SAP BW ABAP Training Lab Bo

Author: Sanjay Chaurasia/Sonia Ghosh

Program Duration: 5 days

Table of Contents

Getting Started

This lab book is a guided tour for learning Business Objects XI Designer. It comprises solved examples and 'To Do' assignments. Follow the steps provided in the solved examples and work out the given 'To Do' assignments

ABAP Fundamentals/Overview

Types of Data and constants

Data Types

In this example, we have a character string of type C with a predefined length 40. STRING is a data type that can be used for any character string of variable length (text strings). Type STRING data objects should generally be used for character-like content where fixed length is not important.

```
REPORT YR_SEP_12.
DATA text_line TYPE C LENGTH 40.
text_line = 'A Chapter on Data Types'.
Write text_line.
DATA text_string TYPE STRING.
text_string = 'A Program in ABAP'.
Write / text_string.
DATA d_date TYPE D.
d_date = SY-DATUM.
Write / d_date.
```

Constants

Constants are used to store a value under a name. We must specify the value when we declare a constant and the value can't be changed later in the program.

Use **CONSTANTS** keyword to declare a constant.

```
REPORT YR_SEP_12.
CONSTANTS PQR TYPE P DECIMALS 4 VALUE '1.2356'.
Write: / 'The value of PQR is:', PQR.
```

Variables & Its Categories

ABAP Variables are instances of data types. Variables are created during program execution and destroyed after program execution.

Use keyword **DATA** to declare a variable.

- You can find the complete list of system variables in the SYST table in SAP.
- Individual fields of the SYST structure can be accessed by using either “SYST-” or “SY-”.

```
REPORT Z_Test123_01.
WRITE:/'SY-ABCDE', SY-ABCDE,
/'SY-DATUM', SY-DATUM,
/'SY-DBSYS', SY-DBSYS,
/'SY-HOST ', SY-HOST,
/'SY-LANGU', SY-LANGU,
/'SY-MANDT', SY-MANDT,
/'SY-OPSYS', SY-OPSYS,
/'SY-SAPRL', SY-SAPRL,
/'SY-SYSID', SY-SYSID,
/'SY-TCODE', SY-TCODE,
/'SY-UNAME', SY-UNAME,
/'SY-UZEIT', SY-UZEIT.
```

Strings

Strings, which are widely used in ABAP programming, are a sequence of characters.

We use data type C variables for holding alphanumeric characters, with a minimum of 1 character and a maximum of 65,535 characters. By default, these are aligned to the left.

The following declaration and initialization creates a string consisting of the word 'Hello'. The size of the string is exactly the number of Characters in the word 'Hello‘.

Following program is an example of creating strings.

```
REPORT YT_SEP_15.
 DATA: title_1(10) VALUE 'Tutorials',
         title_2(10) VALUE 'Point',
         spaced_title(30) VALUE 'Tutorials Point Limited',
         sep,
         dest1(30),
         dest2(30).
 CONCATENATE title_1 title_2 INTO dest1.
 Write: / 'Concatenation:', dest1.
 CONCATENATE title_1 title_2 INTO dest2 SEPARATED BY sep.
 Write: / 'Concatenation with Space:', dest2.
 CONDENSE spaced_title.
 Write: / 'Condense with Gaps:', spaced_title.
 CONDENSE spaced_title NO-GAPS.
 Write: / 'Condense with No Gaps:', spaced_title.
```

Data Type & Variable Declaration

Assignment Statement

Operator

All ABAP operators are classified into four categories –

- Arithmetic Operators
- Comparison Operators
- Bitwise Operators
- Character String Operators

Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra.

```
    REPORT YS_SEP_08.
DATA: A TYPE I VALUE 150,
      B TYPE I VALUE 50,
      Result TYPE I.
Result = A / B.
WRITE / Result.
```

Comparison Operators

```
REPORT YS_SEP_08.
DATA: A TYPE I VALUE 115,
B TYPE I VALUE 119.
IF A LT B.
WRITE: / 'A is less than B'.
ENDIF
```

Character String Operators

```
REPORT YS_SEP_08.
DATA: P(10) TYPE C VALUE 'APPLE',
      Q(10) TYPE C VALUE 'CHAIR'.
IF P CA Q.
    WRITE: / 'P contains at least one character of Q'.
ENDIF.
```

Data Type Properties

Control Statements

Control Statements or Decision making structures have one or more conditions to be evaluated or tested by the program along with a statement or statements that are to be executed, if the condition is determined to be true, and optionally other statements to be executed, if the condition is determined to be false

If Statement

‘IF’ is a control statement used to specify one or more conditions.

If the expression evaluates to true, then the IF block of code will be executed.

```
Report YH_SEP_15.
Data Title_1(20) TYPE C.
Title_1 = 'Tutorials'.
 IF Title_1 = 'Tutorials'.
    write 'This is IF statement'.
ENDIF.
```

If .. Else Statement

In case of IF….ELSE statements, if the expression evaluates to true then the IF block of code will be executed. Otherwise, ‘ELSE’ block of code will be executed.

```
Report YH_SEP_15.
Data Title_1(20) TYPE C.
Title_1 = 'Tutorials'.
IF Title_1 = 'Tutorial'.
write 'This is IF Statement'.
ELSE.
write 'This is ELSE Statement'.
ENDIF.
```

Nested If Statement

It is always legal to nest IF....ELSE statements, which means you can use one IF or ELSEIF statement inside another IFor ELSEIF statement.

```
Report YH_SEP_15.
Data: Title_1(10) TYPE C,
Title_2(15) TYPE C,
Title_3(10) TYPE C.
Title_1 = 'ABAP'.
Title_2 = 'Programming'.
Title_3 = 'Tutorial'.
```

```
IF Title_1 = 'ABAP'.
IF Title_2 = 'Programming'.
IF Title_3 = 'Tutorial'.
Write 'Yes, It’s Correct'.
ELSE.
Write 'Sorry, It’s Wrong'.
ENDIF.
ENDIF.
ENDIF.
```

Case Control Statement

The CASE control statement is used when you need to compare two or more fields.

```
Report YH_SEP_15.
Data: Title_1(10) TYPE C,
Title_2(15) TYPE C.
Title_1 = 'ABAP'.
Title_2 = 'Programming'.
CASE Title_2.
WHEN 'ABAP'.
Write 'This is not the title'.
WHEN 'Tutorials'.
Write 'This is not the title'.
WHEN 'Limited'.
Write 'This is not the title'.
WHEN 'Programming'.
Write 'Yes, this is the title'.
WHEN OTHERS.
Write 'Sorry, Mismatch'.
ENDCASE.
```

Loop Control

A loop statement allows us to execute a statement or group of statements multiple times.

ABAP programming language provides the following types of loop to handle looping requirements.

Do Loops

```
Report YH_SEP_15.
Do 15 TIMES.
Write: / 'Hello'.
ENDDO.
```

While loop

.

```
REPORT YS_SEP_15
 DATA: a type i.
 a = 0.
 WHILE a <> 8.
 Write: / 'This is the line:', a.
 a = a + 1.
 ENDWHILE
```

Nested Loop

```
REPORT YS_SEP_15.
Data: a1 type I,
b1 type I.
a1 = 0.
b1 = 0.
Do 2 times.
a1 = a1 + 1.
Write: /'Outer', a1.
Do 10 times.
b1 = b1 + 1.
```

```
Write: /'Inner', b1.
ENDDo.
ENDDo.
```

Date Operations

ABAP implicitly references the Gregorian calendar, valid across most of the world. We can convert the output to country specific calendars. A date is a time specified to a precise day, week or month with respect to a calendar. A time is specified to a precise second or minute with respect to a day. ABAP always saves time in 24-hour format. The output can have a country specific format.

Dates and time are usually interpreted as local dates that are valid in the current time zone.

ABAP provides two built-in types to work with dates and time −

D data type

T data type

Following is the basic format

```
REPORT YR_SEP_15.
DATA: date_1 TYPE D.
date_1 = SY-DATUM.
Write: / 'Present Date is:', date_1 DD/MM/YYYY.
date_1 = date_1 + 06.
Write: / 'Date after 6 Days is:', date_1 DD/MM/YYYY.
```

Internal Table concept and Usage

Internal tables provide a means of taking data from a fixed structure and storing it in working memory in ABAP.

The data is stored line by line in memory, and each line has the same structure.

In ABAP, internal tables fulfill the function of arrays.

Internal tables are used to obtain data from a fixed structure for dynamic use in ABAP.

Each line in the internal table has the same field structure.

The main use for internal tables is for storing and formatting data from a database table within a program.

Types of Internal Tables - Standard, Sorted, Hashed

```
types : begin of s_vbap,
          zvbeln type vbap-vbeln,
          zposnr type vbap-posnr,
          zmatnr type vbap-matnr,
          znetwr type vbap-netwr,
        end of s_vbap.

*First Type of internal table
*Standard Table
*data : t_vbap type STANDARD TABLE OF s_vbap WITH HEADER LINE.
*       wa_vbap type s_vbap.

data : t_vbap type STANDARD TABLE OF s_vbap,
          wa_vbap type s_vbap.

*Second Type of internal
* Sorted Table -  Sorted data --> Non-Unique
data : t_vbap_snu type sorted TABLE OF s_vbap with NON-UNIQUE key zvbeln zposnr,
         wa_vbap_snu type s_vbap.

* Sorted Table -  Sorted data --> Unique Key
data : t_vbap_su type sorted TABLE OF s_vbap with UNIQUE key zvbeln zposnr,
       wa_vbap_su type s_vbap.

*Third Type
* Hashed Table
data : t_vbap_h type HASHED TABLE OF s_vbap with UNIQUE key zvbeln zposnr,
       wa_vbap_h type s_vbap.


write:/ '***************with Standard Table*********************************'.
select vbeln posnr matnr netwr
  into table t_vbap
  from vbap
  UP to 10 rows.

sort t_vbap.

DELETE ADJACENT DUPLICATES FROM t_vbap comparing zvbeln.

loop at t_vbap into wa_vbap.
   write:/ wa_vbap-zvbeln, wa_vbap-zposnr, wa_vbap-zmatnr, wa_vbap-znetwr.
**write:/ wa_vbap.
ENDLOOP.
*
write:/.
Write:/ '***************with Sorted Non-Unique key*********************************'.

select vbeln posnr matnr netwr
  into table t_vbap_snu
  from vbap
  UP to 10 rows.
```

```
DELETE ADJACENT DUPLICATES FROM t_vbap_snu comparing zvbeln.

loop at t_vbap_snu into wa_vbap_snu.
   write:/ wa_vbap_snu-zvbeln, wa_vbap_snu-zposnr, wa_vbap_snu-
zmatnr, wa_vbap_snu-znetwr.
**write:/ wa_vbap.
ENDLOOP.


write:/.
Write:/ '***************with Sorted Unique key*********************************'.

select vbeln posnr matnr netwr
  into table t_vbap_su
  from vbap
  UP to 10 rows.

loop at t_vbap_su into wa_vbap_su.
   write:/ wa_vbap_su-zvbeln, wa_vbap_su-zposnr, wa_vbap_su-zmatnr, wa_vbap_su-
znetwr.
**write:/ wa_vbap.
ENDLOOP.

write:/.
Write:/ '***************with Hashed Unique key*********************************'.

select vbeln posnr matnr netwr
  into table t_vbap_h
  from vbap
  UP to 10 rows.

loop at t_vbap_h into wa_vbap_h.
   write:/ wa_vbap_h-zvbeln, wa_vbap_h-zposnr, wa_vbap_h-zmatnr, wa_vbap_h-znetwr.
**write:/ wa_vbap.
ENDLOOP.
```

Database Tables Concepts

A table can contain one or more fields, each defined with its data type and length. The large amount of data stored in a table is distributed among the several fields defined in the table

A table consists of many fields, and each field contains many elements.

Viewing of Database Tables & Selection Functions

Select Statement

SELECT is the Open SQL statement for reading data from one or more database tables into data objects.

The select statement reads a result set (whose structure is determined in **result**) from the database tables specified in **source** and assigns the data from the result set to the data objects specified in **target**. You can restrict the result set using the **WHERE** addition.

```
Data : vbeln type vbap-vbeln,
           posnr type vbap-posnr,
          matnr type vbap-matnr.

write :/ '***********Output using variables*************'.

SELECT vbeln posnr matnr
       into (vbeln, posnr, matnr)
       from vbap
       up to 10 rows.

      write :/ vbeln, posnr, matnr.

ENDSELECT.
```

Read Statement

READ statement is used to read the lines of internal table.

```
REPORT ZREAD_DEMO.
 */Creating an internal table
DATA: BEGIN OF Record1,
                  ColP TYPE I,
                  ColQ TYPE I,
           END OF Record1.
DATA mytable LIKE HASHED TABLE OF Record1 WITH UNIQUE KEY ColP.
DO 6 Times.
          Record1-ColP = SY-INDEX.
         Record1-ColQ = SY-INDEX + 5.
         INSERT Record1 INTO TABLE mytable.
ENDDO.
        Record1-ColP = 4.
        Record1-ColQ = 12.
READ TABLE mytable FROM Record1 INTO Record1 COMPARING ColQ.
        WRITE: 'SY-SUBRC =', SY-SUBRC.
        SKIP.
       WRITE: / Record1-ColP, Record1-ColQ.
```

Other Important ABAP Statements /Keywords

INSERT → Adds a new record to the table. If the row (key) exists, issues an error.

UPDATE → Updates an existing record to the table. If the row (key) does not exist, issues an error.

MODIFY → If the key exists, modifies the record. If the key does not exist, adds the record to the table.

MODIFY….TRANSPORTING → TRANSPORTING Statement tells ABAP that only specific fields will be modified which are given after the TRANSPORTING clause.

APPEND → This adds a new record to the internal table in the last position.

COLLECT → The table is first checked for an entry of the key-fields in the table with the comparison of the coming entry. If the key-fields' entry is present, this adds up all the numeric fields of the record with the existing record. If the record is totally new, then it is appended to the table.

DELETE → is used to delete one or more records from an internal table. The records of an internal table are deleted either by specifying a table key or condition or by finding duplicate entries.

DELETE ADJACENT DUPLICATE → This statement works logically on sorted standard table and sorted table with non-unique key. It compares the adjacent rows. If similar records are found based on the comparing fields then it deletes from the second records onward. The system keeps only the first record.

MOVE-CORRESPONDING → The statement MOVE-CORRESPONDING is used to assign components with the same name in structured data objects to each other.

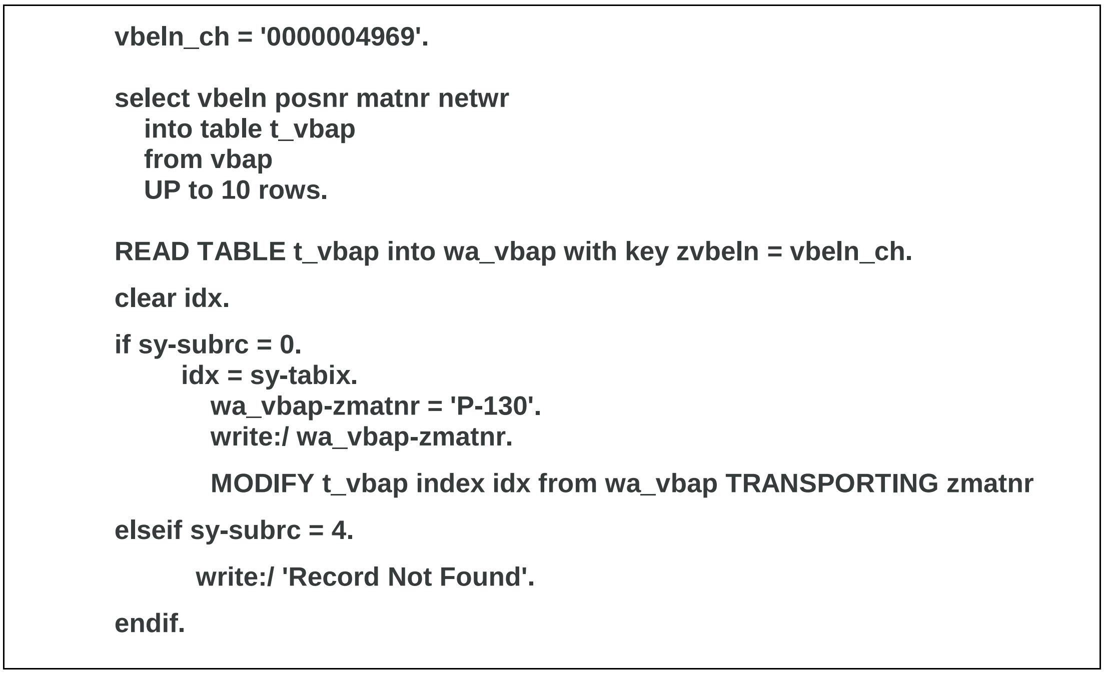
CLEAR → Clears the contents of the variable.
Variables may be internal table, workarea, variables declared using elementary data types etc.

REFRESH → Clears the content of internal table only.

FREE → Clears the contents and releases the memory of internal table. This keyword is applicable only in case of internal table

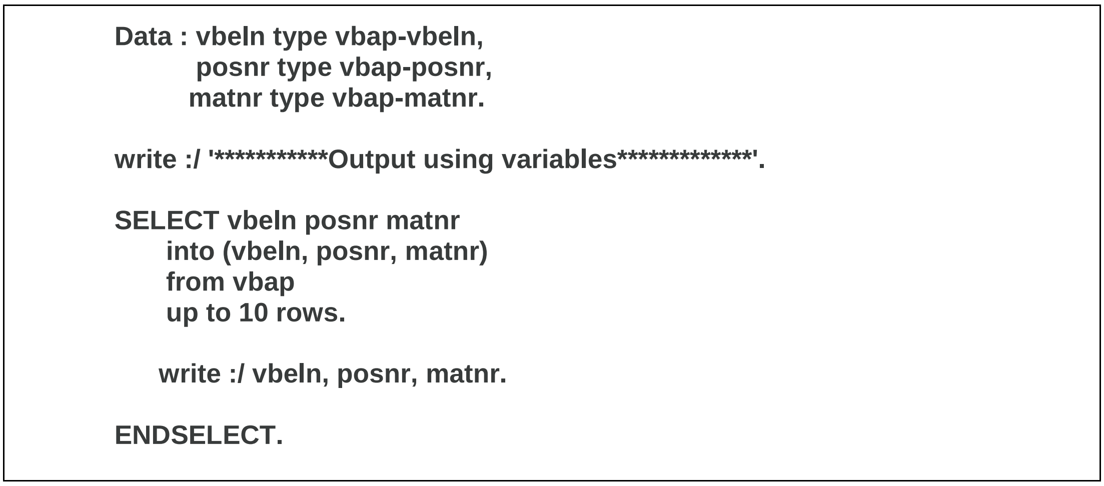
Modify…..Transport Statement

```
types : begin of s_vbap,
          zvbeln type vbap-vbeln,
          zposnr type vbap-posnr,
          zmatnr type vbap-matnr,
          znetwr type vbap-netwr,
        end of s_vbap.

data : t_vbap type standard table of s_vbap,
       wa_vbap type s_vbap,

       vbeln_ch type VBELN_VA,

       idx type sy-tabix.
```



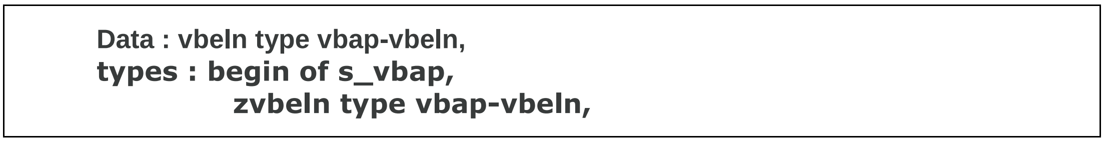
```
vbeln_ch = '0000004969'.

select vbeln posnr matnr netwr
   into table t_vbap
   from vbap
   UP to 10 rows.

READ TABLE t_vbap into wa_vbap with key zvbeln = vbeln_ch.

clear idx.

if sy-subrc = 0.
         idx = sy-tabix.
             wa_vbap-zmatnr = 'P-130'.
             write:/ wa_vbap-zmatnr.

             MODIFY t_vbap index idx from wa_vbap TRANSPORTING zmatnr

elseif sy-subrc = 4.

           write:/ 'Record Not Found'.

endif.
```

Delete Adjacent duplicates, Clear, Refresh, Free Statements



```
Data : vbeln type vbap-vbeln,
           posnr type vbap-posnr,
          matnr type vbap-matnr.

write :/ '***********Output using variables*************'.

SELECT vbeln posnr matnr
       into (vbeln, posnr, matnr)
       from vbap
       up to 10 rows.

      write :/ vbeln, posnr, matnr.

ENDSELECT.
```

Field Symbols



```
Data : vbeln type vbap-vbeln,
types : begin of s_vbap,
                 zvbeln type vbap-vbeln,
             zposnr type vbap-posnr,
             zmatnr type vbap-matnr,
             znetwr type vbap-netwr,
           end of s_vbap.

data : t_vbap type standard table of s_vbap,
       vbeln_ch type vbap-vbeln.

field-symbols :<fs> type s_vbap.

vbeln_ch = '0000004969'.

select vbeln posnr matnr netwr
   into table t_vbap
   from vbap
   up to 10 rows.

Read table t_vbap assigning <fs> with key zvbeln = vbeln_ch.

if sy-subrc = 0.
     <fs>-zmatnr = 'P-130'.
     write:/ <fs>-zmatnr.
elseif sy-subrc = 4.
     write:/ 'Record not found'.
endif.
```

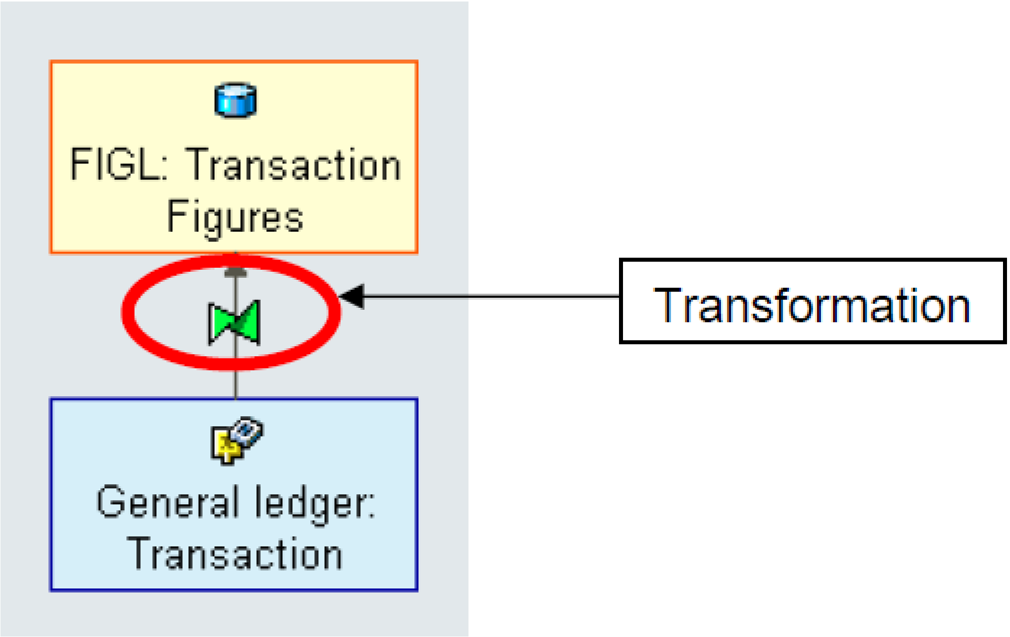
BW transformation Routines

Create Transformation and Map Characteristics & Key Figures

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

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

For purposes of our example, your company is implementing FI-GL and loading data from the 0FI_GL_1 dataSource in SAP R/3. You are loading this data into the level 1 DataStore Object ZFIGL_06. See diagram below:



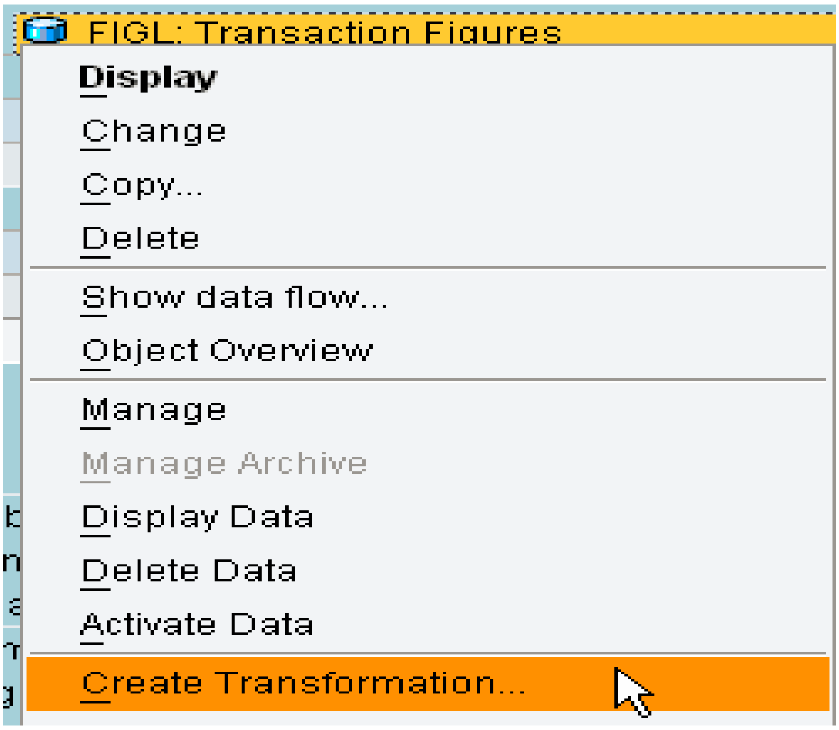
This transformation will be enhanced via routines by performing the following actions:

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

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

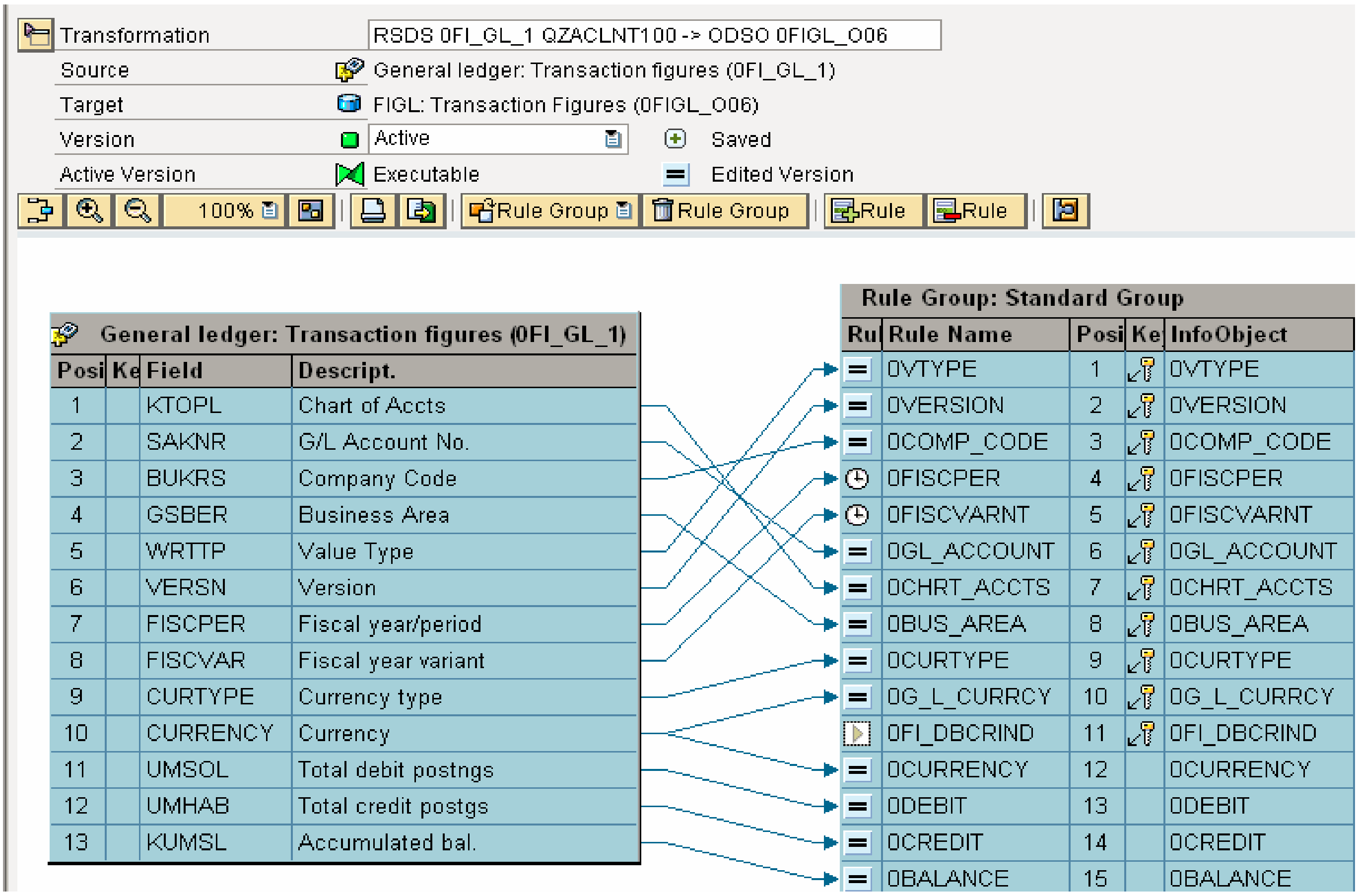
Create Transformation

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



FIGL: Transaction Figures

Create the relevant direct mappings by dragging and dropping the source field to their relevant targets.



