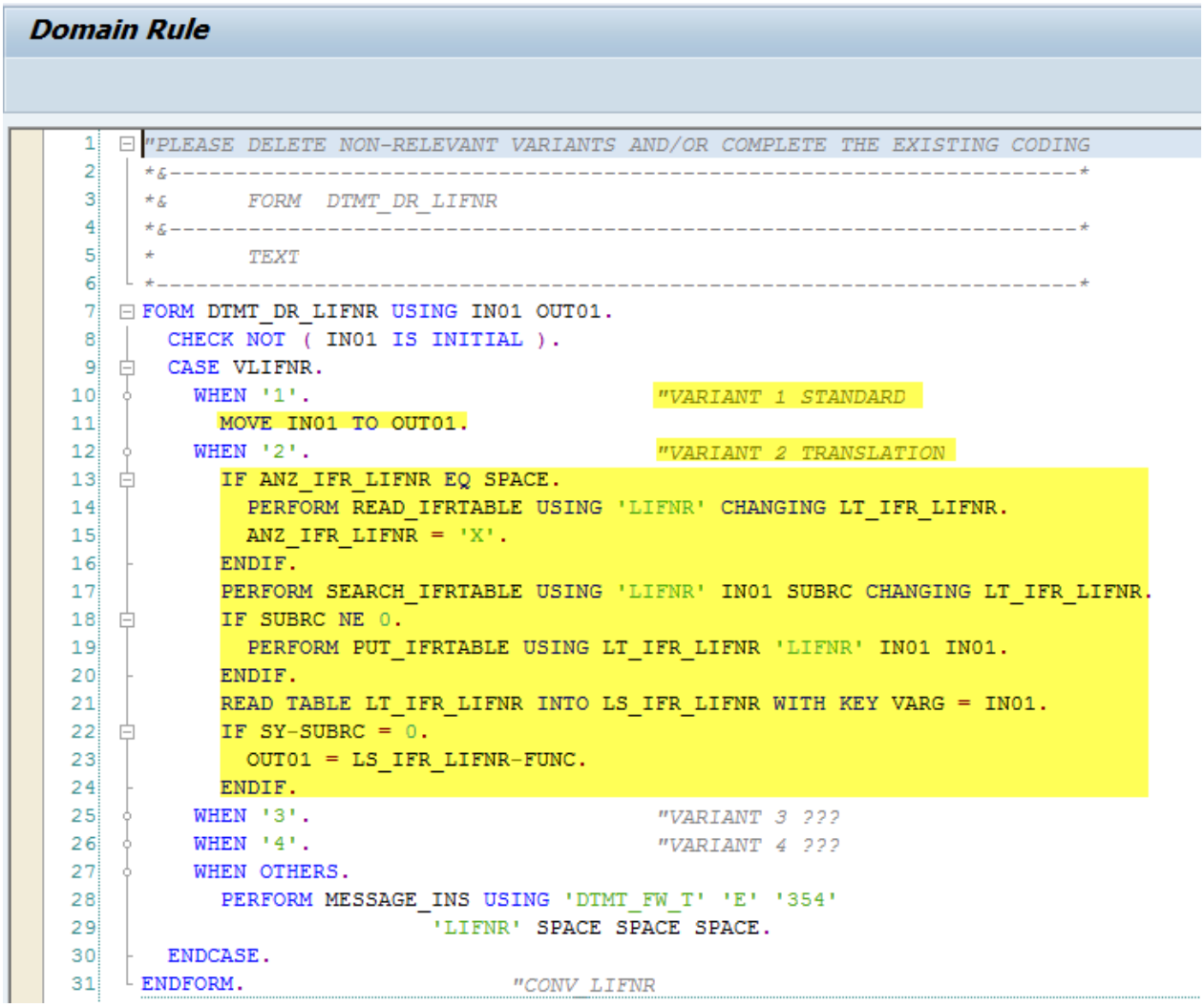


Figure 3.17

4.1.3.2 Domain Rule Coding



As an example, here is the coding proposed by the tool for the defined variants 1 and 2:



4.1.3.3 Domain Rules Call

The call of the domain rule can take place in all relevant locations of the import program. In this example, the domain rule is attached directly to field *LIFNR* of target table *LFA1:*

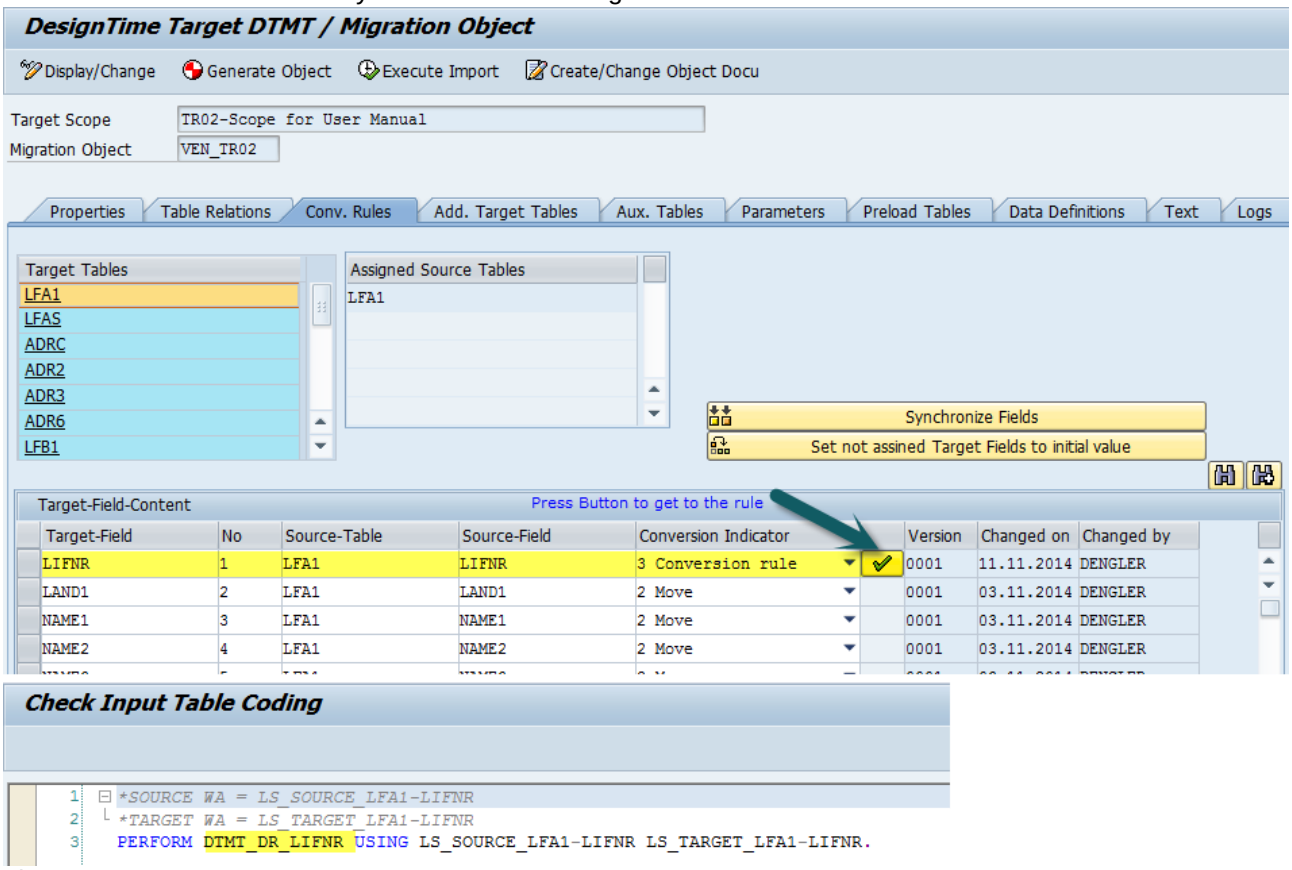
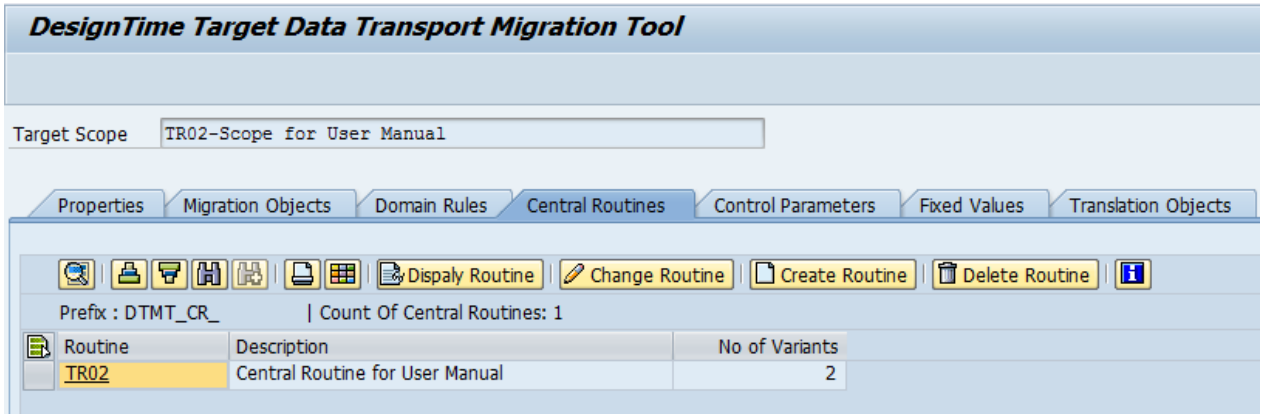


Figure 3.18

4.1.4 Create Central Routines

Central routines are similar to domain rules, but are not connected to a specific domain. They are used to implement coding that can be reused. If more than one variant exists for a central routine, you can activate the variant you want in the customizing step. If you do not activate a variant, the standard variant will be used.



