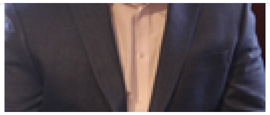
InformationTechnology
PARTNERS, INC

WANT TO AVOID HANA AMDP MISTAKES? READ THIS

| ABAP for HANA | ABAP Objects | SAP HANA | April 17, 2017

Jonathan Andre is a lead ABAP developer at IT Partners. Jon is also the President of Andre Technical Consulting L[illegible]ated out of Centreville, Virginia. Jon has over 6 years experience with SAP, with a concentration on ECC



As we described in a prior blog The ABAP Developer Road Map to SAP HANA, with the advent of SAP HANA, there has been a paradigm shift in the way business applications are developed. The rule-of-thumb is simple: Do as much as you can in the database to get the ***best performance***. This was coined as *"Code Pushdown"* by SAP. So far we have looked at CDS Views as a way to achieve *Code-Pushdown* in the blog Don't Try Coding ABAP Core Data Services Without Reading This First. In this blog, we will continue to examine *Code-Pushdown* Patterns, specifically **A**BAP **M**anaged **D**ata **P**rocedures (AMDP).

An ABAP Managed Data Procedure (AMDP) is a tool that can be utilized to create HANA database procedures that are designed and coded on the ABAP server. In the hierarchy of code-pushdown techniques, **AMDP ranks lowest on the scale of preference behind Open SQL and CDS Views.** While AMDPs are not the preferred approach when coding your code-to-data approach, the technology does offer some unique, albeit seldom used, approaches to interacting with the HANA database. AMDPs also have the added benefit of being transportable and are easy to adapt to for ABAP coders who need functionality Open SQL and CDS Views do not offer. (Note: AMDP code MUST be created and maintained in the Eclipse editor. Attempting to access this code from SAP GUI will not even allow the code to be switch to change mode after it implements the marker interface – more on this later)

We will go over some of the capabilities AMDPs have to offer, as well as how they can be combined with CDS to create powerful, yet easy to use tools.

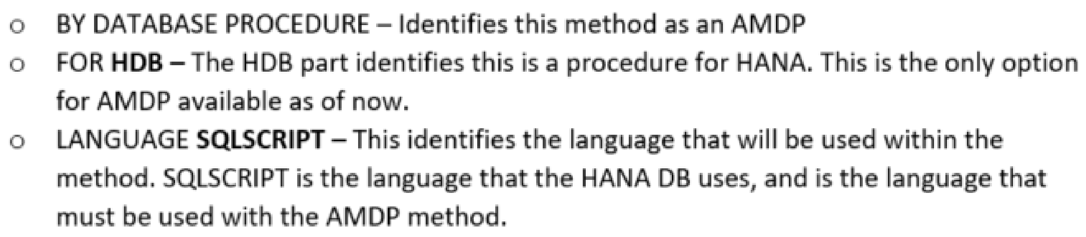
INTRODUCTION TO ABAP MANAGED DATA PROCEDURES (AMDP)

Let's look at the steps.

Step#1: First, they must ensure that the class implements the interface IF_AMDP_MARKER_HDB. Implementing this interface does not add any interface methods, but simply flags the code as an AMDP class.

Step#2: The Method that actually implements the AMDP procedure code must have some specialized method additions to identify itself as an AMDP. These additions also provide some information about the database and language the method should be implemented.

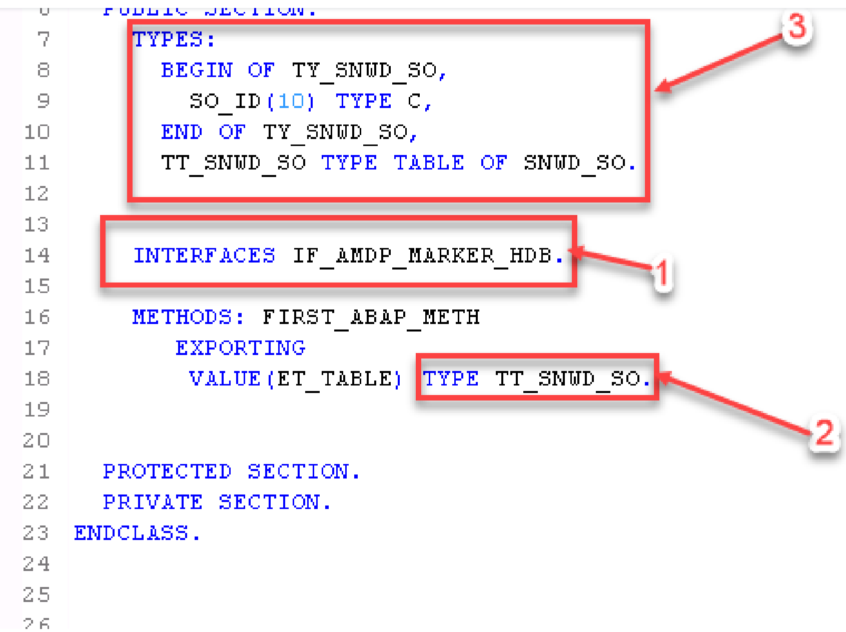
The Method Additions are described below:



- BY DATABASE PROCEDURE – Identifies this method as an AMDP
- FOR **HDB –** The HDB part identifies this is a procedure for HANA. This is the only option for AMDP available as of now.
- LANGUAGE **SQLSCRIPT** – This identifies the language that will be used within the method. SQLSCRIPT is the language that the HANA DB uses, and is the language that must be used with the AMDP method.

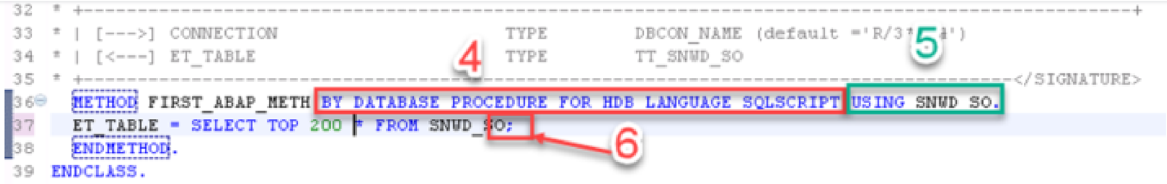
Below is a very basic example of an AMDP method. This method simply selects the 200 records from database table SNWD_SO, with no selection conditions. There are a few features to pay particular attention to, which have been highlighted and numbered.

CREATING YOUR FIRST AMDP



```
 PUBLIC SECTION.
  TYPES:
   BEGIN OF TY_SNWD_SO,
    SO_ID(10) TYPE C,
   END OF TY_SNWD_SO,
   TT_SNWD_SO TYPE TABLE OF SNWD_SO.


  INTERFACES IF_AMDP_MARKER_HDB.

  METHODS: FIRST_ABAP_METH
    EXPORTING
     VALUE(ET_TABLE) TYPE TT_SNWD_SO.


 PROTECTED SECTION.
 PRIVATE SECTION.
ENDCLASS.
```

1. The INTERFACES IF_AMDP_MARKER_HDB statement comes within the PUBLIC SECTION. This may or may not be obvious to you, even if you have worked with OOP on ABAP before. Since this interfaces insertion was usually handled by the Class Development Tool in SE24, this is something that should be noted.
2. When defining your exporting table types within your AMDP class, note that the TYPE TABLE OF statements are not allowed. You will therefore either have to use a DDIC table type or use a TYPES statement and declare you table type prior to your method declaration (as I did in 3). Also note that all IMPORTING and EXPORTING variables must be PASS BY VALUE! Considering that this code will be executed on the HANA DB, it makes sense you cannot pass by reference to another system.
3. In addition to not allowing TYPE TABLE OF statements in the method declaration, the AMDP classes are fairly finicky in general. You can assume that HANA has no access to DDIC types in general, so you should



```
* +--------------------------------------------------------------------------------------+
* | [--->] CONNECTION                  TYPE        DBCON_NAME (default ='R/3*')
* | [<---] ET_TABLE                    TYPE        TT_SNWD_SO
* +------------------------------------------------------------------------</SIGNATURE>
  METHOD FIRST_ABAP_METH BY DATABASE PROCEDURE FOR HDB LANGUAGE SQLSCRIPT USING SNWD_SO.
  ET_TABLE = SELECT TOP 200 * FROM SNWD_SO;
  ENDMETHOD.
ENDCLASS.
```

4. This is the boilerplate text I described in the beginning of the blog when defining an AMDP method. It seems that SAP would like to expand the functionality of AMDPs in the future to support other database types, but as of now this is the only text you will for these methods (other than for CDS Table Functions, which will be covered later). This statement describes the database and the language.
5. You can assume that any code within your AMDP method has NO KNOWLEDGE of any DDIC types. To specify the use of a table or view, you need to call out the tables that will be in used as part of the method header. USING <table_name> will allow you to SELECT, UPDATE, INSERT, or MODIFY <table_name> within your managed procedure call.
6. As stated above, the body of the method must be written in SQLScript. SQLScript does share some similarities with ABAP at a glance, but it is a different language. Note that statements in SQLScript are terminated with ";" and not "."