Transformation: RSDS 0FI_GL_1 QZACLNT100 -> ODSO 0FIGL_O06

Source: General ledger: Transaction figures (0FI_GL_1)

Target: FIGL: Transaction Figures (0FIGL_O06)

Version: Active — Saved

Active Version: Executable — Edited Version

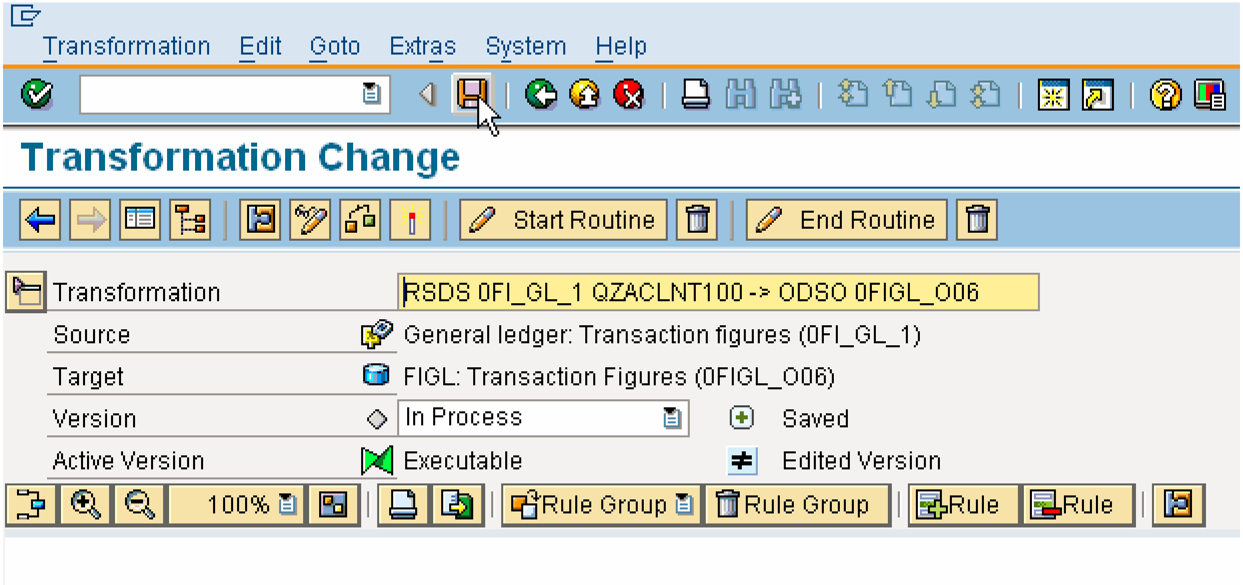
General ledger: Transaction figures (0FI_GL_1)

Posi	Ke	Field	Descript.
1		KTOPL	Chart of Accts
2		SAKNR	G/L Account No.
3		BUKRS	Company Code
4		GSBER	Business Area
5		WRTTP	Value Type
6		VERSN	Version
7		FISCPER	Fiscal year/period
8		FISCVAR	Fiscal year variant
9		CURTYPE	Currency type
10		CURRENCY	Currency
11		UMSOL	Total debit postngs
12		UMHAB	Total credit postgs
13		KUMSL	Accumulated bal.

Rule Group: Standard Group

Ru	Rule Name	Posi	Ke	InfoObject
=	0VTYPE	1	🔑	0VTYPE
=	0VERSION	2	🔑	0VERSION
=	0COMP_CODE	3	🔑	0COMP_CODE
◷	0FISCPER	4	🔑	0FISCPER
◷	0FISCVARNT	5	🔑	0FISCVARNT
=	0GL_ACCOUNT	6	🔑	0GL_ACCOUNT
=	0CHRT_ACCTS	7	🔑	0CHRT_ACCTS
=	0BUS_AREA	8	🔑	0BUS_AREA
=	0CURTYPE	9	🔑	0CURTYPE
=	0G_L_CURRCY	10	🔑	0G_L_CURRCY
▷	0FI_DBCRIND	11	🔑	0FI_DBCRIND
=	0CURRENCY	12		0CURRENCY
=	0DEBIT	13		0DEBIT
=	0CREDIT	14		0CREDIT
=	0BALANCE	15		0BALANCE

Now save the transformation rule group.



Transformation Change

Transformation: RSDS 0FI_GL_1 QZACLNT100 -> ODSO 0FIGL_O06

Source: General ledger: Transaction figures (0FI_GL_1)

Target: FIGL: Transaction Figures (0FIGL_O06)

Version: In Process — Saved

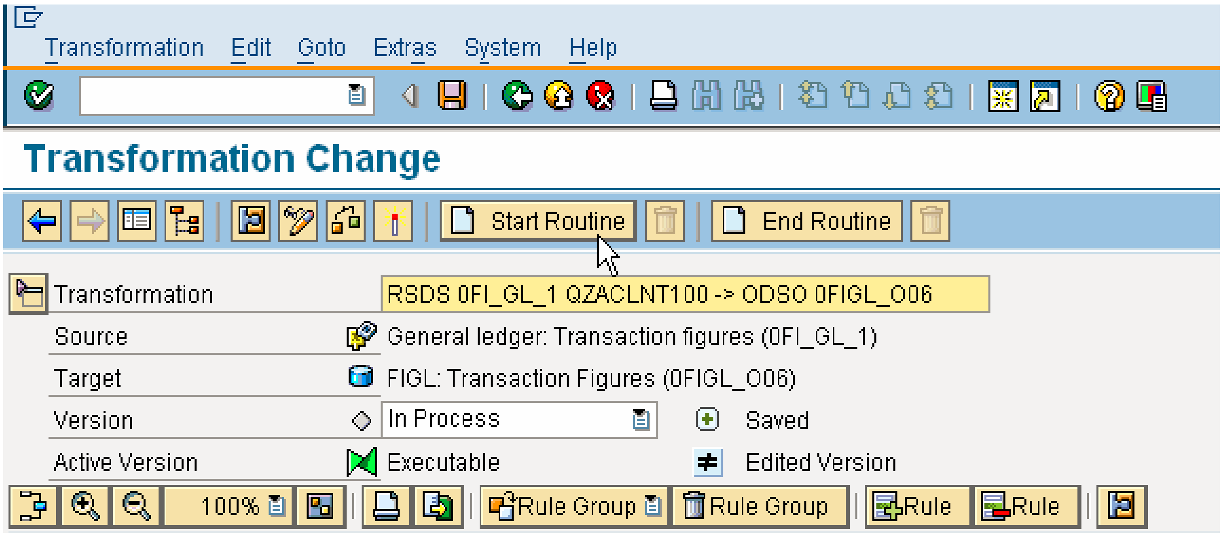
Active Version: Executable — Edited Version

Start Routine in the Transformation

Create a Start Routine

Steps

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



Transformation Change

Transformation RSDS 0FI_GL_1 QZACLNT100 -> ODSO 0FIGL_O06

Source General ledger: Transaction figures (0FI_GL_1)

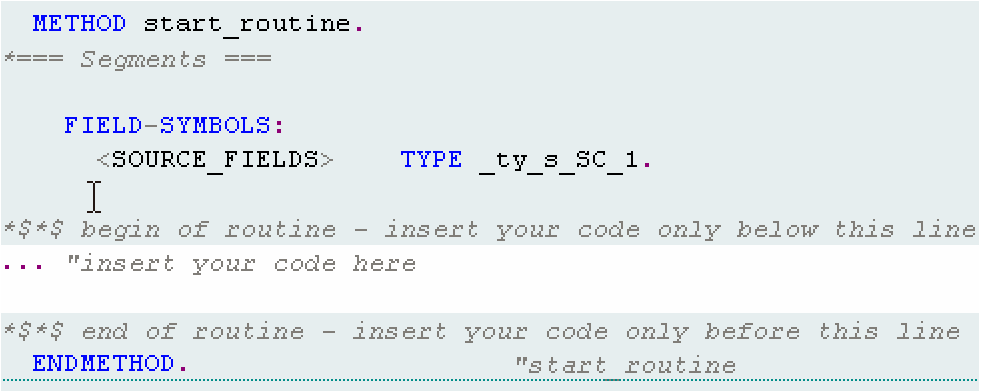
Target FIGL: Transaction Figures (0FIGL_O06)

Version In Process Saved

Active Version Executable Edited Version

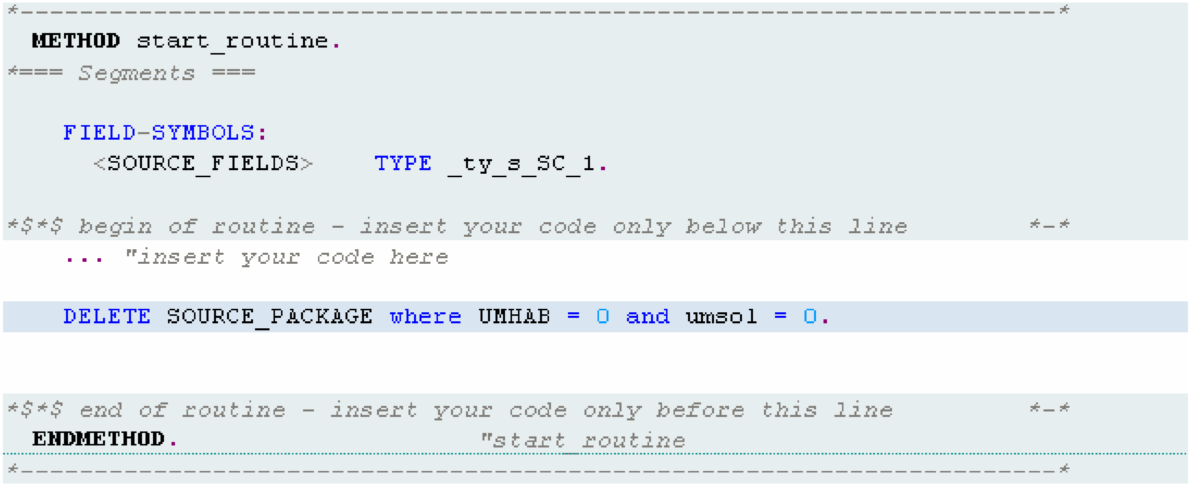
2. From within the start routine there are two sections of code to be filled.
   a. Global Section (optional)
   b. Local Section

   Navigate to the Local section



```
  METHOD start_routine.
*=== Segments ===

    FIELD-SYMBOLS:
      <SOURCE_FIELDS>    TYPE _ty_s_SC_1.

*$*$ begin of routine - insert your code only below this line
... "insert your code here

*$*$ end of routine - insert your code only before this line
  ENDMETHOD.                          "start_routine
```

3. Tasked to eliminate all records that have neither a debit nor a credit, the first and only step is insert a delete statement
   - The source package is filtered.

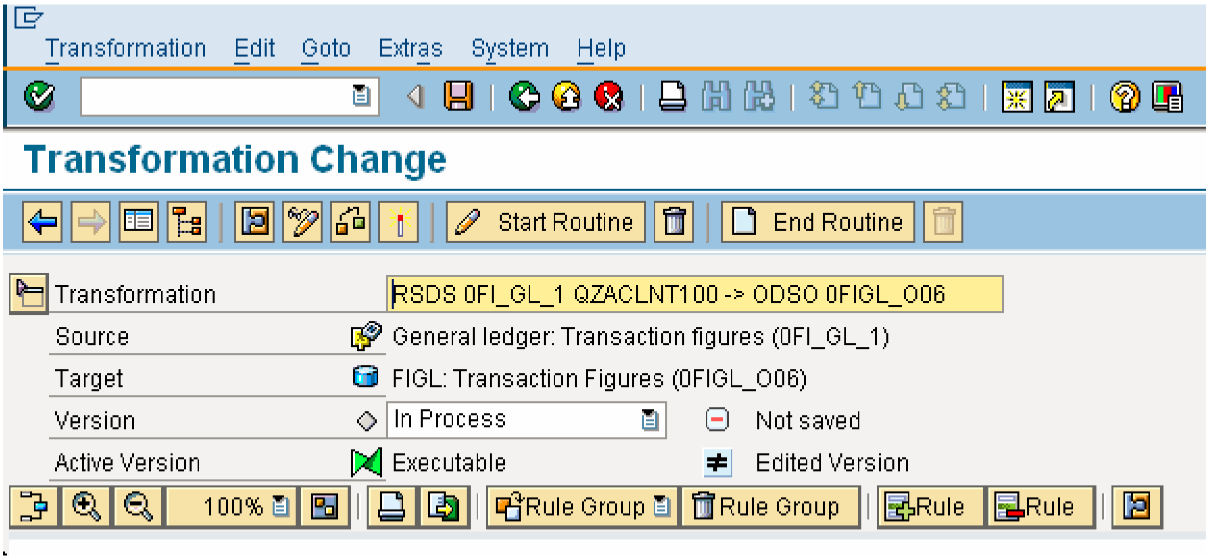


```
*----------------------------------------------------------------------*
  METHOD start_routine.
*=== Segments ===

    FIELD-SYMBOLS:
      <SOURCE_FIELDS>    TYPE _ty_s_SC_1.

*$*$ begin of routine - insert your code only below this line        *-*
    ... "insert your code here

    DELETE SOURCE_PACKAGE where UMHAB = 0 and umsol = 0.


*$*$ end of routine - insert your code only before this line         *-*
  ENDMETHOD.                         "start_routine
*----------------------------------------------------------------------*
```

The Start Routine is now complete

4. Save the Start Routine and enter back into change mode for the Transformation. Save the Transformation as well.

    - There now exists a pencil on the start routine icon which is indicative of the fact that a start routine exists.



Transformation Change

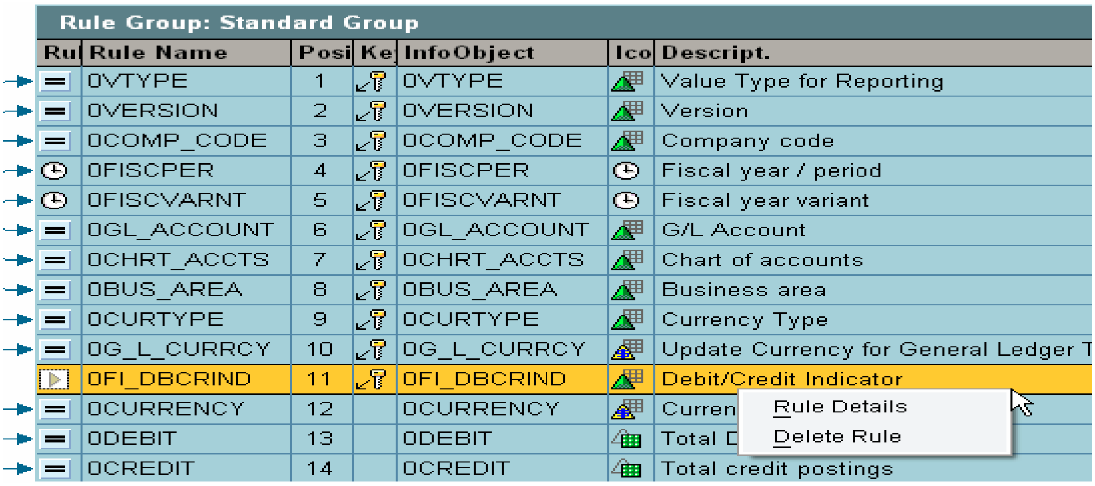
Transformation	RSDS 0FI_GL_1 QZACLNT100 -> ODSO 0FIGL_O06		
Source	General ledger: Transaction figures (0FI_GL_1)		
Target	FIGL: Transaction Figures (0FIGL_O06)		
Version	In Process		Not saved
Active Version	Executable		Edited Version

Create a Routine for Updating Characteristics

Steps

1. As dictated by the customer the next step is to populate the Debit/Credit indicator with a value of ‘D’ if there is a debit posting on the record and a ‘C’ if there is a credit posting on the record.

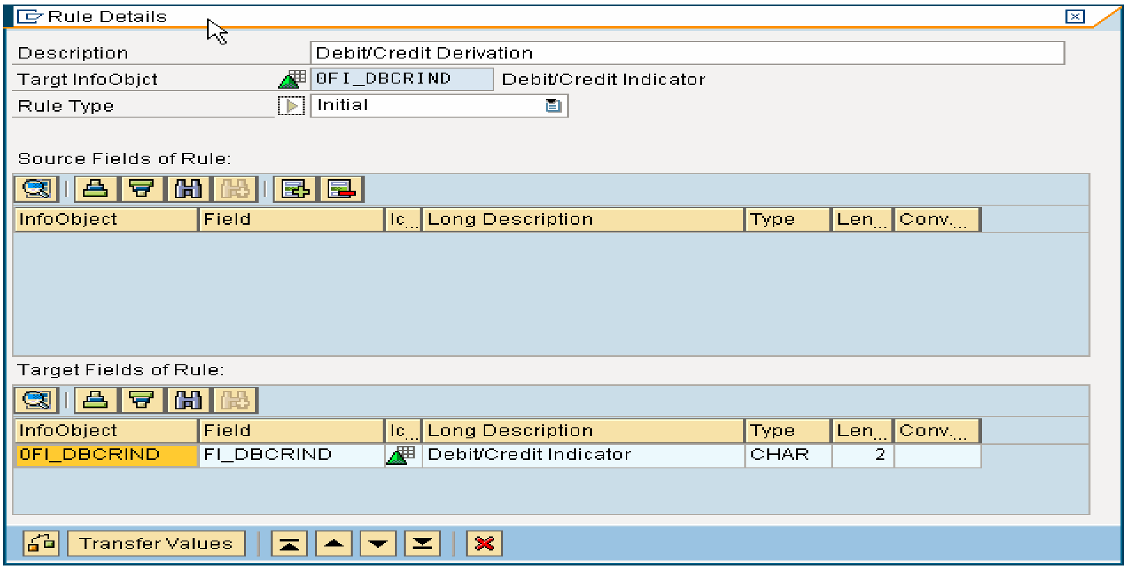
   Right-Click on the Debit/Credit Indicator field within the rule group and click on the Rule Details button.



Rule Group: Standard Group

Ru	Rule Name	Posi	Ke	InfoObject	Ico	Descript.
	0VTYPE	1		0VTYPE		Value Type for Reporting
	0VERSION	2		0VERSION		Version
	0COMP_CODE	3		0COMP_CODE		Company code
	0FISCPER	4		0FISCPER		Fiscal year / period
	0FISCVARNT	5		0FISCVARNT		Fiscal year variant
	0GL_ACCOUNT	6		0GL_ACCOUNT		G/L Account
	0CHRT_ACCTS	7		0CHRT_ACCTS		Chart of accounts
	0BUS_AREA	8		0BUS_AREA		Business area
	0CURTYPE	9		0CURTYPE		Currency Type
	0G_L_CURRCY	10		0G_L_CURRCY		Update Currency for General Ledger T
	0FI_DBCRIND	11		0FI_DBCRIND		Debit/Credit Indicator
	0CURRENCY	12		0CURRENCY		Curren
	0DEBIT	13		0DEBIT		Total D
	0CREDIT	14		0CREDIT		Total credit postings

2. Give a Description to the rule being created.



Rule Details

Description	Debit/Credit Derivation	
Targt InfoObjct	0FI_DBCRIND	Debit/Credit Indicator
Rule Type	Initial	

Source Fields of Rule:

InfoObject	Field	Ic..	Long Description	Type	Len..	Conv...

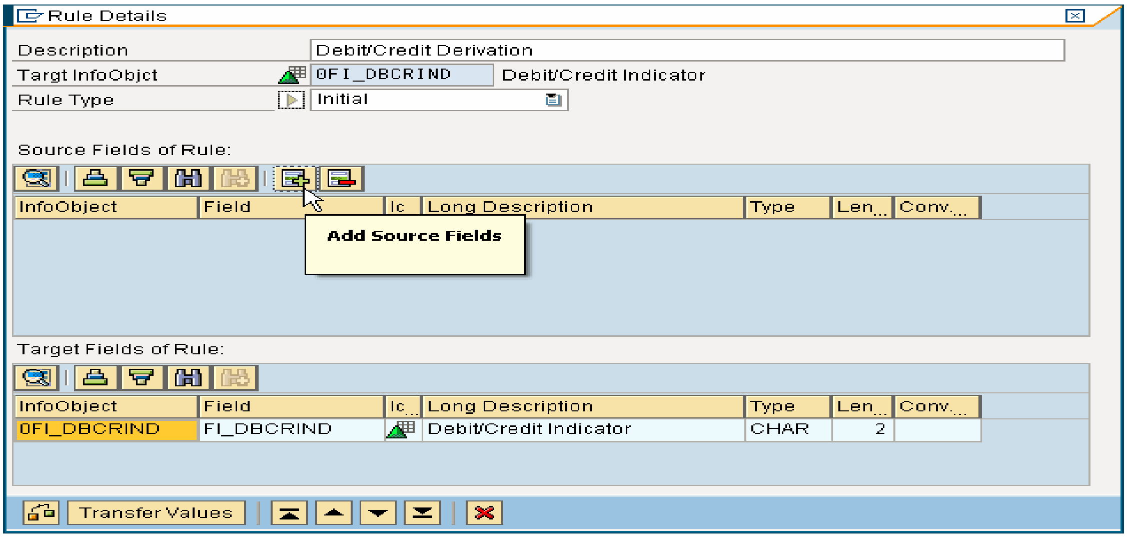
Target Fields of Rule:

InfoObject	Field	Ic..	Long Description	Type	Len..	Conv...
0FI_DBCRIND	FI_DBCRIND		Debit/Credit Indicator	CHAR	2	

3. Now the following source fields need to be added to the rule so they can be accessed within the routine. Add the two fields and hit the Green OK button.

   a. **UMSOL – *Total Debit Postgs***

   b. **UMHAB – *Total Credit Postgs***



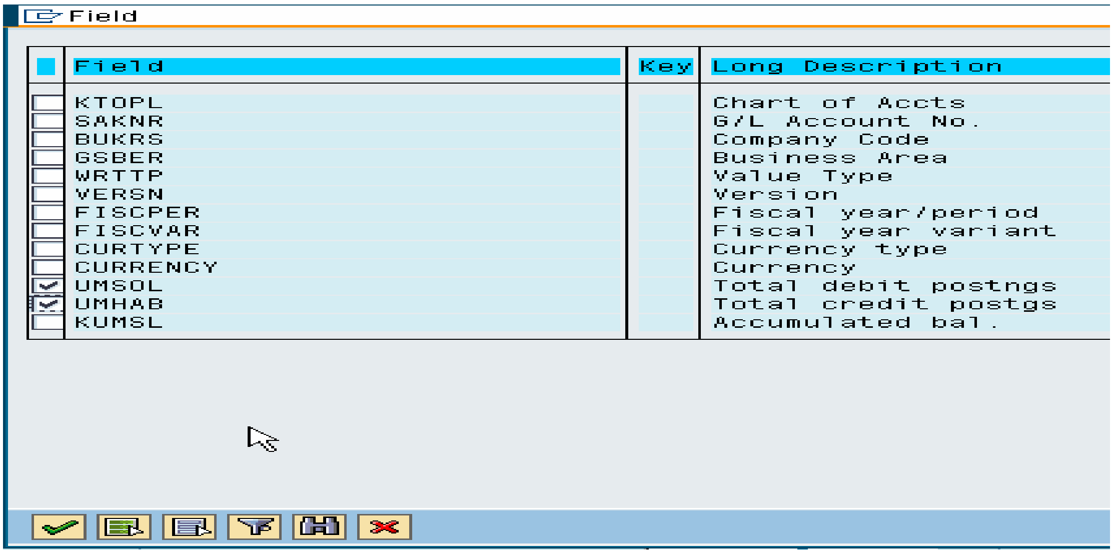
Description	Debit/Credit Derivation
Targt InfoObjct	0FI_DBCRIND Debit/Credit Indicator
Rule Type	Initial

Source Fields of Rule:

InfoObject	Field	Ic...	Long Description	Type	Len...	Conv...

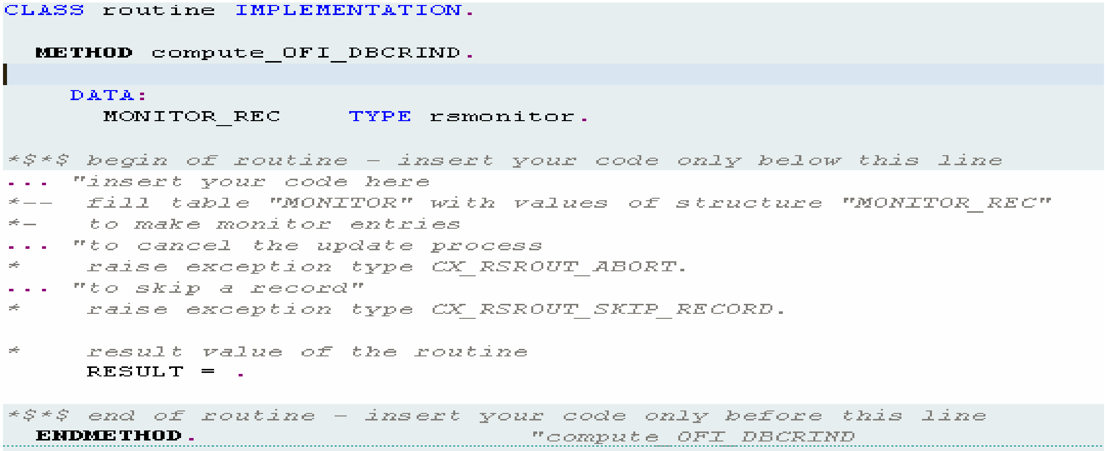
Target Fields of Rule:

InfoObject	Field	Ic...	Long Description	Type	Len...	Conv...
0FI_DBCRIND	FI_DBCRIND		Debit/Credit Indicator	CHAR	2	



	Field	Key	Long Description
□	KTOPL		Chart of Accts
□	SAKNR		G/L Account No.
□	BUKRS		Company Code
□	GSBER		Business Area
□	WRTTP		Value Type
□	VERSN		Version
□	FISCPER		Fiscal year/period
□	FISCVAR		Fiscal year variant
□	CURTYPE		Currency type
□	CURRENCY		Currency
☑	UMSOL		Total debit postngs
☑	UMHAB		Total credit postgs
□	KUMSL		Accumulated bal.

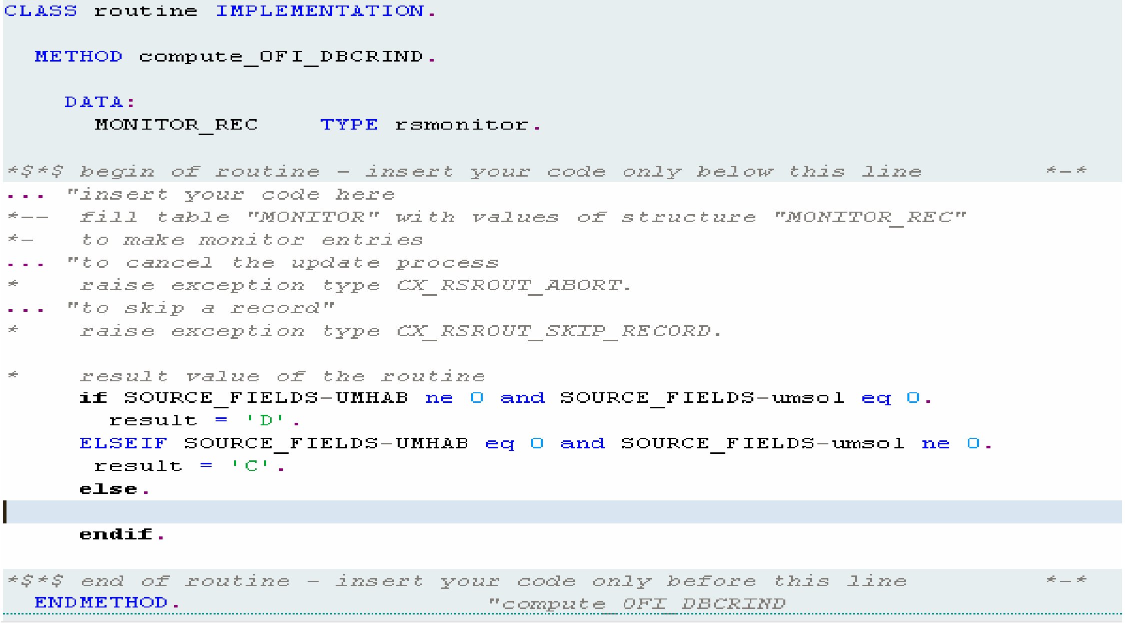
4. Within this piece of code the logic needs to be added to derive either a ‘D’ or a ‘C’ for our result field.



```
CLASS routine IMPLEMENTATION.

  METHOD compute_0FI_DBCRIND.

    DATA:
      MONITOR_REC     TYPE rsmonitor.

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

*    result value of the routine
     RESULT = .

*$*$ end of routine - insert your code only before this line
  ENDMETHOD.                    "compute_0FI_DBCRIND
```

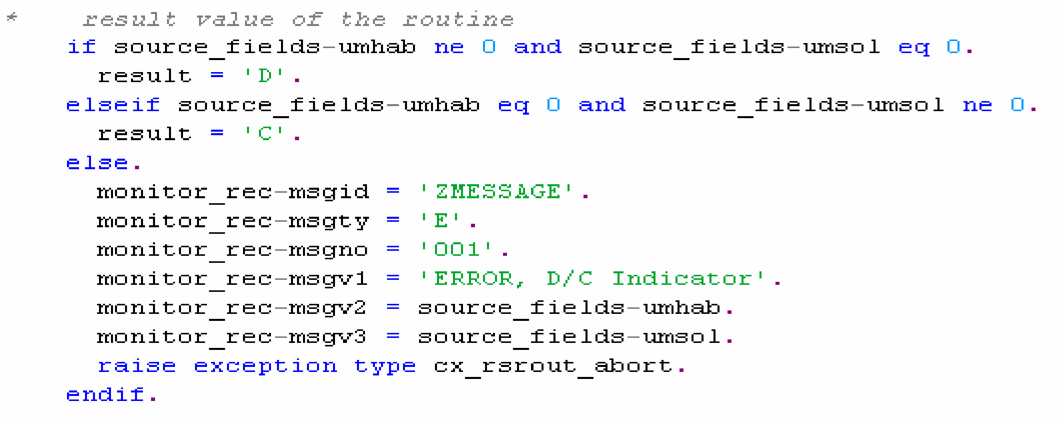
5. A conditional statement needs to be created that determines whether the debit posting field has a value (triggering the population of the the result with a ‘D’) or the credit posting field has a value (triggering the population of the result with a ‘C’).