4.1.4.1 Central Routines Screen

Same as section 3.1.3.1

4.1.4.2 Central Routine Call

The call of central routines can take place in all relevant locations of the import program. In this example, the central routine is attached directly to field *NAME1* of target table *LFA1:*

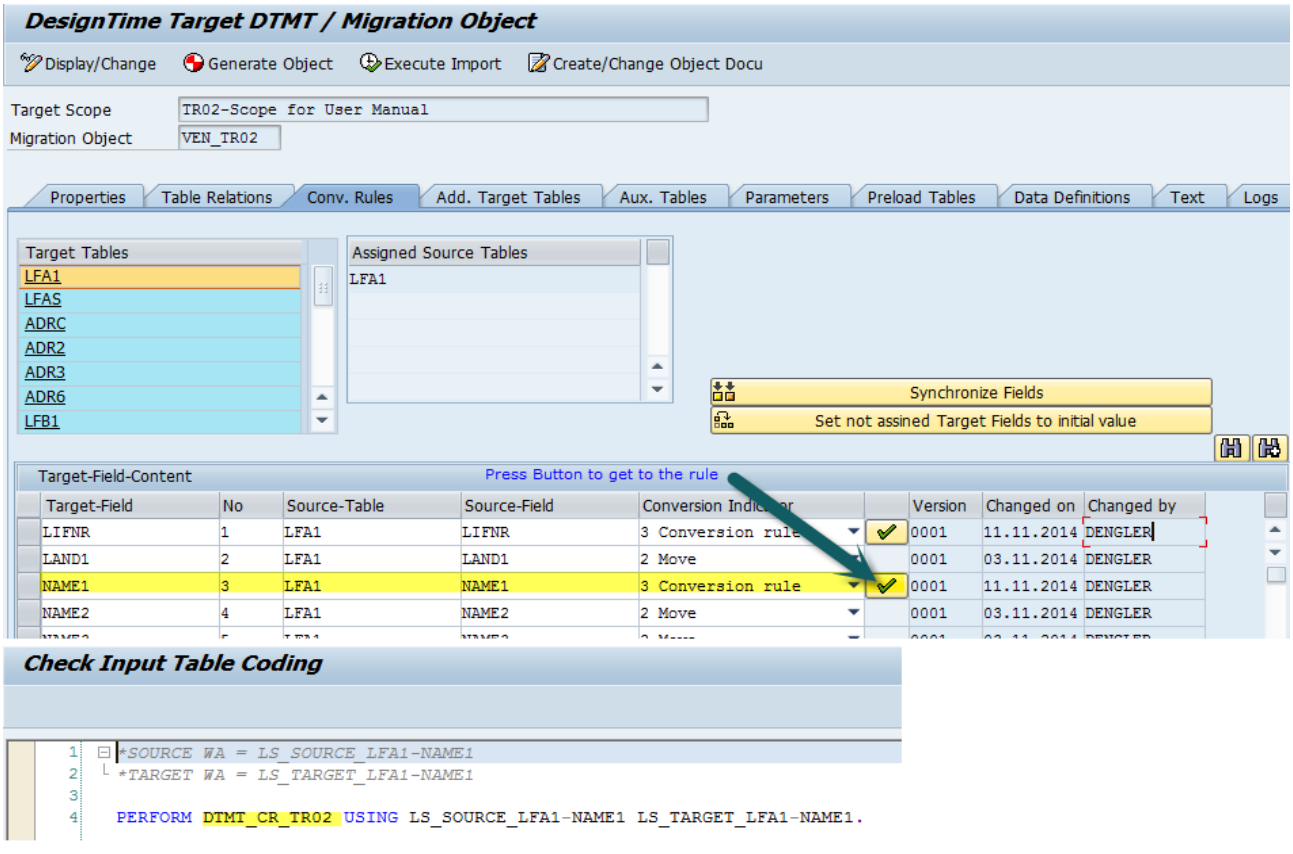


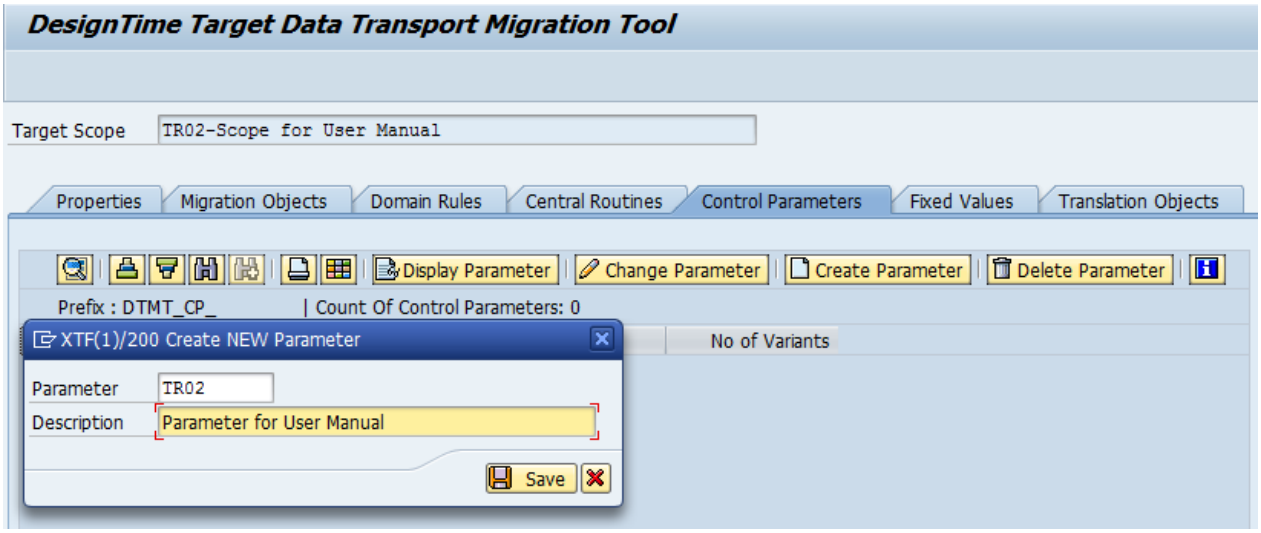
Figure 3.19

4.1.5 Create Control Parameters

Sometimes a customer decision is required to control the standard migration content. This is done by using *Control Parameters* via migration customizing.

Note: Within the AFS2FMS migration, customers need to, for example, decide how long the variant counter (the counter that is appended to the old material number to build the new variant article number) should be. You have the choice between 3 and 4 bytes, which you can determine via the respective control parameter in migration customizing.

Clicking the *Control Parameters* tab on the scope level screen displays a list of all control parameters of a specific scope. In this example, the list is empty because a new source scope has just been created. After pressing *Create Parameter*, you will be asked to enter a parameter name and a description. In this example the migration parameter is called **TR02**.



Control parameters are similar to central rules. If more than one variant exists for a control parameter, you can activate the variant you want in the customizing step. If you do not activate a variant, the standard variant will be used.

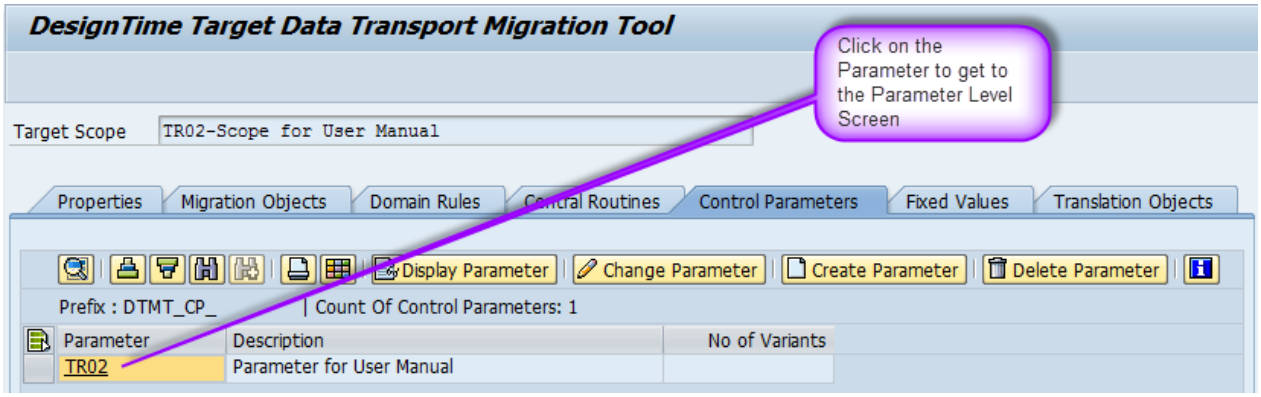
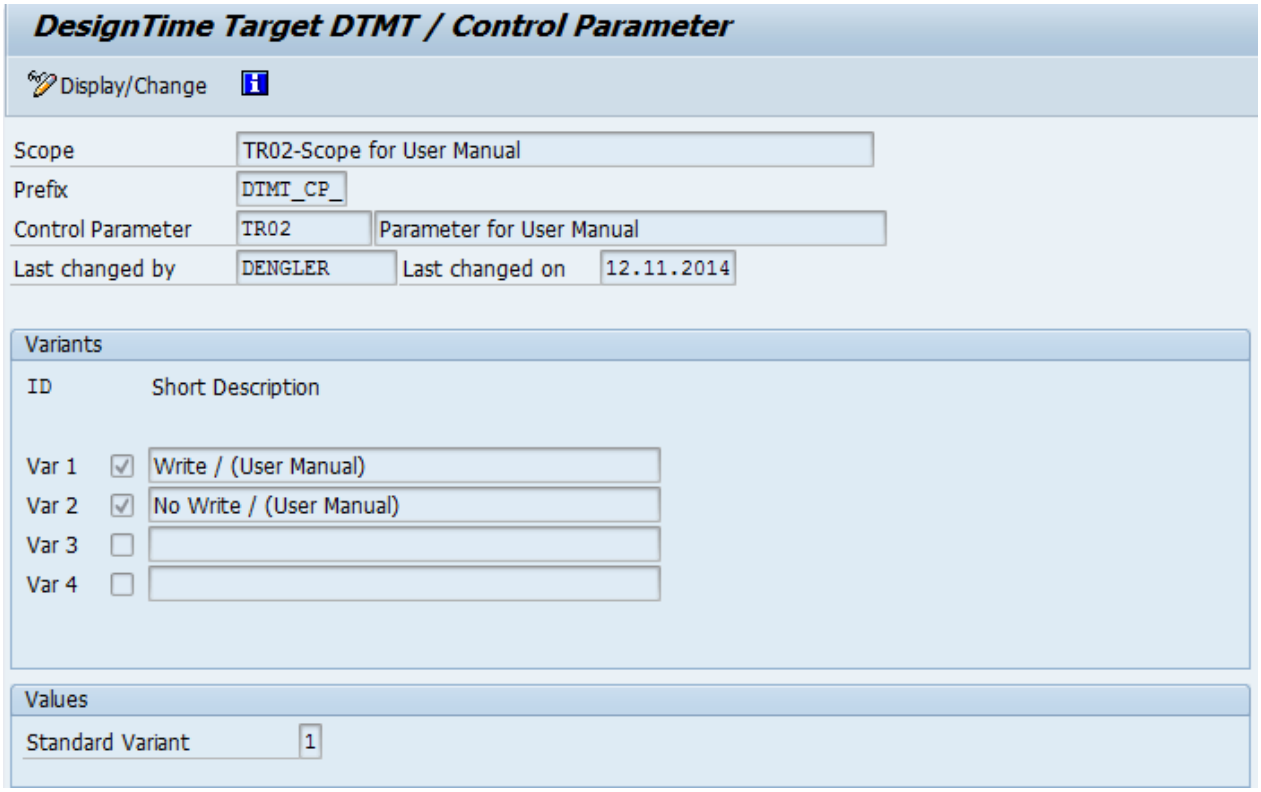


