**Direct Update ADSO ABAP API**

[Содержание](#Содержание)

[ABAP словарь](#ABAP_словарь)

[VALUE # | NEW #](#VALUE_NEW)

# [Internal table](#Internal_table)

[*Типы вн. таблиц*](#Типы_вн_таблиц)

# *[Difference between like and like line of](#Difference_between_like_and_like_line_of)*

[*INITIAL SIZE*](#INITIAL_SIZE)

[From ADSO to internal table](#From_ADSO_to_internal_table)

# [*Declare an internal table*](#declare_an_internal_table)

[Запись из ADSO во внутреннюю таблицу](#Запись_из_ADSO_во_вн_таблицу)

# [How to access a public type of an object](#Access_a_public_type_of_an_object)

# AMDP

# [ABAP AMDP Procedure Output Parameter Type must be Structured](#AMDP_Procedure_Output_Parameter_Type)

[CAST | #CAST](#CAST)

Ошибки

[*using @ to escape host variables*](#err_using_to_escape_host_variables)

[*Reduce the precision of any subexpressions*](#err_Reduce_the_precision)

**ABAP словарь**

[Содержание](#Содержание)

Определение переменной с типом определенного инфо-обекта - /BIC/OI + имя инфо-объекта

data: product\_code type /BIC/***OI***BCOM0000,

Определение типа внутренней таблицы для чтения из ADSO

*Вариант 1*

types:

begin of t\_out,

product\_code type /BIC/OIBCOM0000,

doc\_id type /BIC/OIBCHKNUM0,

doc\_type type /BIC/OIBSLST000,

idin(36) type c,

dt type /BI0/OIDATE,

tm type /BI0/OITIME,

iss\_org\_code type /BIC/OIBEMIT000,

to\_code type /BIC/OIBAZS0000,

rcp\_quantity type /BIC/OIBCREC000,

rcp\_amount type /BIC/OIBSREC000,

curr\_code type /BI0/OICURRENCY,

unit type /BI0/OIUNIT,

end of t\_out,

tt\_output TYPE STANDARD TABLE OF t\_out,

*Вариант 2*

tt\_output1 type standard table of /BIC/ABDINGDS02.

**VALUE # | NEW #**

[Содержание](#Содержание)

## ***Создание экземпляра класса ABAP***

Cтарый вариант синтаксиса

CREATE OBJECT lv\_value\_node EXPORTING iv\_data\_ref = lo\_ref.

Новый вариант

lv\_value\_node = *NEW #* ( iv\_data\_ref = lo\_ref ).

## ***line\_exists*** - Проверка наличия записи во вн. таблице ABAP без считывания

Cтарый вариант

READ TABLE lt\_data *TRANSPORTING NO FIELDS* WITH KEY new\_int = '2'.

IF sy-subrc = 0.

EXIT.

ENDIF.

Новый вариант

IF *line\_exists*( lt\_data[ new\_int = '2' ] ).

EXIT.

ENDIF.

## ***VALUE #*** - Считать данные из вн. таблицы ABAP

Cтарый вариант

DATA: ls\_data TYPE z\_str\_type.

READ TABLE mt\_data *INTO* ls\_data WITH KEY vbeln = lv\_vbeln.

Новый вариант

DATA(ls\_data\_new) = *VALUE #* ( mt\_data[ vbeln = lv\_vbeln ] *OPTIONAL* ).

**From ADSO to internal table**

<https://www.element61.be/en/resource/performance-internal-tables-sap-bw-transformations>

[Содержание](#Содержание)

**Direct Update ADSO ABAP API**

[Содержание](#Содержание)

# Posting the data to BW4/HANA aDSO using BW APIs and RFC connection

*from* <https://blogs.sap.com/2021/12/05/posting-the-data-to-bw4hana-adso-using-bw-apis-and-rfc-connection/>

let's there is a service that *enables third-party APIs to post the data directly into SAP BW ADSO*.

It use an RFC enabled BW function module that allows the external source like CPI to read BW parameters, request the data from an outside source through REST, harmonize it and post the data to BW ADSO in a columnar format.

A sample below is to acquire the data from a time-dependent Workday report with DateFrom and DateTo parameters for multiple periods at once and to post the result to BW.

The data from the Workday report is read using CPI, REST.

CPI makes requests synchronously, it transforms the result to ABAP input structure via a Groovy script. Next, it unions it together, and post to BW.