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



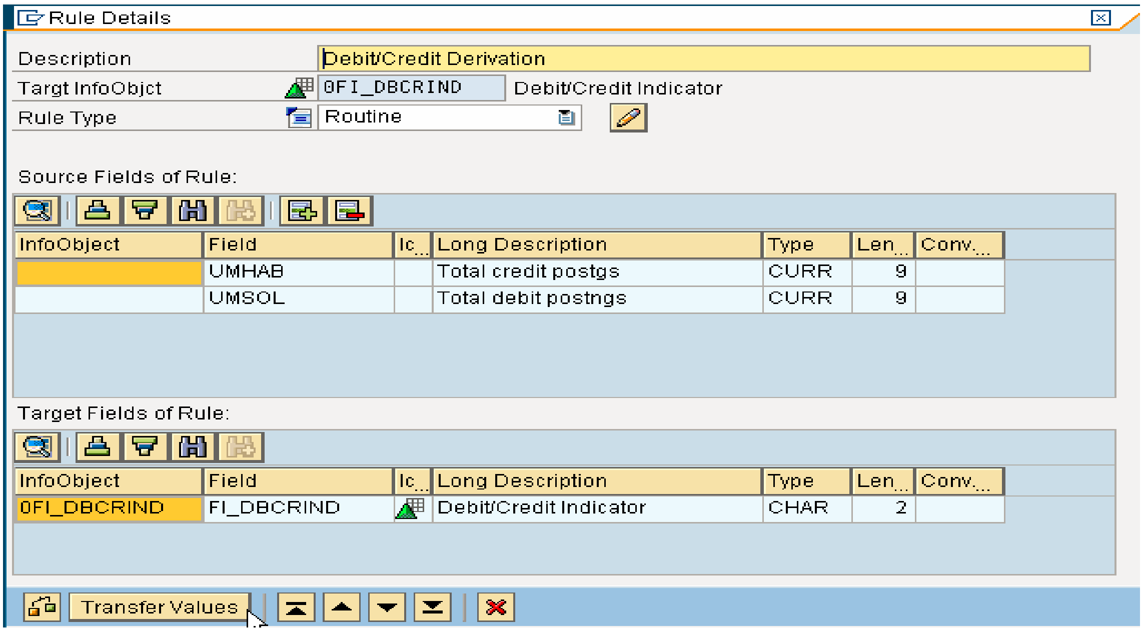
```
CLASS routine IMPLEMENTATION.

  METHOD compute_OFI_DBCRIND.

    DATA:
      MONITOR_REC     TYPE rsmonitor.

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

*    result value of the routine
     if SOURCE_FIELDS-UMHAB ne 0 and SOURCE_FIELDS-umsol eq 0.
       result = 'D'.
     ELSEIF SOURCE_FIELDS-UMHAB eq 0 and SOURCE_FIELDS-umsol ne 0.
      result = 'C'.
     else.

     endif.

*$*$ end of routine - insert your code only before this line         *-*
  ENDMETHOD.                    "compute_OFI_DBCRIND
```

6. The last step is to catch an exception if both the credit and debit fields have a value this is an error so a message needs to be written to the monitor and we will raise an exception to stop the load.



```
*     result value of the routine
    if source_fields-umhab ne 0 and source_fields-umsol eq 0.
      result = 'D'.
    elseif source_fields-umhab eq 0 and source_fields-umsol ne 0.
      result = 'C'.
    else.
      monitor_rec-msgid = 'ZMESSAGE'.
      monitor_rec-msgty = 'E'.
      monitor_rec-msgno = '001'.
      monitor_rec-msgv1 = 'ERROR, D/C Indicator'.
      monitor_rec-msgv2 = source_fields-umhab.
      monitor_rec-msgv3 = source_fields-umsol.
      raise exception type cx_rsrout_abort.
    endif.
```

7. Save the Characteristic Routine and Transfer the values back to the Rule Group. Save your Transformations



Description	Debit/Credit Derivation	
Targt InfoObjct	0FI_DBCRIND	Debit/Credit Indicator
Rule Type	Routine	

Source Fields of Rule:

InfoObject	Field	Ic...	Long Description	Type	Len...	Conv...
	UMHAB		Total credit postgs	CURR	9	
	UMSOL		Total debit postngs	CURR	9	

Target Fields of Rule:

InfoObject	Field	Ic...	Long Description	Type	Len...	Conv...
0FI_DBCRIND	FI_DBCRIND		Debit/Credit Indicator	CHAR	2	

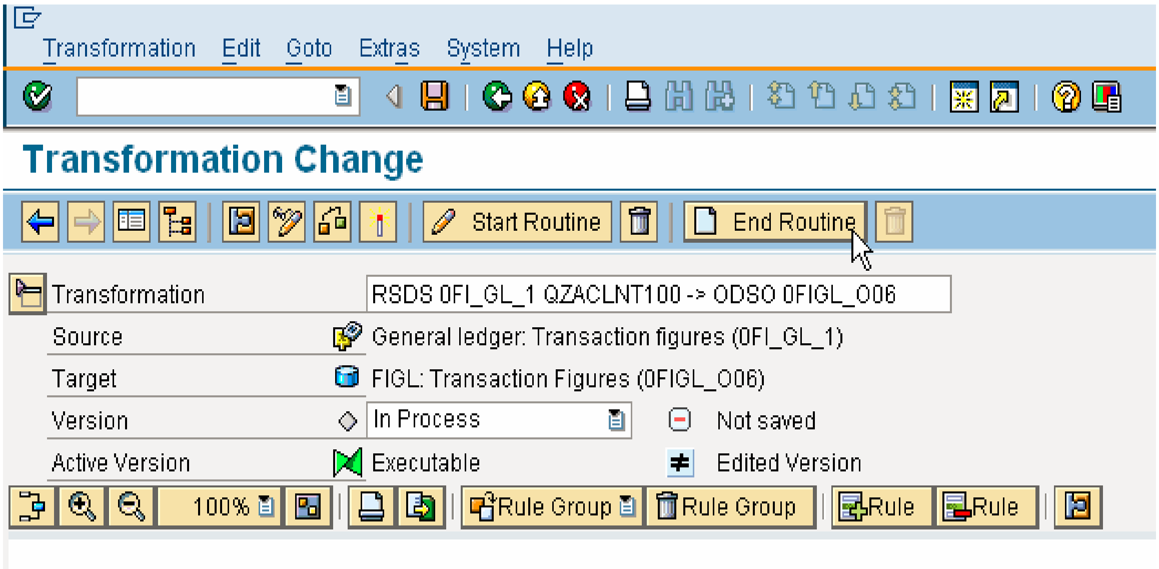
End Routine in the Transformation

Create End Routine

Steps

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

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

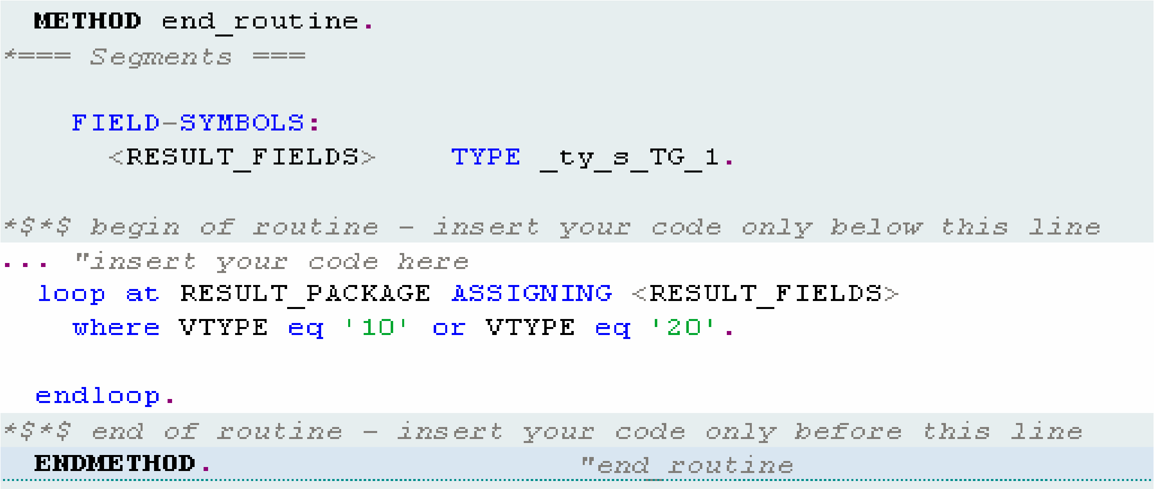


Transformation	RSDS 0FI_GL_1 QZACLNT100 -> ODSO 0FIGL_O06
Source	General ledger: Transaction figures (0FI_GL_1)
Target	FIGL: Transaction Figures (0FIGL_O06)
Version	In Process — Not saved
Active Version	Executable — Edited Version

2. The end routine to be populated looks very similar to the start routine. The result_package

needs to be looped through where the R/3 value types are either plan (20) or actual (10) value types.

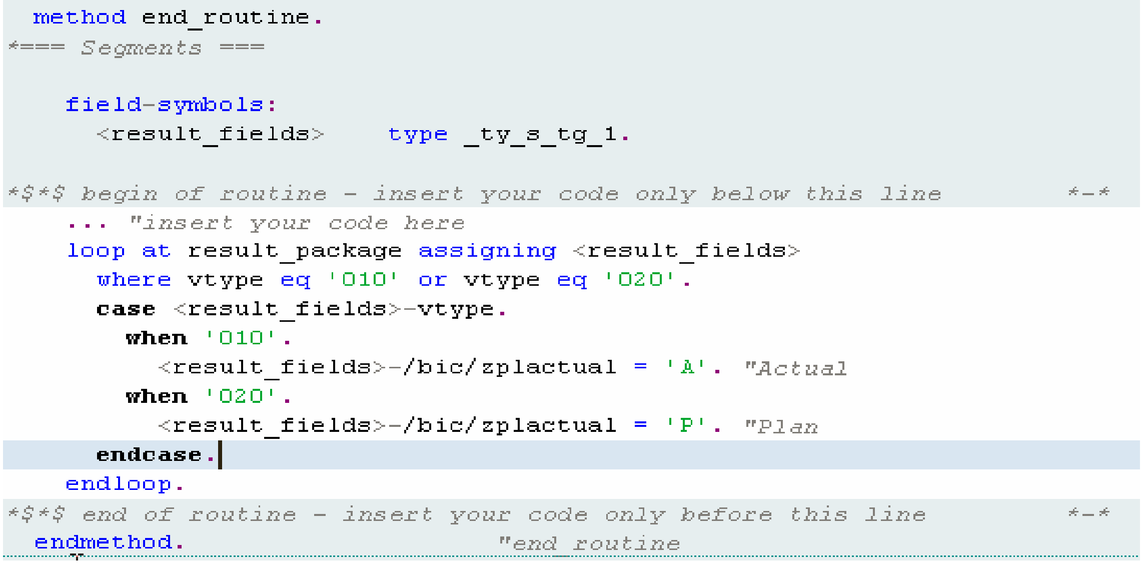
- The code here is looping through the result_package into the field symbol <result_fields> provided by the method only for records that have the value types 10 or 20



```
  METHOD end_routine.
*==== Segments ===

    FIELD-SYMBOLS:
      <RESULT_FIELDS>     TYPE _ty_s_TG_1.

*$*$ begin of routine - insert your code only below this line
... "insert your code here
  loop at RESULT_PACKAGE ASSIGNING <RESULT_FIELDS>
    where VTYPE eq '10' or VTYPE eq '20'.

  endloop.
*$*$ end of routine - insert your code only before this line
  ENDMETHOD.                       "end_routine
```

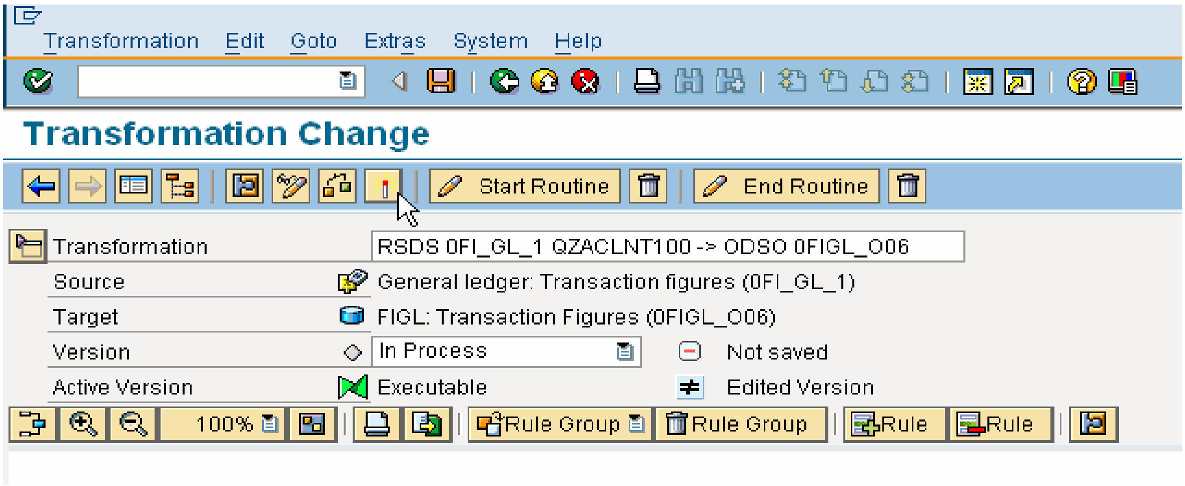
3. If the value type of a given record is 10 the plan/actual indicator receives the value ‘A’ for actual. If the value type is 20 the value passed to the plan/actual indicator is ‘P’ for plan.

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



```
  method end_routine.
*==== Segments ===

    field-symbols:
      <result_fields>     type _ty_s_tg_1.

*$*$ begin of routine - insert your code only below this line        *-*
    ... "insert your code here
    loop at result_package assigning <result_fields>
      where vtype eq '010' or vtype eq '020'.
      case <result_fields>-vtype.
        when '010'.
          <result_fields>-/bic/zplactual = 'A'. "Actual
        when '020'.
          <result_fields>-/bic/zplactual = 'P'. "Plan
      endcase.
    endloop.
*$*$ end of routine - insert your code only before this line         *-*
  endmethod.                          "end_routine
```

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



Transformation Change

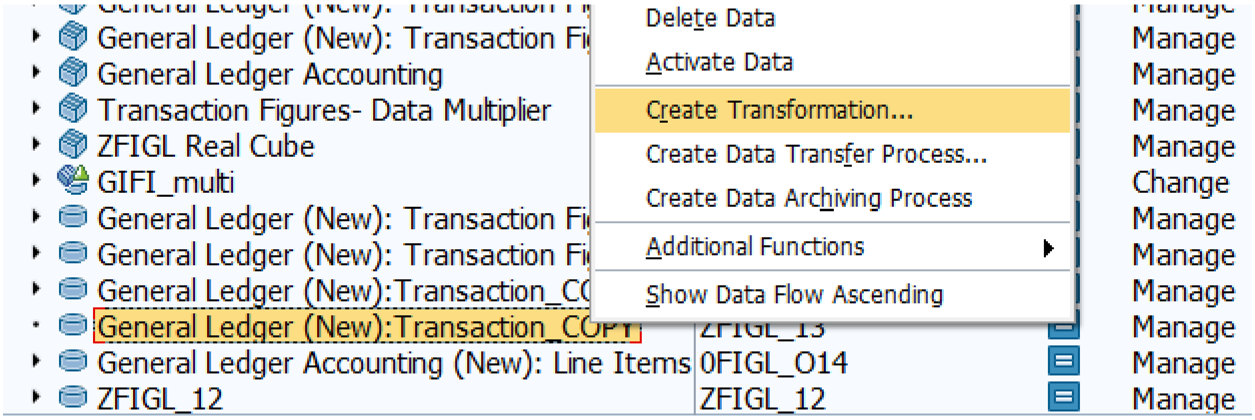
Transformation	RSDS 0FI_GL_1 QZACLNT100 -> ODSO 0FIGL_O06
Source	General ledger: Transaction figures (0FI_GL_1)
Target	FIGL: Transaction Figures (0FIGL_O06)
Version	In Process — Not saved
Active Version	Executable — Edited Version

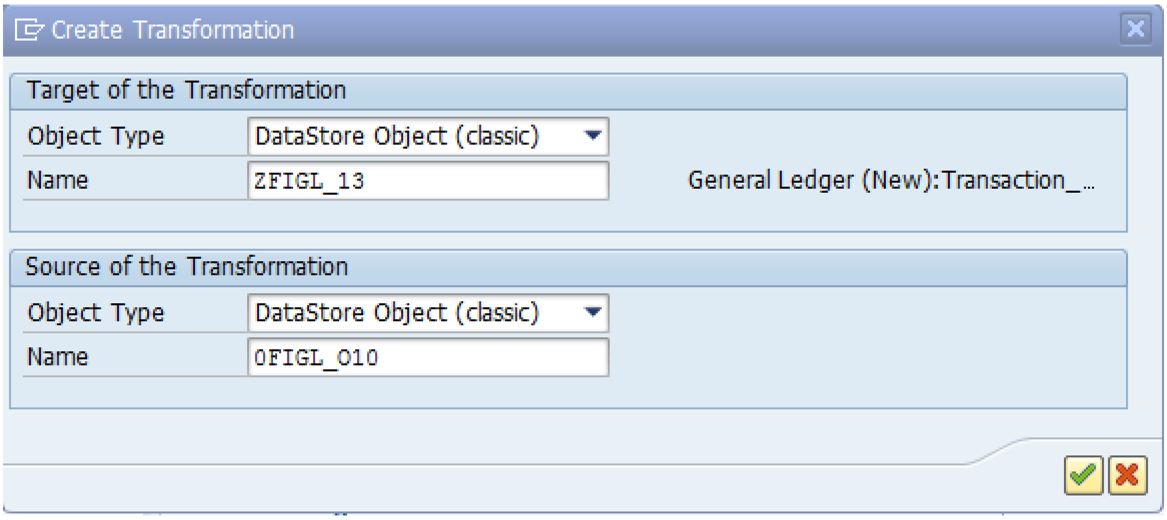
Expert Routine in the Transformation

Create Expert Routine

Steps

1. Let us start with creation of transformation


General Ledger (New): Transaction Fi
General Ledger Accounting
Transaction Figures- Data Multiplier
ZFIGL Real Cube
GIFI_multi
General Ledger (New): Transaction Fi
General Ledger (New): Transaction Fi
General Ledger (New):Transaction_CC
General Ledger (New):Transaction_COPY
ZFIGL_13
General Ledger Accounting (New): Line Items
0FIGL_O14
ZFIGL_12
ZFIGL_12
Manage
Change
Delete Data
Activate Data
Create Transformation...
Create Data Transfer Process...
Create Data Archiving Process
Additional Functions
Show Data Flow Ascending



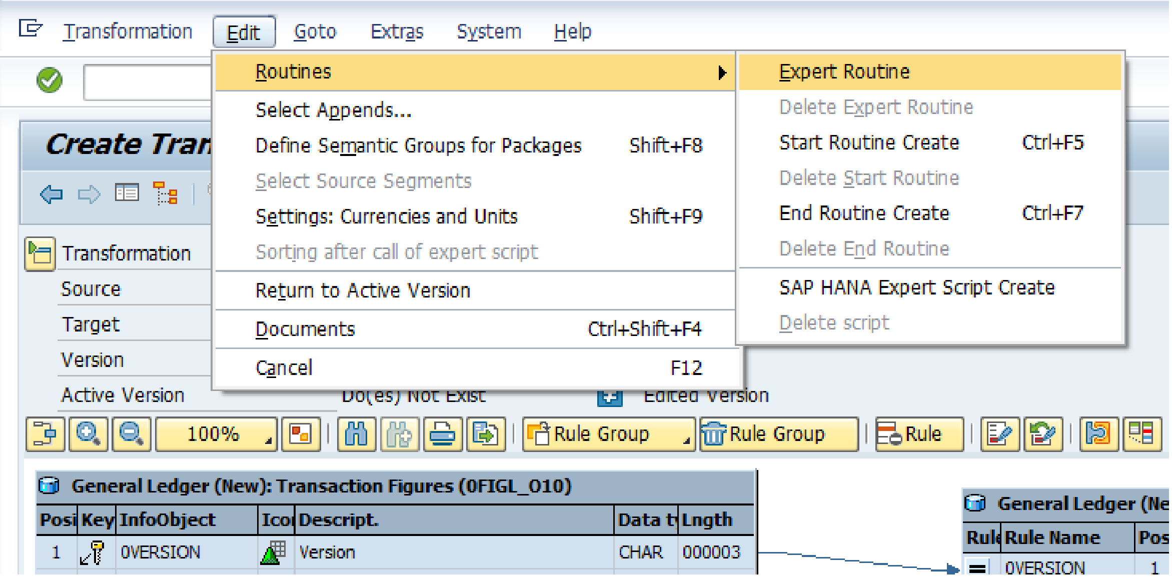
Target of the Transformation

Object Type	DataStore Object (classic)
Name	ZFIGL_13 — General Ledger (New):Transaction_...

Source of the Transformation

Object Type	DataStore Object (classic)
Name	0FIGL_O10

2. Create Expert Routine



General Ledger (New): Transaction Figures (0FIGL_O10)

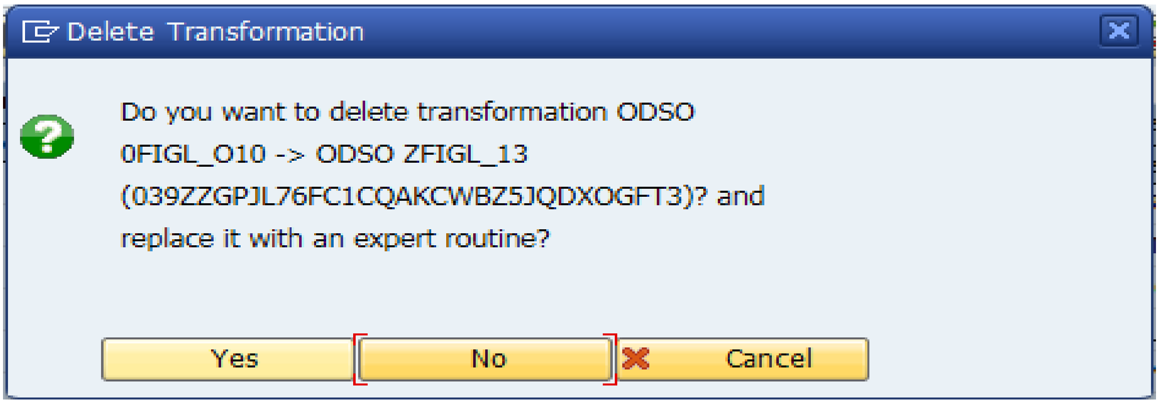
Posi	Key	InfoObject	Ico	Descript.	Data t	Lngth
1		0VERSION		Version	CHAR	000003

General Ledger (Ne

Rul	Rule Name	Pos
=	0VERSION	1

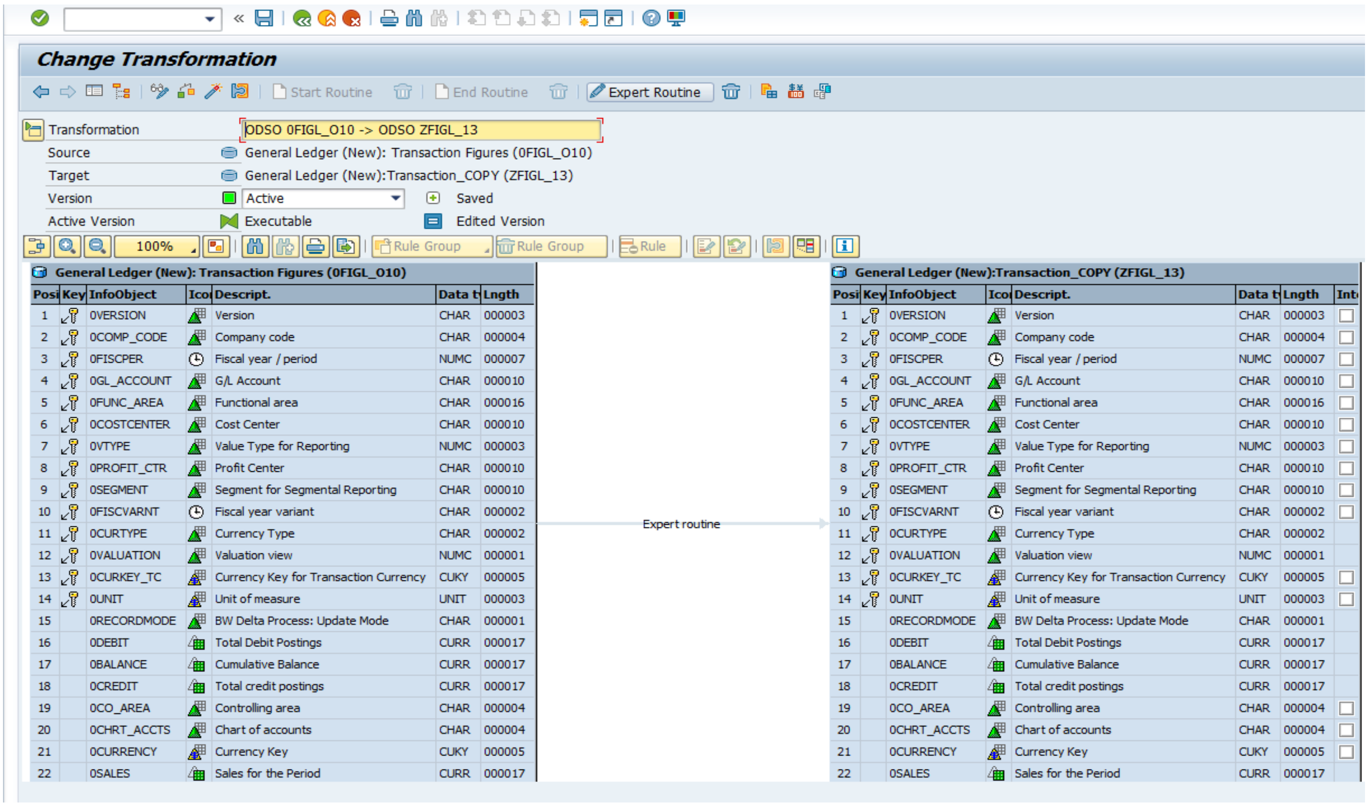
3. It will ask you for deleting transformation. Click on Yes.

If in case start and end routines were already present in the transformation then while creating expert routine they will get deleted.


Delete Transformation

Do you want to delete transformation ODSO 0FIGL_O10 -> ODSO ZFIGL_13 (039ZZGPJL76FC1CQAKCWBZ5JQDXOGFT3)? and replace it with an expert routine?

4. Click on Save, Activate and Expert routine will be created. In order to write code for expert routine click on ‘Expert Routine’ button.



Transformation	ODSO 0FIGL_O10 -> ODSO ZFIGL_13
Source	General Ledger (New): Transaction Figures (0FIGL_O10)
Target	General Ledger (New):Transaction_COPY (ZFIGL_13)
Version	Active
	Saved
Active Version	Executable
	Edited Version

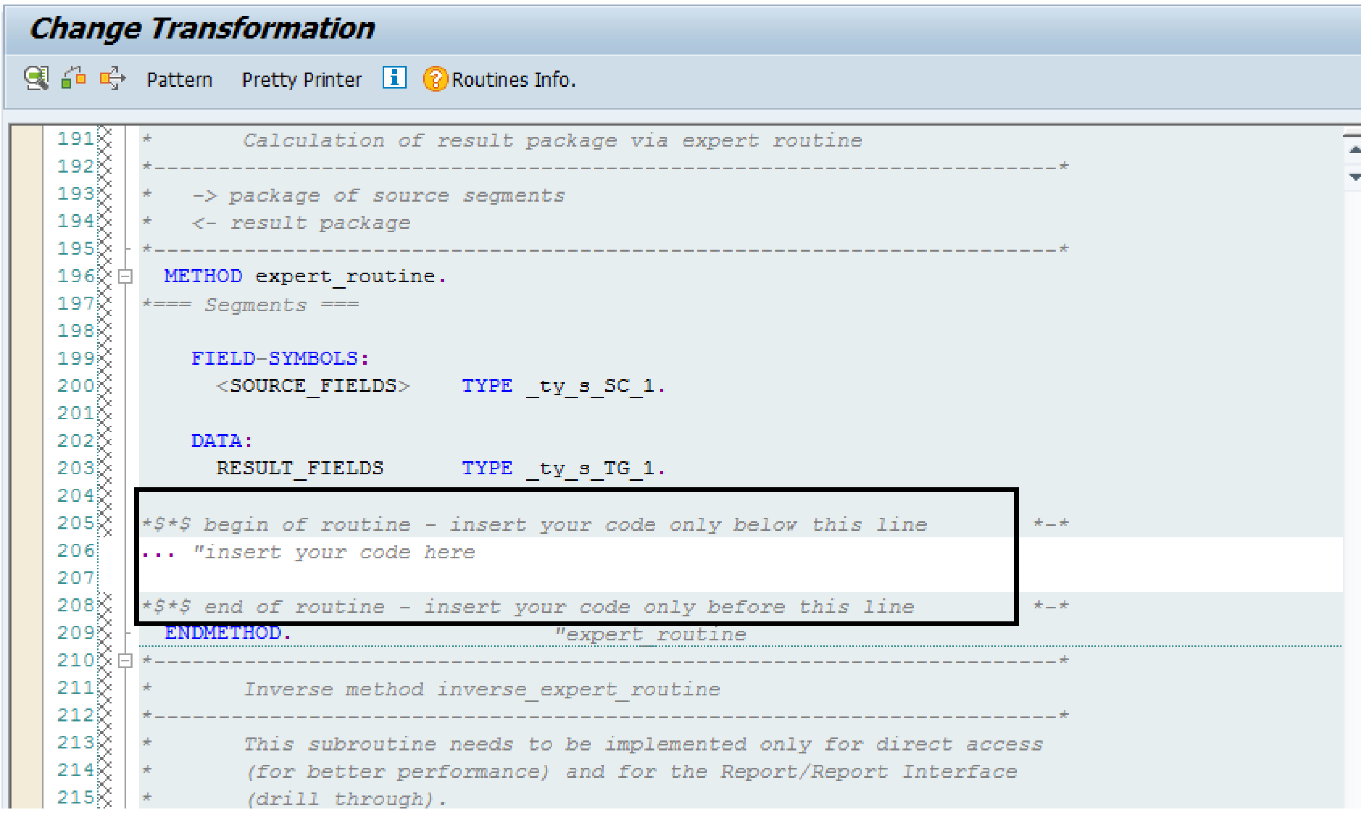
General Ledger (New): Transaction Figures (0FIGL_O10)

Posi	Key	InfoObject	Ico	Descript.	Data t	Lngth
1		0VERSION		Version	CHAR	000003
2		0COMP_CODE		Company code	CHAR	000004
3		0FISCPER		Fiscal year / period	NUMC	000007
4		0GL_ACCOUNT		G/L Account	CHAR	000010
5		0FUNC_AREA		Functional area	CHAR	000016
6		0COSTCENTER		Cost Center	CHAR	000010
7		0VTYPE		Value Type for Reporting	NUMC	000003
8		0PROFIT_CTR		Profit Center	CHAR	000010
9		0SEGMENT		Segment for Segmental Reporting	CHAR	000010
10		0FISCVARNT		Fiscal year variant	CHAR	000002
11		0CURTYPE		Currency Type	CHAR	000002
12		0VALUATION		Valuation view	NUMC	000001
13		0CURKEY_TC		Currency Key for Transaction Currency	CUKY	000005
14		0UNIT		Unit of measure	UNIT	000003
15		0RECORDMODE		BW Delta Process: Update Mode	CHAR	000001
16		0DEBIT		Total Debit Postings	CURR	000017
17		0BALANCE		Cumulative Balance	CURR	000017
18		0CREDIT		Total credit postings	CURR	000017
19		0CO_AREA		Controlling area	CHAR	000004
20		0CHRT_ACCTS		Chart of accounts	CHAR	000004
21		0CURRENCY		Currency Key	CUKY	000005
22		0SALES		Sales for the Period	CURR	000017

General Ledger (New):Transaction_COPY (ZFIGL_13)