Figure 3.20

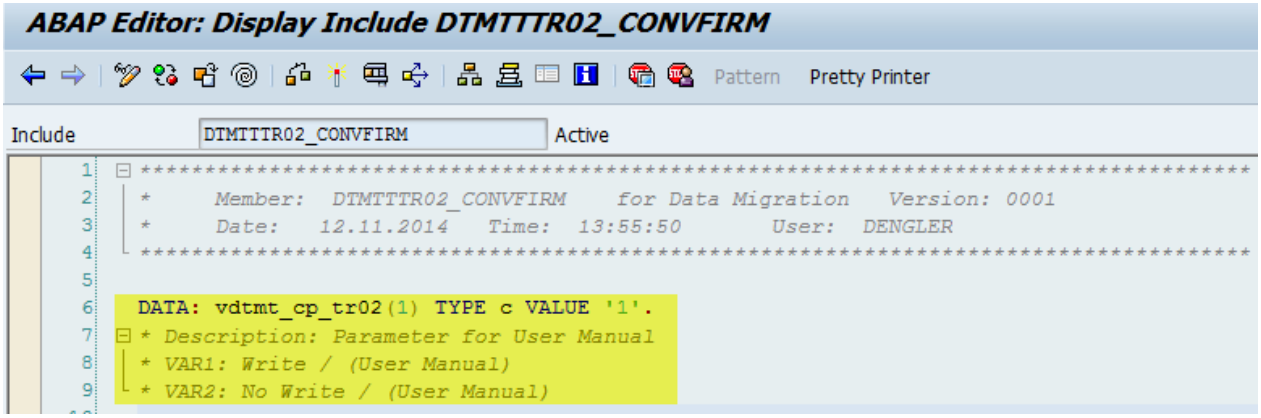
4.1.5.1 Control Parameter Screen

Up to four variants can be defined for a control parameter. You use migration customizing to determine which variant is actually used in the import. The tool makes the variant that is set in Customizing available, or if one is not set, the standard variant is made available at runtime so that you can react appropriately to it in the coding.

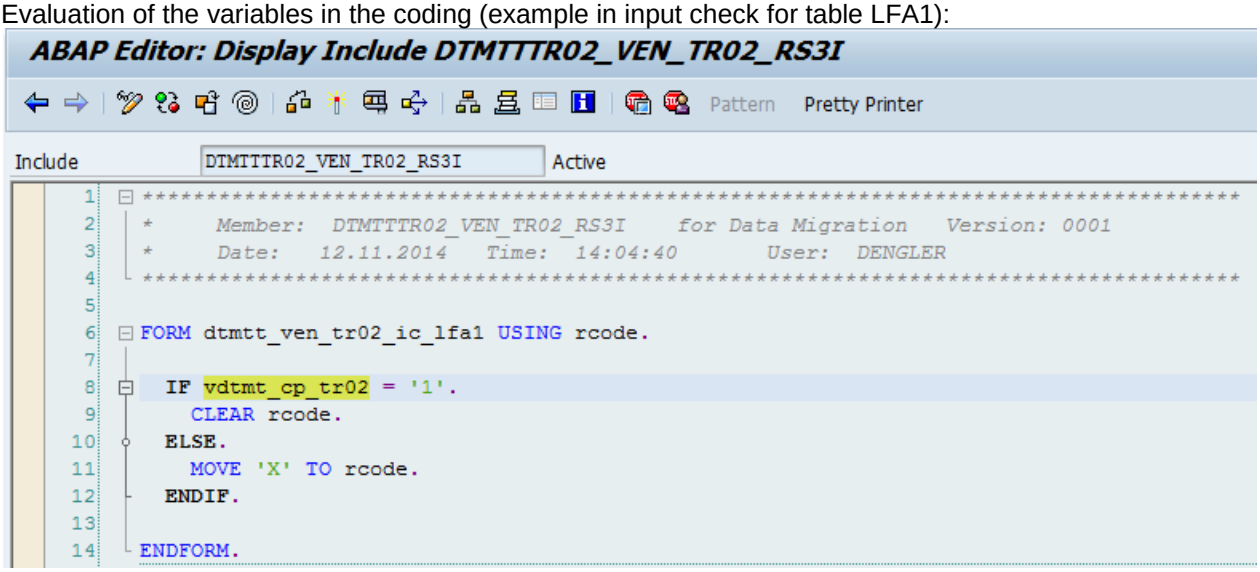


The query in the coding takes place via the variable *VDTMT_CP_NAME* (name = TR02 in this example). Since the customizing has not yet taken place, the variable is automatically evaluated with the standard variable.

Generated variable definition:



Evaluation of the variables in the coding (example in input check for table LFA1):



4.1.6 Create Fixed Values

Fixed values are centrally-defined variables whose values can be maintained from outside, via the migration customizing.
If you click the *Fixed Values* tab on the scope level screen, a list of all fixed values of a specific scope will be displayed. In this example, the list is empty because a new source scope has just been created.
After pressing *Create Fixed Value*, you will be asked to enter a value name and a description. In this example, the migration fixed value is called **TR02**.