In this example, the CPI request should have the following structure

<FM\_NAME>

<INPUT>

<IT\_DATA>

<item>

<key\_field>value</key\_field>

<RECORDMODE></RECORDMODE>

<field>value</field>

</item>

</IT\_DATA>

</FM\_NAME>

Step 1 - *Create a wrapper Function Module*

It needs to

* *throw errors* like the API provided by SAP;
* *add errors* if a process chain has failed;
* *have an input structure* of the active Table of the target ADSO. Please note this table has a *record mode* field after the key specification. Therefore, it should always be posted as empty.

The function module needs to pass multiple parameters to CPI.

FUNCTION ***Z\_FM\_NAME***

IMPORTING

VALUE(IT\_DATA) TYPE ZTY\_<ADSONAME>1 OPTIONAL

EXPORTING

VALUE(ET\_PARAMS) TYPE ZTT\_<2columns\_table\_type>

VALUE(E\_LINES\_INSERTED) TYPE INT4

VALUE(E\_COLD\_LINES\_INSERTED) TYPE INT4

VALUE(ET\_MSG) TYPE RS\_T\_MSG

VALUE(E\_UPD\_REQ\_TSN) TYPE RSPM\_REQUEST\_TSN

VALUE(ET\_ACT\_REQ\_TSN) TYPE RSDSO\_T\_TSN

EXCEPTIONS

WRITE\_FAILED

ACTIVATION\_FAILED

DATASTORE\_NOT\_FOUND

CHAIN\_FAILED.

*<Content>*

ENDFUNCTION.

Step 2 - *Add a check if the parameters are filled and if the CPI request has data*

FUNCTION ***Z\_FM\_NAME***

…

CONSTANTS: lc\_adso\_open\_pos TYPE rsoadsonm VALUE '<ADSO\_NAME>',

lc\_pc\_id TYPE rspc\_chain VALUE '<PC\_NAME>'.

SELECT start\_date, end\_date

FROM *<parameters\_table>* INTO TABLE *@et\_params*.

*<content>*

ENDFUNCTION.

Step 3 - *Add executions of the RSDSO\_WRITE\_API*

It enables the new master data values /sid generation/ specification via a parameter *i\_allow\_new\_sids*.  
In addition, it allows specification of whether the data is activated after posting via a parameter *i\_activate\_data*. It returns a full activation log and the number of inserted & activated lines.

…

IF it\_data[] IS NOT INITIAL AND et\_params[] IS NOT INITIAL.

CALL FUNCTION '*RSDSO\_WRITE\_API*'

EXPORTING

i\_adsonm = lc\_adso\_open\_pos

i\_allow\_new\_sids = rs\_c\_true

i\_activate\_data = rs\_c\_true

it\_data = it\_data

IMPORTING

e\_lines\_inserted = e\_lines\_inserted

e\_cold\_lines\_inserted = e\_cold\_lines\_inserted

et\_msg = et\_msg

e\_upd\_req\_tsn = e\_upd\_req\_tsn

et\_act\_req\_tsn = et\_act\_req\_tsn

EXCEPTIONS

write\_failed = 1

activation\_failed = 2

datastore\_not\_found = 3

OTHERS = 4.

CASE sy-subrc.

*<content>*

ENDIF.

ENDFUNCTION.

Step 4 - *Add error handling and a process chain execution*

…

WHEN 0.

CALL FUNCTION 'RSPC\_API\_CHAIN\_START' EXPORTING i\_chain = lc\_pc\_id.

IF sy-subrc <> 0.

MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno

WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4

RAISING chain\_failed.

ENDIF.

WHEN 1.

MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno

WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4 RAISING write\_failed.

WHEN 2.

MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno

WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4

RAISING activation\_failed.

WHEN 3.

MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno

WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4

RAISING datastore\_not\_found.

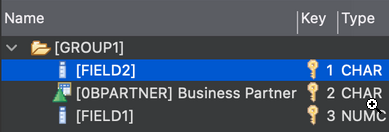
ENDCASE.

ENDIF.

ENDFUNCTION.

The easy way is just to use the official function module [delivered by SAP](https://help.sap.com/viewer/107a6e8a38b74ede94c833ca3b7b6f51/2.0.5/en-US/72e16c936fb94cffb71ce90edd5f8f8e.html).

Example ADSO



Example code

REPORT zadsoamdp.

DATA: lt\_data TYPE STANDARD TABLE OF /bic/a*adsoamdp*2,

lt\_msg TYPE rs\_t\_msg.

APPEND *VALUE #*( field1 = '1' field2 = '2' ) TO lt\_data.

CALL FUNCTION *'RSDSO\_DU\_WRITE\_API'*

EXPORTING

i\_adsonm = 'ADSOAMDP'

it\_data = lt\_data

IMPORTING

et\_msg = lt\_msg

EXCEPTIONS

write\_failed = 1

datastore\_not\_found = 2

OTHERS = 3.