Posi	Key	InfoObject	Ico	Descript.	Data t	Lngth	Int
1		0VERSION		Version	CHAR	000003	□
2		0COMP_CODE		Company code	CHAR	000004	□
3		0FISCPER		Fiscal year / period	NUMC	000007	□
4		0GL_ACCOUNT		G/L Account	CHAR	000010	□
5		0FUNC_AREA		Functional area	CHAR	000016	□
6		0COSTCENTER		Cost Center	CHAR	000010	□
7		0VTYPE		Value Type for Reporting	NUMC	000003	□
8		0PROFIT_CTR		Profit Center	CHAR	000010	□
9		0SEGMENT		Segment for Segmental Reporting	CHAR	000010	□
10		0FISCVARNT		Fiscal year variant	CHAR	000002	□
11		0CURTYPE		Currency Type	CHAR	000002	
12		0VALUATION		Valuation view	NUMC	000001	
13		0CURKEY_TC		Currency Key for Transaction Currency	CUKY	000005	□
14		0UNIT		Unit of measure	UNIT	000003	□
15		0RECORDMODE		BW Delta Process: Update Mode	CHAR	000001	
16		0DEBIT		Total Debit Postings	CURR	000017	
17		0BALANCE		Cumulative Balance	CURR	000017	
18		0CREDIT		Total credit postings	CURR	000017	
19		0CO_AREA		Controlling area	CHAR	000004	□
20		0CHRT_ACCTS		Chart of accounts	CHAR	000004	□
21		0CURRENCY		Currency Key	CUKY	000005	□
22		0SALES		Sales for the Period	CURR	000017	

5. Expert routine code can be written in below highlighted section



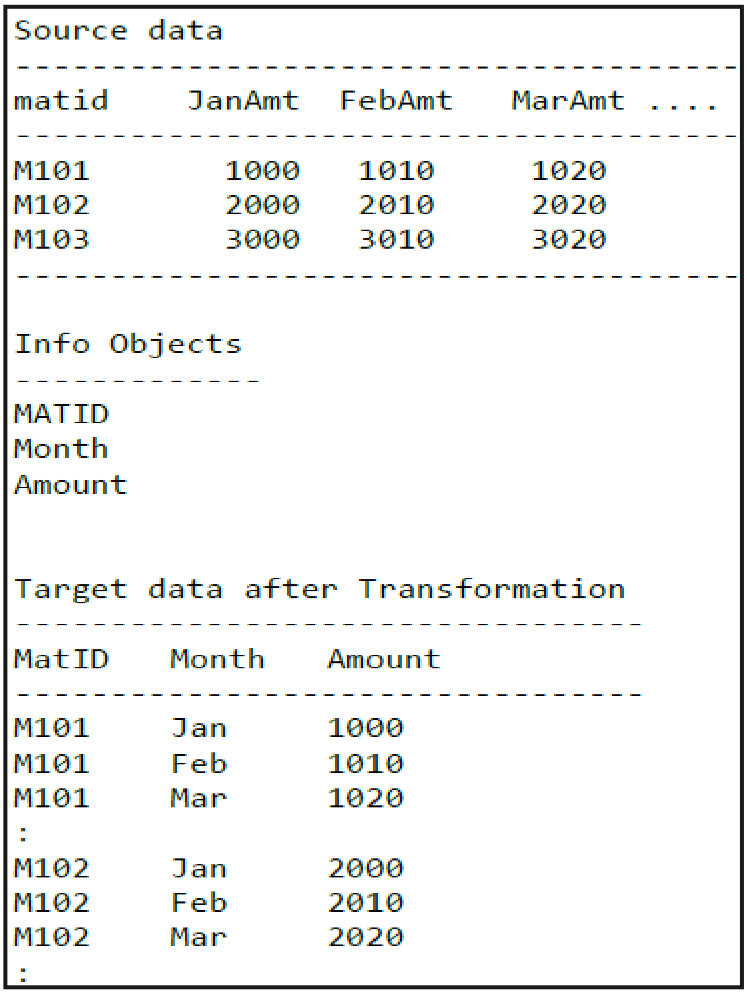
```
*       Calculation of result package via expert routine
*----------------------------------------------------------------------*
*    -> package of source segments
*    <- result package
*----------------------------------------------------------------------*
  METHOD expert_routine.
*=== Segments ===

    FIELD-SYMBOLS:
      <SOURCE_FIELDS>    TYPE _ty_s_SC_1.

    DATA:
      RESULT_FIELDS      TYPE _ty_s_TG_1.

*$*$ begin of routine - insert your code only below this line        *-*
...  "insert your code here

*$*$ end of routine - insert your code only before this line         *-*
  ENDMETHOD.                    "expert_routine
*----------------------------------------------------------------------*
*       Inverse method inverse_expert_routine
*----------------------------------------------------------------------*
*       This subroutine needs to be implemented only for direct access
*       (for better performance) and for the Report/Report Interface
*       (drill through).
```

6. Scenario on Expert Routine: Converting 1 record of source into Several records of target

   Only routine possible here is "Expert Routine"

   Logic of Expert Routine

```
----------------------
Loop at source_package assigning <source_fields>.
result_fields-/bic/matid = <source_fields>-matid.
result_fields-/bic/month = 'Jan'.
result_fields-/bic/amount = <source_fields>-JanAmt.
append result_fields to result_package.
clear result_fields.
result_fields-/bic/matid = <source_fields>-matid.
result_fields-/bic/month = 'Feb'.
result_fields-/bic/amount = <source_fields>-FebAmt.
append result_fields to result_package.
clear result_fields.
result_fields-/bic/matid = <source_fields>-matid.
result_fields-/bic/month = 'Mar'.
result_fields-/bic/amount = <source_fields>-MarAmt.
append result_fields to result_package.
clear result_fields.
....................
endloop.
```



```
Source data
--------------------------------------
matid    JanAmt  FebAmt   MarAmt ....
--------------------------------------
M101       1000   1010     1020
M102       2000   2010     2020
M103       3000   3010     3020
--------------------------------------

Info Objects
-------------
MATID
Month
Amount


Target data after Transformation
---------------------------------
MatID   Month   Amount
---------------------------------
M101    Jan     1000
M101    Feb     1010
M101    Mar     1020
:
M102    Jan     2000
M102    Feb     2010
M102    Mar     2020
:
```

User Exits and BADIs in the Extraction Process

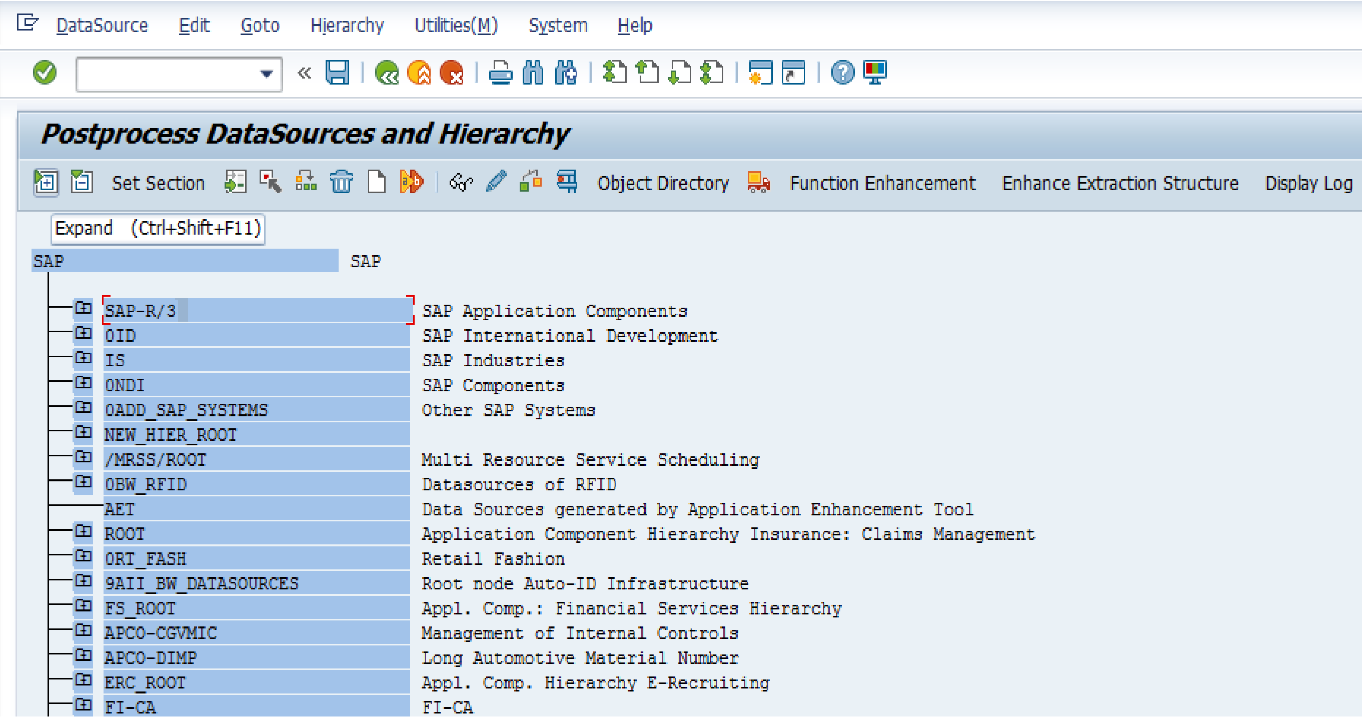
CMOD Enhancement

Example: Enhancement of Datasource 0FI_GL_14

Let us enhance FI datasource 0FI_GL_14.

Steps

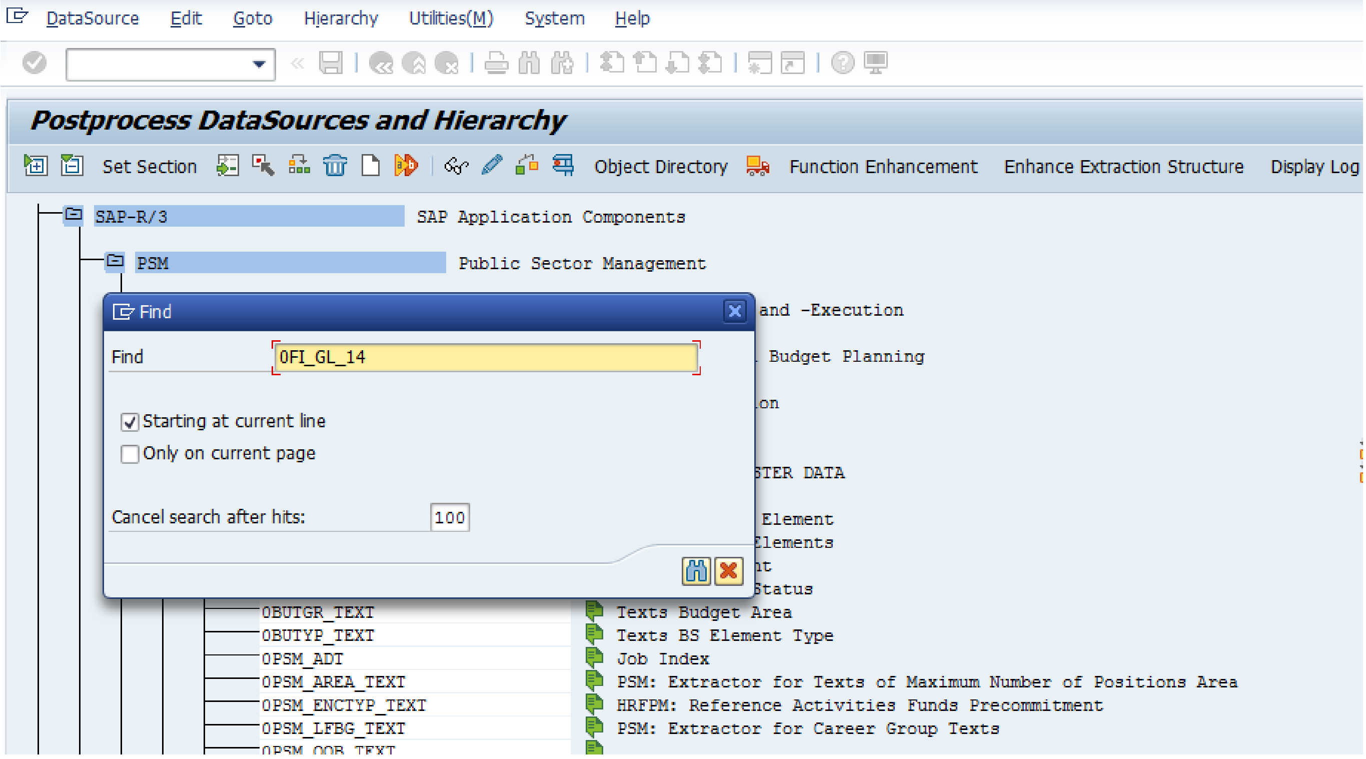
1. Go to tcode RSA6, select SAP-R/3 (SAP Application Components) and click on Expand



SAP	SAP
SAP-R/3	SAP Application Components
0ID	SAP International Development
IS	SAP Industries
0NDI	SAP Components
0ADD_SAP_SYSTEMS	Other SAP Systems
NEW_HIER_ROOT	
/MRSS/ROOT	Multi Resource Service Scheduling
0BW_RFID	Datasources of RFID
AET	Data Sources generated by Application Enhancement Tool
ROOT	Application Component Hierarchy Insurance: Claims Management
0RT_FASH	Retail Fashion
9AII_BW_DATASOURCES	Root node Auto-ID Infrastructure
FS_ROOT	Appl. Comp.: Financial Services Hierarchy
APCO-CGVMIC	Management of Internal Controls
APCO-DIMP	Long Automotive Material Number
ERC_ROOT	Appl. Comp. Hierarchy E-Recruiting
FI-CA	FI-CA

2. Click on search icon and below search screen will pop-up. Put the name of the datasource 0FI_GL_14 and click on search.

   It will take you to the next screen.



Postprocess DataSources and Hierarchy

```
SAP-R/3            SAP Application Components
  PSM              Public Sector Management
                   and -Execution
                   Budget Planning
                   on
                   STER DATA
                   Element
                   Elements
                   nt
                   Status
      0BUTGR_TEXT        Texts Budget Area
      0BUTYP_TEXT        Texts BS Element Type
      0PSM_ADT           Job Index
      0PSM_AREA_TEXT     PSM: Extractor for Texts of Maximum Number of Positions Area
      0PSM_ENCTYP_TEXT   HRFPM: Reference Activities Funds Precommitment
      0PSM_LFBG_TEXT     PSM: Extractor for Career Group Texts
      0PSM_OOB_TEXT
```

Find

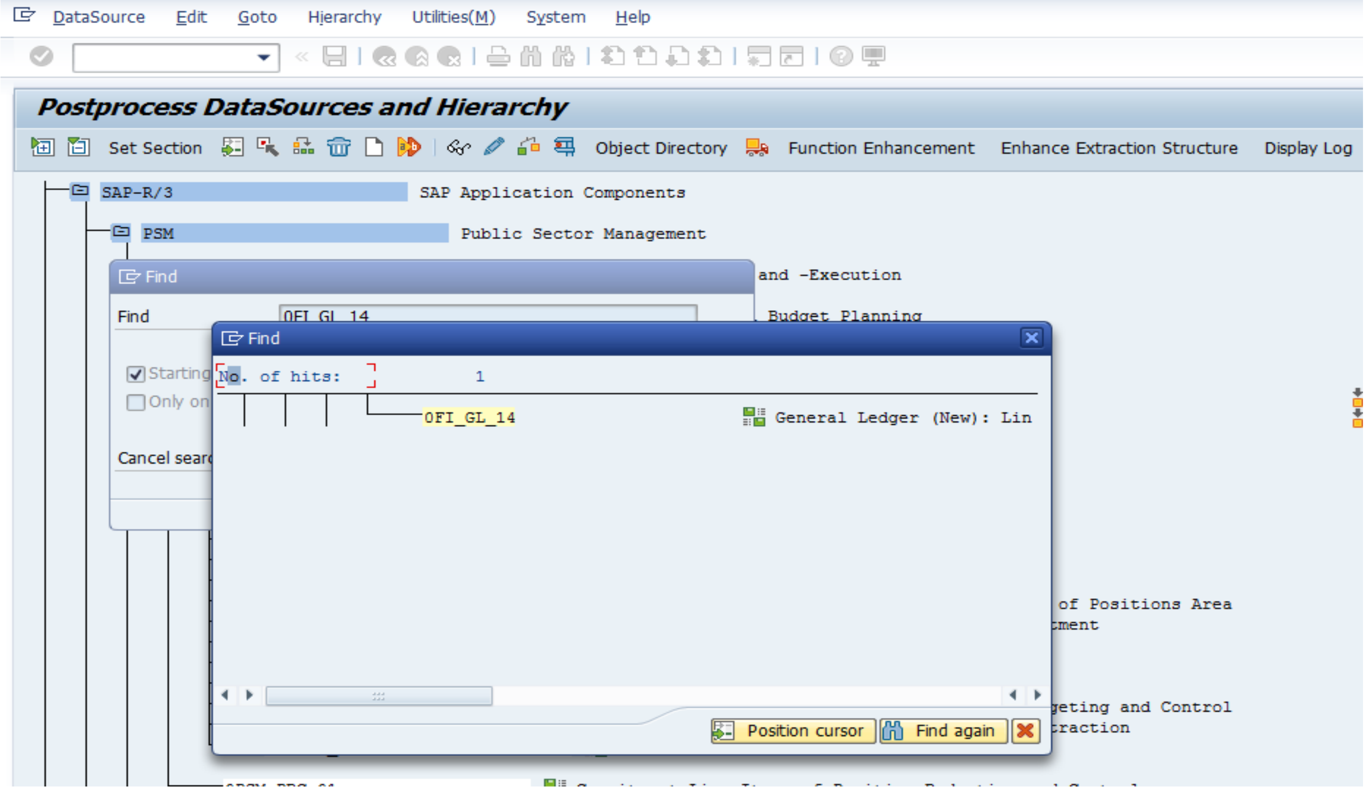
Find: 0FI_GL_14

☑ Starting at current line

☐ Only on current page

Cancel search after hits: 100

3. Select and click on the datasource 0FI_GL_14 in below screen. It will take you to the next screen



Postprocess DataSources and Hierarchy

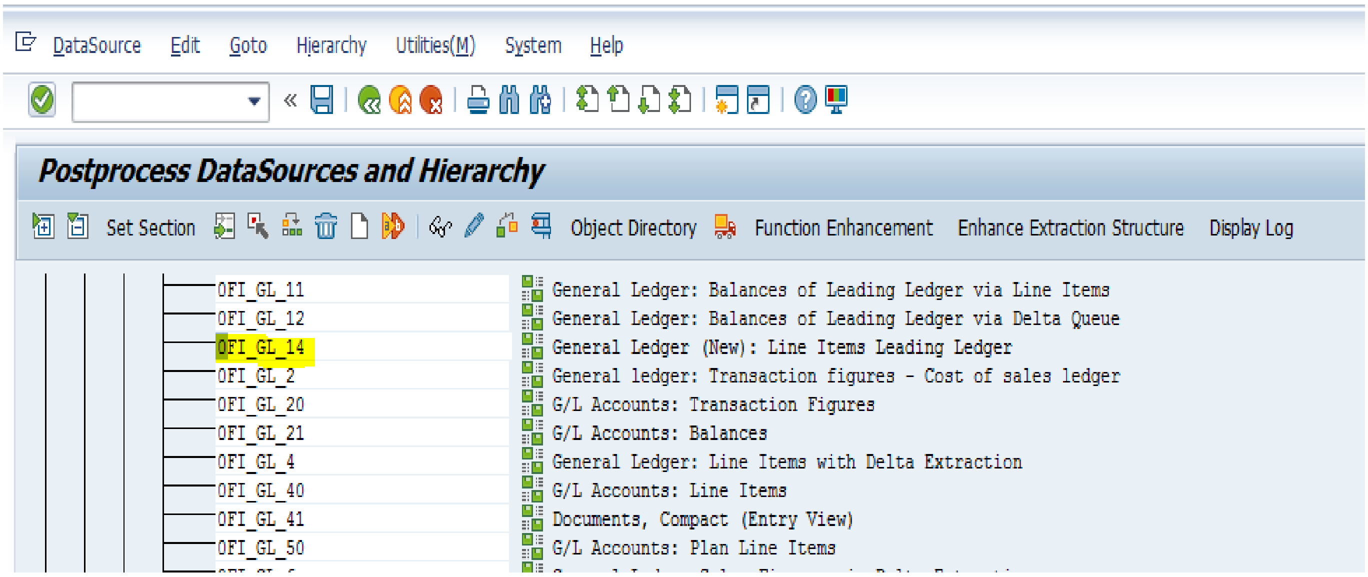
```
SAP-R/3            SAP Application Components
  PSM              Public Sector Management
                   and -Execution
                   Budget Planning
```

Find

No. of hits: 1

```
0FI_GL_14                      General Ledger (New): Lin
```

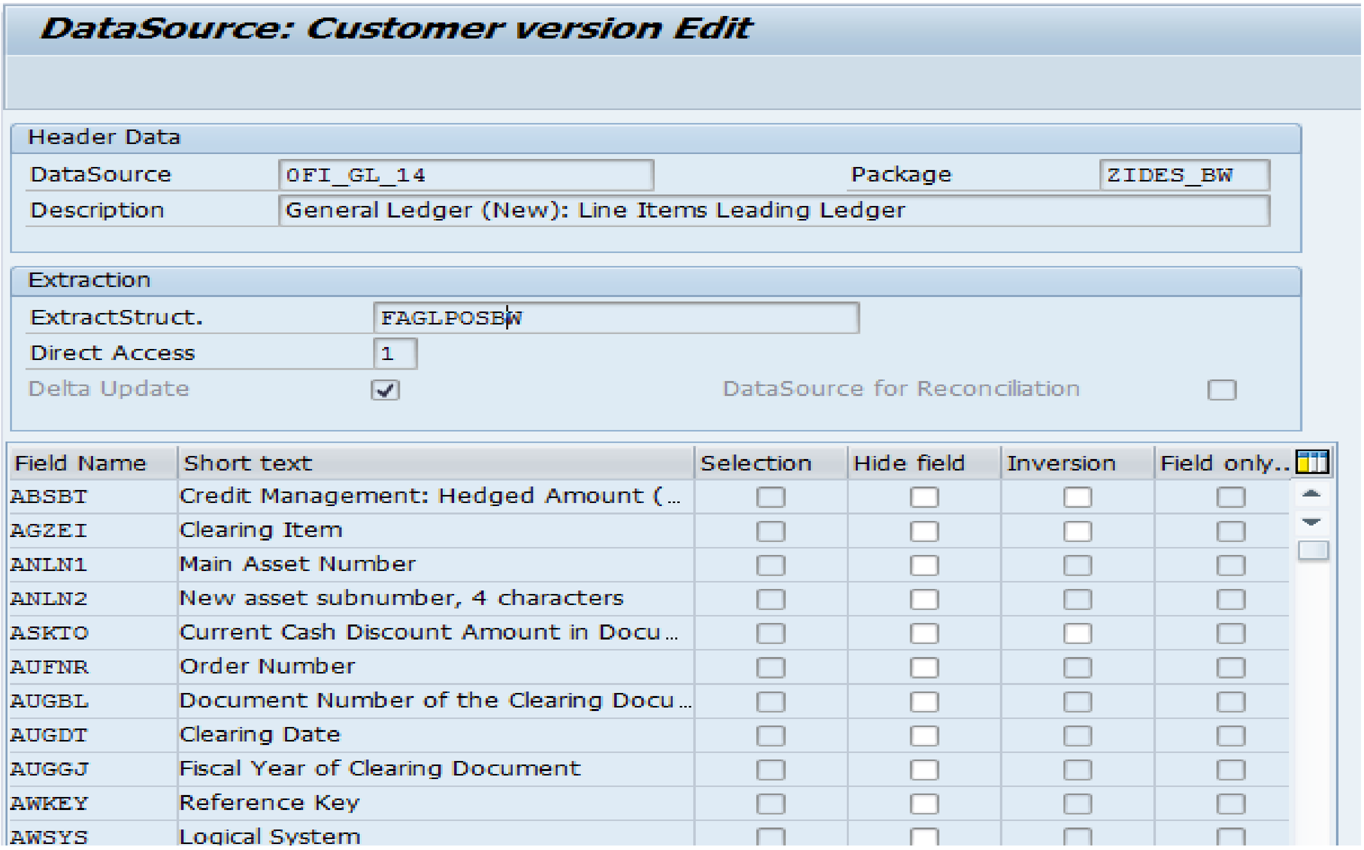
4. Select datasource 0FI_GL_14 and click on change button. It will take you to the next screen.



Postprocess DataSources and Hierarchy

0FI_GL_11	General Ledger: Balances of Leading Ledger via Line Items
0FI_GL_12	General Ledger: Balances of Leading Ledger via Delta Queue
0FI_GL_14	General Ledger (New): Line Items Leading Ledger
0FI_GL_2	General ledger: Transaction figures - Cost of sales ledger
0FI_GL_20	G/L Accounts: Transaction Figures
0FI_GL_21	G/L Accounts: Balances
0FI_GL_4	General Ledger: Line Items with Delta Extraction
0FI_GL_40	G/L Accounts: Line Items
0FI_GL_41	Documents, Compact (Entry View)
0FI_GL_50	G/L Accounts: Plan Line Items

5. Double click on the extract structure and it will take you to the extract structure screen



DataSource: Customer version Edit

Header Data

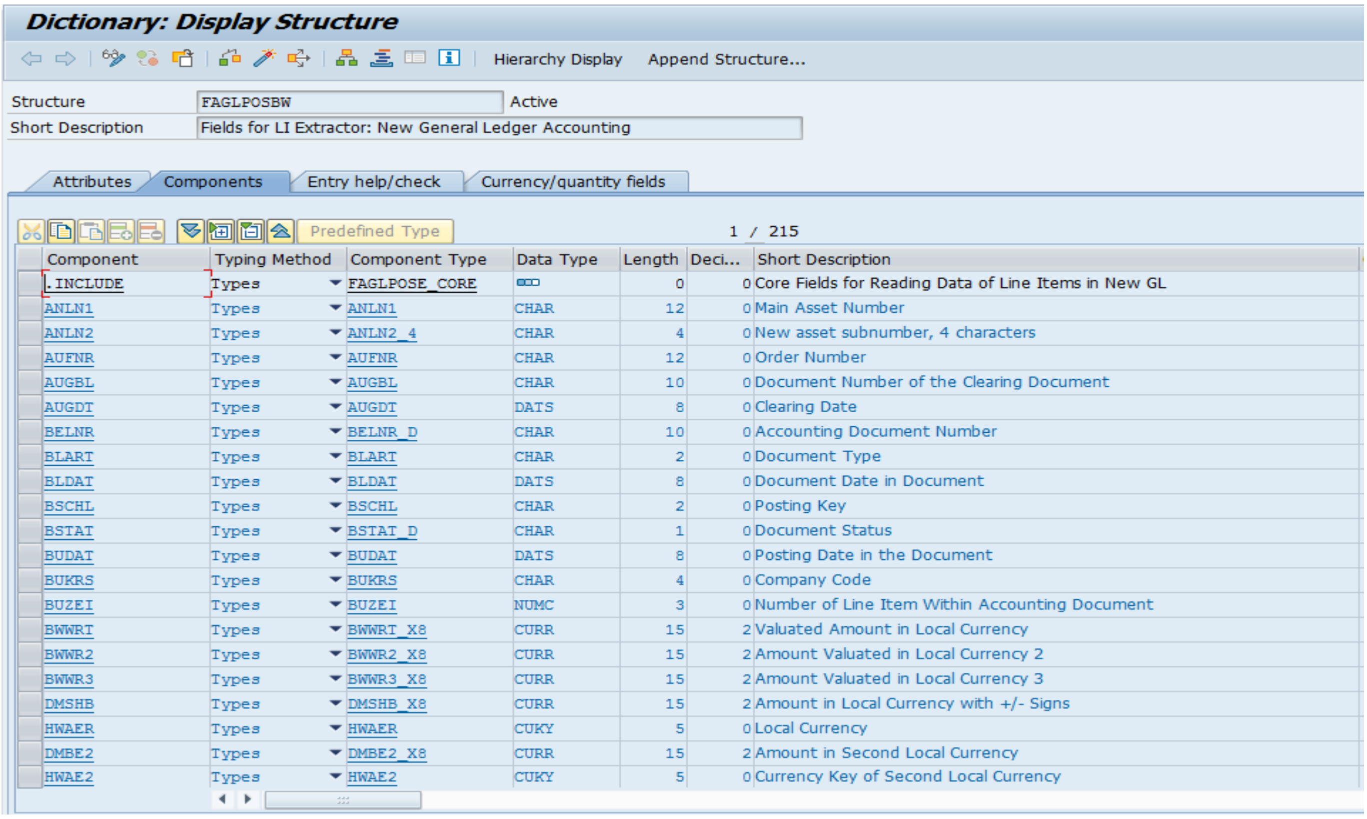
DataSource	0FI_GL_14	Package	ZIDES_BW
Description	General Ledger (New): Line Items Leading Ledger		

Extraction

ExtractStruct.	FAGLPOSBW		
Direct Access	1		
Delta Update	☑	DataSource for Reconciliation	□

Field Name	Short text	Selection	Hide field	Inversion	Field only..
ABSBT	Credit Management: Hedged Amount (...	□	□	□	□
AGZEI	Clearing Item	□	□	□	□
ANLN1	Main Asset Number	□	□	□	□
ANLN2	New asset subnumber, 4 characters	□	□	□	□
ASKTO	Current Cash Discount Amount in Docu...	□	□	□	□
AUFNR	Order Number	□	□	□	□
AUGBL	Document Number of the Clearing Docu...	□	□	□	□
AUGDT	Clearing Date	□	□	□	□
AUGGJ	Fiscal Year of Clearing Document	□	□	□	□
AWKEY	Reference Key	□	□	□	□
AWSYS	Logical System	□	□	□	□

6. Alternative we can enhance datasource from RSA2 as well and reach to the below screen which is shown in next few screens.