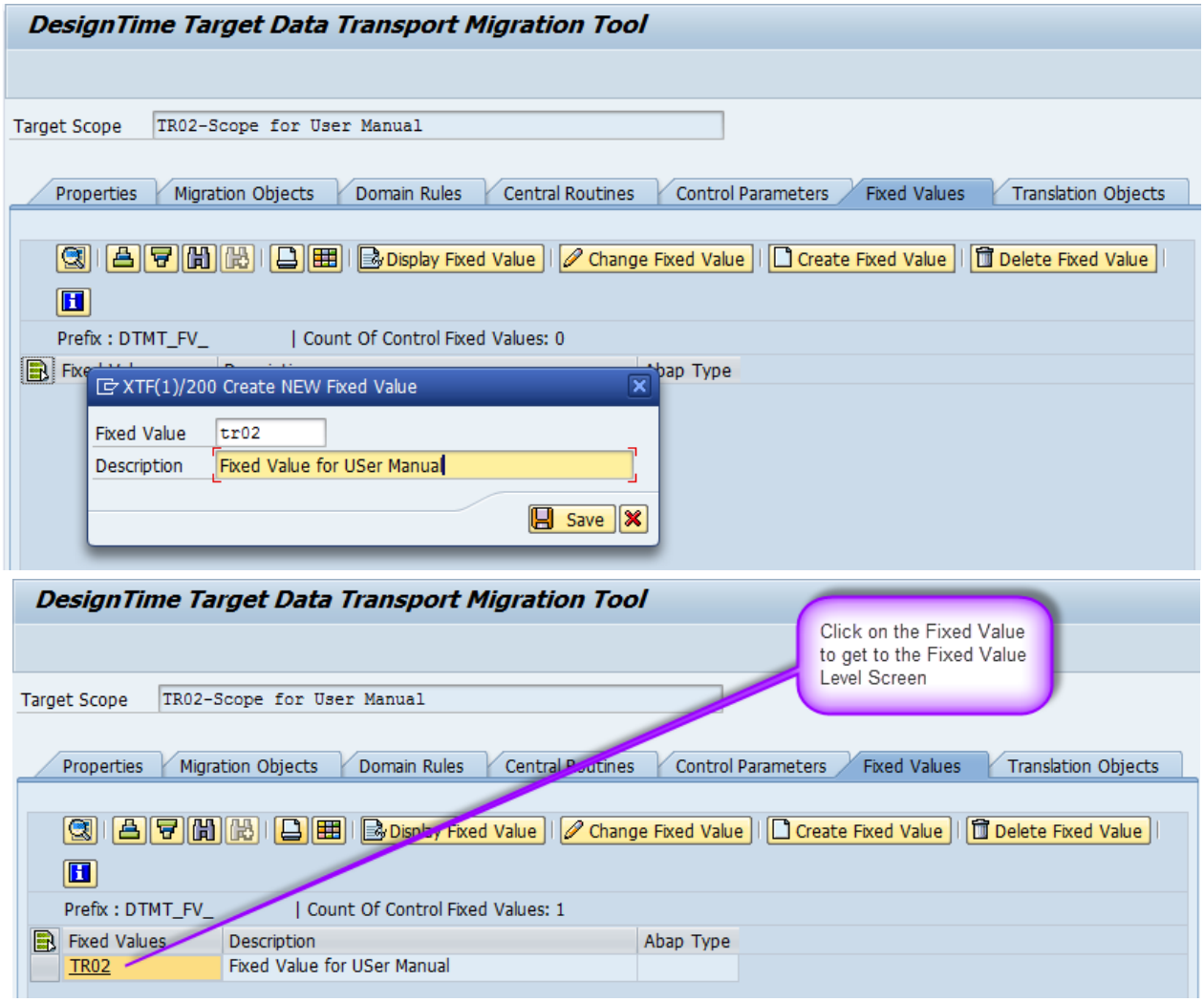


Figure 3.21

4.1.6.1 Fixed Value Screen

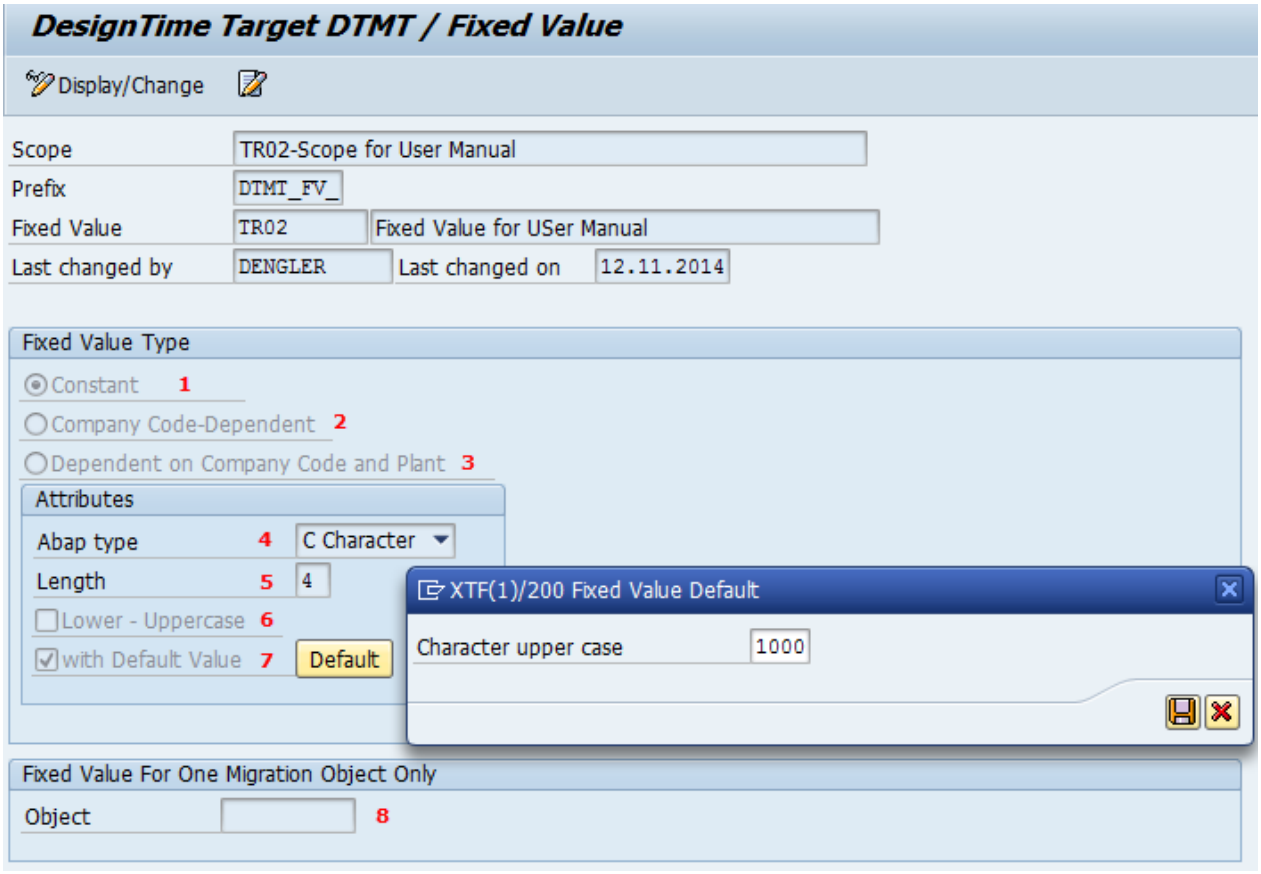


Figure 3.22

1. The fixed value is defined as a constant
2. The fixed value is company-code-dependent
3. The fixed value is dependent on the company code and plant
4. Typing of the fixed value
   - C Character
   - D Date
   - N Numeric
   - T Time
5. Length of the fixed value
6. Mixed case is allowed
7. Definition of a default value (possible only if defined as a constant)
8. Fixed value allowed only for specified object

4.1.7 Create Translation Objects

Converting data via translation tables is state of the art migration technology. Whenever there is a translation variant activated via migration customizing, it is required to maintain this translation table via Customizing. There is functionality embedded into DTMT that makes the customizing of translation objects more convenient.

Note: A translation object can be active by default.

Note: If you activate a translation variant in the migration customizing of domain rules or central rules, **all** potential field values are to be maintained in the respective translation object.

The processing of the translation object during the import takes place either via a domain rule or a central routine. A translation object always has a direct relation to a domain rule or central routine, which must contain a translation variant.

Note: The name of the translation object must correspond to that of the domain rule or central routine.

In this example, the related translation object is created for the existing domain rule *LIFNR.*

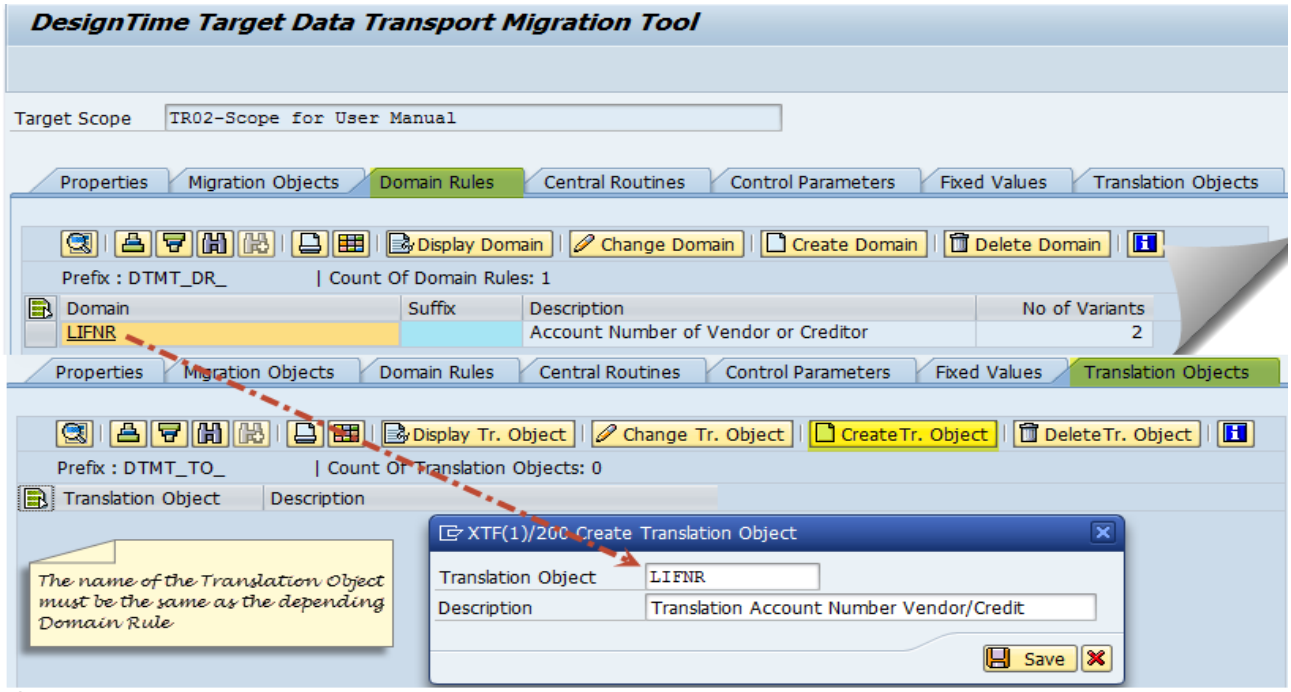


Figure 3.23

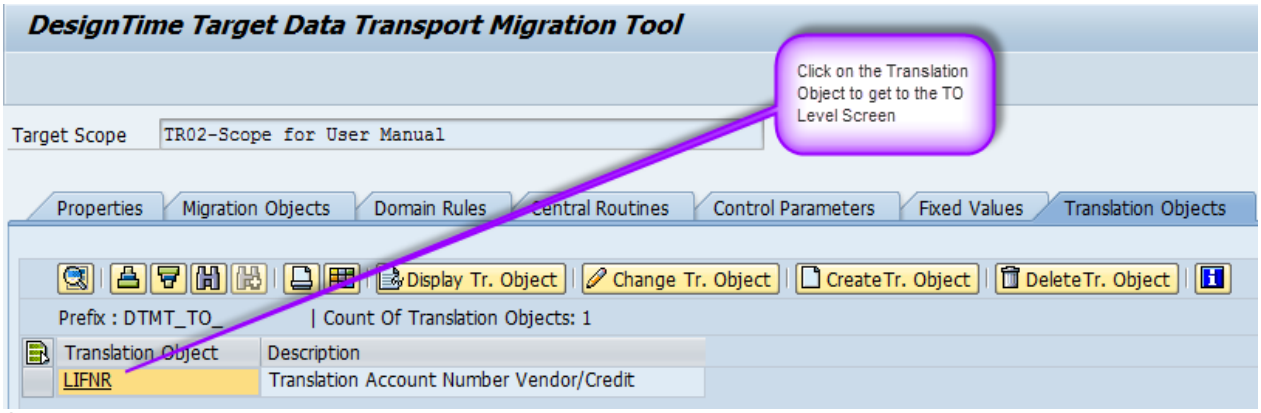


Figure 3.24

4.1.7.1 Translation Objects Screen

The screen *Translation Object* is separated into three areas:

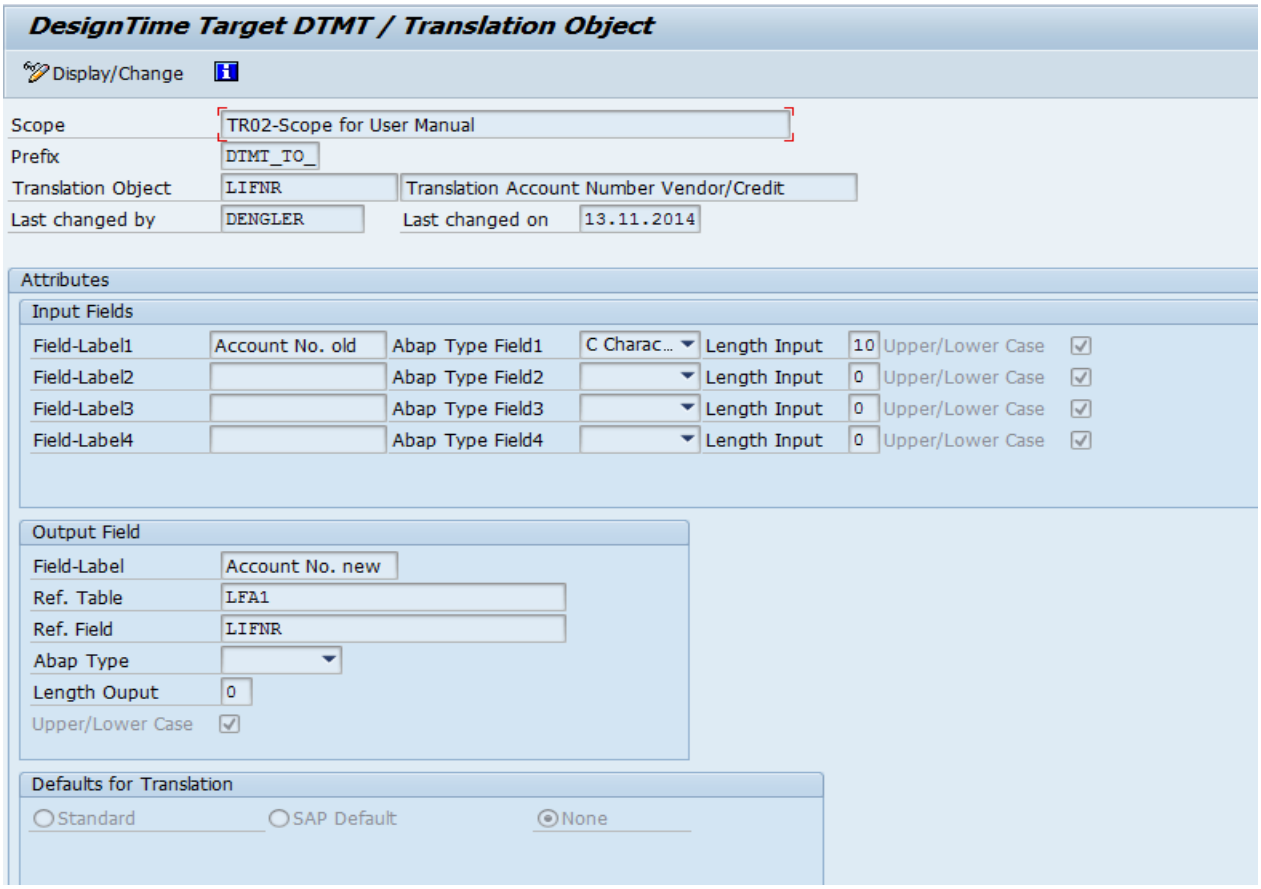
1. Input fields:
   Describes the source value that should be translated. This source value is saved in the field *VARG* of table *DTMT_FW_T_UMFR,* and has a maximum length of 50 bytes. These 50 bytes can be distributed to four labels, which are correspondingly displayed when the translation object is displayed or changed. An input field is assigned for each label. These input fields must be correspondingly linked in the coding of the domain rule or the central routine when more than one label is used.

   Note:
   If more than one label is used, the coding of the corresponding domain rule or central routine must be enhanced.

2. Output field
   Describes the target value into which the translation occurs. This target value is saved in field *FUNC* of table *DTMT_FW_T_UMFR*, and has a maximum value of 50 bytes. You can refer to a reference field of a reference table, for the display of the translation object.

3. Defaults for translation (SAP-internal)
   This differentiation describes whether SAP defaults, generally valid proposals, or no proposals at all are delivered. This is relevant for the anonymization using after-import methods, but irrelevant for delivery to the customer and the import process.

Definition of the translation object



4.1.7.2 *Customizing of the translation object*

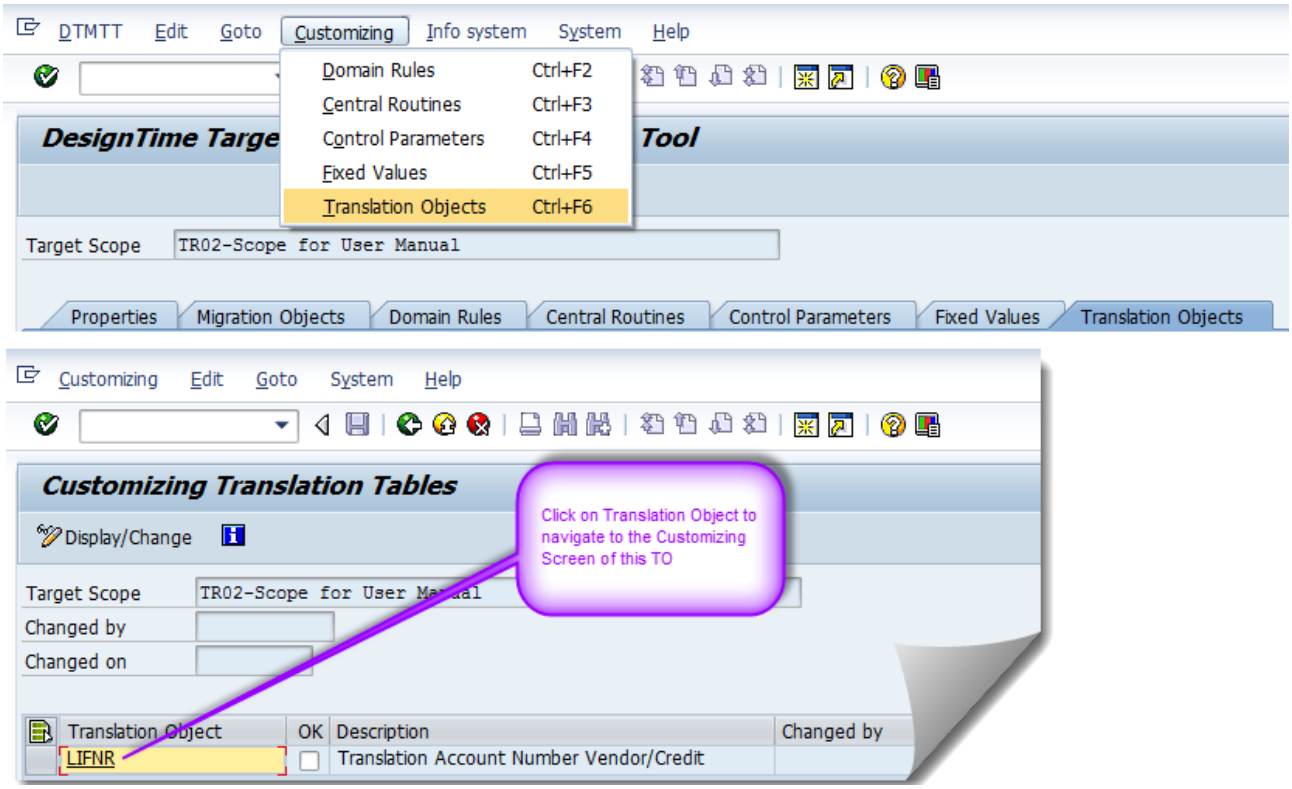
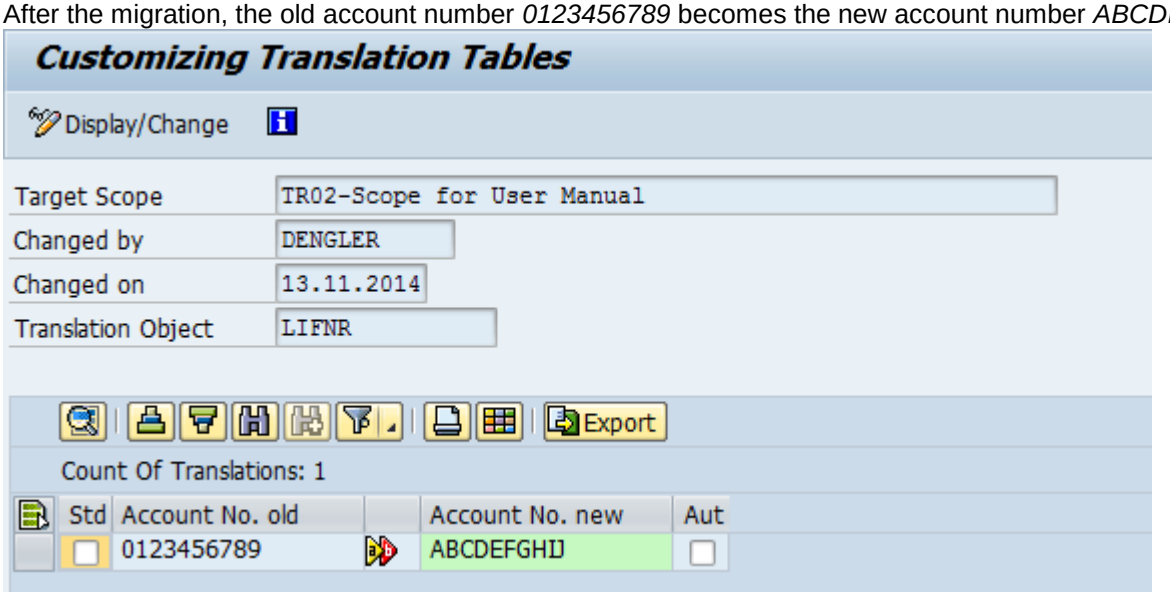