IF sy-subrc <> 0.

MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno

WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.

ENDIF.

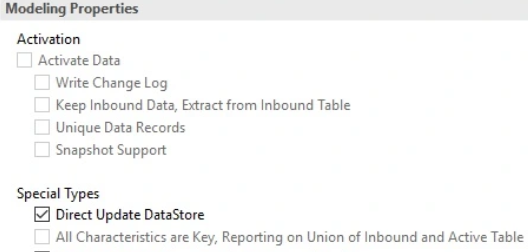
cl\_demo\_output=>display(

EXPORTING

data = lt\_msg

).

*RSDSO\_DU\_WRITE\_API* используется для ADSO типа *Direct Update Datastore* /ADSO прямой записи/



Вместо таблицы Новых данных - использоваться таблица Активных данных ⇒ следует определить семантический ключ, без него активировать ADSO не получится.

Ракурс экстракции и ракурс отчётности берёт данные из таблицы Активных данных.

Хоть в инф. при активации указано, что такой ADSO должен быть заполнен только через API, построить поток данных через трансформацию в этот ADSO возможно.

Тип агрегации показателя изменить нельзя - он всегда будет *Move*, т.е. будет брать последнее значение при совпадении строк по ключу.

**!** В такой ADSO в принципе не получится записать несколько записей с одним ключом в одном реквесте - будет ругаться на неуникальный индекс. Т.е. если в других вариантах ADSO можно взять последнее значение при совпадении строк по ключу, то тут *из источника должна приходить ровно одна запись с одним ключом*.  Следующими реквестами /запросами/ пожалуйста, можете грузить с уже загруженным ключом, произойдёт перезапись.

### **How to read/write/delete from/to aDSO objects**

<https://blog.maruskin.eu/2022/12/how-to-readwritedelete-fromto-adso.html>

[Содержание](#Содержание)

There is a set of SAP standard function modules for reading, writing and deleting data from/to the A[DSO](https://blog.maruskin.eu/2022/03/flavors-of-adso-object.html). They can easily be used in [custom ABAP code](https://blog.maruskin.eu/2018/12/bw4hana-custom-abap-code-in.html) in [BW’s transformations](https://blog.maruskin.eu/search/label/Transformations).

There are function modules representing APIs for *DSO* objects under name space *RSDRI\_\** or *RSDRD\_\**.

In case of *aDSO* objects the API are there under name space *RSDSO\_\**.

|  |  |  |
| --- | --- | --- |
| **aDSO type** | **Operation** | **Method / API** |
| **Standard** | READ | Open SQL *SELECT*;  *RSDRI\_INFOPROV\_READ* |
|  | WRITE | *RSDSO\_WRITE\_API*;  *RSDSO\_WRITE\_API\_RFC*  *RSDSO\_ACTIVATE\_REQ\_API\_RFC* - Activates requests in aDSO. Multiple requests can be activated separately or whether or system can activate as many requests as possible at one shot. |
|  | DELELE | N/A |
| **Direct Update** | READ | Open SQL *SELECT* |
|  | WRITE | *RSDSO\_DU\_WRITE\_API*;  *RSDSO\_DU\_WRITE\_API\_RFC*– Writes data from an itab into active data table of the aDSO. Each API call results in a new request. Database transaction is committed automatically. |
|  | DELETE | *RSDSO\_DU\_DELETE\_API\_RFC* - Deletes data from an aDSO. Whole content can be deleted or data can be deleted based on a selective deletion. |
|  | CLEANUP | *RSDSO\_DU\_CLEANUP\_API\_RFC* - Changes status of red requests in aDSO to green. Red requests are blocking further data loads thus have to be corrected. Only requests that were loaded via API are considered. |

Note - FMs ending with \**RFC* are supposed to be used for remote scenarios whereas other FMs are locally to be used where the aDSO and calling code resides within the same system.

Supporting FMs

*RSDSO\_DEBUG\_API* - It enables a user to debug above listed RFC enabled APIs for aDSO.