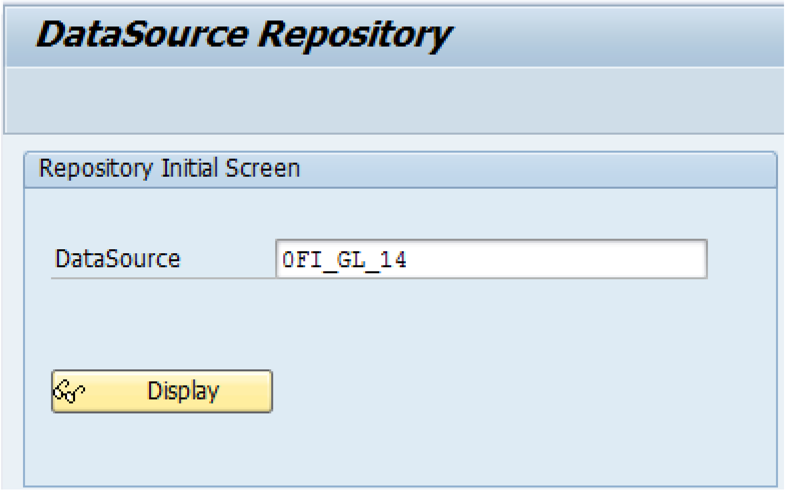
Dictionary: Display Structure

Structure FAGLPOSBW Active

Short Description Fields for LI Extractor: New General Ledger Accounting

Component	Typing Method	Component Type	Data Type	Length	Deci...	Short Description
.INCLUDE	Types	FAGLPOSE_CORE		0	0	Core Fields for Reading Data of Line Items in New GL
ANLN1	Types	ANLN1	CHAR	12	0	Main Asset Number
ANLN2	Types	ANLN2_4	CHAR	4	0	New asset subnumber, 4 characters
AUFNR	Types	AUFNR	CHAR	12	0	Order Number
AUGBL	Types	AUGBL	CHAR	10	0	Document Number of the Clearing Document
AUGDT	Types	AUGDT	DATS	8	0	Clearing Date
BELNR	Types	BELNR_D	CHAR	10	0	Accounting Document Number
BLART	Types	BLART	CHAR	2	0	Document Type
BLDAT	Types	BLDAT	DATS	8	0	Document Date in Document
BSCHL	Types	BSCHL	CHAR	2	0	Posting Key
BSTAT	Types	BSTAT_D	CHAR	1	0	Document Status
BUDAT	Types	BUDAT	DATS	8	0	Posting Date in the Document
BUKRS	Types	BUKRS	CHAR	4	0	Company Code
BUZEI	Types	BUZEI	NUMC	3	0	Number of Line Item Within Accounting Document
BWWRT	Types	BWWRT_X8	CURR	15	2	Valuated Amount in Local Currency
BWWR2	Types	BWWR2_X8	CURR	15	2	Amount Valuated in Local Currency 2
BWWR3	Types	BWWR3_X8	CURR	15	2	Amount Valuated in Local Currency 3
DMSHB	Types	DMSHB_X8	CURR	15	2	Amount in Local Currency with +/- Signs
HWAER	Types	HWAER	CUKY	5	0	Local Currency
DMBE2	Types	DMBE2_X8	CURR	15	2	Amount in Second Local Currency
HWAE2	Types	HWAE2	CUKY	5	0	Currency Key of Second Local Currency

7. Go to tcode RSA2 and enter the name of the datasource to be enhanced and then click on display button

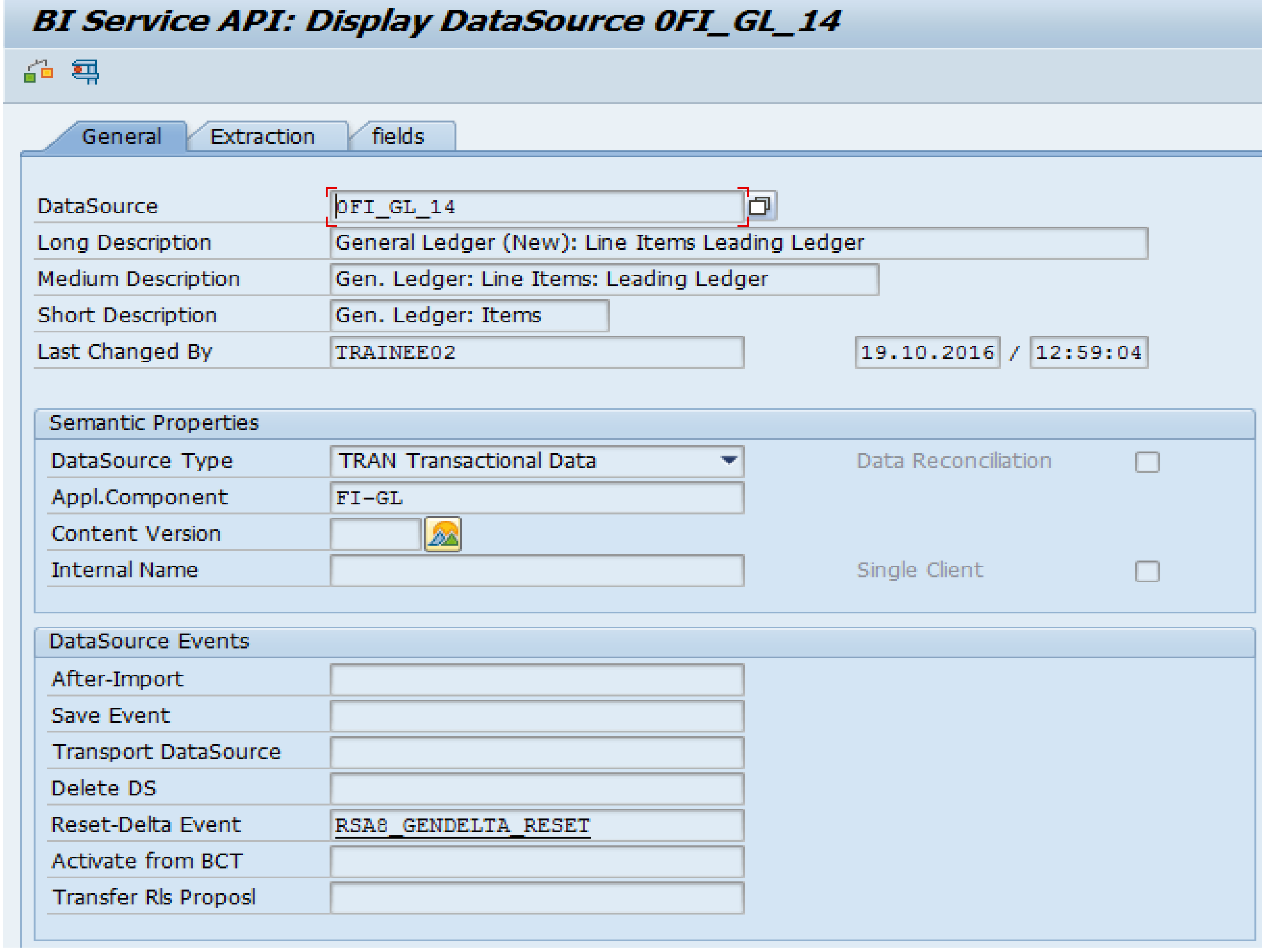


DataSource Repository

Repository Initial Screen

DataSource 0FI_GL_14

8. Click on the extraction tab



BI Service API: Display DataSource 0FI_GL_14

DataSource	0FI_GL_14
Long Description	General Ledger (New): Line Items Leading Ledger
Medium Description	Gen. Ledger: Line Items: Leading Ledger
Short Description	Gen. Ledger: Items
Last Changed By	TRAINEE02 19.10.2016 / 12:59:04

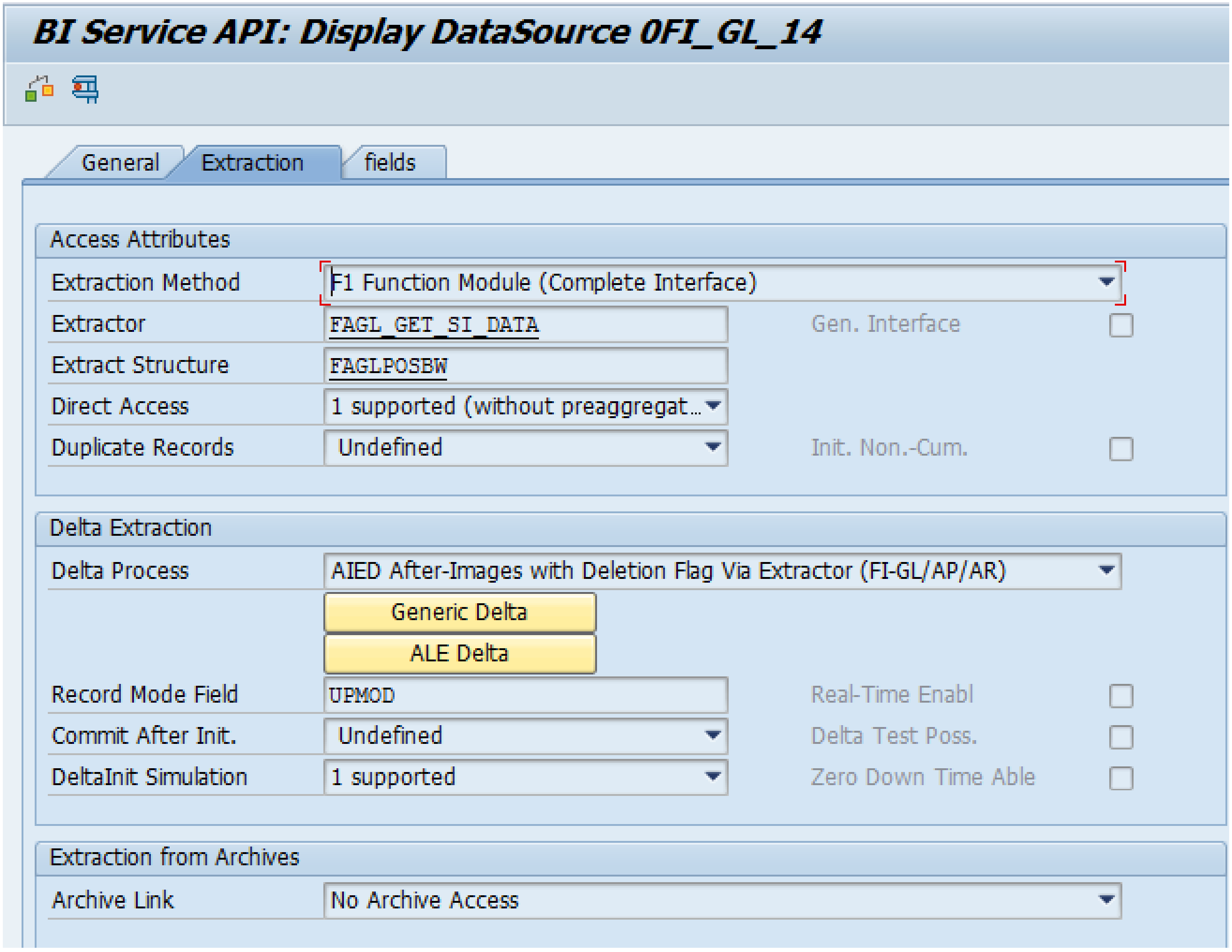
Semantic Properties

DataSource Type	TRAN Transactional Data	Data Reconciliation	□
Appl.Component	FI-GL		
Content Version			
Internal Name		Single Client	□

DataSource Events

After-Import	
Save Event	
Transport DataSource	
Delete DS	
Reset-Delta Event	RSA8_GENDELTA_RESET
Activate from BCT	
Transfer Rls Proposl	

9. Select extract structure FAGLPOSBW and double click on it. It will take you to the extract structure screen.



BI Service API: Display DataSource 0FI_GL_14

Access Attributes

Extraction Method	F1 Function Module (Complete Interface)		
Extractor	FAGL_GET_SI_DATA	Gen. Interface	□
Extract Structure	FAGLPOSBW		
Direct Access	1 supported (without preaggregat...		
Duplicate Records	Undefined	Init. Non.-Cum.	□

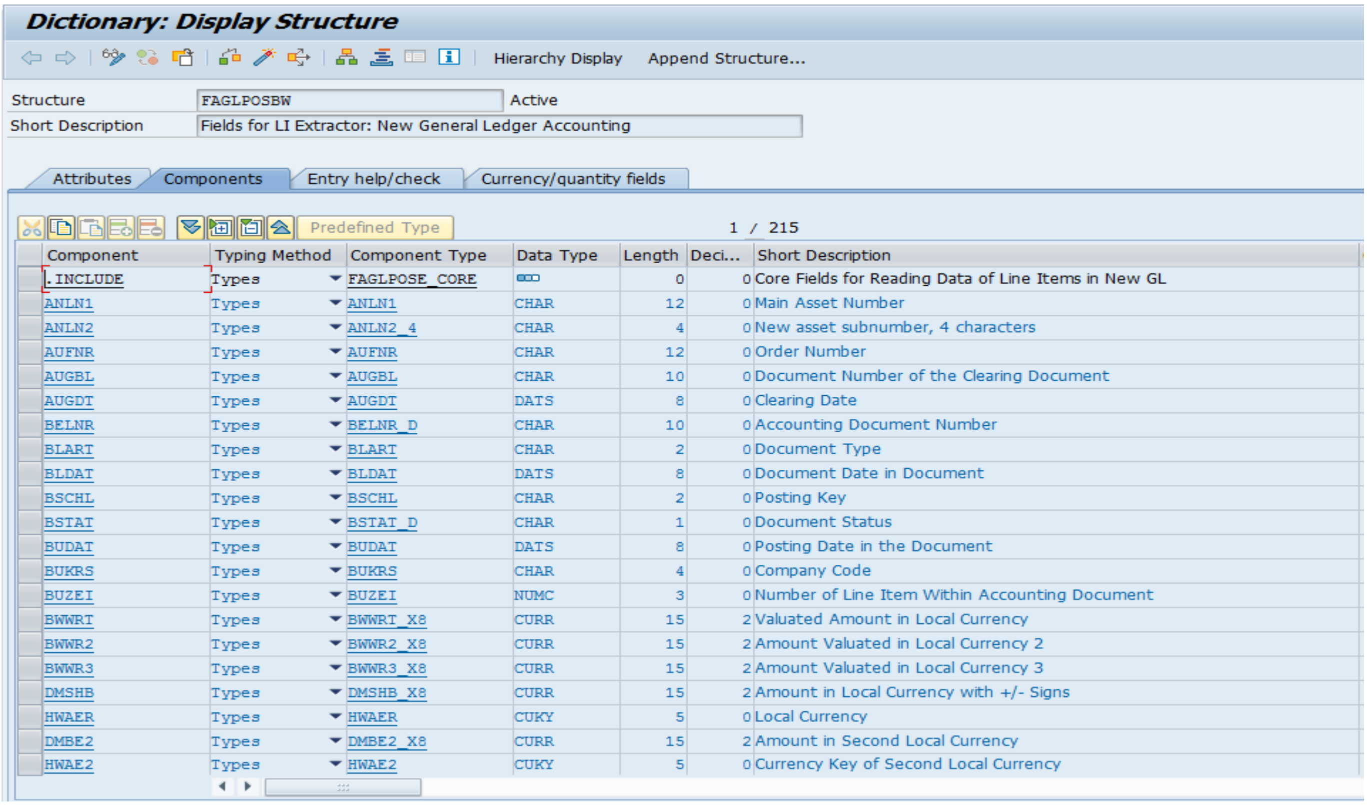
Delta Extraction

Delta Process	AIED After-Images with Deletion Flag Via Extractor (FI-GL/AP/AR)		
	Generic Delta		
	ALE Delta		
Record Mode Field	UPMOD	Real-Time Enabl	□
Commit After Init.	Undefined	Delta Test Poss.	□
DeltaInit Simulation	1 supported	Zero Down Time Able	□

Extraction from Archives

Archive Link	No Archive Access

10. This is the same screen which we have seen from RSA6. Click on Append Structure in order to append new fields in the extract structure



Dictionary: Display Structure

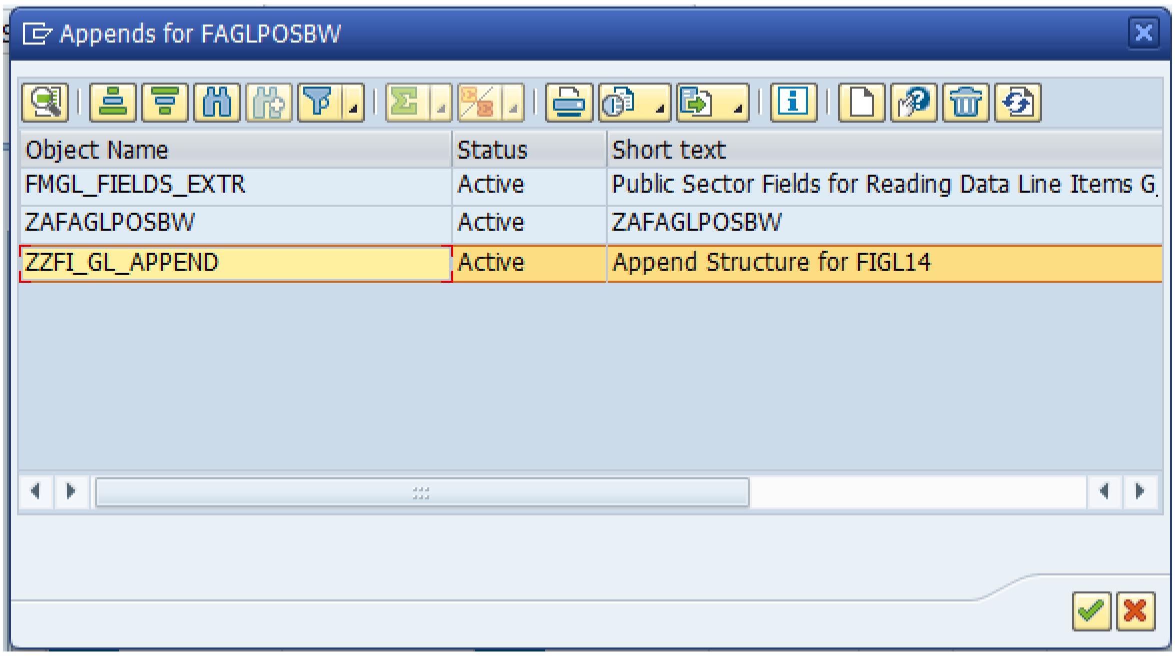
Structure FAGLPOSBW Active

Short Description Fields for LI Extractor: New General Ledger Accounting

1 / 215

Component	Typing Method	Component Type	Data Type	Length	Deci...	Short Description
.INCLUDE	Types	FAGLPOSE_CORE		0	0	Core Fields for Reading Data of Line Items in New GL
ANLN1	Types	ANLN1	CHAR	12	0	Main Asset Number
ANLN2	Types	ANLN2_4	CHAR	4	0	New asset subnumber, 4 characters
AUFNR	Types	AUFNR	CHAR	12	0	Order Number
AUGBL	Types	AUGBL	CHAR	10	0	Document Number of the Clearing Document
AUGDT	Types	AUGDT	DATS	8	0	Clearing Date
BELNR	Types	BELNR_D	CHAR	10	0	Accounting Document Number
BLART	Types	BLART	CHAR	2	0	Document Type
BLDAT	Types	BLDAT	DATS	8	0	Document Date in Document
BSCHL	Types	BSCHL	CHAR	2	0	Posting Key
BSTAT	Types	BSTAT_D	CHAR	1	0	Document Status
BUDAT	Types	BUDAT	DATS	8	0	Posting Date in the Document
BUKRS	Types	BUKRS	CHAR	4	0	Company Code
BUZEI	Types	BUZEI	NUMC	3	0	Number of Line Item Within Accounting Document
BWWRT	Types	BWWRT_X8	CURR	15	2	Valuated Amount in Local Currency
BWWR2	Types	BWWR2_X8	CURR	15	2	Amount Valuated in Local Currency 2
BWWR3	Types	BWWR3_X8	CURR	15	2	Amount Valuated in Local Currency 3
DMSHB	Types	DMSHB_X8	CURR	15	2	Amount in Local Currency with +/- Signs
HWAER	Types	HWAER	CUKY	5	0	Local Currency
DMBE2	Types	DMBE2_X8	CURR	15	2	Amount in Second Local Currency
HWAE2	Types	HWAE2	CUKY	5	0	Currency Key of Second Local Currency

11. Select Append structure ZZFI_GL_APPEND and double click on it. It will take you to the next screen



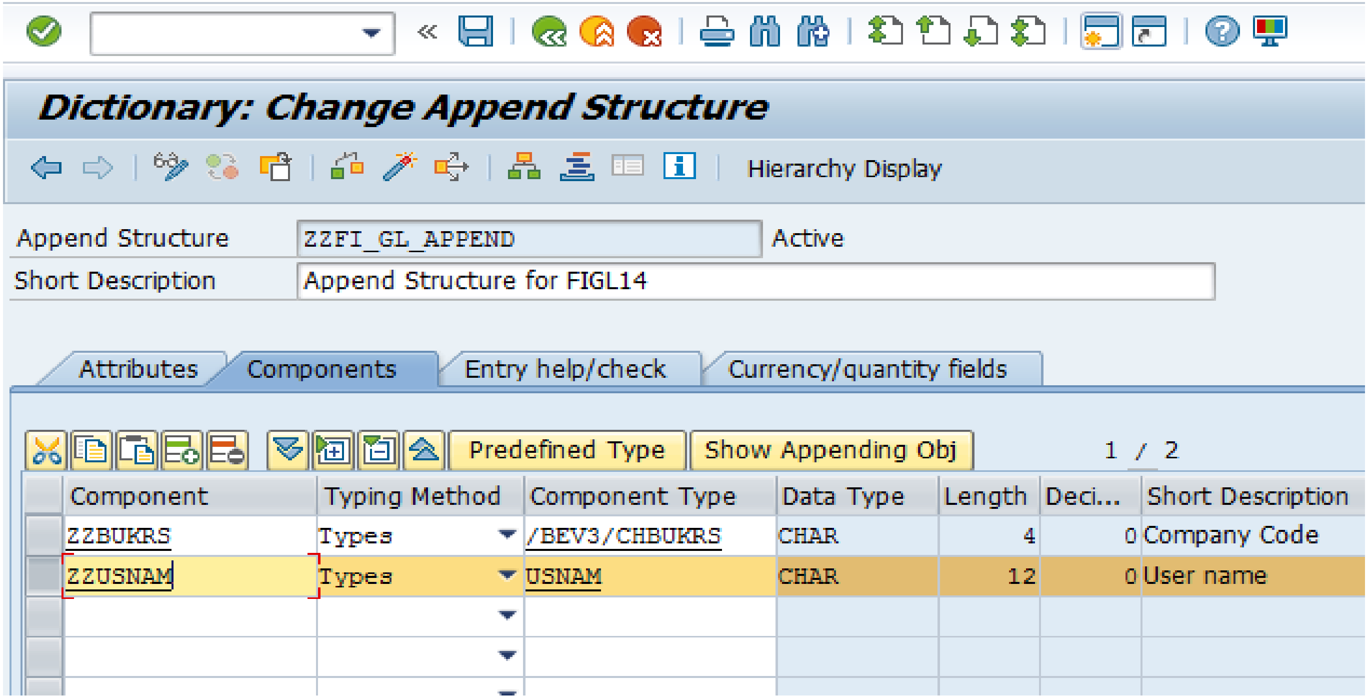
Appends for FAGLPOSBW

Object Name	Status	Short text
FMGL_FIELDS_EXTR	Active	Public Sector Fields for Reading Data Line Items G
ZAFAGLPOSBW	Active	ZAFAGLPOSBW
ZZFI_GL_APPEND	Active	Append Structure for FIGL14

12. Add new field ZZUSNAM under component column with USNAM as it’s component type.

Click on Save, check and activate buttons.

In the last, click on back button to go back to the extract structure screen.



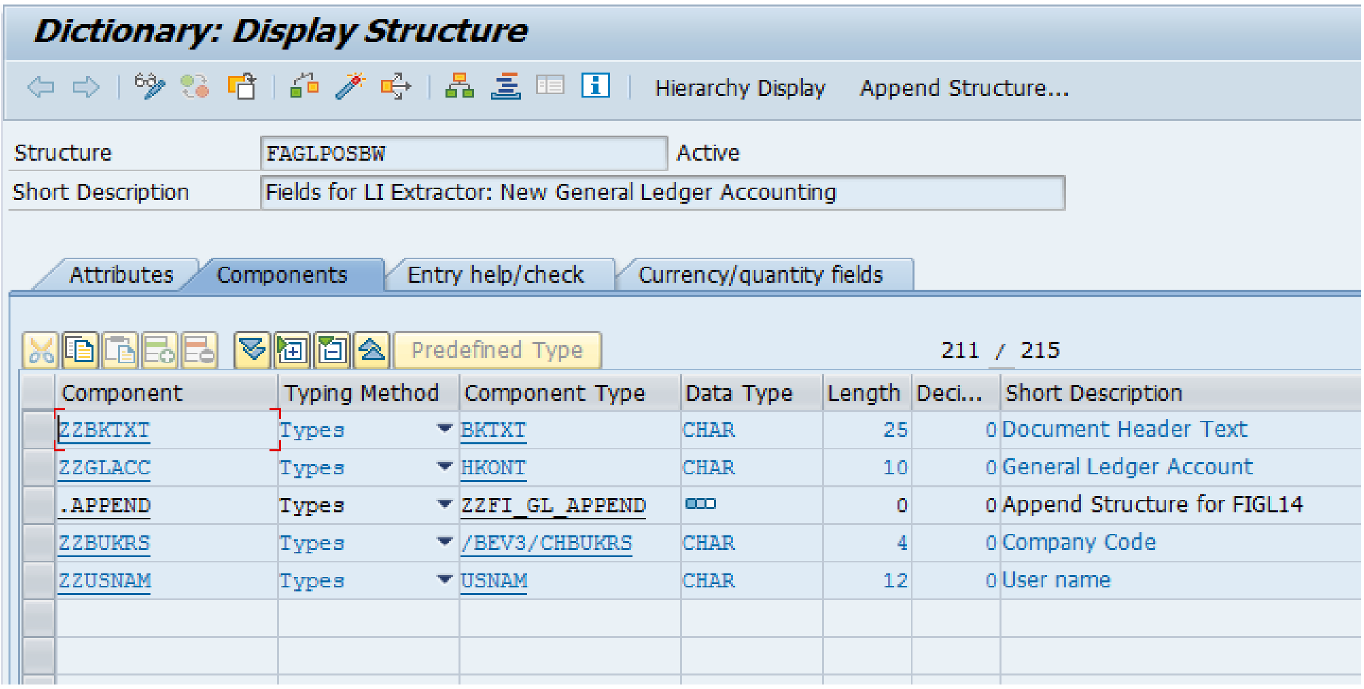
Dictionary: Change Append Structure

Append Structure	ZZFI_GL_APPEND	Active
Short Description	Append Structure for FIGL14	

1 / 2

Component	Typing Method	Component Type	Data Type	Length	Deci...	Short Description
ZZBUKRS	Types	/BEV3/CHBUKRS	CHAR	4	0	Company Code
ZZUSNAM	Types	USNAM	CHAR	12	0	User name

13. Newly added field ZZUSNAM would be appended to extract structure FAGLPOSBW



Dictionary: Display Structure

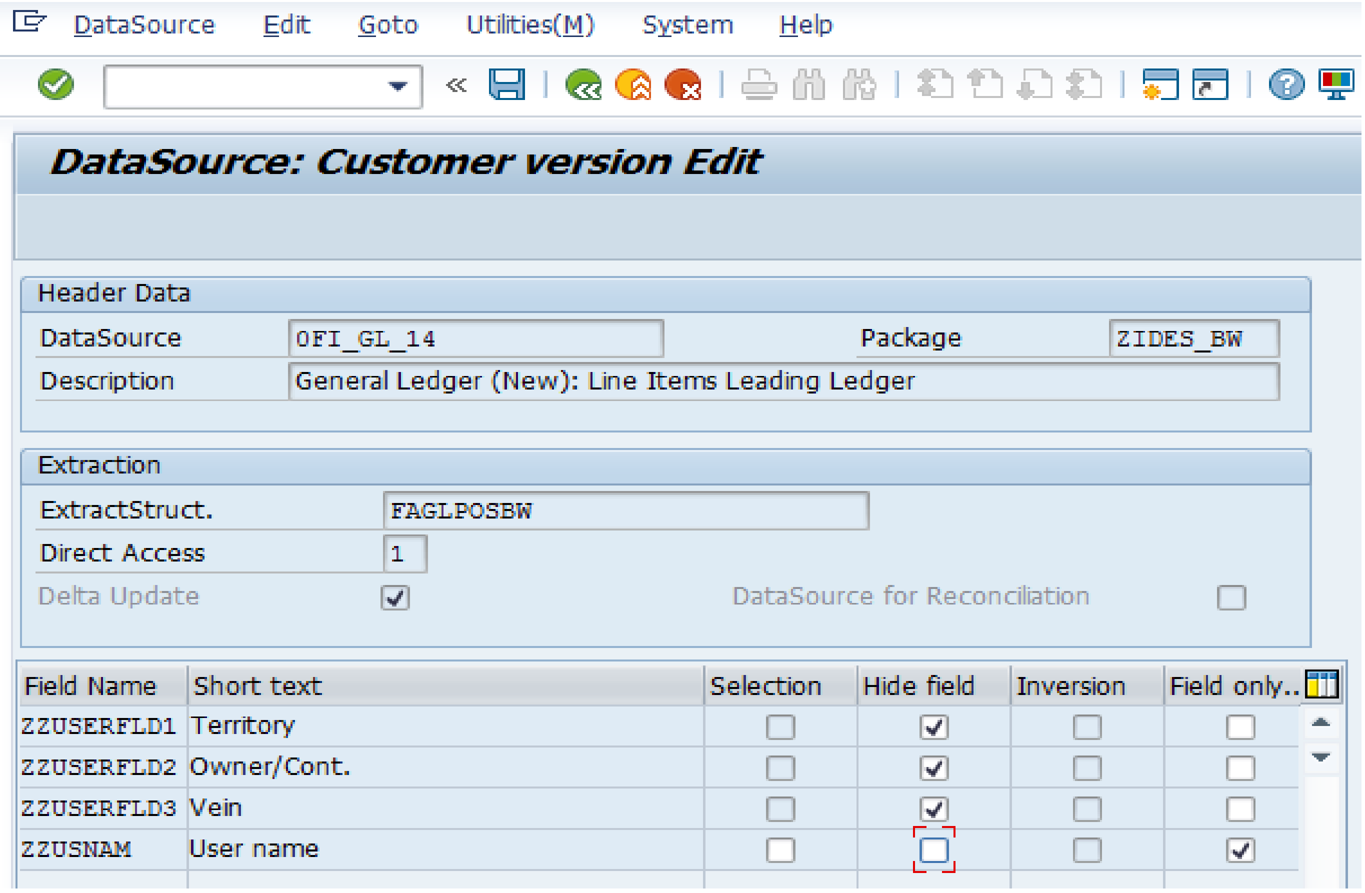
Structure	FAGLPOSBW	Active
Short Description	Fields for LI Extractor: New General Ledger Accounting	

211 / 215

Component	Typing Method	Component Type	Data Type	Length	Deci...	Short Description
ZZBKTXT	Types	BKTXT	CHAR	25	0	Document Header Text
ZZGLACC	Types	HKONT	CHAR	10	0	General Ledger Account
.APPEND	Types	ZZFI_GL_APPEND		0	0	Append Structure for FIGL14
ZZBUKRS	Types	/BEV3/CHBUKRS	CHAR	4	0	Company Code
ZZUSNAM	Types	USNAM	CHAR	12	0	User name

14. Go to RSA6, find the enhanced datasource (0FI_GL_14) and then open it in change mode.

    Initially the newly added field ZZUSNAM would be hidden mode. Uncheck the ‘Hide field’ checkbox in order to unhide the newly appended field in the datasource and click on Save button.