The above basic example would not be justification enough to use an AMDP, since this could obvious be easily accomplished using either Open SQL or CDS Views. So how do we know what is the best ***Code-to-Data*** or ***Code-Pushdown*** Technique?

Below is a cursory set of Guidelines. This is not meant to be a definitive guide, but rather a starting point.

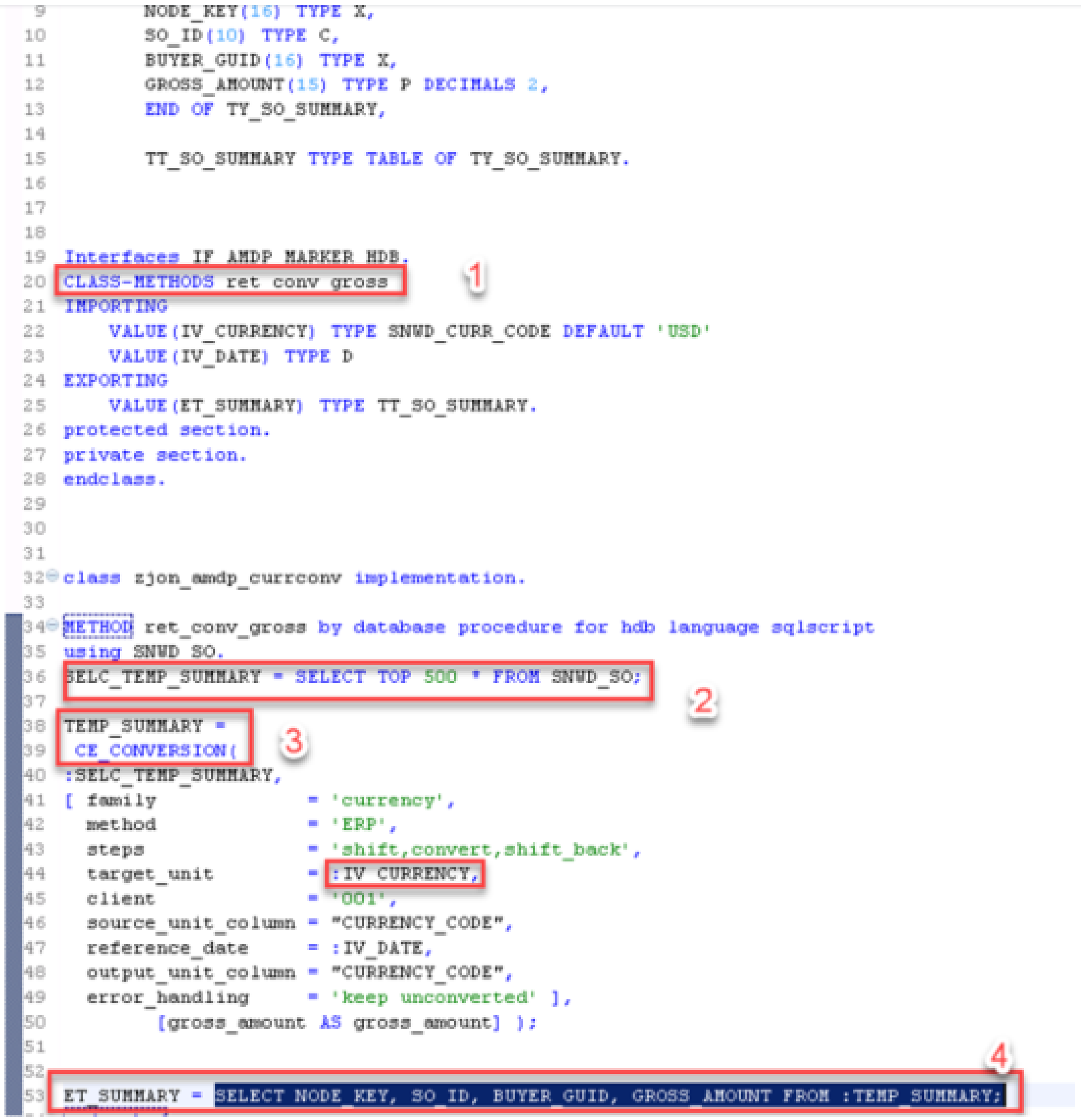
CDS Views

Only ONE result set can be returned from a CDS View

AMDP
Independent SQL Statement those are not often used in other objects
MULTIPLE result sets are needed
Powerful features of native SQL such as currency conversion and CE functions can be leveraged.

Open SQL
If the SQL queries are for the specific object and won't be needed elsewhere (not reusable)

OK, The basic example would not be justification enough to use an AMDP. **BUT** a reason to use AMDP, however, is its ability to access Native SQL Script and internal **HANA functions**. I will replicate some code from the previous blog while correcting a cardinal coding sin: ignoring currency conversion. In the following code, I will attempt to address this issue, while also pointing out some of the other perks of using AMDPs.



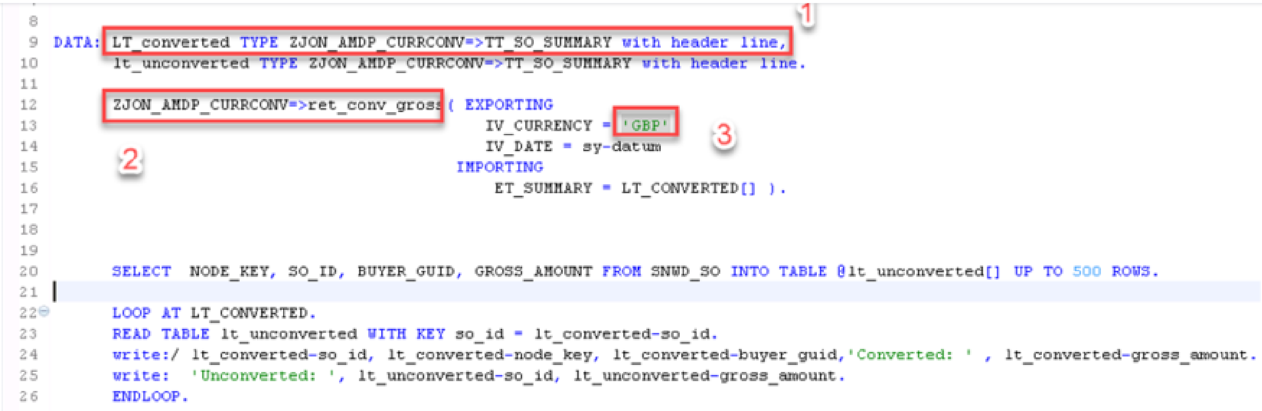
```
        NODE_KEY(16) TYPE X,
        SO_ID(10) TYPE C,
        BUYER_GUID(16) TYPE X,
        GROSS_AMOUNT(15) TYPE P DECIMALS 2,
        END OF TY_SO_SUMMARY,

        TT_SO_SUMMARY TYPE TABLE OF TY_SO_SUMMARY.



Interfaces IF_AMDP_MARKER_HDB.
CLASS-METHODS ret_conv_gross
IMPORTING
    VALUE(IV_CURRENCY) TYPE SNWD_CURR_CODE DEFAULT 'USD'
    VALUE(IV_DATE) TYPE D
EXPORTING
    VALUE(ET_SUMMARY) TYPE TT_SO_SUMMARY.
protected section.
private section.
endclass.



class zjon_amdp_currconv implementation.

METHOD ret_conv_gross by database procedure for hdb language sqlscript
using SNWD_SO.
SELC_TEMP_SUMMARY = SELECT TOP 500 * FROM SNWD_SO;

TEMP_SUMMARY =
 CE_CONVERSION(
:SELC_TEMP_SUMMARY,
[ family             = 'currency',
  method             = 'ERP',
  steps              = 'shift,convert,shift_back',
  target_unit        = :IV_CURRENCY,
  client             = '001',
  source_unit_column = "CURRENCY_CODE",
  reference_date     = :IV_DATE,
  output_unit_column = "CURRENCY_CODE",
  error_handling     = 'keep unconverted' ],
        [gross_amount AS gross_amount] );


ET_SUMMARY = SELECT NODE_KEY, SO_ID, BUYER_GUID, GROSS_AMOUNT FROM :TEMP_SUMMARY;
```

method into a function module call.