More information

* [Online docu for aDSO objects API – BW4/HANA](https://help.sap.com/docs/SAP_BW4HANA/107a6e8a38b74ede94c833ca3b7b6f51/72e16c936fb94cffb71ce90edd5f8f8e.html)
* [Online docu for aDSO objects API – BW 7.5](https://help.sap.com/doc/saphelp_nw75/7.5.5/en-US/72/e16c936fb94cffb71ce90edd5f8f8e/content.htm?no_cache=true)

You can use the following APIs for a ADSO object with the properties ***Activate/Compress Data*** and ***Write Change Log***

| **Type** | **Target Table** | **API for Local Use** | **API for RFC Use** | **Description** |
| --- | --- | --- | --- | --- |
| Load | Inbound Table | RSDSO\_WRITE\_API | RSDSO\_WRITE\_API\_RFC | Loads data from an internal table to the inbound table. The activation can then be started. |
| Activate | Active Data | RSDSO\_ACTIVATE\_ REQ\_API\_RFC | RSDSO\_ACTIVATE\_REQ\_ API\_RFC | Activates loaded requests |

You can use the following APIs for a DataStore object (advanced) for ***direct writing***

| **Type** | **Target Table** | **API for Local Use** | **API for RFC Use** | **Description** |
| --- | --- | --- | --- | --- |
| Load | Active Data | RSDSO\_DU\_WRITE\_ API | RSDSO\_DU\_WRITE\_ API\_RFC | Loads data from an internal table to the inbound table. |
| Delete | Active Data | RSDSO\_DU\_DELETE\_ API\_RFC | RSDSO\_DU\_DELETE\_ API\_RFC | Deletes data from the table of active data. The table of active data can be truncated or selectively deleted. |
| Status | Active Data | RSDSO\_DU\_CLEANUP\_ API\_RFC | RSDSO\_DU\_CLEANUP\_ API\_RFC | Deletes API requests with errors. Red requests block further load requests by DTP or by API. |

# Posting the data to BW4/HANA aDSO using BW APIs and RFC connection

<https://blogs.sap.com/2021/12/05/posting-the-data-to-bw4hana-adso-using-bw-apis-and-rfc-connection/>

[Содержание](#Содержание)

# Пишем в ADSO из SQL Console в HANA

<https://helpbw.wordpress.com/2021/11/25/%D0%BF%D0%B8%D1%88%D0%B5%D0%BC-%D0%B2-adso-%D0%B8%D0%B7-sql-console-%D0%B2-hana/>

[Содержание](#Содержание)

Расскажу, как записать данные в любую таблицу в схеме ***SAPHANADB*** из SQL в HANA.

Данный подход можно использовать, чтобы через SQL скрипт записать данные в любую таблицу не только в своей схеме, но и схеме SAP HANA DB.

В обычном сценарии использования такой подход не применим. Однако, например для прототипирования, или срочного расчёта каких-то данных, вполне возможен для использования, особенно если процесс переноса транспортов в продуктив сильно бюрократизирован и затруднён.

Итак, что бы записать данные в схему SAPHANADB, где находятся системные таблицы и таблицы BW объектов, нам понадобится следующие операции

1. Выполняем SQL скрипт, и записываем результат в своей пользовательской схеме в таблицу.
2. Добавляем полномочия на запись данных своей таблицы в схему SAPHANADB.
3. После этого через специальный ФМ записываем данные из нашей таблицы нашей схемы в таблицу ADSO или любую другую таблицу в схеме SAPHANADB.

Для начала выполняем какую-то логику в скрипте и записываем результат в свою таблицу в своей схеме. Это будет примерно так

create table "USERNAME"."TABLE" as (select \* from :internal\_table);

В дальнейшем, когда вы будете для отладки запускать этот скрипт много раз, придётся сначала убивать таблицу, потом создавать заново.

drop table "USERNAME"."TABLE";

create table "USERNAME"."TABLE" as (select \* from :internal\_table);

*:internal\_table* - это таблица с данными, которая у вас получается по результатам выполнения SQL скрипта.

В результате выполненных выше действий мы получили физическую таблицу в БД HANA в своей схеме.

Теперь нужно получить полномочия для возможности переноса ей в другую схему. В SQL скрипте выполняем операцию

grant select on "USERNAME"."TABLE" to "SAPHANADB";

После успешного выполнения /на какое-то непродолжительное время, минут 10/ предоставляются полномочия для переноса данных в системную схему.

После этого идём в ФМ ***RSDU\_EXEC\_SQL\_HDB*** через транзакцию SE37. Этот ФМ позволяет выполнять SQL код с расширенными полномочиями.

В параметре I\_T\_STMT этого ФМ указываем в строке LINE код SQL.

insert into "SAPHANADB"."/BIC/AADSONAME1" (select \* from "USERNAME".TABLE");

Код выше перенесёт данные в таблицу новых даных ADSO *ADSONAME* из таблицы *TABLE* в вашей схеме.

Но пример выше не позволяет в ADSO увидеть реквест в его администрировании. Это неудобство можно обойти следующим образом.