Figure 3.25

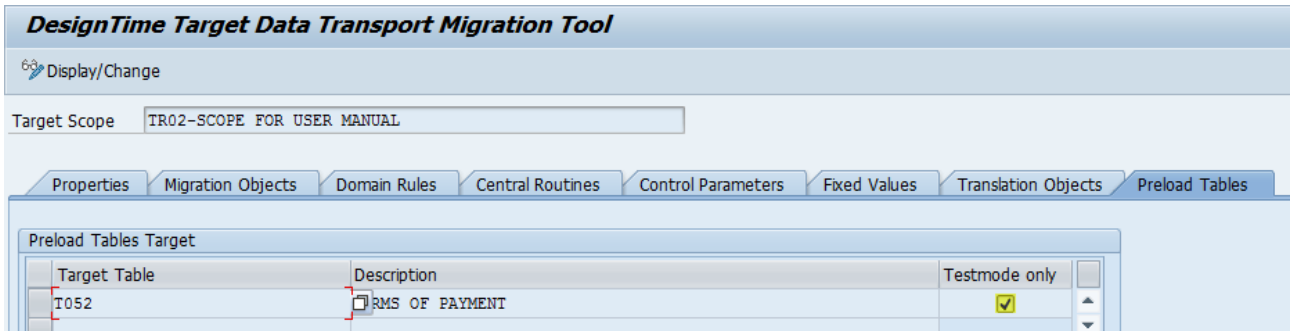
After the migration, the old account number *0123456789* becomes the new account number *ABCDEFGHIJ.*



4.1.8 Scope Level "Preload Tables"

Preload tables are customizing tables that are read during the processing of the data import, and loaded into internal tables (for performance reasons), to avoid too many instances of read access to the database. These internal tables are filled directly after the start of the data import.
They are accessible for each object of a scope.

LS_PRELOAD_SCOPE_Table = Name of work area of the internal tables
LT_PRELOAD_SCOPE_Table = Name of the internal table



*If **"Testmode only"** is selected, the preload function is just called in mode "Test-Run".*

```
159   *Internal Preloaded-Customizing-Tables on Scope Level
160     DEFINE_PRELOAD_TABLE_SCOPE T052.
161     IF MG_TESTL = 'X'.
162       SELECT * FROM T052 INTO TABLE LT_PRELOAD_SCOPE_T052
163                                 ORDER BY PRIMARY KEY.
164     ENDIF.
```

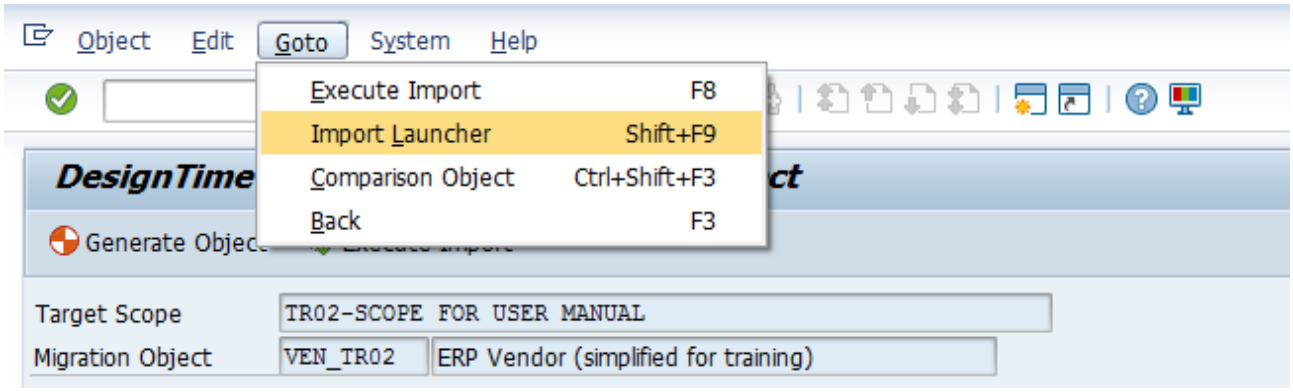
5 SPECIAL FEATURES ON TARGET SIDE

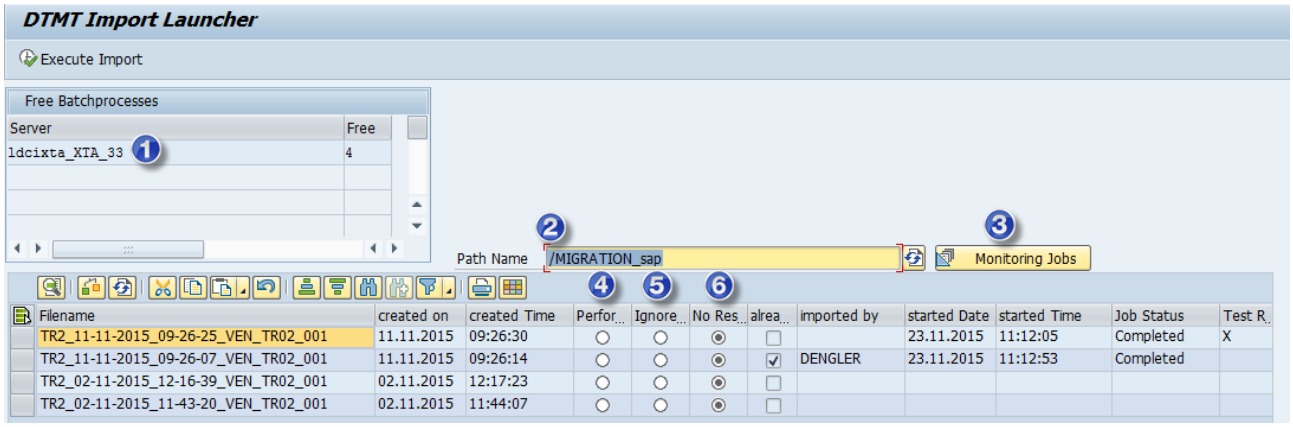
5.1 Object

5.1.1 DTMT Import Launcher

The import launcher starts the import of the files in the background, regardless of the available servers and free batch processes for the selected files. The load distribution to the servers and batch processes is controlled automatically. If more files are imported than can be handled by the batch processes, they are put into wait mode, and start only when a batch process is free.

The monitoring of the processes can be carried out by using SM37 (direct access to screen: Monitoring Jobs), or within DTMT by using the log overview. The successful import of a file can be monitored by using the log overview.



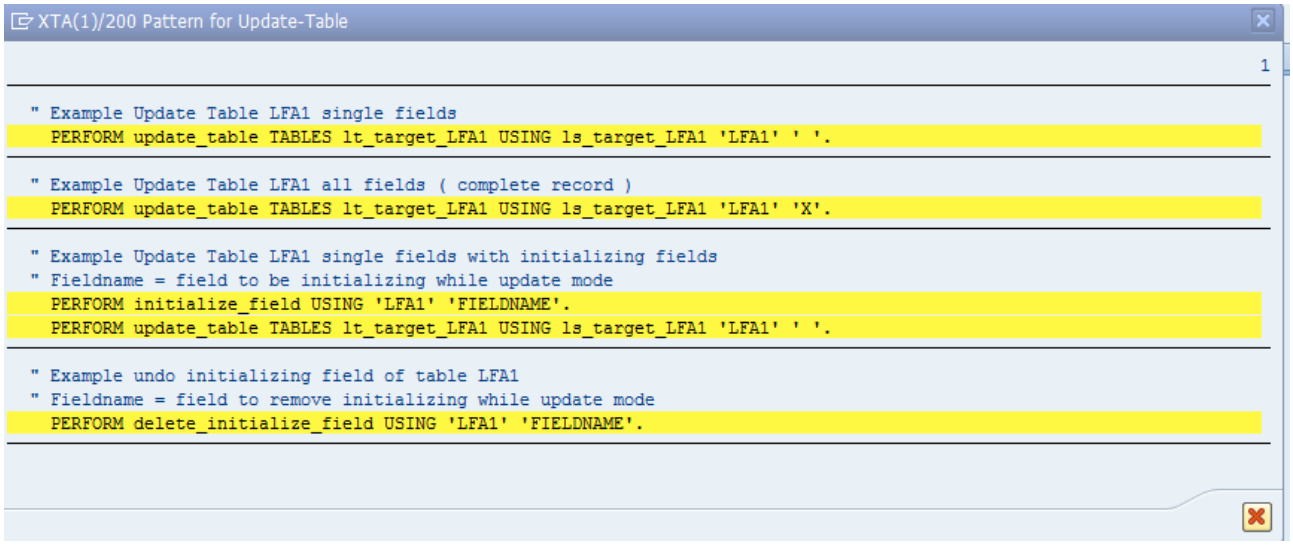


Free batch processes, distributed to the available servers
Path name of the displayed files
SM37 monitoring of the batch processes
Perform Restart : The import of this file was canceled. The file was not completely processed. The system set a restart point. The file can now be further processed from the restart point, if you select this radio button.
Ignore-Restart : If a restart point was set, it can be ignored if you select this radio button. The processing of the file to be imported then starts again.
No Restart : There is no restart point.

5.1.2 Update Mode

The import can also take place in update mode, that is, the tables are enriched, or the existing field content is overwritten. To do this, replace the APPEND in ENDLOOP with additional coding. You can display this additional coding by choosing button *Example Update Table* on tab *Table Relations* as an example, and manually copy it into the ENDLOOP of the respective table.