DataSource: Customer version Edit

Header Data

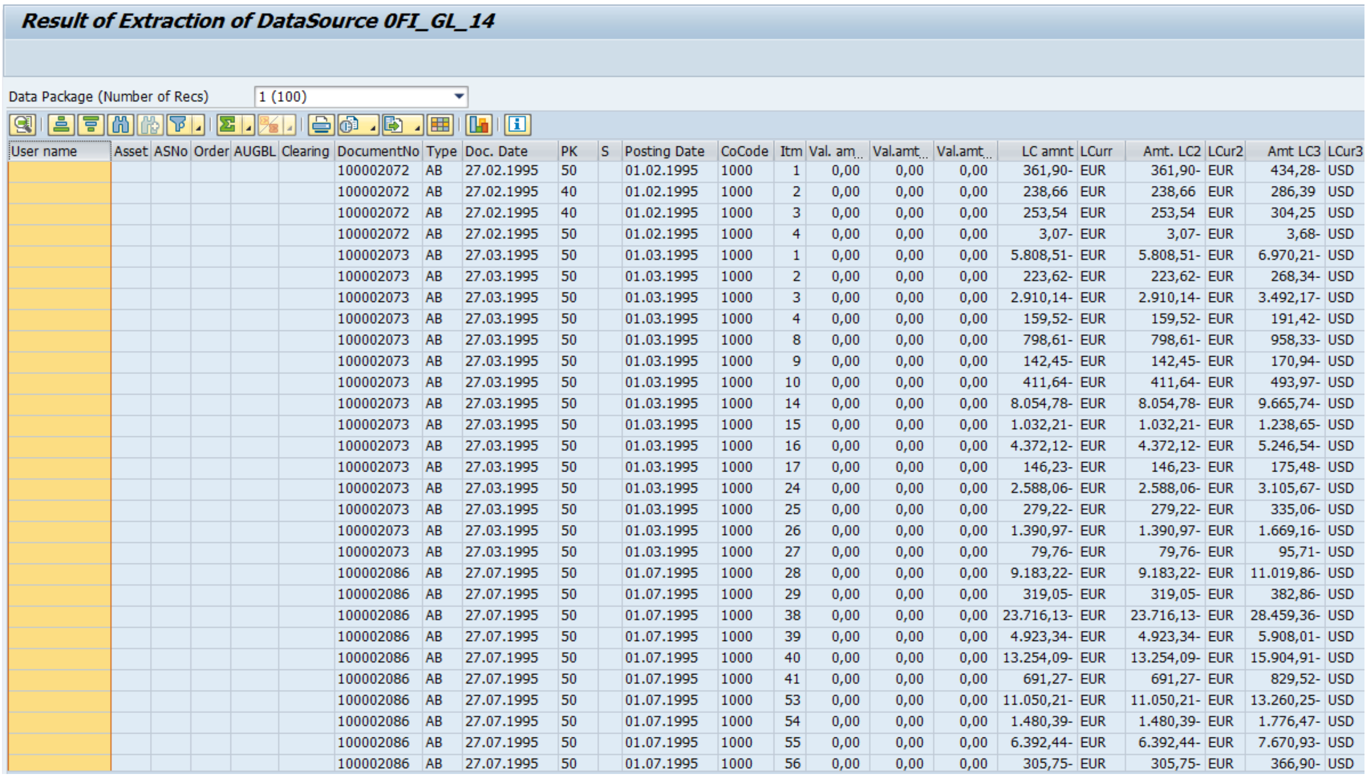
DataSource	0FI_GL_14	Package	ZIDES_BW
Description	General Ledger (New): Line Items Leading Ledger		

Extraction

ExtractStruct.	FAGLPOSBW		
Direct Access	1		
Delta Update	☑	DataSource for Reconciliation	□

Field Name	Short text	Selection	Hide field	Inversion	Field only..
ZZUSERFLD1	Territory	□	☑	□	□
ZZUSERFLD2	Owner/Cont.	□	☑	□	□
ZZUSERFLD3	Vein	□	☑	□	□
ZZUSNAM	User name	□	□	□	☑

15. Go to RSA3 to see the datasource output. Since enhancement code is not written in CMOD until now, newly added field ZZUSNAM (User Name) is blank. Let us write enhancement code in CMOD in order to populate this field in the datasource. Please check further slides for the same

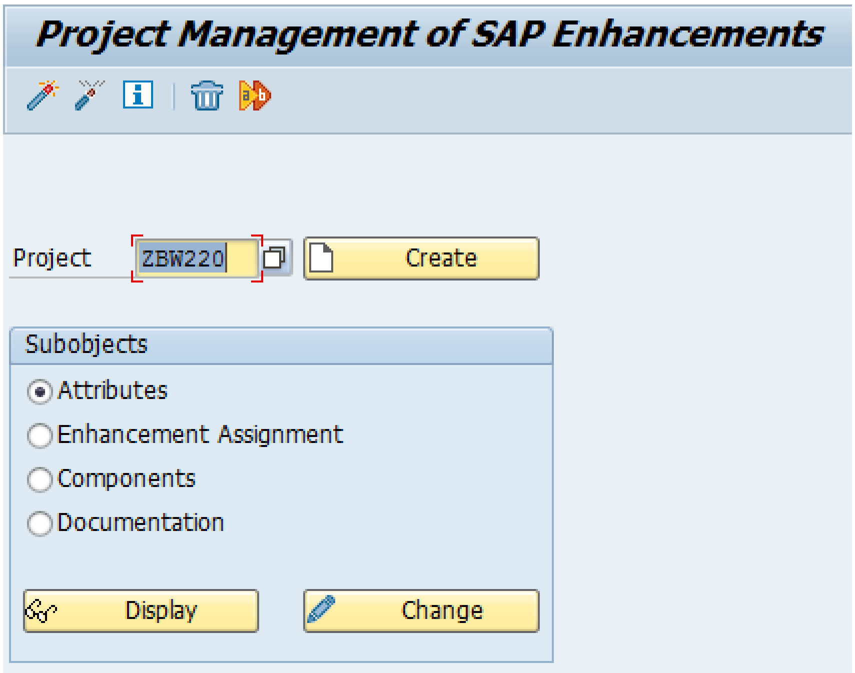


Result of Extraction of DataSource 0FI_GL_14

Data Package (Number of Recs) 1 (100)

User name	Asset	ASNo	Order	AUGBL	Clearing	DocumentNo	Type	Doc. Date	PK	S	Posting Date	CoCode	Itm	Val. am...	Val.amt...	Val.amt...	LC amnt	LCurr	Amt. LC2	LCur2	Amt LC3	LCur3
						100002072	AB	27.02.1995	50		01.02.1995	1000	1	0,00	0,00	0,00	361,90-	EUR	361,90-	EUR	434,28-	USD
						100002072	AB	27.02.1995	40		01.02.1995	1000	2	0,00	0,00	0,00	238,66	EUR	238,66	EUR	286,39	USD
						100002072	AB	27.02.1995	40		01.02.1995	1000	3	0,00	0,00	0,00	253,54	EUR	253,54	EUR	304,25	USD
						100002072	AB	27.02.1995	50		01.02.1995	1000	4	0,00	0,00	0,00	3,07-	EUR	3,07-	EUR	3,68-	USD
						100002073	AB	27.03.1995	50		01.03.1995	1000	1	0,00	0,00	0,00	5.808,51-	EUR	5.808,51-	EUR	6.970,21-	USD
						100002073	AB	27.03.1995	50		01.03.1995	1000	2	0,00	0,00	0,00	223,62-	EUR	223,62-	EUR	268,34-	USD
						100002073	AB	27.03.1995	50		01.03.1995	1000	3	0,00	0,00	0,00	2.910,14-	EUR	2.910,14-	EUR	3.492,17-	USD
						100002073	AB	27.03.1995	50		01.03.1995	1000	4	0,00	0,00	0,00	159,52-	EUR	159,52-	EUR	191,42-	USD
						100002073	AB	27.03.1995	50		01.03.1995	1000	8	0,00	0,00	0,00	798,61-	EUR	798,61-	EUR	958,33-	USD
						100002073	AB	27.03.1995	50		01.03.1995	1000	9	0,00	0,00	0,00	142,45-	EUR	142,45-	EUR	170,94-	USD
						100002073	AB	27.03.1995	50		01.03.1995	1000	10	0,00	0,00	0,00	411,64-	EUR	411,64-	EUR	493,97-	USD
						100002073	AB	27.03.1995	50		01.03.1995	1000	14	0,00	0,00	0,00	8.054,78-	EUR	8.054,78-	EUR	9.665,74-	USD
						100002073	AB	27.03.1995	50		01.03.1995	1000	15	0,00	0,00	0,00	1.032,21-	EUR	1.032,21-	EUR	1.238,65-	USD
						100002073	AB	27.03.1995	50		01.03.1995	1000	16	0,00	0,00	0,00	4.372,12-	EUR	4.372,12-	EUR	5.246,54-	USD
						100002073	AB	27.03.1995	50		01.03.1995	1000	17	0,00	0,00	0,00	146,23-	EUR	146,23-	EUR	175,48-	USD
						100002073	AB	27.03.1995	50		01.03.1995	1000	24	0,00	0,00	0,00	2.588,06-	EUR	2.588,06-	EUR	3.105,67-	USD
						100002073	AB	27.03.1995	50		01.03.1995	1000	25	0,00	0,00	0,00	279,22-	EUR	279,22-	EUR	335,06-	USD
						100002073	AB	27.03.1995	50		01.03.1995	1000	26	0,00	0,00	0,00	1.390,97-	EUR	1.390,97-	EUR	1.669,16-	USD
						100002073	AB	27.03.1995	50		01.03.1995	1000	27	0,00	0,00	0,00	79,76-	EUR	79,76-	EUR	95,71-	USD
						100002086	AB	27.07.1995	50		01.07.1995	1000	28	0,00	0,00	0,00	9.183,22-	EUR	9.183,22-	EUR	11.019,86-	USD
						100002086	AB	27.07.1995	50		01.07.1995	1000	29	0,00	0,00	0,00	319,05-	EUR	319,05-	EUR	382,86-	USD
						100002086	AB	27.07.1995	50		01.07.1995	1000	38	0,00	0,00	0,00	23.716,13-	EUR	23.716,13-	EUR	28.459,36-	USD
						100002086	AB	27.07.1995	50		01.07.1995	1000	39	0,00	0,00	0,00	4.923,34-	EUR	4.923,34-	EUR	5.908,01-	USD
						100002086	AB	27.07.1995	50		01.07.1995	1000	40	0,00	0,00	0,00	13.254,09-	EUR	13.254,09-	EUR	15.904,91-	USD
						100002086	AB	27.07.1995	50		01.07.1995	1000	41	0,00	0,00	0,00	691,27-	EUR	691,27-	EUR	829,52-	USD
						100002086	AB	27.07.1995	50		01.07.1995	1000	53	0,00	0,00	0,00	11.050,21-	EUR	11.050,21-	EUR	13.260,25-	USD
						100002086	AB	27.07.1995	50		01.07.1995	1000	54	0,00	0,00	0,00	1.480,39-	EUR	1.480,39-	EUR	1.776,47-	USD
						100002086	AB	27.07.1995	50		01.07.1995	1000	55	0,00	0,00	0,00	6.392,44-	EUR	6.392,44-	EUR	7.670,93-	USD
						100002086	AB	27.07.1995	50		01.07.1995	1000	56	0,00	0,00	0,00	305,75-	EUR	305,75-	EUR	366,90-	USD

16. In order to write enhancement code, go to CMOD and give the project name (ZBW220) and then click on display button



Project Management of SAP Enhancements

Project ZBW220

Subobjects

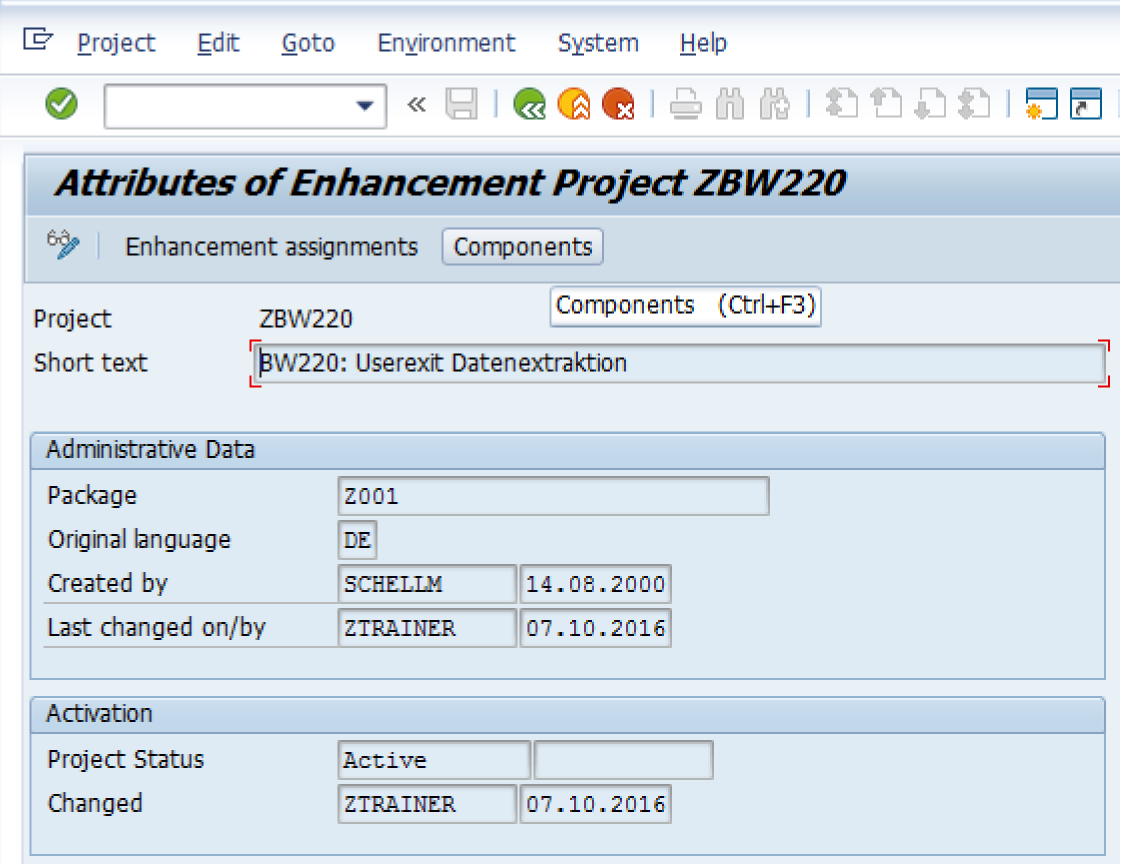
● Attributes

○ Enhancement Assignment

○ Components

○ Documentation

17. Click on Components button which will take you to the next screen



Attributes of Enhancement Project ZBW220

Project	ZBW220
Short text	BW220: Userexit Datenextraktion

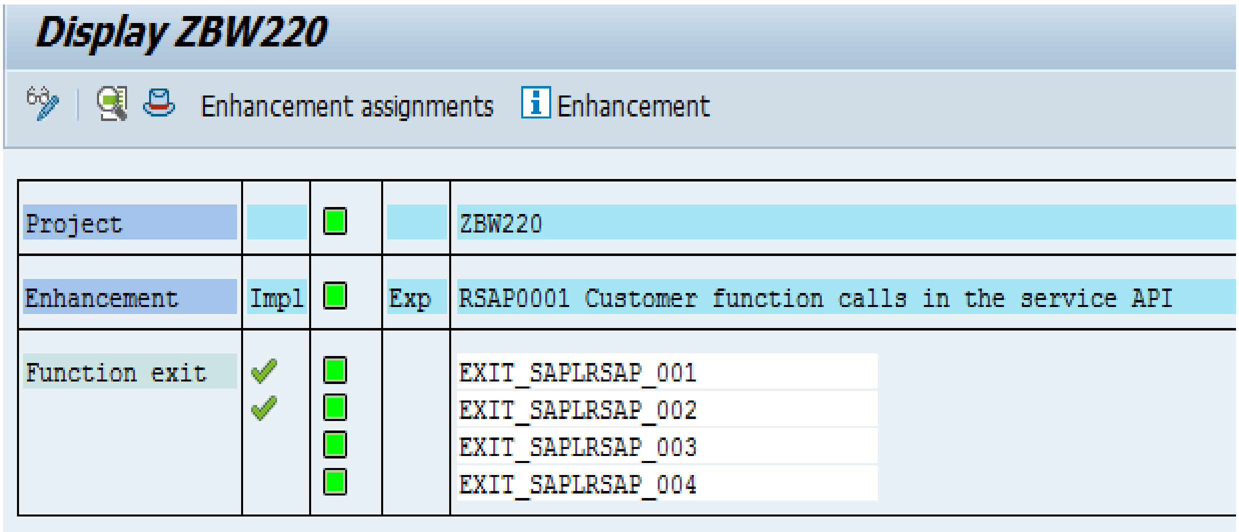
Administrative Data

Package	Z001	
Original language	DE	
Created by	SCHELLM	14.08.2000
Last changed on/by	ZTRAINER	07.10.2016

Activation

Project Status	Active	
Changed	ZTRAINER	07.10.2016

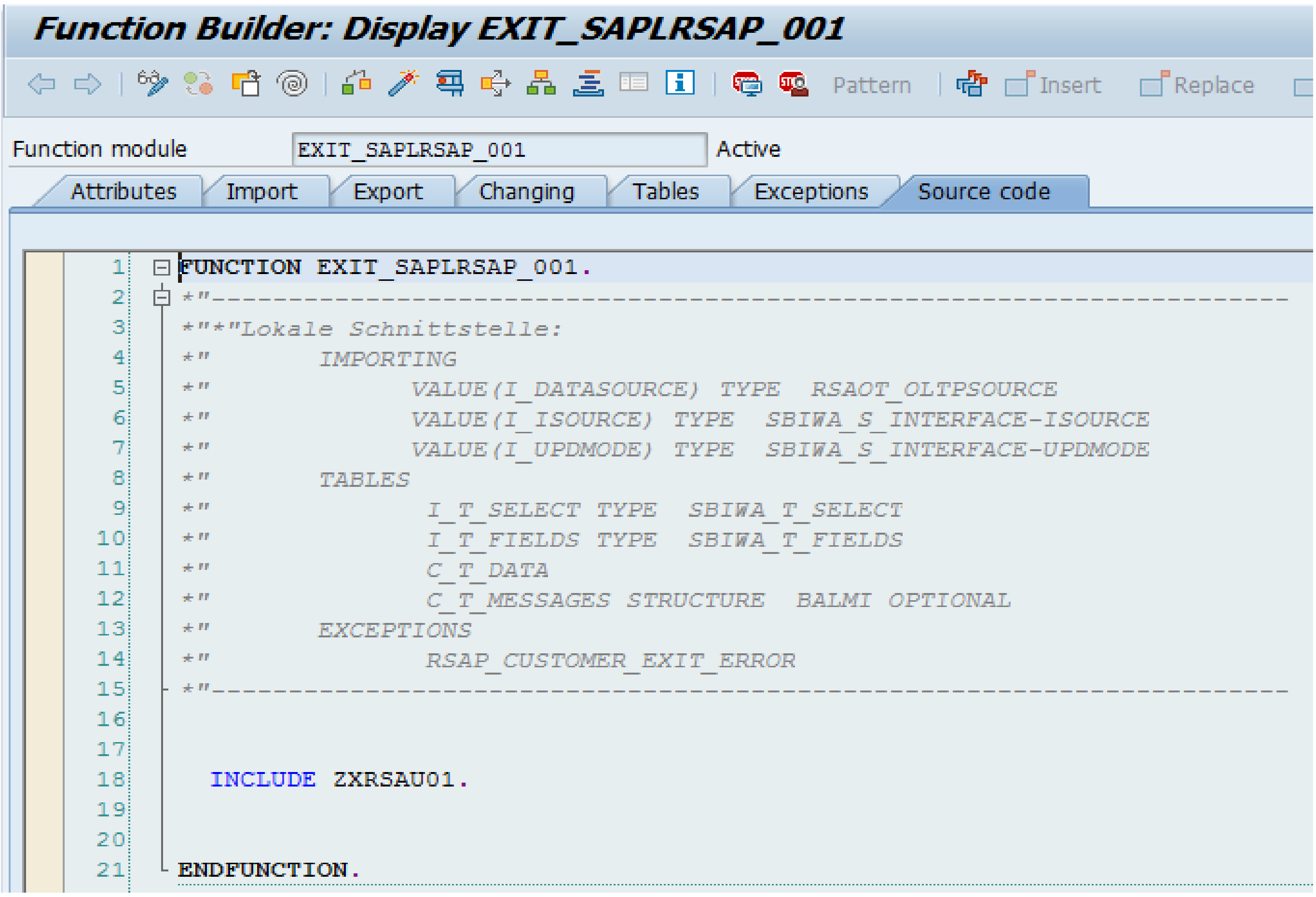
18. Since we are enhancing transaction datasource, double click on EXIT_SAPLRSAP_001.



Display ZBW220

Project		🟩		ZBW220
Enhancement	Impl	🟩	Exp	RSAP0001 Customer function calls in the service API
Function exit	✓ ✓	🟩 🟩 🟩 🟩		EXIT_SAPLRSAP_001 EXIT_SAPLRSAP_002 EXIT_SAPLRSAP_003 EXIT_SAPLRSAP_004

19. Double click on include ZXRSAU01. It would open up editor for writing enhancement code. Write enhancement code which is given in below code block.



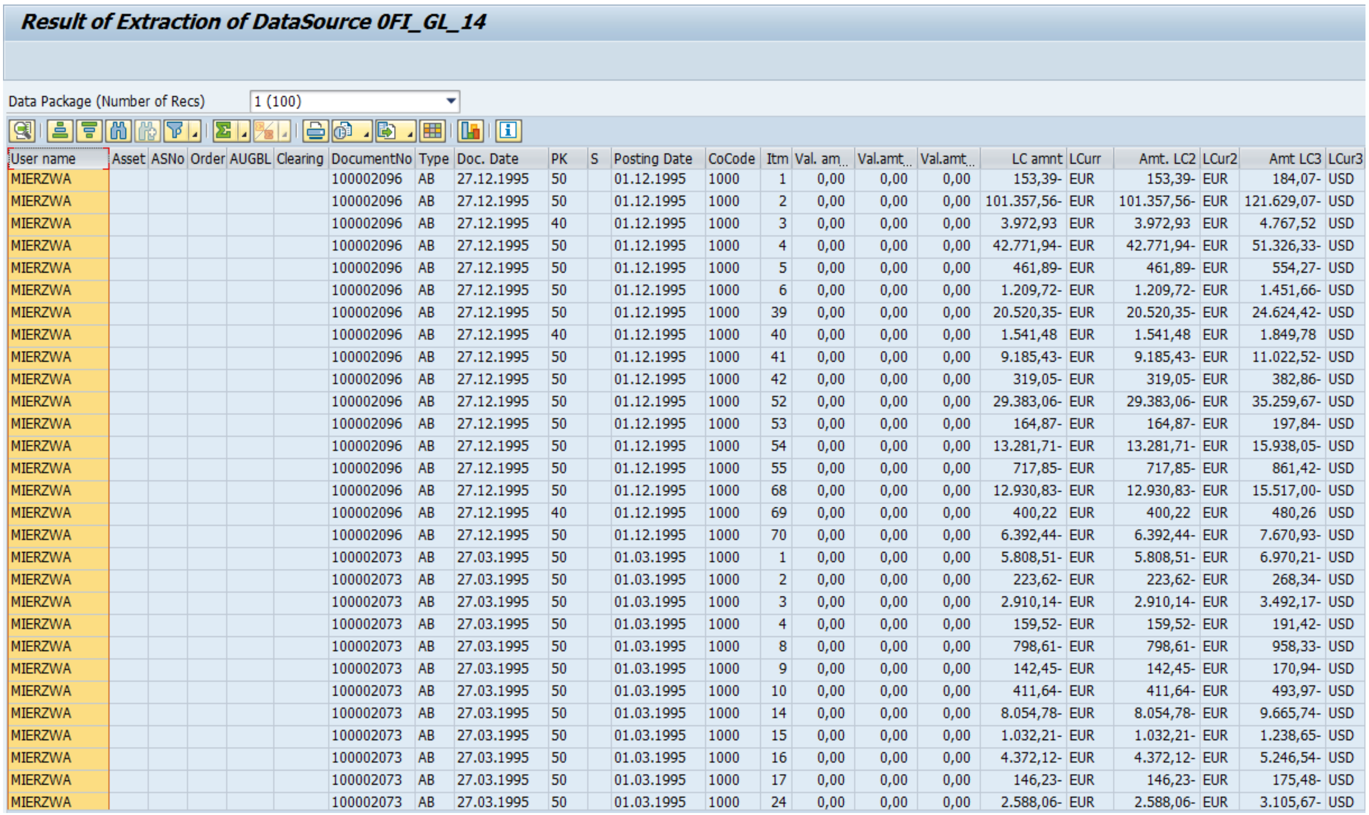
Function Builder: Display EXIT_SAPLRSAP_001

Function module EXIT_SAPLRSAP_001 Active

```
FUNCTION EXIT_SAPLRSAP_001.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"             VALUE(I_DATASOURCE) TYPE  RSAOT_OLTPSOURCE
*"             VALUE(I_ISOURCE) TYPE  SBIWA_S_INTERFACE-ISOURCE
*"             VALUE(I_UPDMODE) TYPE  SBIWA_S_INTERFACE-UPDMODE
*"       TABLES
*"              I_T_SELECT TYPE  SBIWA_T_SELECT
*"              I_T_FIELDS TYPE  SBIWA_T_FIELDS
*"              C_T_DATA
*"              C_T_MESSAGES STRUCTURE  BALMI OPTIONAL
*"       EXCEPTIONS
*"              RSAP_CUSTOMER_EXIT_ERROR
*"----------------------------------------------------------------------


  INCLUDE ZXRSAU01.


ENDFUNCTION.
```

```
       when '0FI_GL_14'.

        types : begin of s_bkpf,
                 bukrs type bkpf-bukrs,
                 belnr TYPE bkpf-belnr,
                 gjahr type bkpf-gjahr,
                 usnam type bkpf-usnam,
     end of s_bkpf.

 data: t_bkpf type STANDARD TABLE OF s_bkpf,
    Zc_t_data type STANDARD TABLE OF FAGLPOSBW,
    wa_BKPF type s_bkpf.

 FIELD-SYMBOLS : <FS_CT_DATA> type FAGLPOSBW,
                  <FS_BKPF> type s_bkpf.

 if c_t_data[] is not INITIAL.
  Zc_t_data[] = c_t_data[].

  select bukrs belnr gjahr usnam
     from bkpf
     into TABLE t_bkpf
     FOR ALL ENTRIES IN Zc_t_data
     WHERE BUKRS = Zc_t_data-BUKRS
     AND BELNR = Zc_t_data-BELNR
     AND GJAHR = Zc_t_data-GJAHR.

   sort t_bkpf by BUKRS BELNR GJAHR.
if t_bkpf is not INITIAL.


     loop at c_t_data ASSIGNING <FS_CT_DATA>.


       READ TABLE t_bkpf ASSIGNING <FS_BKPF>
         with KEY BUKRS = <FS_CT_DATA>-BUKRS
         BELNR = <FS_CT_DATA>-BELNR
         GJAHR = <FS_CT_DATA>-GJAHR
         BINARY SEARCH.

*      READ TABLE t_bkpf into wa_BKPF
*         with KEY BUKRS = wa_BKPF-BUKRS
*         BELNR = wa_BKPF-BELNR
*         GJAHR = wa_BKPF-GJAHR.
*
       if sy-subrc = 0.
*         <FS_CT_DATA>-zzusnam = wa_BKPF-usnam.
         <FS_CT_DATA>-zzusnam = <FS_BKPF>-usnam.
       endif.


     endloop.


   endif.


FREE Zc_t_data.

 endif.
```

20. Go to RSA3 to see the datasource output. Newly added field ZZUSNAM (User Name) is populated with values



Result of Extraction of DataSource 0FI_GL_14

Data Package (Number of Recs) 1 (100)

User name	Asset	ASNo	Order	AUGBL	Clearing	DocumentNo	Type	Doc. Date	PK	S	Posting Date	CoCode	Itm	Val. am...	Val.amt...	Val.amt...	LC amnt	LCurr	Amt. LC2	LCur2	Amt LC3	LCur3
MIERZWA						100002096	AB	27.12.1995	50		01.12.1995	1000	1	0,00	0,00	0,00	153,39-	EUR	153,39-	EUR	184,07-	USD
MIERZWA						100002096	AB	27.12.1995	50		01.12.1995	1000	2	0,00	0,00	0,00	101.357,56-	EUR	101.357,56-	EUR	121.629,07-	USD
MIERZWA						100002096	AB	27.12.1995	40		01.12.1995	1000	3	0,00	0,00	0,00	3.972,93	EUR	3.972,93	EUR	4.767,52	USD
MIERZWA						100002096	AB	27.12.1995	50		01.12.1995	1000	4	0,00	0,00	0,00	42.771,94-	EUR	42.771,94-	EUR	51.326,33-	USD
MIERZWA						100002096	AB	27.12.1995	50		01.12.1995	1000	5	0,00	0,00	0,00	461,89-	EUR	461,89-	EUR	554,27-	USD
MIERZWA						100002096	AB	27.12.1995	50		01.12.1995	1000	6	0,00	0,00	0,00	1.209,72-	EUR	1.209,72-	EUR	1.451,66-	USD
MIERZWA						100002096	AB	27.12.1995	50		01.12.1995	1000	39	0,00	0,00	0,00	20.520,35-	EUR	20.520,35-	EUR	24.624,42-	USD
MIERZWA						100002096	AB	27.12.1995	40		01.12.1995	1000	40	0,00	0,00	0,00	1.541,48	EUR	1.541,48	EUR	1.849,78	USD
MIERZWA						100002096	AB	27.12.1995	50		01.12.1995	1000	41	0,00	0,00	0,00	9.185,43-	EUR	9.185,43-	EUR	11.022,52-	USD
MIERZWA						100002096	AB	27.12.1995	50		01.12.1995	1000	42	0,00	0,00	0,00	319,05-	EUR	319,05-	EUR	382,86-	USD
MIERZWA						100002096	AB	27.12.1995	50		01.12.1995	1000	52	0,00	0,00	0,00	29.383,06-	EUR	29.383,06-	EUR	35.259,67-	USD
MIERZWA						100002096	AB	27.12.1995	50		01.12.1995	1000	53	0,00	0,00	0,00	164,87-	EUR	164,87-	EUR	197,84-	USD
MIERZWA						100002096	AB	27.12.1995	50		01.12.1995	1000	54	0,00	0,00	0,00	13.281,71-	EUR	13.281,71-	EUR	15.938,05-	USD
MIERZWA						100002096	AB	27.12.1995	50		01.12.1995	1000	55	0,00	0,00	0,00	717,85-	EUR	717,85-	EUR	861,42-	USD
MIERZWA						100002096	AB	27.12.1995	50		01.12.1995	1000	68	0,00	0,00	0,00	12.930,83-	EUR	12.930,83-	EUR	15.517,00-	USD
MIERZWA						100002096	AB	27.12.1995	40		01.12.1995	1000	69	0,00	0,00	0,00	400,22	EUR	400,22	EUR	480,26	USD
MIERZWA						100002096	AB	27.12.1995	50		01.12.1995	1000	70	0,00	0,00	0,00	6.392,44-	EUR	6.392,44-	EUR	7.670,93-	USD
MIERZWA						100002073	AB	27.03.1995	50		01.03.1995	1000	1	0,00	0,00	0,00	5.808,51-	EUR	5.808,51-	EUR	6.970,21-	USD
MIERZWA						100002073	AB	27.03.1995	50		01.03.1995	1000	2	0,00	0,00	0,00	223,62-	EUR	223,62-	EUR	268,34-	USD
MIERZWA						100002073	AB	27.03.1995	50		01.03.1995	1000	3	0,00	0,00	0,00	2.910,14-	EUR	2.910,14-	EUR	3.492,17-	USD
MIERZWA						100002073	AB	27.03.1995	50		01.03.1995	1000	4	0,00	0,00	0,00	159,52-	EUR	159,52-	EUR	191,42-	USD
MIERZWA						100002073	AB	27.03.1995	50		01.03.1995	1000	8	0,00	0,00	0,00	798,61-	EUR	798,61-	EUR	958,33-	USD
MIERZWA						100002073	AB	27.03.1995	50		01.03.1995	1000	9	0,00	0,00	0,00	142,45-	EUR	142,45-	EUR	170,94-	USD
MIERZWA						100002073	AB	27.03.1995	50		01.03.1995	1000	10	0,00	0,00	0,00	411,64-	EUR	411,64-	EUR	493,97-	USD
MIERZWA						100002073	AB	27.03.1995	50		01.03.1995	1000	14	0,00	0,00	0,00	8.054,78-	EUR	8.054,78-	EUR	9.665,74-	USD
MIERZWA						100002073	AB	27.03.1995	50		01.03.1995	1000	15	0,00	0,00	0,00	1.032,21-	EUR	1.032,21-	EUR	1.238,65-	USD
MIERZWA						100002073	AB	27.03.1995	50		01.03.1995	1000	16	0,00	0,00	0,00	4.372,12-	EUR	4.372,12-	EUR	5.246,54-	USD
MIERZWA						100002073	AB	27.03.1995	50		01.03.1995	1000	17	0,00	0,00	0,00	146,23-	EUR	146,23-	EUR	175,48-	USD
MIERZWA						100002073	AB	27.03.1995	50		01.03.1995	1000	24	0,00	0,00	0,00	2.588,06-	EUR	2.588,06-	EUR	3.105,67-	USD