1. Загружаем любые данные в любом количестве в ADSO.
2. Выполняем SQL код через ФМ *RSDU\_EXEC\_SQL\_HDB*для очистки данных в ADSO для реквеста в столбце *REQTSN*. Код примерно такой

delete from "/BIC/AADSONAME1" where REQTSN = '20211210132355000040000'

Получить корректный REQTSN можно через ФМ ***CONVERSION\_EXIT\_RSTSN\_INPUT***, подав на вход реквест в формате, который даёт эклипс {1234 433…}

1. Мы имеем ADSO с реквестом в администрировании, но без данных в самой ADSO по этому реквесту. После этого мы можем вставить в этот ADSO данные как было рассказано выше.  Только естественно не забудьте в таблице *TABLE* в каждую строку поля REQTSN вставить константное значение '20211210132355000040000'.

insert into "SAPHANADB"."/BIC/AADSONAME1" (select \* from "USERNAME".TABLE");

Таблица *TABLE*должна полностью повторять структуру полей в ADSO, т.е. в ADSO без активации в вашем случае должно быть такие служебные поля

…….

'20211210132355000040000' as REQTSN,

' ' as DATAPAKID,

row\_number() over() as RECORD,

' ' as RECORDMODE,

\*

from :TABLE

После таких манипуляций, удалив реквест из ADSO стандартным образом, мы удалим и все данные этого реквеста из ADSO. Единственный момент, только количество записей в реквесте не будет совпадать с количеством записей в ADSO фактически.

Повторюсь ещё раз, это отнюдь не стандартный и не нормальный режима загрузки данных в ADSO, и не должен использоваться в обычных ситуациях. Но могут возникнуть определённые случае, когда таким образом вынужден записывать данные в хранилище. Однако целевая схема не должна предусматривать в принципе такой вариант загрузки данных в хранилище BW.

# Posting the data to BW4/HANA aDSO using BW APIs and RFC connection

<https://blogs.sap.com/2021/12/05/posting-the-data-to-bw4hana-adso-using-bw-apis-and-rfc-connection/>

**Internal table**

[Содержание](#Содержание)

*Direct access* to internal table content is not provided

Using ***header line***, we could directly use the table variable name to access the table contents, whilst *without header line*, contents are accessed via a ***working area***.

Пример

*“ Структура x\_personnel*

TYPES : BEGIN OF *x\_personnel*,

           id(10) TYPE C,

           name(25) TYPE C,

        END OF x\_personnel.

*“ Две вн.таблицы - t\_itab1* /*с заголовком/ и t\_itab2 /без заголовка/*

DATA:  *t\_itab1* TYPE STANDARD TABLE OF x\_personnel *WITH HEADER LINE*,

*t\_itab2* TYPE STANDARD TABLE OF x\_personnel,

*“ working area - d\_personnel*

*d\_personnel* TYPE x\_personnel.

To access ***all contents*** of an internal table

LOOP AT *t\_itab1*.

  WRITE: / *t\_itab1*-name.

ENDLOOP.

LOOP AT *t\_itab2* INTO *d\_personnel*.

  WRITE: / *d\_personnel*-name.

ENDLOOP.

To access a ***single content*** with certain key

READ TABLE *t\_itab1* WITH KEY id = '1234'.

WRITE : *t\_itab1*-name.

READ TABLE *t\_itab1* INTO *d\_personnel* WITH KEY id = '1234'.

WRITE : *d\_personnel*-name.

Using *header line* can cause ambiguity in code

CLEAR *t\_itab2*. *“ clear entire contents of table t\_itab2*

CLEAR *t\_itab1*. *“clear the contents of t\_itab1 header*.

CLEAR *t\_itab1*[]. “ clear the *entire* contents of *t\_itab1*

In *ABAP OO* you cannot use internal tables *with* header line

Try to modify your program using a ***field symbol***

DATA: *it\_bseg* TYPE TABLE it\_vbseg.

*FIELD-SYMBOLS*: <fs\_bseg> *like line of* it\_bseg.

LOOP AT *it\_bseg* ASSIGNING *<fs\_bseg*>.

<fs\_bseg>-FIELX = <fs\_bseg>-FIELX + ….

ENDLOOP.

**Типы вн. таблиц**

[Содержание](#Содержание)

Типы вн. таблиц

* TYPE ***STANDARD*** *TABLE* OF and TYPE *TABLE* OF is exactly the same.

*Rem*

In the ***OO-context*** you would not be able to leave out the STANDARD addition.

* TYPE ***HASHED*** *TABLE* - an internal HASH algorithm, allow read from the table with the cost is [by approximation] independent from the size of the table.

*Rem*

It is suitable for large data-sets with a lot of reads, but comparatively few writes. When declaring a hash table you have to also declare a ***UNIQUE KEY***, as the HASH algorithm is dependent on this.