2. A nice feature of AMDP classes is that you can define variables on the fly. SELC_TEMP_SUMMARY will automatically take on the types and structure returned by the SELECT.
3. This conversion works for functions as well. The return value of CE_CONVERSION is variable depending on what value is passed, so TEMP_SUMMARY will take on the structure necessary to hold what is returned. Also, note that to reference an internal objects values, you must prefix the variable name with ":". In this scenario we use the values from :SELC_TEMP_SUMMARY as input to CE_CONVERSION, and stored the generated output in TEMP_SUMMARY.
4. Here we find another useful feature of SQLScript/AMDP. We can perform SELECTs on our generated "internal tables" as if they were database tables, and store the results. The select uses the TEMP_SUMMARY (with the required ":" in front) and pulls back only the fields required to populate our returning table ET_SUMMARY.

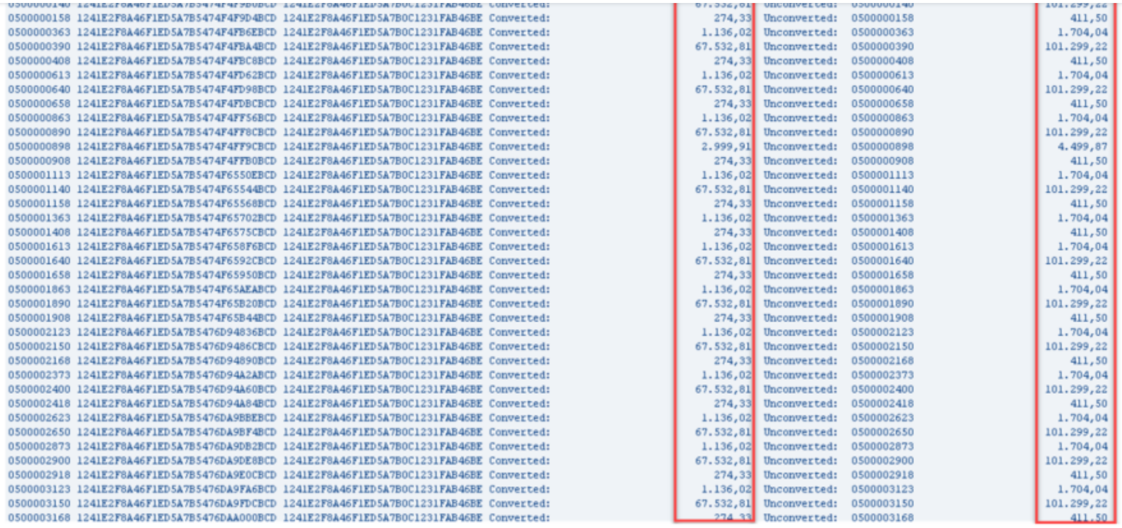
INTEGRATING AN AMDP INTO YOUR ABAP CODE

With our class built, we can now use a simple ABAP program to pull back pertinent information from SNWD_SO and display it. The sample program below will use the class method (using a static method call) to pull back the records from SNWD_SO with the gross amount converted into British Pounds ('GBP'). These records will be displayed alongside the unconverted gross_amount values, so we can compare the two.



```
DATA: LT_converted TYPE ZJON_AMDP_CURRCONV=>TT_SO_SUMMARY with header line,
      lt_unconverted TYPE ZJON_AMDP_CURRCONV=>TT_SO_SUMMARY with header line.

      ZJON_AMDP_CURRCONV=>ret_conv_gross( EXPORTING
                                            IV_CURRENCY = 'GBP'
                                            IV_DATE = sy-datum
                                          IMPORTING
                                             ET_SUMMARY = LT_CONVERTED[] ).



      SELECT  NODE_KEY, SO_ID, BUYER_GUID, GROSS_AMOUNT FROM SNWD_SO INTO TABLE @lt_unconverted[] UP TO 500 ROWS.

      LOOP AT LT_CONVERTED.
      READ TABLE lt_unconverted WITH KEY so_id = lt_converted-so_id.
      write:/ lt_converted-so_id, lt_converted-node_key, lt_converted-buyer_guid,'Converted: ' , lt_converted-gross_amount.
      write:  'Unconverted: ', lt_unconverted-so_id, lt_unconverted-gross_amount.
      ENDLOOP.
```

1. As OOP refresher, recall that since we defined our TT_SO_SUMMARY type in the public section ZJON_AMDP_CURRCONV class, we can directly reference it from our program with the above syntax. This prevents type mismatches and also prevents you from having to create this TYPE again in your program.
2. As mentioned above, since we defined this method as a “CLASS-METHODS” type method, we can reference our method without creating an instance. This allows us to call this method in the manner resembling a normal function call.
3. Note that we are specifying that we want the currency of the GROSS_AMOUNT converted from its listed table currency (currently all in ‘USD’) into ‘GBP’. We can obviously reuse this method and pass in any target currency we desire.