Here is an example from table LFA1:



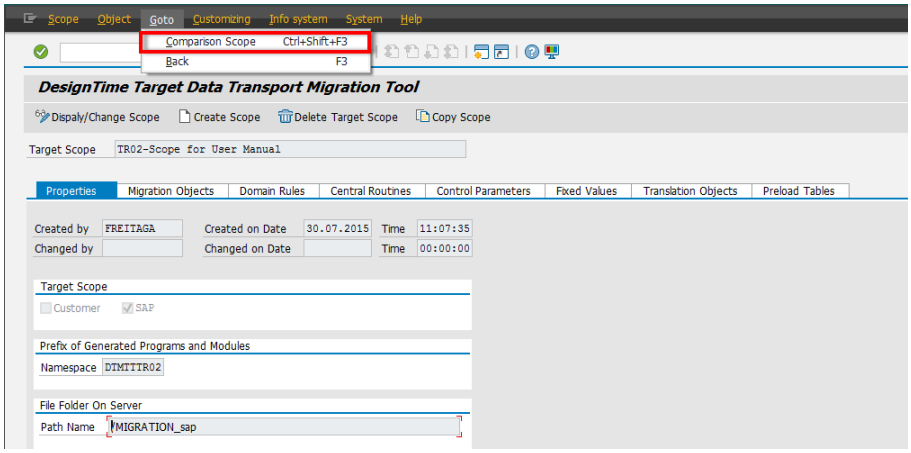
5.2 Code Comparison

SAP delivers standard content (scope: V600) that you can copy to the customer namespace as a template for your migration. Corrections and enhancements in the standard scope are delivered with each support package. There is now a function that you can use to recognize what has changed in the standard scope, in comparison with your custom scope: the coding comparison.

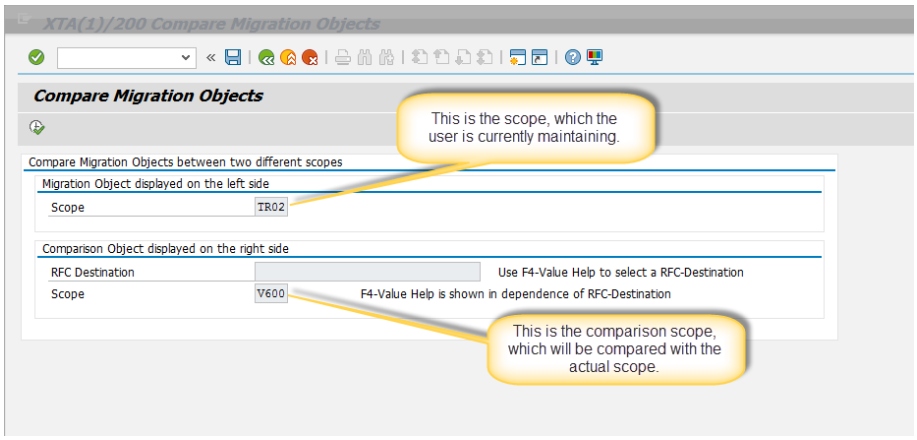
You can carry out the coding comparison in two different ways; on scope level and on object level.

5.2.1 Code Comparison on Scope Level

Call transaction DTMT_DTT. Choose *Goto -> Comparison Scope*.

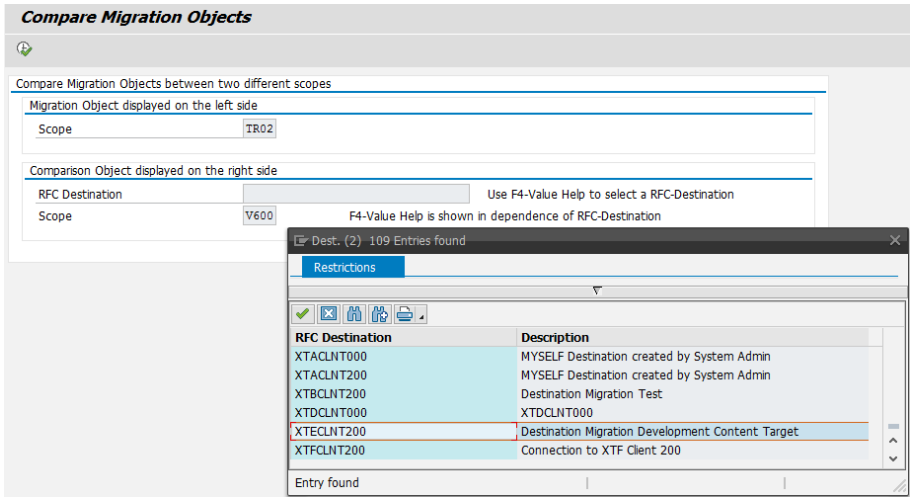


The scope that you are currently in as a user (in this example, TR02) is shown in the upper part of the screen, and the scope delivered by SAP (V600) is shown as the comparison scope in the lower part.

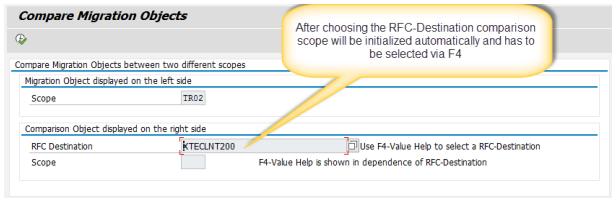


You can use the F4 help to change both the current scope and the comparison scope. You can also compare scopes from different systems.

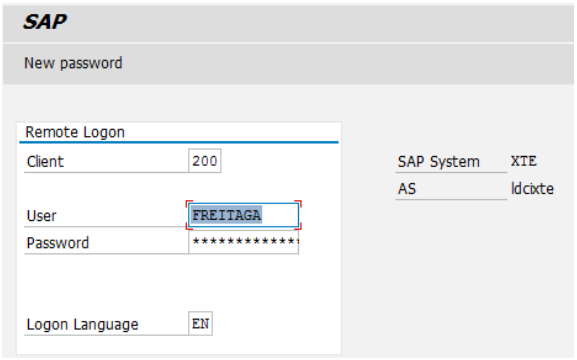
To do this, choose the corresponding RFC connection from the F4 help.



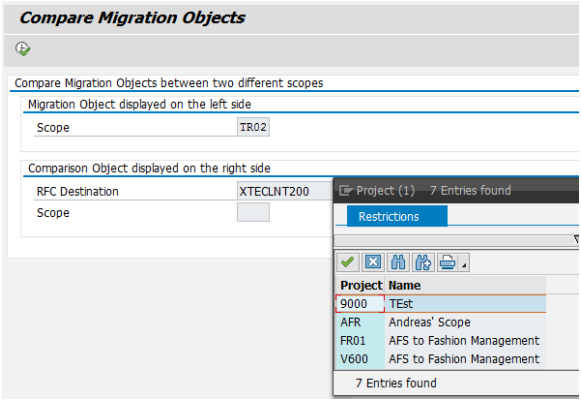
After you have selected the RFC connection and thereby the system that should be used for the comparison, the associated scope is automatically initialized. You then need to choose it again by using the F4 help. This is necessary because the preset scope for the RFC connection may no longer be valid.



When you try to choose the scope, a logon screen appears. Log on.

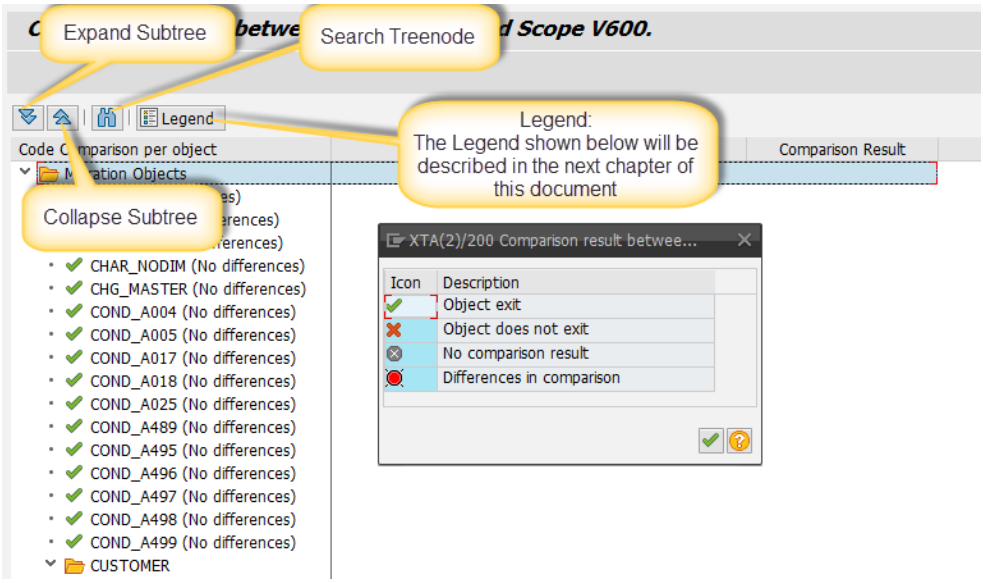


After the logon, you can choose the comparison scope by using F4.



You do not have to choose the original scope (V600). It is also possible to compare two custom scopes.

Choose *Execute* .
The system now carries out a coding comparison on scope level. Objects with the same names are always compared with each other.
The result of the comparison is displayed for each object, in the analysis monitor. Each object has its own subnode in the tree structure. Objects that are flagged with ✔ contain no differences, unlike the objects that are shown with Section 3.2.2 contains a detailed description of the differences.



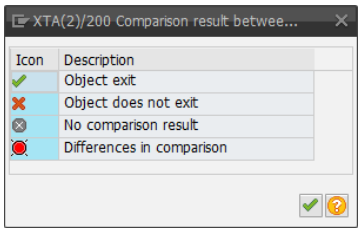
Function description for the individual buttons

: Opens all subnodes of the selected node

: Closes all subnodes of the selected node

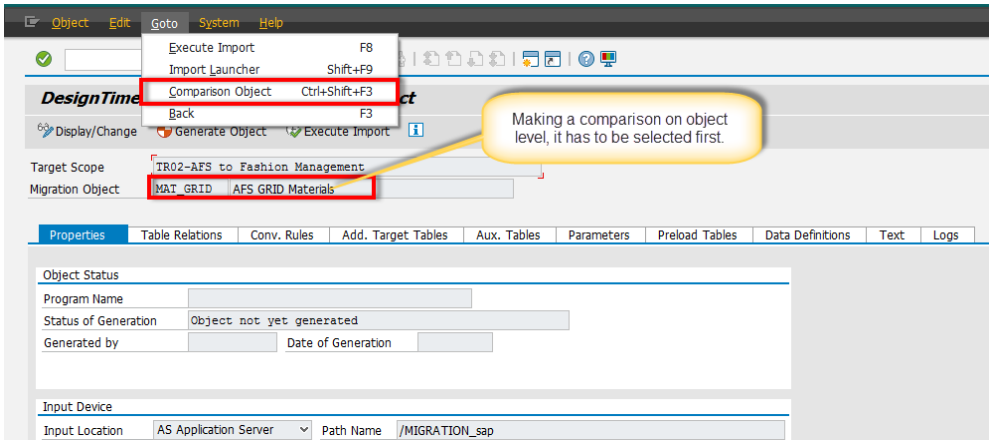
: Searches for a node (for example, CUSTOMER). The cursor is then automatically positioned on the search string that is found.