* TYPE ***SORTED*** TABLE - can be declared with either an ***UNIQUE*** or a ***NON-UNIQUE*** key; allows a BINARY SEARCH.

*Rem*

The cost *to read* a record - HASHED table < *SORTED* TABLE < STANDARD table

The cost to write - STANDARD table < *SORTED* TABLE < HASHED table

# D[ifference between *like* and *like line of*](https://stackoverflow.com/questions/24925097/what-is-the-difference-between-like-and-like-line-of-in-abap)

[Содержание](#Содержание)

wa LIKE it\_one *“ a table without headerline*

wa LIKE LINE OF it\_one *“a table with headerline*

**INITIAL SIZE**

[Содержание](#Содержание)

INITIAL SIZE ***0*** is the *default*. If you wanted memory allocation to happen in sets of 10 times the number of lines in your internal table, you would have used - INITIAL SIZE 10, but in most cases leaving the default memory allocation is better than trying to force it.

Global variables

Whereas in the transactional ***SAP ECC*** you can write *includes* and have *global variables* access in these includes, in ***SAP BW*** you have to keep in mind that the transformation is [usually] going to be executed in parallel mode, with *no real global area* across all parallel executions.

Work area vs. Header line

In older ABAP releases, there used to be an internal table ***header line*** [work area] - a special line that contain the last accessed line of internal table. It was a *confusing mechanism* because the *variable name* to access the contents of the header line was the same as the *table* itself, plus it was difficult to keep track of what was actually in that line.

For backward compatibility reasons it is still supported, but you should refrain from using it and declare your own structure.

Example - This internal table definition would be useful for containing the *red cars* from the DSO of vehicles from our business case.

DATA: BEGIN OF *LS\_CARS*,

/BIC/ZVEHICLE type /BIC/OIZVEHICLE, *"unique id for each car*

/BIC/ZBRAND type /BIC/OIZBRAND,

/BIC/ZMODEL type /BIC/OIZMODEL,

/BIC/ZCOLOR type /BIC/OIZCOLOR,

/BIC/ZMSTART type /BI0/OICALMONTH, *"month/year of entrance*

END OF LS\_CARS,

*LT\_REDCARS* like STANDARD TABLE of *LS\_CARS*.

When declaring ***sorted*** or ***hashed*** tables, you need to specify a ***key*** as well. For sorted tables this may be unique or not, hashed tables must be created with a unique key.

#### Filling the internal table

select \* from */BIC/ADSO\_CARS00*

into table *LT\_REDCARS*

where /BIC/ZTYPE eq ‘CAR’.

*“ Указание списка полей лучше в плане экономии памяти*

select /BIC/ZVEHICLE, /BIC/ZBRAND, /BIC/ZMODEL, /BIC/ZCOLOR, /BIC/ZMSTART

from */BIC/ADSO\_CARS00*

into table *LT\_REDCARS*

where /BIC/ZTYPE eq ‘CAR’ and /BIC/ZCOLOR eq ‘RED’.

INTO CORRESPONDING FIELDS OF

The *select … into table* statement inserts the fields into the internal table in the *order* in which they were selected. *If the field order in the database table would change*, the data would end up in the wrong columns of the internal table ⇒ use the ***corresponding fields*** of addition.

select /BIC/ZVEHICLE, /BIC/ZBRAND, /BIC/ZMODEL, /BIC/ZCOLOR, /BIC/ZMSTART

from */BIC/ADSO\_CARS00*

into *CORRESPONDING FIELDS* of table *LT\_REDCARS*

where /BIC/ZTYPE eq ‘CAR’ and /BIC/ZCOLOR eq ‘RED’.

#### Cleaning up

***Refresh*** the internal table using empty brackets [] *if you* *still need the structure further on*

*refresh* LT\_REDCARS[].

or f***ree*** it to release all the related memory

*free* LT\_REDCARS.

#### For all entries

Imagine you want to get the rental information about *red cars* into an internal table. A developer not knowing about the existence of the *for all entries* statement could have come up with the following valid but *slow* approach

loop at *LT\_REDCAR* into LS\_REDCAR.

select /BIC/ZVEHICLE, /BIC/ZDATFROM, /BIC/ZDATTO, /BIC/ZCLIENT, /BIC/ZMILES

from */BIC/ADSO\_RENT00*

appending CORRESPONDING FIELDS of table *LT\_REDCAR\_RENT*

where /BIC/ZVEHICLE eq LS\_REDCAR-/BIC/ZVEHICLE.

endloop.