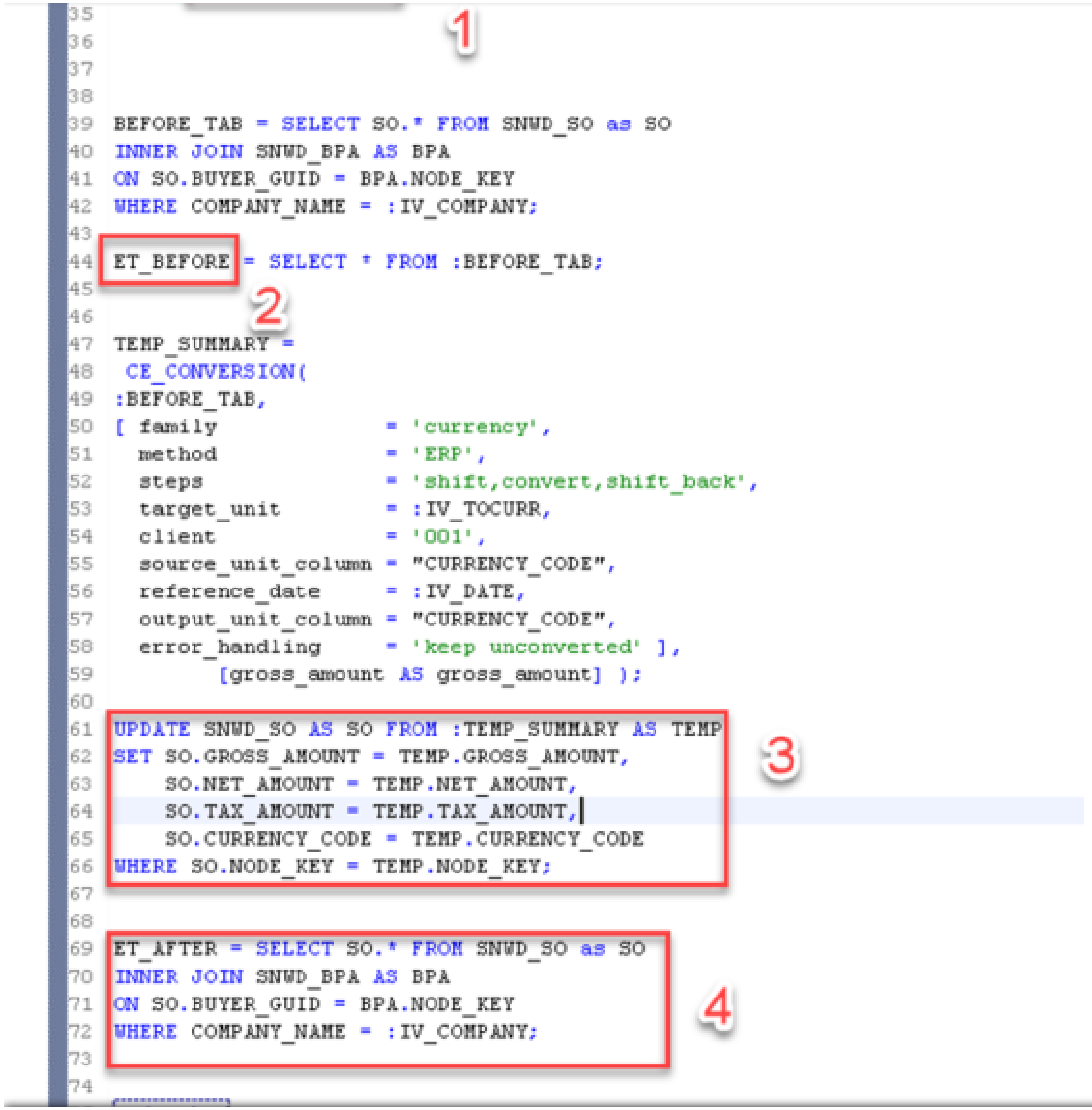


```
0500000140 1241E2F8A46F1ED5A7B5474F4F9B0BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:  67.532,81 Unconverted:  0500000140             101.299,22
0500000158 1241E2F8A46F1ED5A7B5474F4F9D4BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:     274,33 Unconverted:  0500000158                 411,50
0500000363 1241E2F8A46F1ED5A7B5474F4FB6EBCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:   1.136,02 Unconverted:  0500000363               1.704,04
0500000390 1241E2F8A46F1ED5A7B5474F4FBA4BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:  67.532,81 Unconverted:  0500000390             101.299,22
0500000408 1241E2F8A46F1ED5A7B5474F4FBC8BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:     274,33 Unconverted:  0500000408                 411,50
0500000613 1241E2F8A46F1ED5A7B5474F4FD62BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:   1.136,02 Unconverted:  0500000613               1.704,04
0500000640 1241E2F8A46F1ED5A7B5474F4FD98BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:  67.532,81 Unconverted:  0500000640             101.299,22
0500000658 1241E2F8A46F1ED5A7B5474F4FDBCBCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:     274,33 Unconverted:  0500000658                 411,50
0500000863 1241E2F8A46F1ED5A7B5474F4FF56BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:   1.136,02 Unconverted:  0500000863               1.704,04
0500000890 1241E2F8A46F1ED5A7B5474F4FF8CBCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:  67.532,81 Unconverted:  0500000890             101.299,22
0500000898 1241E2F8A46F1ED5A7B5474F4FF9CBCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:   2.999,91 Unconverted:  0500000898               4.499,87
0500000908 1241E2F8A46F1ED5A7B5474F4FFB0BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:     274,33 Unconverted:  0500000908                 411,50
0500001113 1241E2F8A46F1ED5A7B5474F6550EBCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:   1.136,02 Unconverted:  0500001113               1.704,04
0500001140 1241E2F8A46F1ED5A7B5474F65544BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:  67.532,81 Unconverted:  0500001140             101.299,22
0500001158 1241E2F8A46F1ED5A7B5474F65568BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:     274,33 Unconverted:  0500001158                 411,50
0500001363 1241E2F8A46F1ED5A7B5474F65702BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:   1.136,02 Unconverted:  0500001363               1.704,04
0500001408 1241E2F8A46F1ED5A7B5474F6575CBCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:     274,33 Unconverted:  0500001408                 411,50
0500001613 1241E2F8A46F1ED5A7B5474F658F6BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:   1.136,02 Unconverted:  0500001613               1.704,04
0500001640 1241E2F8A46F1ED5A7B5474F6592CBCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:  67.532,81 Unconverted:  0500001640             101.299,22
0500001658 1241E2F8A46F1ED5A7B5474F65950BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:     274,33 Unconverted:  0500001658                 411,50
0500001863 1241E2F8A46F1ED5A7B5474F65AEABCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:   1.136,02 Unconverted:  0500001863               1.704,04
0500001890 1241E2F8A46F1ED5A7B5474F65B20BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:  67.532,81 Unconverted:  0500001890             101.299,22
0500001908 1241E2F8A46F1ED5A7B5474F65B44BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:     274,33 Unconverted:  0500001908                 411,50
0500002123 1241E2F8A46F1ED5A7B5476D94836BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:   1.136,02 Unconverted:  0500002123               1.704,04
0500002150 1241E2F8A46F1ED5A7B5476D9486CBCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:  67.532,81 Unconverted:  0500002150             101.299,22
0500002168 1241E2F8A46F1ED5A7B5476D94890BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:     274,33 Unconverted:  0500002168                 411,50
0500002373 1241E2F8A46F1ED5A7B5476D94A2ABCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:   1.136,02 Unconverted:  0500002373               1.704,04
0500002400 1241E2F8A46F1ED5A7B5476D94A60BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:  67.532,81 Unconverted:  0500002400             101.299,22
0500002418 1241E2F8A46F1ED5A7B5476D94A84BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:     274,33 Unconverted:  0500002418                 411,50
0500002623 1241E2F8A46F1ED5A7B5476DA9BBEBCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:   1.136,02 Unconverted:  0500002623               1.704,04
0500002650 1241E2F8A46F1ED5A7B5476DA9BF4BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:  67.532,81 Unconverted:  0500002650             101.299,22
0500002873 1241E2F8A46F1ED5A7B5476DA9DB2BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:   1.136,02 Unconverted:  0500002873               1.704,04
0500002900 1241E2F8A46F1ED5A7B5476DA9DE8BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:  67.532,81 Unconverted:  0500002900             101.299,22
0500002918 1241E2F8A46F1ED5A7B5476DA9E0CBCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:     274,33 Unconverted:  0500002918                 411,50
0500003123 1241E2F8A46F1ED5A7B5476DA9FA6BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:   1.136,02 Unconverted:  0500003123               1.704,04
0500003150 1241E2F8A46F1ED5A7B5476DA9FDCBCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:  67.532,81 Unconverted:  0500003150             101.299,22
0500003168 1241E2F8A46F1ED5A7B5476DAA000BCD 1241E2F8A46F1ED5A7B0C1231FAB46BE Converted:     274,33 Unconverted:  0500003168                 411,50
```

USING AN AMDP TO RETRIEVE MULTIPLE RESULTS SETS

Some other abilities that set AMDPs apart from Open SQL and CDS Views is their ability to retrieve multiple result sets and provide a reusable way to mass insert or update records.

The next example will illustrate these capabilities in one AMDP method. This scenario will assume we wish to convert all sales orders from SAP within the SNWD_SO table to 'EUR'. To do this, we want to convert the gross_amount, net_amount, and tax_amount to 'EUR', in addition to updating the record to the correct currency code. We will also return a before-conversion table and an after-conversion table.