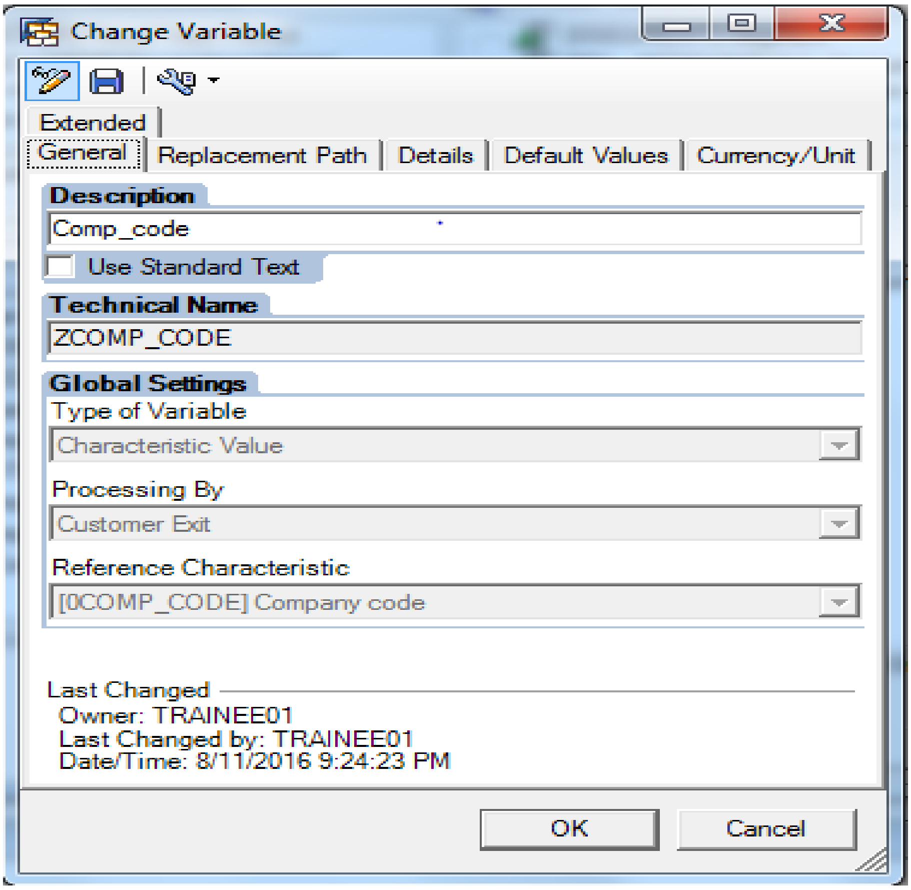
User Exits and BADIs in Reporting.

Customer exit Variable in BEX reporting (I_STEP = 0,1,2,3)

Example 1 : I_STEP =1 – Automatically populating single value /Multiple single value in variable screen

Let us consider a scenario where a characteristic variable of type customer exit & ready for input is to be populated with a default value when the report is executed & when the selection screen appears

Scenario 1: Automatically populating single value in variable screen.



Description

Comp_code

□ Use Standard Text

Technical Name

ZCOMP_CODE

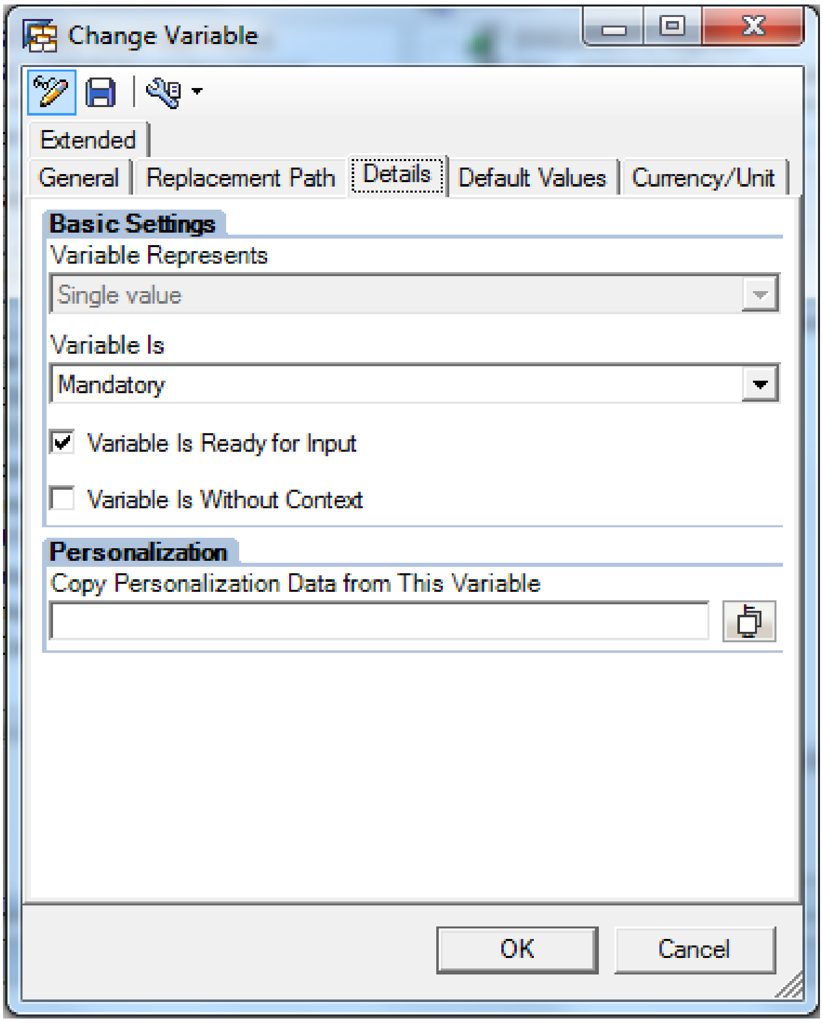
Global Settings

Type of Variable: Characteristic Value

Processing By: Customer Exit

Reference Characteristic: [0COMP_CODE] Company code

Last Changed
Owner: TRAINEE01
Last Changed by: TRAINEE01
Date/Time: 8/11/2016 9:24:23 PM



Basic Settings

Variable Represents

Single value

Variable Is

Mandatory

☑ Variable Is Ready for Input

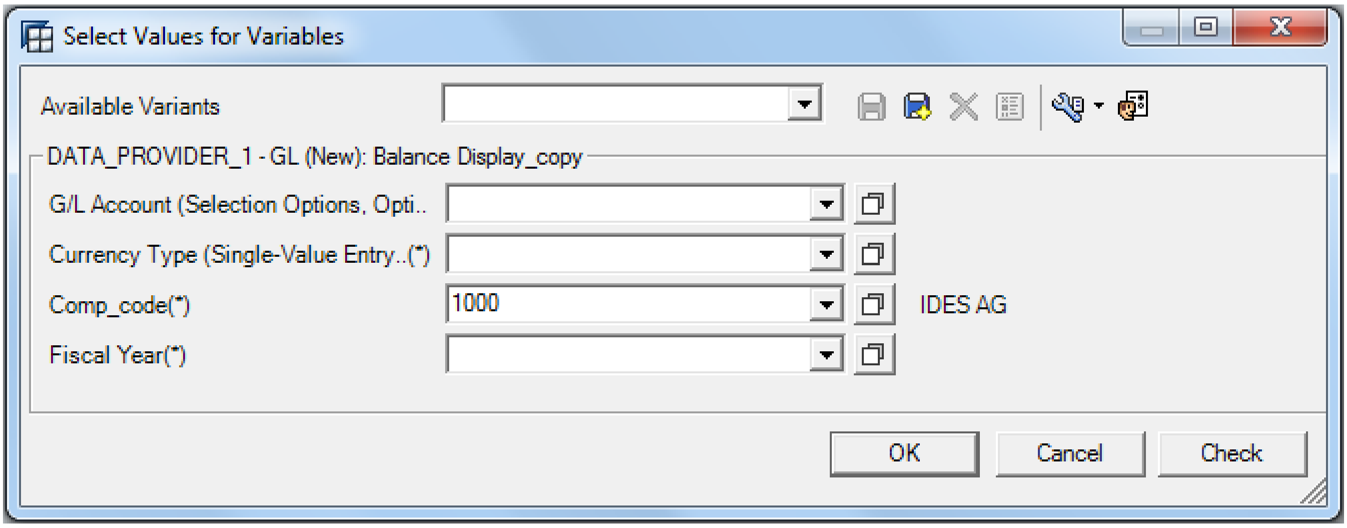
□ Variable Is Without Context

Personalization

Copy Personalization Data from This Variable

Code for automatically populating single value in variable screen

```
when 'ZCOMP_CODE'.
 IF i_step = 1.

        clear s_range.

        s_range-low = '1000'.
        s_range-sign = 'I'.
        s_range-opt = 'EQ'.
       Append s_range to e_t_range.


  endif.
```



Select Values for Variables

Available Variants

DATA_PROVIDER_1 - GL (New): Balance Display_copy

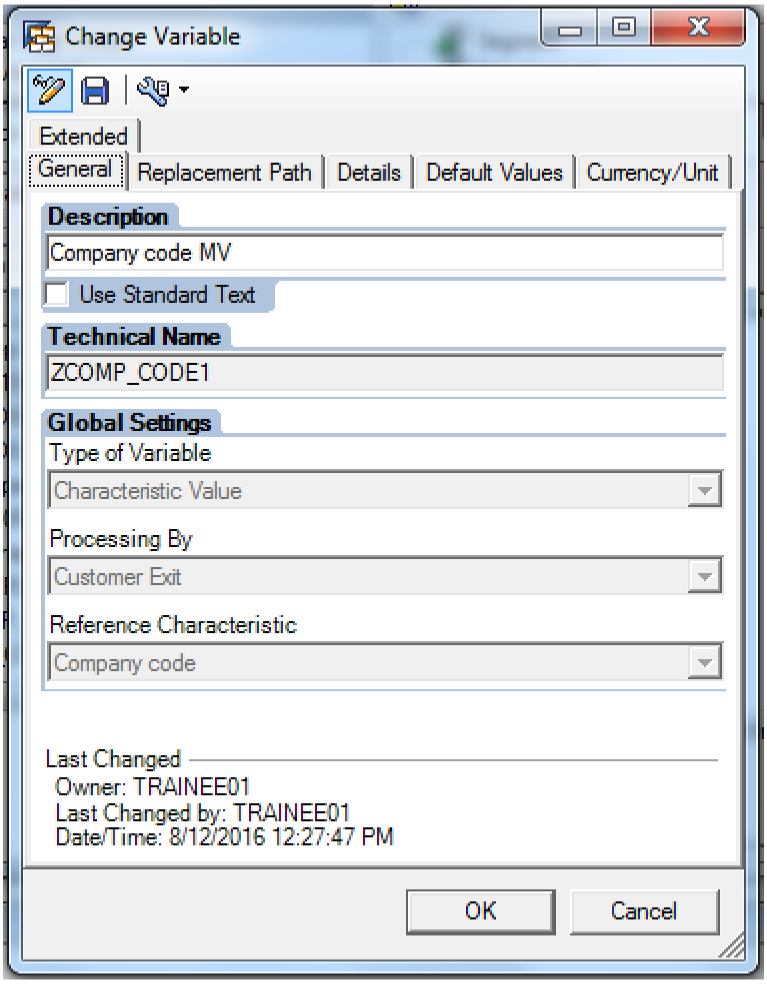
G/L Account (Selection Options, Opti..

Currency Type (Single-Value Entry..(*)

Comp_code(*): 1000 IDES AG

Fiscal Year(*)

Scenario 2: Automatically populating multiple single value in variable screen.



Change Variable

Description

Company code MV

□ Use Standard Text

Technical Name

ZCOMP_CODE1

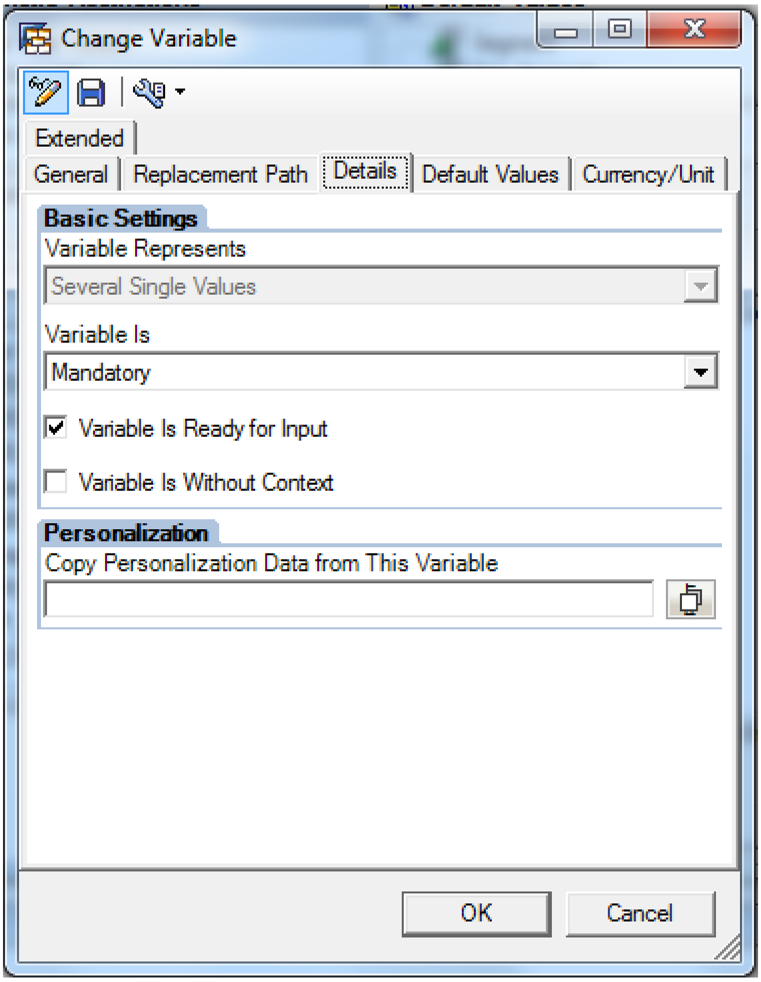
Global Settings

Type of Variable: Characteristic Value

Processing By: Customer Exit

Reference Characteristic: Company code

Last Changed
Owner: TRAINEE01
Last Changed by: TRAINEE01
Date/Time: 8/12/2016 12:27:47 PM



Basic Settings

Variable Represents: Several Single Values

Variable Is: Mandatory

☑ Variable Is Ready for Input

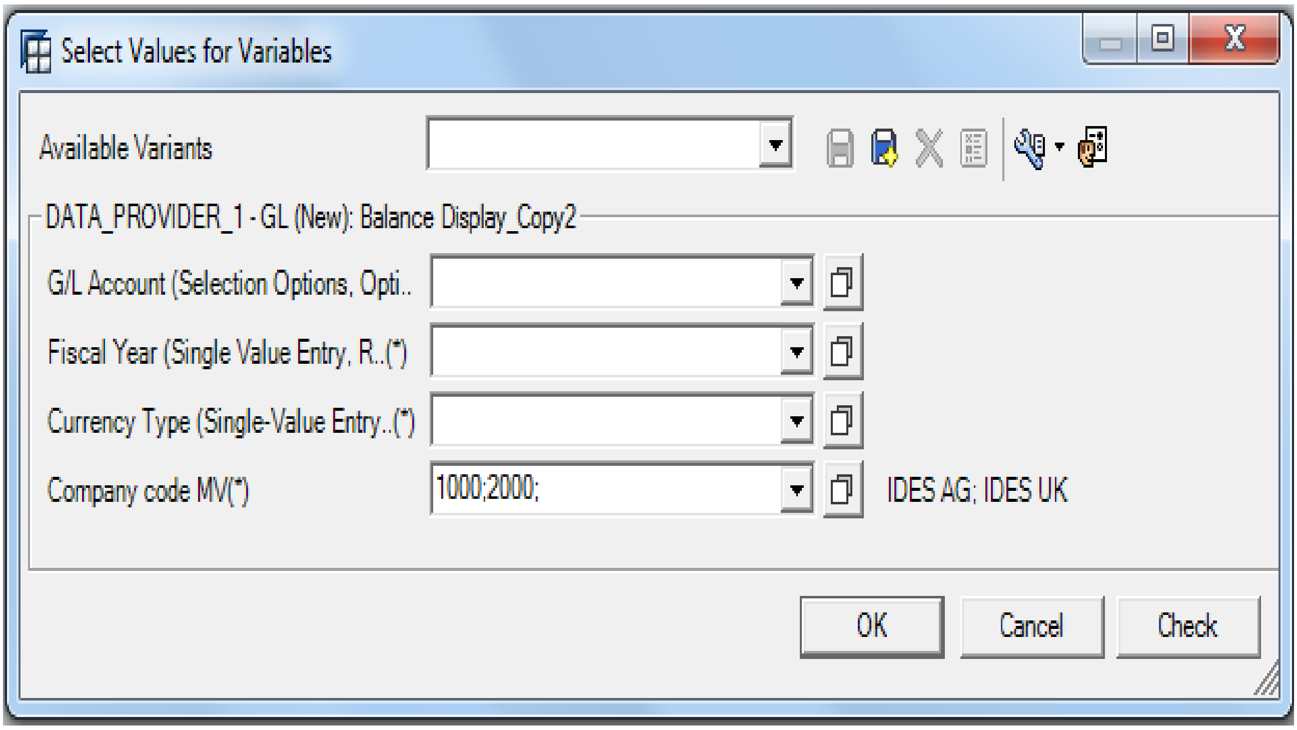
□ Variable Is Without Context

Personalization

Copy Personalization Data from This Variable

Code for populating multiple single values in variable screen.

```
when 'ZCOMP_CODE1'.

  IF i_step = 1.
   clear s_range.

   s_range-low = '1000'.
   s_range-sign = 'I'.
   s_range-opt = 'EQ'.
   Append s_range to e_t_range.

   s_range-low = '2000'.
   s_range-sign = 'I'.
   s_range-opt = 'EQ'.
   Append s_range to e_t_range.

  endif.
```



Available Variants

DATA_PROVIDER_1 - GL (New): Balance Display_Copy2

G/L Account (Selection Options, Opti..

Fiscal Year (Single Value Entry, R..(*)

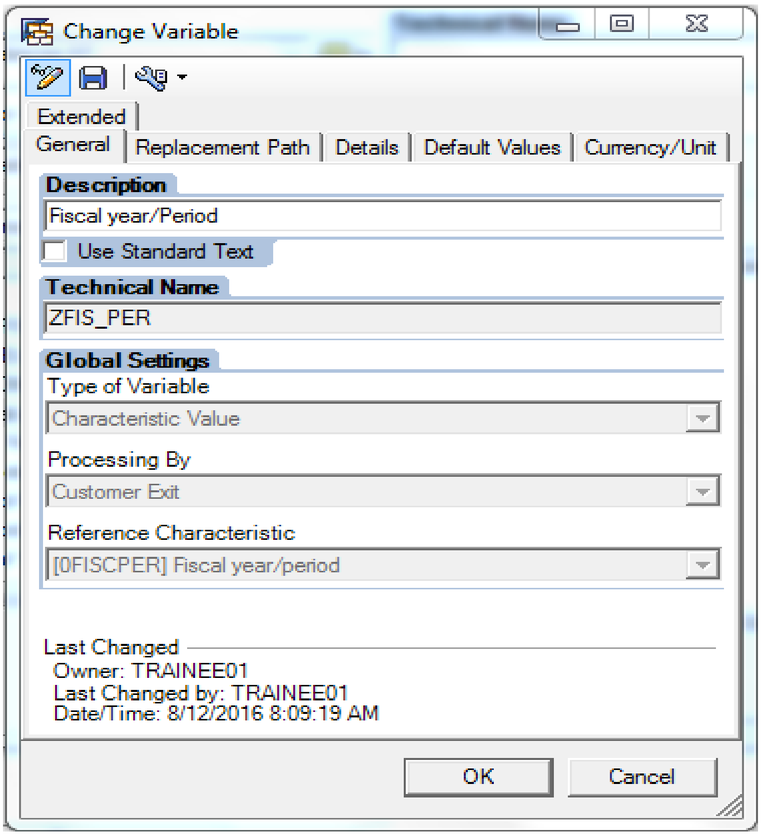
Currency Type (Single-Value Entry..(*)

Company code MV(*) 1000;2000; IDES AG; IDES UK

Example 2: I_STEP =2 - the variable to be processed after the report's selection appears (ie after user input)

Scenario : Let us consider a scenario where a characteristic variable of type customer exit is to be populated with a value based on the value entered by the user in another input variable during selection

In the following screen shots the scenario considered is, the user would input Fiscal Year for the variable 0P_FYEAR. Inside the user exit, this value needs to be captured and we would have to derive the first 2 quarters of the Fiscal Year entered by user and should be passed to the customer exit variable ZFIS_PER



Description

Fiscal year/Period

□ Use Standard Text

Technical Name

ZFIS_PER

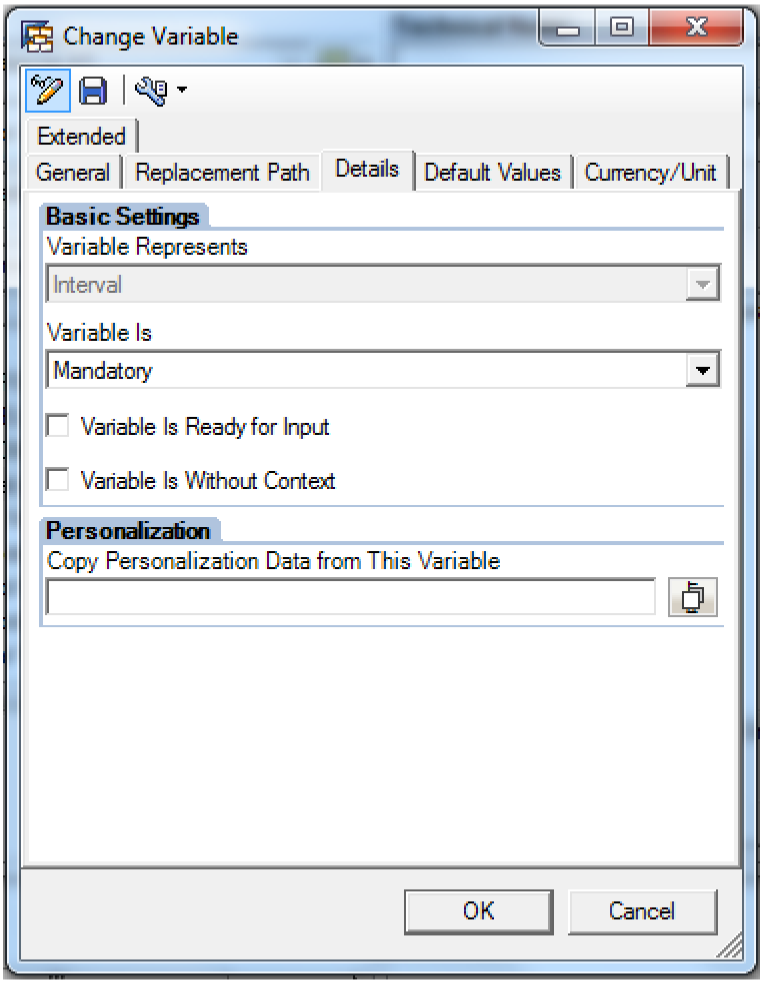
Global Settings

Type of Variable: Characteristic Value

Processing By: Customer Exit

Reference Characteristic: [0FISCPER] Fiscal year/period

Last Changed
Owner: TRAINEE01
Last Changed by: TRAINEE01
Date/Time: 8/12/2016 8:09:19 AM



Basic Settings

Variable Represents: Interval

Variable Is: Mandatory

□ Variable Is Ready for Input

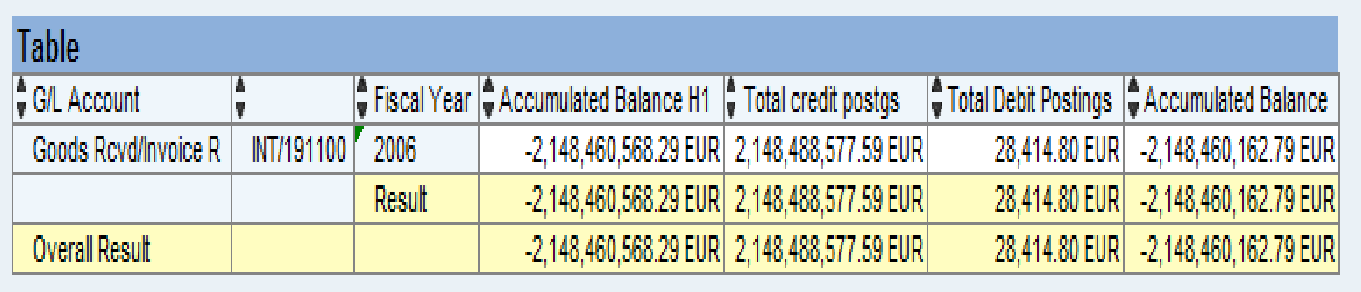
□ Variable Is Without Context

Personalization

Copy Personalization Data from This Variable

Code snippet

```
when 'ZFIS_PER'.

Data : zfyear1 type /BI0/OIFISCYEAR,
       ZFISCPER1(7) type c,
       ZFISCPER2(7) type c.

  if i_step = 2.
    READ TABLE i_t_var_range into s_var_range
          with key vnam = '0P_FYEAR'.

   zfyear1 = s_var_range-low.

  CONCATENATE zfyear1 '001' into ZFISCPER1.
  CONCATENATE zfyear1 '006' into ZFISCPER2.
*2016001 to 2016006
       s_range-sign = 'I'.
       s_range-opt = 'BT'.
       s_range-low = ZFISCPER1.
       s_range-High = ZFISCPER2.

   Append s_range to e_t_range.

  endif.
```



Table

G/L Account		Fiscal Year	Accumulated Balance H1	Total credit postgs	Total Debit Postings	Accumulated Balance
Goods Rcvd/Invoice R	INT/191100	2006	-2,148,460,568.29 EUR	2,148,488,577.59 EUR	28,414.80 EUR	-2,148,460,162.79 EUR
		Result	-2,148,460,568.29 EUR	2,148,488,577.59 EUR	28,414.80 EUR	-2,148,460,162.79 EUR
Overall Result			-2,148,460,568.29 EUR	2,148,488,577.59 EUR	28,414.80 EUR	-2,148,460,162.79 EUR

ABAP OOPs Concept with examples.

Classes and Objects

This example covers creation of a local Class and its Objects.

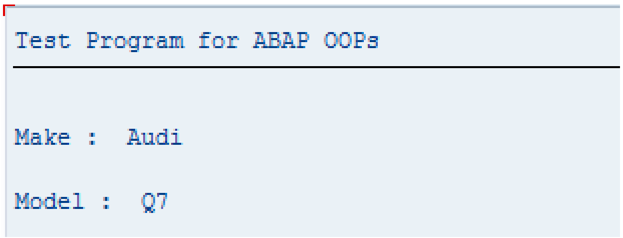
Steps

1. Create a class LCL_VEHICLE:

Private Section	
Attribute MAKE of type STRING	
Attribute MODEL of type STRING	
Public Section	
Method SET_ATTRIBUTES to set the values of MAKE and MODEL	Importing: I_MAKE I_MODEL
Method DISPLAY_ATTRIBUTES to display the values of MAKE and MODEL attributes.	N/A

2. Create an object LREF_VEHICLE of type LCL_VEHICLE.
3. Call method SET_ATTRIBUTES using reference object LREF_VEHICLE to set the values of attributes MAKE and MODEL.
4. Call method DISPLAY_ATRIBUTES using reference object LREF_VEHICLE to display the values of attributes MAKE and MODEL.