As opposed to using ***for all entries***

select /BIC/ZVEHICLE, /BIC/ZDATFROM, /BIC/ZDATTO, /BIC/ZCLIENT, /BIC/ZMILES

from */BIC/ADSO\_RENT00*

into CORRESPONDING FIELDS of table *LT\_REDCAR\_RENT*

*FOR ALL ENTRIES* in *LT\_REDCAR*

where /BIC/ZVEHICLE eq LT\_REDCAR-/BIC/ZVEHICLE.

It may not look like a big difference, but the first solution will go back and forth to the database, returning the rows one by one, whereas the second one will create one SQL statement for the database to get all records in one roundtrip.

##### *Behind the scenes*

The ABAP programming layer is making abstraction of the underlying database platform in order to make it work on multiple platforms ⇒ all SQL statements are translated into the real database statements. The ***for all entries*** generates a where clause with IN-operator, e.g. on a DB2 database

select "/BIC/ZVEHICLE”, …

from *"/BIC/ADSO\_RENT00”*

where *"/BIC/ZVEHICLE”* in (‘0000003’, ‘0000006’, ‘0000013’, ‘0000019’, ‘0000020’, ‘0000024’, ‘0000029’, ‘0000030’)

#### Merry-stop-going-round

Like our LT\_REDCAR internal table, containing all red cars, we would like to have one which also contains the sum of mileages that were driven with it, *LT\_REDCAR\_MILES*. The mileage info can be found in the LT\_REDCAR\_RENT internal table from before.

loop at *LT\_REDCAR\_MILES* into *LS\_REDCAR\_MILES*.

LV-INDEX = SY-INDEX.

clear LV\_COUNTER.

loop at *LT\_REDCAR\_RENT* into *LS\_REDCAR\_RENT*

where /BIC/ZVEHICLE = LS\_REDCAR\_MILES-/BIC/ZVEHICLE.

LV-COUNTER = LV-COUNTER + LS\_REDCAR\_RENT-/BIC/ZMILES.

endloop.  
 LS\_REDCAR\_MILES-/BIC/ZMILES = LV-COUNTER.

update *LT\_REDCAR\_MILES* from *LS\_REDCAR\_MILES* index LV-INDEX.

endloop.

There is a tool at our disposal to optimize such processing - ***field symbols***. These are special variables which do not contain a value themselves, but *point to something with a value* instead. This pointer can be a whole work area or just one field [and other options not discussed here].

When using field symbols it can work more efficiently

FIELD-SYMBOLS:

TYPE ANY,

TYPE ANY.

…

loop at *LT\_REDCAR\_MILES* assigning.

assign component ‘/BIC/ZMILES’ of structure to.

loop at *LT\_REDCAR\_RENT* into *LS\_REDCAR\_RENT*

where /BIC/ZVEHICLE = -/BIC/ZVEHICLE.

add LS\_REDCAR\_RENT-/BIC/ZMILES to .

endloop.

endloop.

The code above shows how individual fields can still be accessed via the field symbol that points to a structure (the where clause), and how direct access to a field can be done (add statement).

Tip: by using field symbols in a smart way, you could make your code even more dynamic, so that it can be used for e.g. a variable amount of fields in your structure.

# Declare an internal table

<https://abap-python.com/how-to-declare-an-internal-table-in-three-different-ways/>

[Содержание](#Содержание)

Объявление вн. таблицы

Please declare the internal type explicitly like below /otherwise it will consider as structure/

DATA It\_internal\_table *TYPE STANDARD TABLE OF* ztt\_db\_table1.

Объявление структуры

*“ Here Is\_structure is considered as internal table. It shows error which assigning values to the structure. The compilation error – ‘Is\_structure is a table without a header line’.*

DATA Is\_structure *LIKE* It\_internal\_table.

*“ If it is written like below - it will show compilation error - ’Type It\_internal\_table is unknown’.*

DATA Is\_structure *TYPE* It\_internal\_table.

DATA Is\_structure *LIKE LINE OF* ztt\_db\_table1.

Is\_structure-field1 = 10.

Is\_structure-field2 = ‘Ten’.

Is\_ structure-fieid3 = ‘Number: 10’.

INSERT INTO ztt\_db\_table1 VALUES Is\_structure.

SELECT \* FROM ztt\_db\_table1 INTO TABLE It\_internal\_table.

LOOP AT It\_internal\_table ASSIGNING FIELD-SYMBOL(<Is\_internal\_table>).

WRITE:/ <ls\_internaltable>-field1.

WRITE:/ <Is\_internal\_table›-field2.

WRITE:/ <Is\_internal\_table>-field3.

ENDLOOP.

Declaration of local internal table and structure

Обрати внимание – если это

* определение *типа* - то TYPE STANDARD TABLE OF;
* определение *данных* – TYPE.