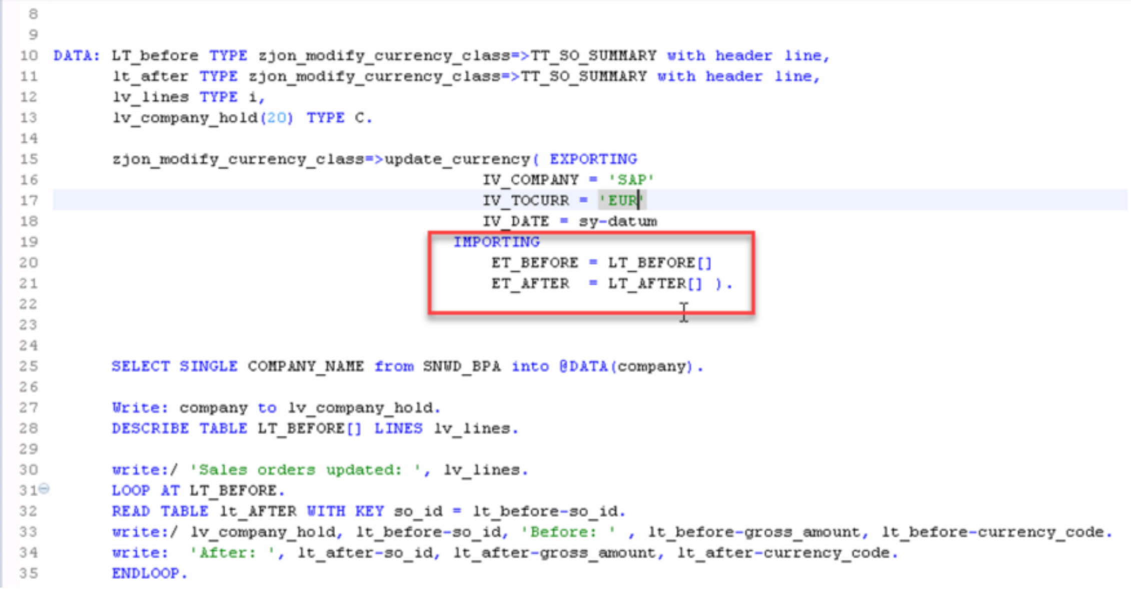


```
BEFORE_TAB = SELECT SO.* FROM SNWD_SO as SO
INNER JOIN SNWD_BPA AS BPA
ON SO.BUYER_GUID = BPA.NODE_KEY
WHERE COMPANY_NAME = :IV_COMPANY;

ET_BEFORE = SELECT * FROM :BEFORE_TAB;


TEMP_SUMMARY =
 CE_CONVERSION(
:BEFORE_TAB,
[ family             = 'currency',
  method             = 'ERP',
  steps              = 'shift,convert,shift_back',
  target_unit        = :IV_TOCURR,
  client             = '001',
  source_unit_column = "CURRENCY_CODE",
  reference_date     = :IV_DATE,
  output_unit_column = "CURRENCY_CODE",
  error_handling     = 'keep unconverted' ],
        [gross_amount AS gross_amount] );

UPDATE SNWD_SO AS SO FROM :TEMP_SUMMARY AS TEMP
SET SO.GROSS_AMOUNT = TEMP.GROSS_AMOUNT,
    SO.NET_AMOUNT = TEMP.NET_AMOUNT,
    SO.TAX_AMOUNT = TEMP.TAX_AMOUNT,
    SO.CURRENCY_CODE = TEMP.CURRENCY_CODE
WHERE SO.NODE_KEY = TEMP.NODE_KEY;


ET_AFTER = SELECT SO.* FROM SNWD_SO as SO
INNER JOIN SNWD_BPA AS BPA
ON SO.BUYER_GUID = BPA.NODE_KEY
WHERE COMPANY_NAME = :IV_COMPANY;
```

or a CDS view.

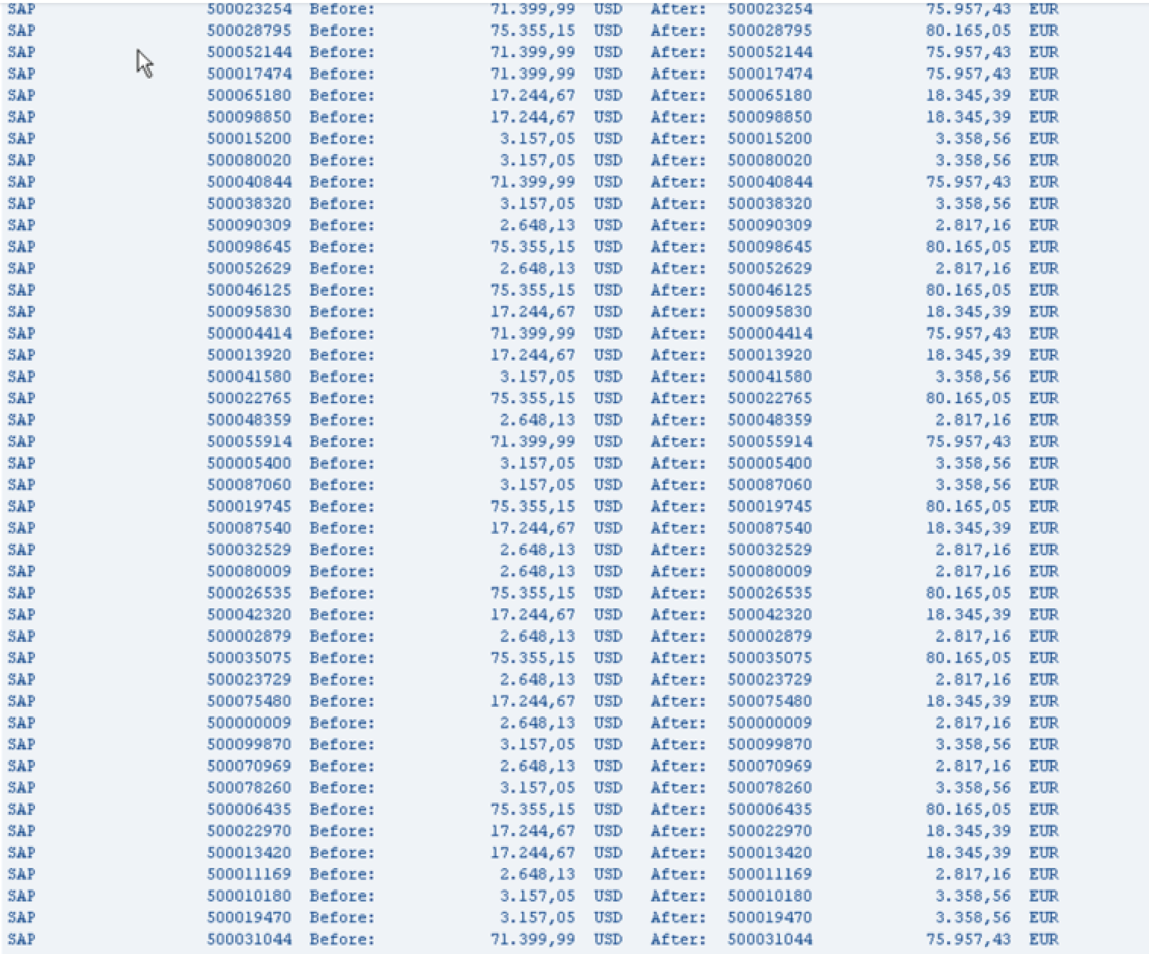
3. After modifying the records pulled back from the SNWD_SO table, we are able to modify SNWD_SO database records based on a matching NODE_KEY.
4. We now create our second return table, which is derived by performing another select on the now updated SNWD_SO table. This is a capability unique to AMDPs, and proves useful in this scenario and other scenarios where multiple sets of data would need to be retrieved in unison.

Let's use this in an ABAP Report. The report source created to utilize this class is straightforward and closely resembles the original AMDP currency conversion program. I have highlighted the fact that this program is able to pull back to different sets of data from one statement. We could run this program for different companies, or different currencies without modifying the class.



```
DATA: LT_before TYPE zjon_modify_currency_class=>TT_SO_SUMMARY with header line,
      lt_after TYPE zjon_modify_currency_class=>TT_SO_SUMMARY with header line,
      lv_lines TYPE i,
      lv_company_hold(20) TYPE C.

      zjon_modify_currency_class=>update_currency( EXPORTING
                                      IV_COMPANY = 'SAP'
                                      IV_TOCURR = 'EUR'
                                      IV_DATE = sy-datum
                                   IMPORTING
                                       ET_BEFORE = LT_BEFORE[]
                                       ET_AFTER  = LT_AFTER[] ).



      SELECT SINGLE COMPANY_NAME from SNWD_BPA into @DATA(company).

      Write: company to lv_company_hold.
      DESCRIBE TABLE LT_BEFORE[] LINES lv_lines.

      write:/ 'Sales orders updated: ', lv_lines.
      LOOP AT LT_BEFORE.
      READ TABLE lt_AFTER WITH KEY so_id = lt_before-so_id.
      write:/ lv_company_hold, lt_before-so_id, 'Before: ' , lt_before-gross_amount, lt_before-currency_code.
      write:  'After: ', lt_after-so_id, lt_after-gross_amount, lt_after-currency_code.
      ENDLOOP.
```