Legend: Displays the legend for the icons that are used, with an explanation. Section 3.2.2 contains more information about this.

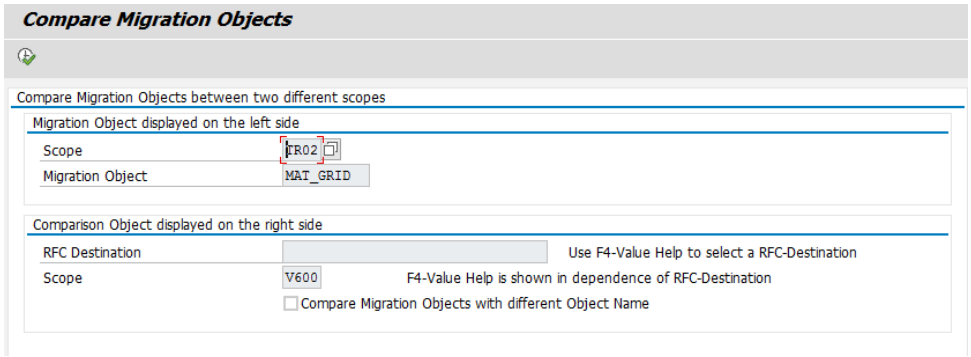


5.2.2 Code Comparison on Object Level

To carry out an analysis on object level, you need to have selected a migration object in the migration tool. You can then call the analysis. Call transaction DTMT_DTS. Choose *Goto -> Comparison Object*.

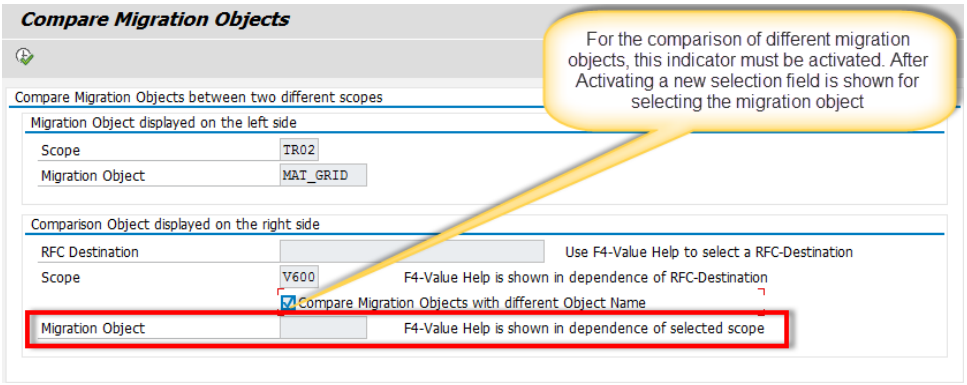


The scope that you are currently in as a user (in this example, TR02) is shown in the upper part of the screen, and the scope delivered by SAP (V600) is shown as the comparison scope in the lower part.



Here, it is also possible to choose a comparison scope other than V600. You can use the F4 help to choose any other scope, even from another system (see also section 3.2.2.2).

To choose a different migration object for comparison, select the checkbox *Compare Migration Object with Different Object Name*. This may be necessary if, for example, the customer used an object name other than the proposed one when copying the object.



The following sections describe the differences in more detail.

5.2.2.1 Migration Objects ith the Same Name

See section 3.2.2.1

5.2.2.2 Migration Objects with Different Names

See section 3.2.2.2

5.2.2.3 Object-independent Functions

In contrast to the source scope, there are functions in the target scope that are object-independent:

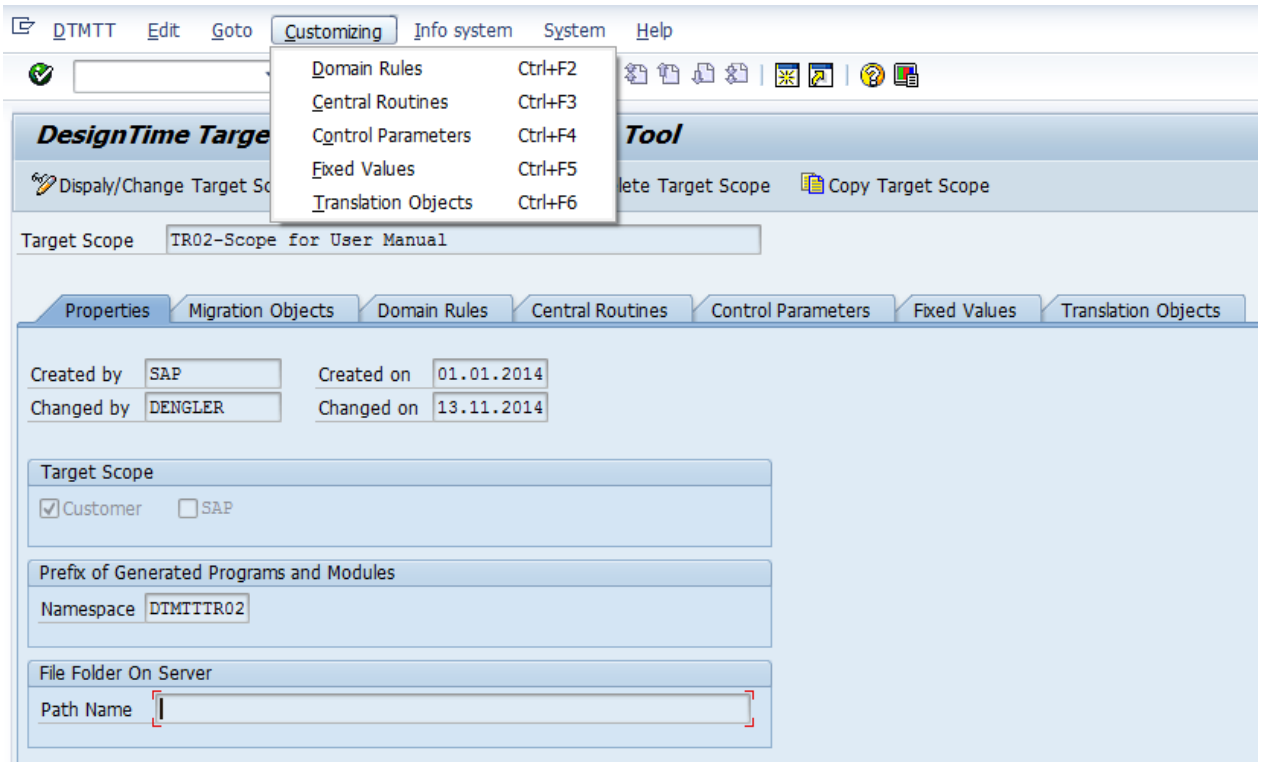
1. Domain Rules (see 4.1.3)
2. Central Routines (see 4.1.4)
3. Control Parameters (see 4.1.5)
4. Fixed Values (see 4.1.6)
5. Translation Objects (see 4.1.7)
6. Preload Tables

Almost all points listed above are executed under the node *Scope Lev.* in the analysis. The preload tables are an exception, since they can be either object-dependent or object-independent. The analysis recognizes this, and collects the information on either an object-dependent or object-independent level.

6 MIGRATION CUSTOMIZING

The migration content delivered by SAP can be customized to a certain degree. This allows customers to influence the way data is migrated if SAP foresees multiple options.

Customizing is carried out in the target system on scope level, and is therefore valid across all migration objects within a scope. It has to be done before the first data import is performed.
It is split into the following sections:



6.1 Domain Rules

Domain rules are central conversion rules that can be used in all MOs of a scope. They are always connected to a domain, and are used to convert fields assigned to the same domain, to ensure a consistent conversion across MOs within a scope. Some of the domain rules consist of two or more variants reflecting different types of conversion rules (for example **Move;Translate**). In this case you will see them in the *Migration Customizing/Domain Rules* section and you can activate the variant you want. If you do not activate a variant, the standard variant will be used.

Note: Domain rules are provided for most of the domains referring to organizational entities like company codes, plants, and so on. They are also provided for other central fields like customer, vendor, or material numbers because the values of those fields are often the subject of value changes. If you need to consistently change the values of fields where a domain rule exists, you can do this easily. However, it is your responsibility to make sure that the changes make sense. Renumbering of plants or company codes, for example, becomes very simple. Should the variants of a domain rule designated in the system not be sufficient for your environment, new variants can be created or existing variants can be changed using the DTMT design time. However, domain rule changes should be undertaken only by experienced migration experts with sufficient knowledge of the respective application area.

Note: If you activate a translation variant via migration customizing, you must also maintain the translation object for this.

6.2 Central Routines

Central routines are similar to domain rules but are not connected to a specific domain. They are used to implement coding that can be reused. If more than one variant exists for a central routine, you can activate the variant you want.

6.3 Control Parameters

Sometimes a customer decision is required to control the standard migration content. This is done by using control parameters via migration customizing.

Note: Within the AFS2FMS migration, for example, customers need to decide how long the variant counter (the counter that is appended to the old material number to build the new variant article number) should be. You have the choice between 3 and 4 bytes, which you can determine via the respective control parameter in migration customizing.

6.4 Fixed Values

Fixed values are centrally-defined variables whose values can be maintained from outside, via migration customizing.

6.5 Translation Objects

Converting data via translation tables is state of the art migration technology. Whenever there is a translation variant activated via migration customizing, it is necessary to maintain this translation table via Customizing. There is functionality embedded into DTMT that makes the customizing of translation objects more convenient. Refer to the DTMT user manual for more details.

Note: A translation object can be active by default.

Note: If you activate a translation variant in the migration customizing of domain rules or central rules, **all** potential field values are to be maintained in the respective translation object.