Output:

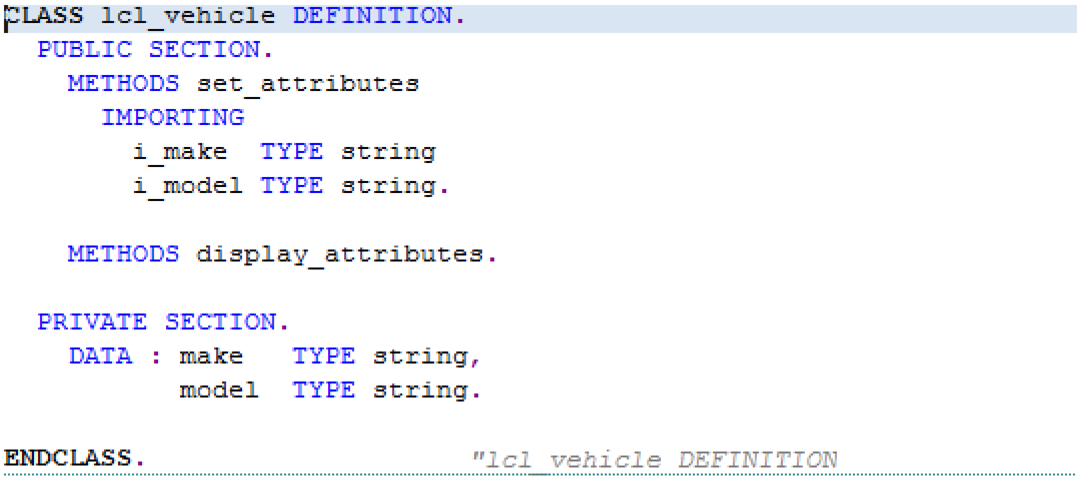


```
Test Program for ABAP OOPs

Make :  Audi

Model :  Q7
```

Solution:

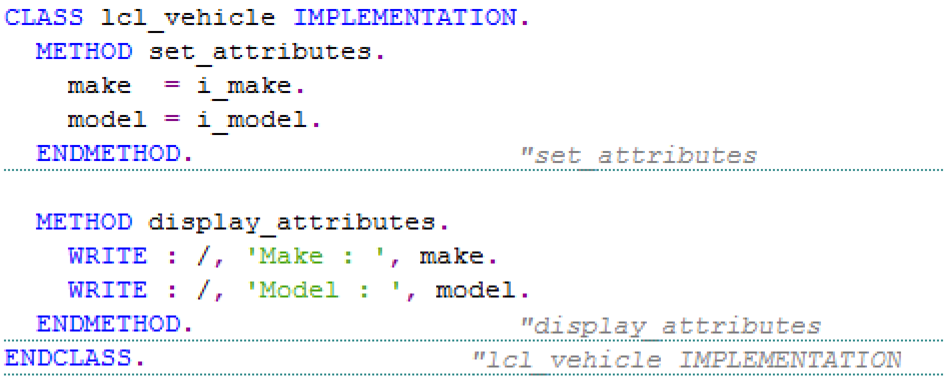
1. Go to transaction SE38 and create an executable program Z_TEST_OOP_XX.

2. Create the class definition as below.



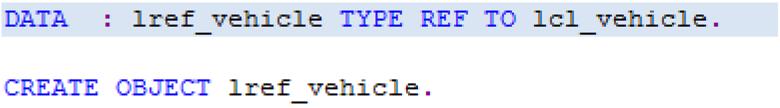
```
CLASS lcl_vehicle DEFINITION.
  PUBLIC SECTION.
    METHODS set_attributes
      IMPORTING
        i_make  TYPE string
        i_model TYPE string.

    METHODS display_attributes.

  PRIVATE SECTION.
    DATA : make    TYPE string,
           model   TYPE string.

ENDCLASS.                    "lcl_vehicle DEFINITION
```

3. Create the class implementation as below.



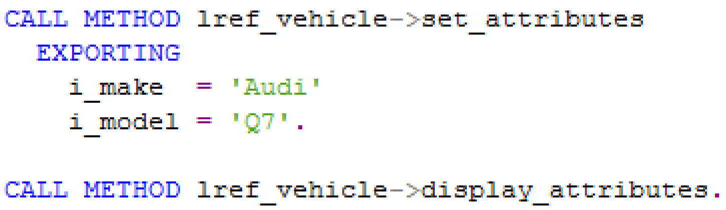
```
CLASS lcl_vehicle IMPLEMENTATION.
  METHOD set_attributes.
    make  = i_make.
    model = i_model.
  ENDMETHOD.                    "set_attributes

  METHOD display_attributes.
    WRITE : /, 'Make : ', make.
    WRITE : /, 'Model : ', model.
  ENDMETHOD.                    "display_attributes
ENDCLASS.                    "lcl_vehicle IMPLEMENTATION
```

4. Declare and create the object reference of the class LCL_VEHICLE.



```
DATA  : lref_vehicle TYPE REF TO lcl_vehicle.

CREATE OBJECT lref_vehicle.
```

5. Call methods using the above object reference.



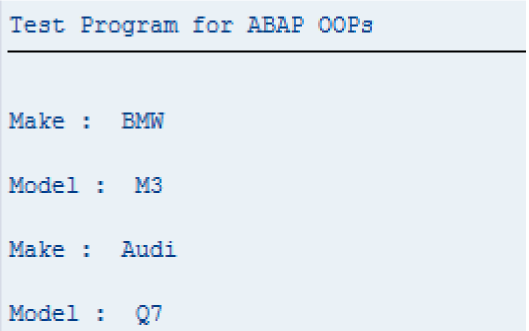
```
CALL METHOD lref_vehicle->set_attributes
  EXPORTING
    i_make  = 'Audi'
    i_model = 'Q7'.

CALL METHOD lref_vehicle->display_attributes.
```

Constructor

Steps

1. Create a CONSTRUCTOR in the above class to initialize the values of MAKE and MODEL**.**
2. Create the object reference LREF_VEHICLE.
3. Display the Initial values of MAKE and MODEL using DISPLAY_ATTRIBUTES method.

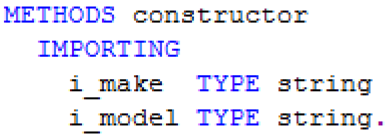
Output:



```
Test Program for ABAP OOPs


Make :  BMW

Model :  M3

Make :  Audi

Model :  Q7
```

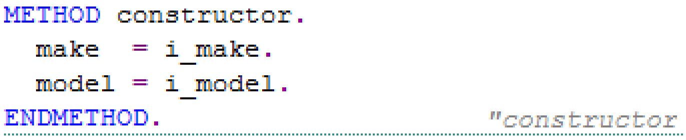
Solution:

1. Define the CONSTRUCTOR in the class.



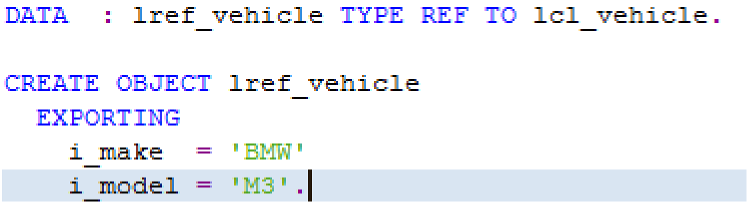
```
METHODS constructor
  IMPORTING
    i_make  TYPE string
    i_model TYPE string.
```

2. Implement the source code of the constructor.



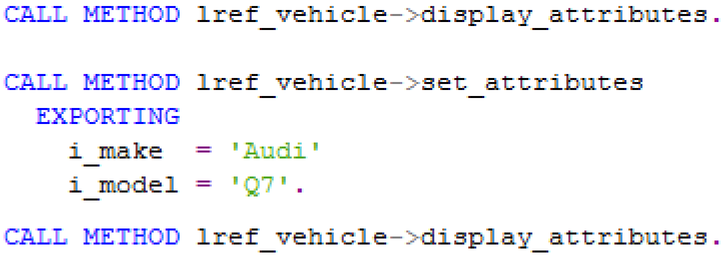
```
METHOD constructor.
  make  = i_make.
  model = i_model.
ENDMETHOD.                              "constructor
```

3. Declare and create the object reference of the class LCL_VEHICLE.



```
DATA  : lref_vehicle TYPE REF TO lcl_vehicle.

CREATE OBJECT lref_vehicle
  EXPORTING
    i_make  = 'BMW'
    i_model = 'M3'.
```

4. Call methods using the above object reference.



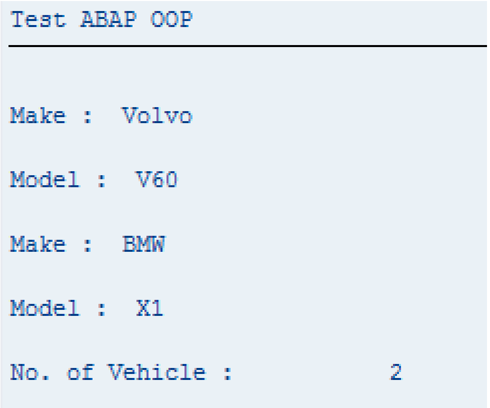
```
CALL METHOD lref_vehicle->display_attributes.

CALL METHOD lref_vehicle->set_attributes
  EXPORTING
    i_make  = 'Audi'
    i_model = 'Q7'.

CALL METHOD lref_vehicle->display_attributes.
```

Static Attributes and Methods

Steps

1. Create a Static attribute VEHICLE_COUNT of type integer (i) in the class LCL_VEHICLE to maintain the count of object reference for the class.
2. Increment the count of vehicle in Constructor of the class.
3. Create a Static method DISPLAY_STAT_ATTRIBUTE to display the Static attribute.
4. Call method DISPLAY_STAT_ATTRIBUTE to display the count of objects.

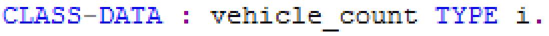
Output:



```
Test ABAP OOP

Make :  Volvo

Model :  V60

Make :  BMW

Model :  X1

No. of Vehicle :           2
```

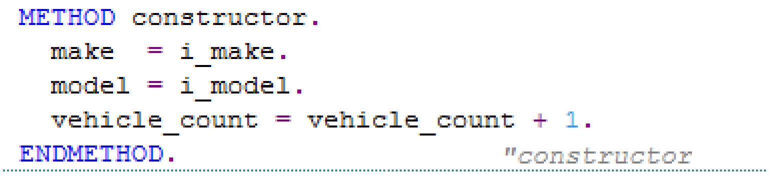
Solution:

1. Create a Static Attribute VEHICLE_COUNT.



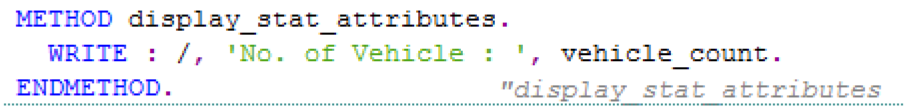
```
CLASS-DATA : vehicle_count TYPE i.
```

2. Increment the count of vehicle in Constructor of class.



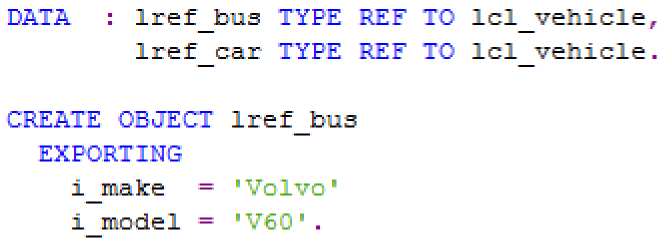
```
METHOD constructor.
  make  = i_make.
  model = i_model.
  vehicle_count = vehicle_count + 1.
ENDMETHOD.                    "constructor
```

3. Implement the statis method DISPLAY_STAT_ATTRIBUTE to display Static attributes.

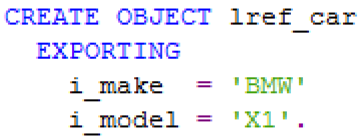


```
METHOD display_stat_attributes.
  WRITE : /, 'No. of Vehicle : ', vehicle_count.
ENDMETHOD.                    "display_stat_attributes
```

4. Create and instantiate two objects. LREF_BUS and LREF_CAR.

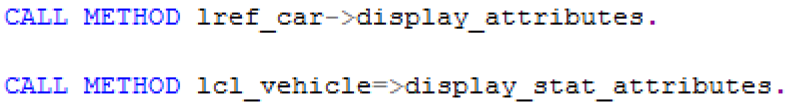


```
DATA  : lref_bus TYPE REF TO lcl_vehicle,
        lref_car TYPE REF TO lcl_vehicle.

CREATE OBJECT lref_bus
  EXPORTING
    i_make  = 'Volvo'
    i_model = 'V60'.
```



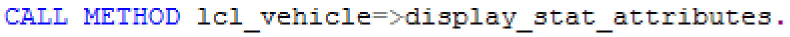
```
CREATE OBJECT lref_car
  EXPORTING
    i_make  = 'BMW'
    i_model = 'X1'.
```

5. Display attributes of both objects.



```
CALL METHOD lref_car->display_attributes.

CALL METHOD lcl_vehicle=>display_stat_attributes.
```

6. Display No. Of vehicles.



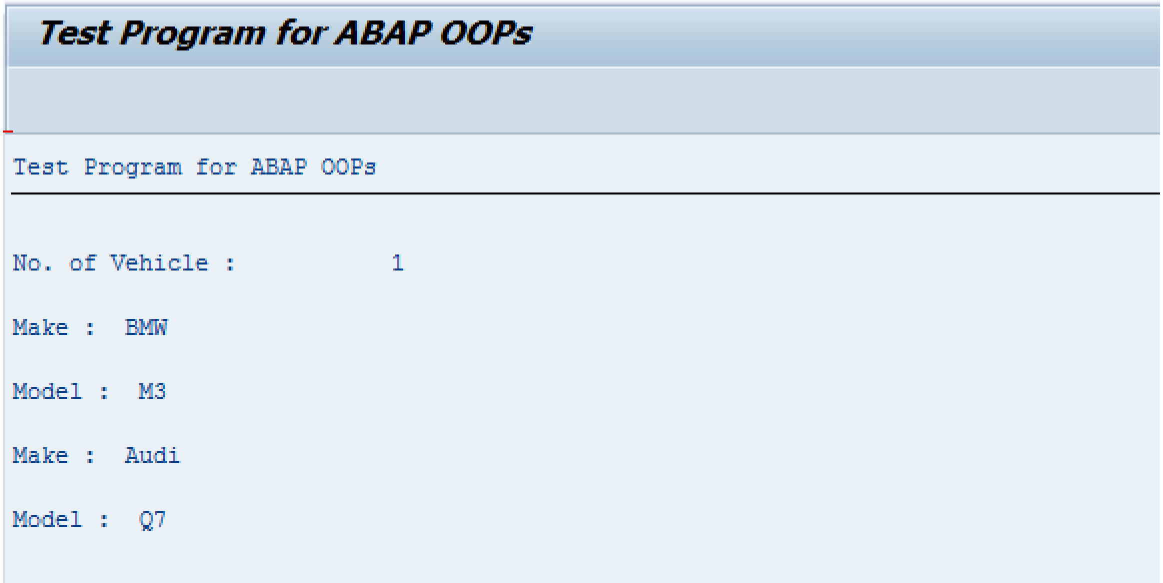
```
CALL METHOD lcl_vehicle=>display_stat_attributes.
```

Global Classes

Steps

1. Create a Global class ZCL_VEHICLE_XX by importing above class.
2. Create an object LREF_VEHICLE of type ZCL_VEHICLE.
3. Call method DISPLAY_ATRIBUTES using reference object LREF_VEHICLE to display the values of attributes MAKE and MODEL.

Output:

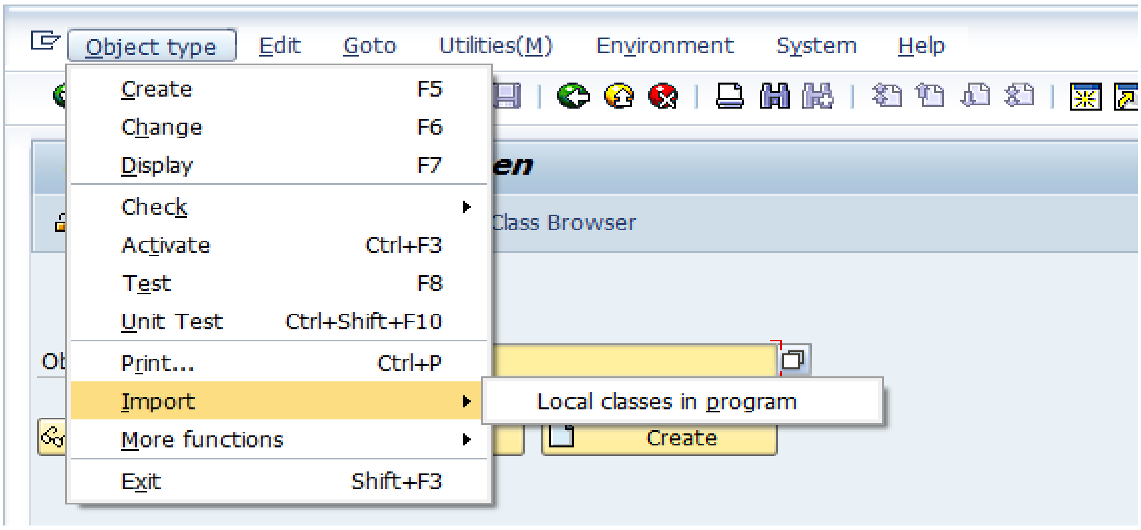


Test Program for ABAP OOPs

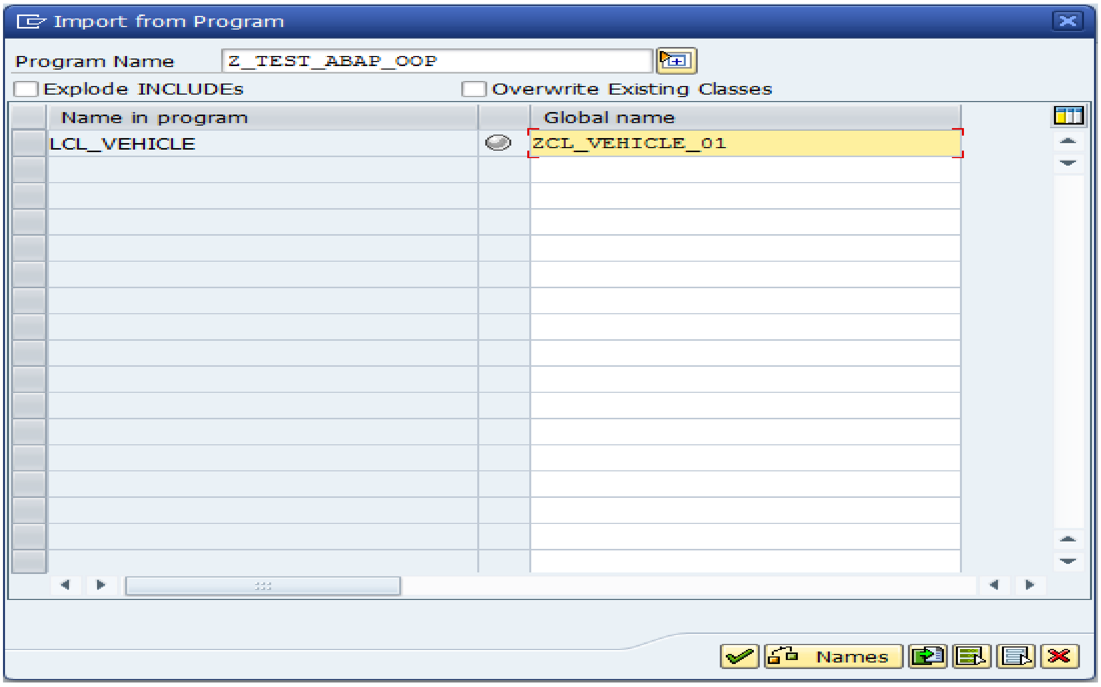
```
Test Program for ABAP OOPs

No. of Vehicle :            1

Make :  BMW

Model :  M3

Make :  Audi

Model :  Q7
```

Solution:

1. Go to transaction SE24 and select Object type->Import->Local classes in program.



2. Enter the name of your program in the Pop up and hit Enter. Provide a global name starting with Z for your global class.



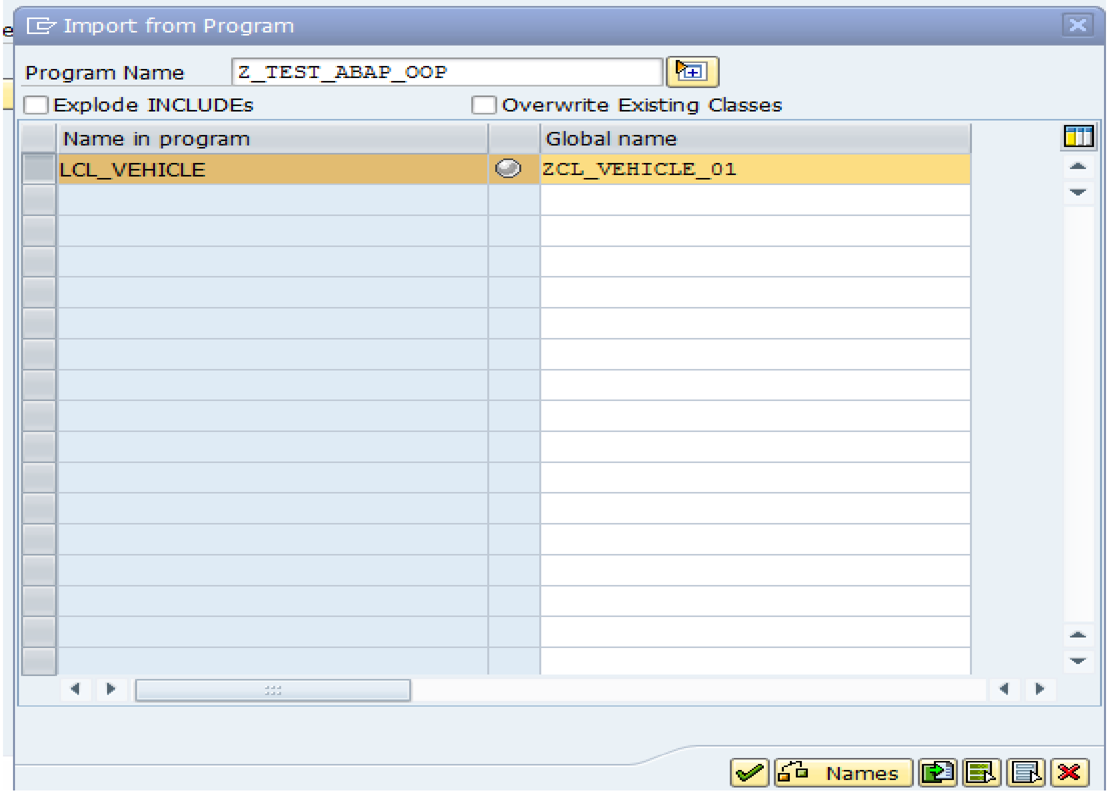
Import from Program

Program Name Z_TEST_ABAP_OOP

□ Explode INCLUDEs □ Overwrite Existing Classes

Name in program		Global name
LCL_VEHICLE	○	ZCL_VEHICLE_01

3. Select the class and click on import button.



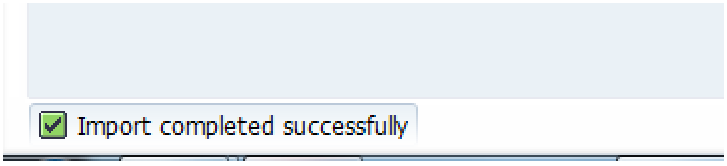
Import from Program

Program Name Z_TEST_ABAP_OOP

□ Explode INCLUDEs □ Overwrite Existing Classes

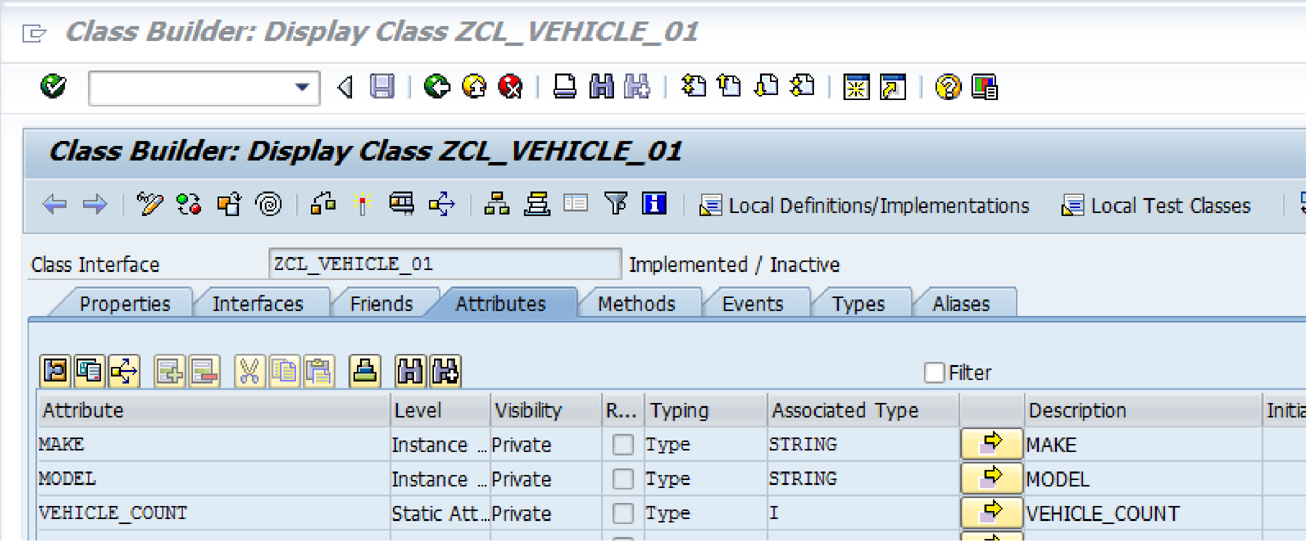
Name in program		Global name
LCL_VEHICLE	○	ZCL_VEHICLE_01

4. Save in a transport. Class is imported successfully.



☑ Import completed successfully

5. Display the class in class builder and Activate it.
   a. Attributes:

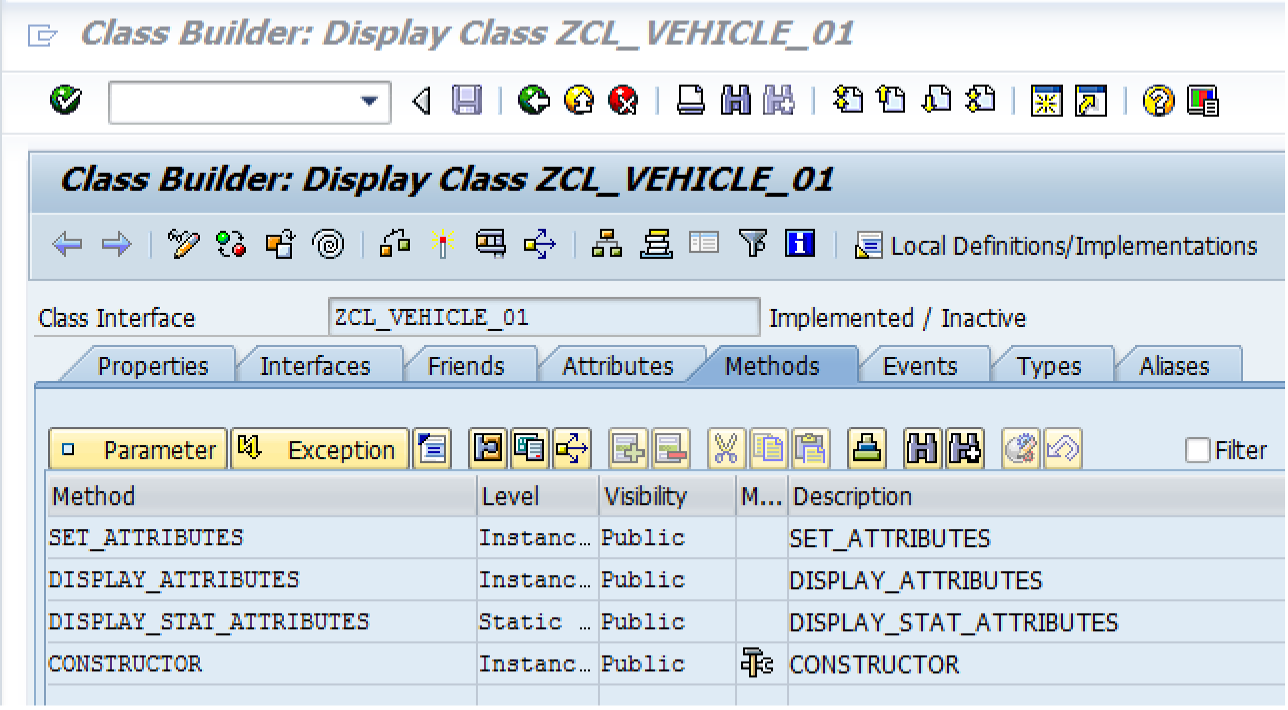


Class Builder: Display Class ZCL_VEHICLE_01

Class Interface ZCL_VEHICLE_01 Implemented / Inactive

Attribute	Level	Visibility	R...	Typing	Associated Type		Description	Initia
MAKE	Instance ...	Private	□	Type	STRING	⇨	MAKE	
MODEL	Instance ...	Private	□	Type	STRING	⇨	MODEL	
VEHICLE_COUNT	Static Att...	Private	□	Type	I	⇨	VEHICLE_COUNT	

b. Methods:


Class Builder: Display Class ZCL_VEHICLE_01
Class Builder: Display Class ZCL_VEHICLE_01
Local Definitions/Implementations
Properties
Interfaces
Friends
Attributes
Methods
Events
Types
Aliases
Parameter
Exception
Filter

Class Interface ZCL_VEHICLE_01 Implemented / Inactive

Method	Level	Visibility	M...	Description
SET_ATTRIBUTES	Instanc...	Public		SET_ATTRIBUTES
DISPLAY_ATTRIBUTES	Instanc...	Public		DISPLAY_ATTRIBUTES
DISPLAY_STAT_ATTRIBUTES	Static ...	Public		DISPLAY_STAT_ATTRIBUTES
CONSTRUCTOR	Instanc...	Public		CONSTRUCTOR

6. Declare and create the object reference of the class ZCL_VEHICLE_XX.

```
DATA  : lref_vehicle TYPE REF TO zcl_vehicle_01.

CREATE OBJECT lref_vehicle
  EXPORTING
    i_make  = 'BMW'
    i_model = 'M3'.
```

7. Call methods using the above object reference.

```
CALL METHOD zcl_vehicle_01=>display_stat_attributes.

CALL METHOD lref_vehicle->display_attributes.

CALL METHOD lref_vehicle->set_attributes
  EXPORTING
    i_make  = 'Audi'
    i_model = 'Q7'.

CALL METHOD lref_vehicle->display_attributes.
```

Speech balloon

About Capgemini

With more than 190,000 people, Capgemini is present in over 40 countries and celebrates its 50th Anniversary year in 2017. A global leader in consulting, technology and outsourcing services, the Group reported 2016 global revenues of EUR 12.5 billion. Together with its clients, Capgemini creates and delivers business, technology and digital solutions that fit their needs, enabling them to achieve innovation and competitiveness. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

Learn more about us at www.capgemini.com.

People matter, results count.

This message is intended only for the person to whom it is addressed. If you are not the intended recipient, you are not authorized to read, print, retain, copy, dissemina distribute, or use this message or any part thereof. If you receive this message in error, please notify the sender immediately and delete all copies of this message

This message contains information that may be privileged or confidential and is the property of the Capgemini Group.