The resulting output displays both the before and after for all ‘SAP’ sales orders in the SNWD_SO table

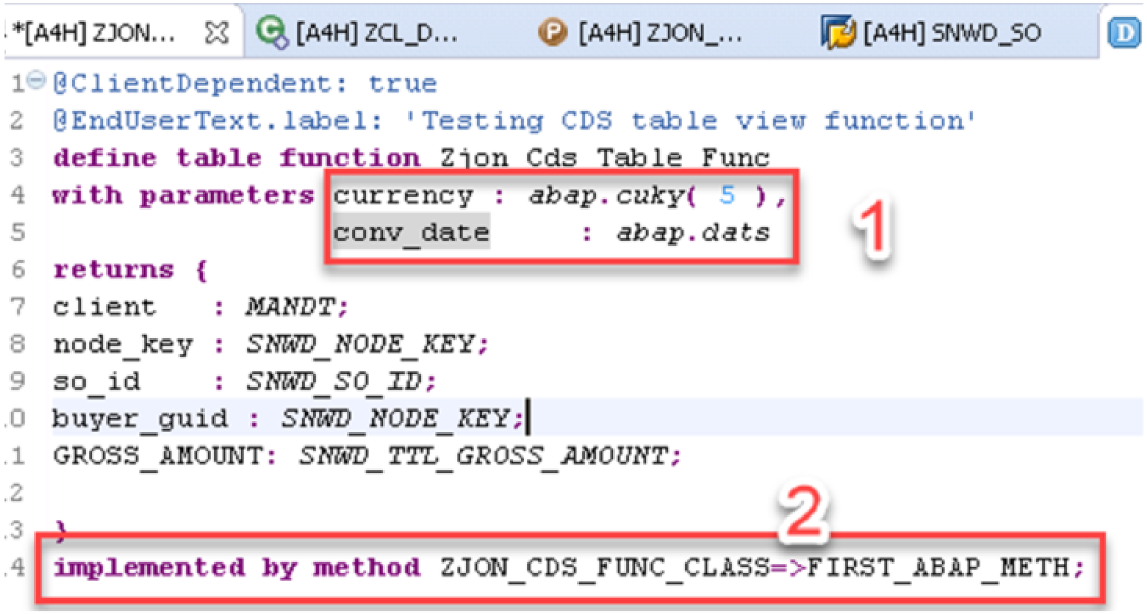


```
SAP          500023254  Before:      71.399,99  USD   After:  500023254      75.957,43  EUR
SAP          500028795  Before:      75.355,15  USD   After:  500028795      80.165,05  EUR
SAP          500052144  Before:      71.399,99  USD   After:  500052144      75.957,43  EUR
SAP          500017474  Before:      71.399,99  USD   After:  500017474      75.957,43  EUR
SAP          500065180  Before:      17.244,67  USD   After:  500065180      18.345,39  EUR
SAP          500098850  Before:      17.244,67  USD   After:  500098850      18.345,39  EUR
SAP          500015200  Before:       3.157,05  USD   After:  500015200       3.358,56  EUR
SAP          500080020  Before:       3.157,05  USD   After:  500080020       3.358,56  EUR
SAP          500040844  Before:      71.399,99  USD   After:  500040844      75.957,43  EUR
SAP          500038320  Before:       3.157,05  USD   After:  500038320       3.358,56  EUR
SAP          500090309  Before:       2.648,13  USD   After:  500090309       2.817,16  EUR
SAP          500098645  Before:      75.355,15  USD   After:  500098645      80.165,05  EUR
SAP          500052629  Before:       2.648,13  USD   After:  500052629       2.817,16  EUR
SAP          500046125  Before:      75.355,15  USD   After:  500046125      80.165,05  EUR
SAP          500095830  Before:      17.244,67  USD   After:  500095830      18.345,39  EUR
SAP          500004414  Before:      71.399,99  USD   After:  500004414      75.957,43  EUR
SAP          500013920  Before:      17.244,67  USD   After:  500013920      18.345,39  EUR
SAP          500041580  Before:       3.157,05  USD   After:  500041580       3.358,56  EUR
SAP          500022765  Before:      75.355,15  USD   After:  500022765      80.165,05  EUR
SAP          500048359  Before:       2.648,13  USD   After:  500048359       2.817,16  EUR
SAP          500055914  Before:      71.399,99  USD   After:  500055914      75.957,43  EUR
SAP          500005400  Before:       3.157,05  USD   After:  500005400       3.358,56  EUR
SAP          500087060  Before:       3.157,05  USD   After:  500087060       3.358,56  EUR
SAP          500019745  Before:      75.355,15  USD   After:  500019745      80.165,05  EUR
SAP          500087540  Before:      17.244,67  USD   After:  500087540      18.345,39  EUR
SAP          500032529  Before:       2.648,13  USD   After:  500032529       2.817,16  EUR
SAP          500080009  Before:       2.648,13  USD   After:  500080009       2.817,16  EUR
SAP          500026535  Before:      75.355,15  USD   After:  500026535      80.165,05  EUR
SAP          500042320  Before:      17.244,67  USD   After:  500042320      18.345,39  EUR
SAP          500002879  Before:       2.648,13  USD   After:  500002879       2.817,16  EUR
SAP          500035075  Before:      75.355,15  USD   After:  500035075      80.165,05  EUR
SAP          500023729  Before:       2.648,13  USD   After:  500023729       2.817,16  EUR
SAP          500075480  Before:      17.244,67  USD   After:  500075480      18.345,39  EUR
SAP          500000009  Before:       2.648,13  USD   After:  500000009       2.817,16  EUR
SAP          500099870  Before:       3.157,05  USD   After:  500099870       3.358,56  EUR
SAP          500070969  Before:       2.648,13  USD   After:  500070969       2.817,16  EUR
SAP          500078260  Before:       3.157,05  USD   After:  500078260       3.358,56  EUR
SAP          500006435  Before:      75.355,15  USD   After:  500006435      80.165,05  EUR
SAP          500022970  Before:      17.244,67  USD   After:  500022970      18.345,39  EUR
SAP          500013420  Before:      17.244,67  USD   After:  500013420      18.345,39  EUR
SAP          500011169  Before:       2.648,13  USD   After:  500011169       2.817,16  EUR
SAP          500010180  Before:       3.157,05  USD   After:  500010180       3.358,56  EUR
SAP          500019470  Before:       3.157,05  USD   After:  500019470       3.358,56  EUR
SAP          500031044  Before:      71.399,99  USD   After:  500031044      75.957,43  EUR
```

The examples above show that AMDPs have a lot to offer in terms of really taking advantage of HANAs features. As far simplicity goes however, AMDPs simply do not offer the ease of use offered by CDS or Open SQL. Thankfully, SAP introduced another technology called CDS table functions, which provide a familiar and

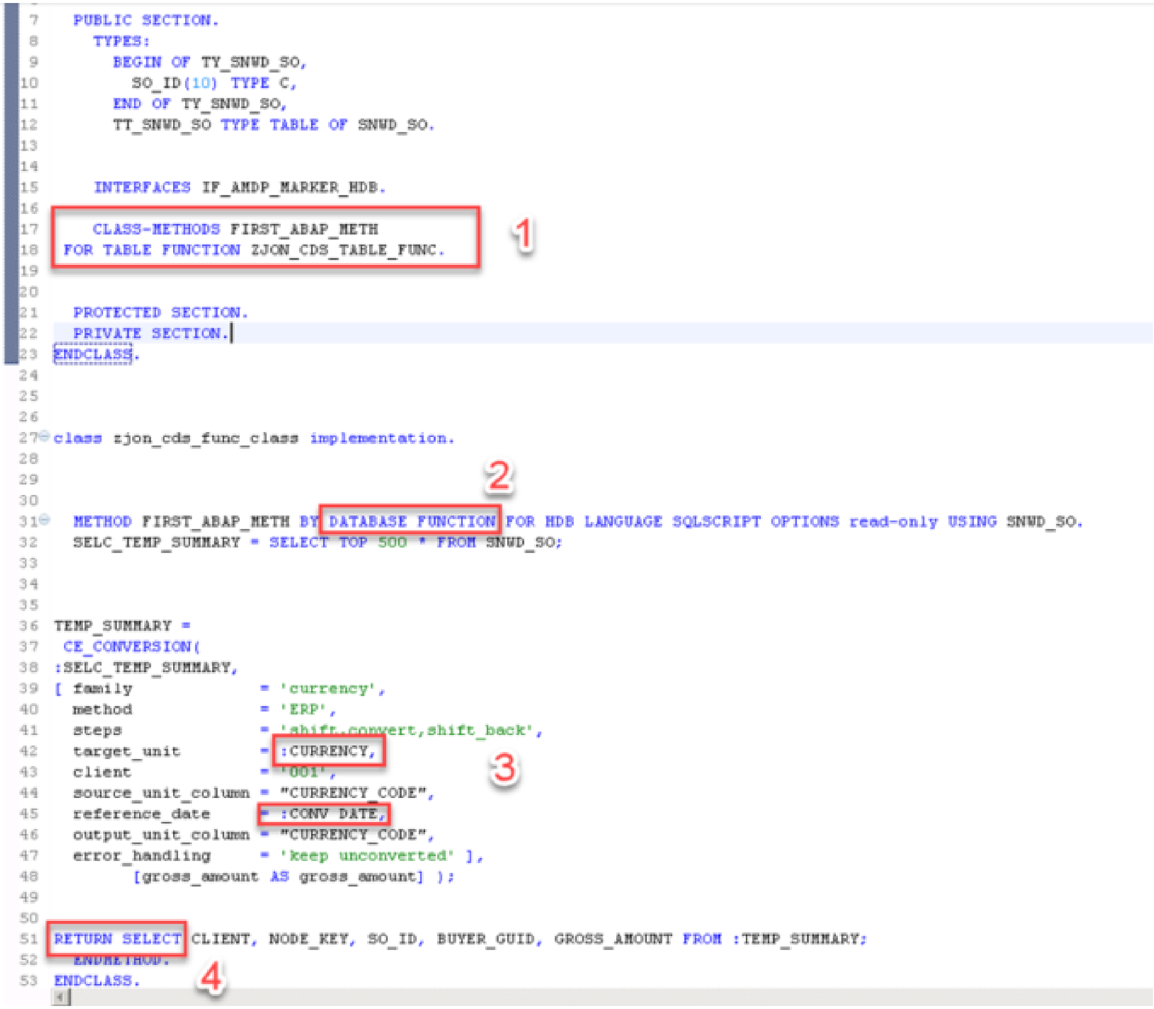
CDS table functions are essentially wrappers for AMDPs that make them more user-friendly for developers. By using a CDS table function, users are able to access AMDPs with CDS View-like SQL syntax, and my not even realize that an AMDP is being used behind the scenes. The next example will illustrate how CDS table functions and AMDPs interact, in addition to demonstrating how to call a CDS table function.

Lets, build the CDS Table Function (see below)



```
@ClientDependent: true
@EndUserText.label: 'Testing CDS table view function'
define table function Zjon_Cds_Table_Func
with parameters currency : abap.cuky( 5 ),
                conv_date      : abap.dats
returns {
client     : MANDT;
node_key : SNWD_NODE_KEY;
so_id      : SNWD_SO_ID;
buyer_guid : SNWD_NODE_KEY;
GROSS_AMOUNT: SNWD_TTL_GROSS_AMOUNT;


}
implemented by method ZJON_CDS_FUNC_CLASS=>FIRST_ABAP_METH;
```

1. CDS Parameters are passed implicitly to the AMDPs methods parameters that implements the CDS table function. Therefore, if you want the table function to have importing parameters, this is the place to do it.
2. The CDS table function is strictly bound to a specific method of a specific class. You can name that method



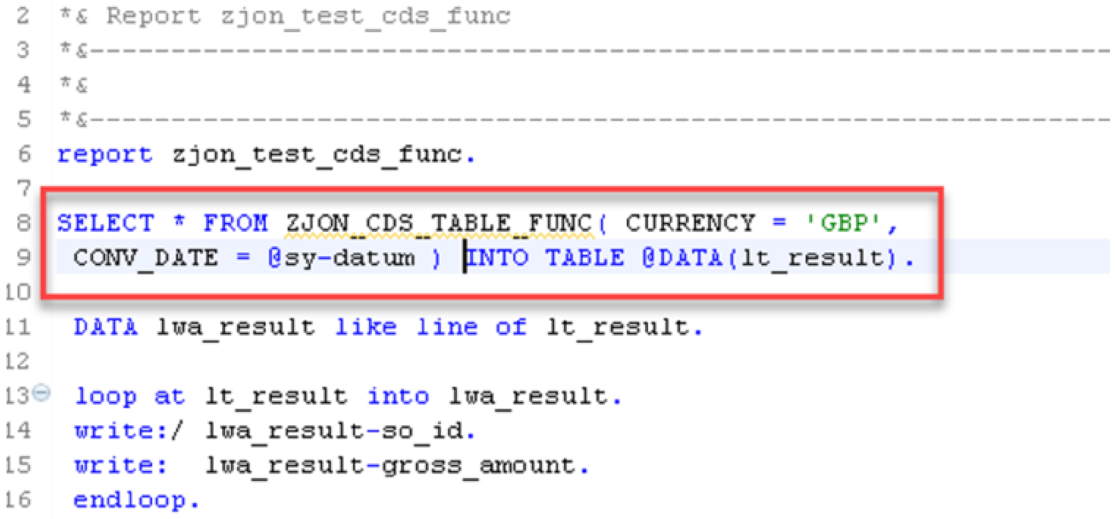
```
  PUBLIC SECTION.
    TYPES:
      BEGIN OF TY_SNWD_SO,
        SO_ID(10) TYPE C,
      END OF TY_SNWD_SO,
      TT_SNWD_SO TYPE TABLE OF SNWD_SO.


    INTERFACES IF_AMDP_MARKER_HDB.

    CLASS-METHODS FIRST_ABAP_METH
 FOR TABLE FUNCTION ZJON_CDS_TABLE_FUNC.


  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.



class zjon_cds_func_class implementation.



  METHOD FIRST_ABAP_METH BY DATABASE FUNCTION FOR HDB LANGUAGE SQLSCRIPT OPTIONS read-only USING SNWD_SO.
  SELC_TEMP_SUMMARY = SELECT TOP 500 * FROM SNWD_SO;



TEMP_SUMMARY =
 CE_CONVERSION(
:SELC_TEMP_SUMMARY,
[ family             = 'currency',
  method             = 'ERP',
  steps              = 'shift,convert,shift_back',
  target_unit        = :CURRENCY,
  client             = '001',
  source_unit_column = "CURRENCY_CODE",
  reference_date     = :CONV_DATE,
  output_unit_column = "CURRENCY_CODE",
  error_handling     = 'keep unconverted' ],
        [gross_amount AS gross_amount] );


RETURN SELECT CLIENT, NODE_KEY, SO_ID, BUYER_GUID, GROSS_AMOUNT FROM :TEMP_SUMMARY;
  ENDMETHOD.
ENDCLASS.
```

1. Note that the implementing method of the implementing class for the CDS table function MUST be a CLASS METHOD. In addition, the method declaration must include the syntax FOR TABLE FUNCTION

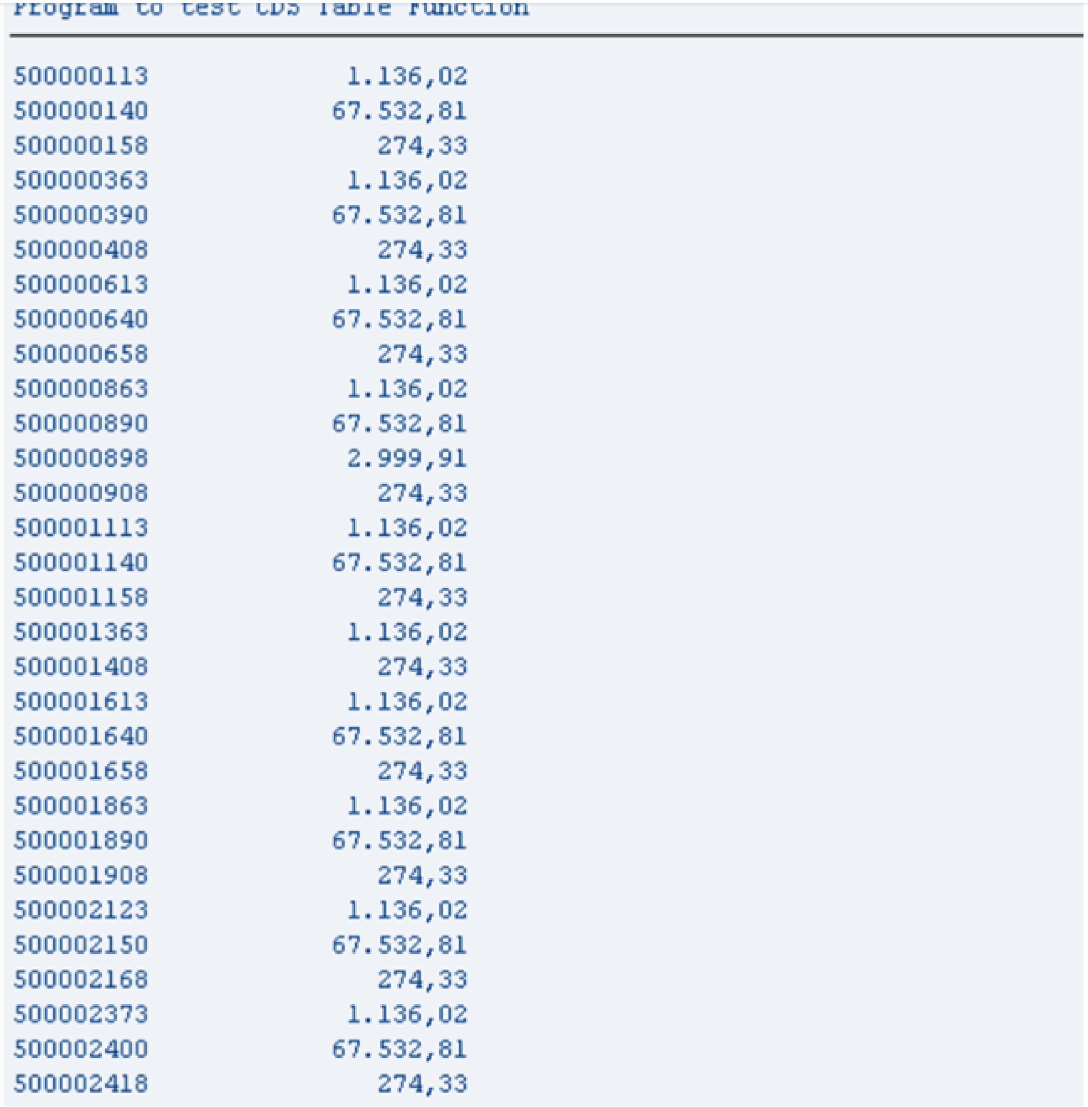
as we have used up until this point, the syntax for this method is DATABASE FUNCTION. Leaving this change off will prevent your code from compiling.

3. Even though the parameters CURRENCY and CONV_DATE are defined nowhere in the CLASS or METHOD, the method already knows and can use the parameters that were defined in the CDS table function.
4. Lastly, a method that implements a CDS table function MUST return exactly one table as returning value. While all AMDPs can have a returning parameter (as of ABAP 7.50), it is required for this table function.

To call a CDS Table Function, simply perform a SELECT as you would for any ABAP CDS entity. Note that the code is using the AMDP, yet does not need to mention the class or method anywhere.



```
*& Report zjon_test_cds_func
*&---------------------------------------------------------------------
*&
*&---------------------------------------------------------------------
report zjon_test_cds_func.

SELECT * FROM ZJON_CDS_TABLE_FUNC( CURRENCY = 'GBP',
 CONV_DATE = @sy-datum ) INTO TABLE @DATA(lt_result).

 DATA lwa_result like line of lt_result.

 loop at lt_result into lwa_result.
 write:/ lwa_result-so_id.
 write:  lwa_result-gross_amount.
 endloop.
```



Program to test CDS Table Function

```
500000113             1.136,02
500000140            67.532,81
500000158               274,33
500000363             1.136,02
500000390            67.532,81
500000408               274,33
500000613             1.136,02
500000640            67.532,81
500000658               274,33
500000863             1.136,02
500000890            67.532,81
500000898             2.999,91
500000908               274,33
500001113             1.136,02
500001140            67.532,81
500001158               274,33
500001363             1.136,02
500001408               274,33
500001613             1.136,02
500001640            67.532,81
500001658               274,33
500001863             1.136,02
500001890            67.532,81
500001908               274,33
500002123             1.136,02
500002150            67.532,81
500002168               274,33
500002373             1.136,02
500002400            67.532,81
500002418               274,33
```

AMDPs provide powerful abstractions that enable ABAP developers to combine the benefits of the high-speed in-memory execution of SAP HANA with the well-established ABAP runtime and lifecycle model. Along with CDS Views, CDS Table Functions, and OPEN SQL, allow ABAP developers to renovate their existing assets and create new and innovative applications for SAP Business Suite on SAP HANA without losing the platform-independence of ABAP.

IF YOU ENJOYED THIS BLOG, WANT TO AVOID HANA AMDP MISTAKES? READ THIS, PLEASE FILL OUT THE FORM BELOW TO SIGN UP FOR OUR NEWSLETTER. WE DELIVER SAP TECHNICAL TIPS & TRICKS, SAP NEWS, AND THE CURRENT MONTH'S BLOG RIGHT TO YOUR INBOX!

ITP NEWSLETTER SIGN UP

Sign up to receive ITPSAP's industry insights, tips, tricks, news, and more!

* Email

* First Name

* Last Name

By submitting this form, you are consenting to receive marketing emails from: IT Partners. You can revoke your consent to receive emails at any time by using the SafeUnsubscribe® link, found at the bottom of every email.

RELATED POSTS

- An Introduction to UI5
- An Introduction to ABAP Unit – Part 2
- An Introduction to ABAP Unit – Part 1

abap ABAP for HANA ABAP Managed Data Procedures AMDP CDS Table Functions Core Data Services HANA HANA AMDP IT Partners IT staffing sap sap consulting SAP HANA SAP Integration SAP staffing

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

ALSO ON IT PARTNERS, INC

Using Memory Inspector to Analyze Usage in Real Time

1 comment • 5 years ago

Avatar **sudha** — I love all these posts in this blog. They are so informative . Thank You!

The New Enhancement Framework – Part 2

3 comments • 6 years ago

Avatar **the tao of badass scam** — Thank you for your website publish. Manley and i also have been saving to get a whole new guidebook about …

ABAP OO in your Custom Workflow – Part 2

3 comments • 6 years ago

Avatar **Bill Craig** — Hello Robert,Thanks for you interest in my blog. To answer your questions, the element im_lifnr is an import parameter for the …

IDocs: A Guide for New Developers – Part 5

3 comments • 5 years ago

Avatar **Anthony Cecchini** — Hi GV... interesting question. Let me try and see if I can explain - A Remote Function Call (RFC) makes direct and …

✉ Subscribe D Add Disqus to your siteAdd DisqusAdd 🔒 Disqus' Privacy PolicyPrivacy PolicyPrivacy

OUR PROFILE

Since 1993, IT Partners has been providing reliable, cost-effective solutions to meet our customer's goals and objectives in the Commercial and Federal ERP Marketplace.

Toll Free: (877) 288-6044
Local: (724) 582-4307
FAX: (412) 291-2901
Email: info@itpsap.com

OUR CERTIFICATIONS

U.S. Small Business Administration

AFCEA

NEWS

IT Partners Ranks on Inc. 5000 List for 5th Consecutive Year!
September 5, 2018

IT Partners, Inc Ranked #43 on Washington Technology Fast 50
September 5, 2018

NEWSLETT

Sign up to re
more!

First Name

Last Name

Email Address*

SIGN UP

CORPORATE MEMBER

SMALL BUSINESS MEMBER

Privacy Policy