TYPES: BEGIN OF *st\_structure*,

              fld\_1 TYPE zdel\_id,

              fld\_1 TYPE zdel\_name,

              fld\_1 TYPE zdel\_desc,

           END OF st\_structure.

*TYPES*: *tt\_internal\_table* *TYPE STANDARD TABLE OF* st\_structure.

*DATA* *pt\_internal\_table* *TYPE* tt\_internal\_table.

*“ work area*

DATA *wa* *LIKE LINE OF* pt\_internal\_table.

*“ from wa to ztt\_db\_table1*

wa-fld\_1 = 20.

wa-fld\_2 = ‘Twenty’.

wa-fld\_3 = ‘Number: 20’.

INSERT INTO ztt\_db\_table1 *VALUES* *wa*.

*“ from ztt\_db\_table1 to pt\_internal\_table*

SELECT \* FROM ztt\_db\_table1 INTO TABLE pt\_internal\_table.

*“ Вывод содержимого вн. таблицы pt\_internal\_table*

LOOP AT *pt\_internal\_table* ASSIGNING *<fs>*.

              WRITE:/ <fs>-fld\_1.

              WRITE:/ <fs>-fld\_2.

              WRITE:/ <<fs>-fld\_3.

 ENDLOOP.

**Запись из ADSO во внутреннюю таблицу**

[Содержание](#Содержание)

# G[et ERROR "Internal tables cannot be used as work areas" inside of method](https://stackoverflow.com/questions/70019751/get-error-internal-tables-cannot-be-used-as-work-areas-inside-of-method)

I duplicate a *table* to a *local table* and remove all duplicates in it, this works fine /first 3 code lines/

DATA *localTable* TYPE STANDARD TABLE OF *table*.

SELECT columnName FROM table INTO TABLE localTable.

SORT localTable BY columnName ASCENDING.

DELETE *ADJACENT DUPLICATES* FROM localTable COMPARING columnName.

Now I want to loop over *local table* and send all matching data into an structure with INTO CORRESPONDING FIELDS OF - unfortunately I always get the following error - *Internal tables cannot be used as work areas*.

***Rem***

I'm working inside of a method.

LOOP AT localTable ASSIGNING FIELD-SYMBOL(<fs\_table>).

SELECT \* FROM anotherTable as p

WHERE p~CN1 = @localVariable

AND p~CN2 = @<fs\_table>-columnName

INTO CORRESPONDING FIELDS OF @exportStructure *"<-- Here I always get my error*

ENDSELECT.

ENDLOOP.

− *First.* I've read that I have to sort my internal table before using command DELETE ADJACENT DUPLICATES FROM localTable COMPARING columnName. So I've added following code line in between

SORT localTable BY columnName ASCENDING.

*− Second.* Instead of using *INTO CORRESPONDING FIELDS OF TABLE* I've used APPENDING CORRESPONDING FIELDS OF TABLE because INTO overwrites every line with itself, so in total I have only one line in my exported structure.

APPENDING adds a new line every time my statements are true.

− IMO, it is better to use ***FOR ALL ENTRIES*** using LOCALTABLE rather than looping over it. Also, an INTO TABLE on your select statement would remove your ENDSELECT loop.

SELECT \* FROM /BIC/ADSO00 AS tb1

INNER JOIN /BIC/PMASTER as tbl2

ON tbl1~field1 = tbl2~field1 AND tbl12~field2 = tbl2~field2

INTO CORRESPONDING FIELDS OF TABLE gt\_itab

FOR ALL ENTRIES IN SOURCE\_PACKAGE

WHERE tbl1~field3 = SOURCE\_PACKAGE-field3

AND tbl1~field4 BETWEEN lv\_minper AND lv\_maxper

AND tbl1~field5 = SOURCE\_PACKAGE-field5

AND tbl1~field6 = '0100'

AND tbl2~OBJVERS = 'A'.

SELECT \* FROM /BIC/ABDINGDS02 AS tb1

INTO CORRESPONDING FIELDS OF lt\_data

# [How to access a public type of an object](https://stackoverflow.com/questions/15303575/how-to-access-a-public-type-of-an-object)

[Содержание](#Содержание)

I have a class named ZCL\_RM\_SPREADSHEETML. It has in the Types tab a type called TY\_STYLE with visibility *public* and it’s defined with Direct Type Entry.

− You have to use the appropriate [*component selector*](http://help.sap.com/abapdocu_702/en/abencomponent_selector_glosry.htm)

* a structure component selector (-),
* a class component selector (=>),
* an interface component selector (~),
* and an object component selector (->).

In this case, you're accessing a type (component) of a class, so you have to use =>.

DATA : wa\_blue\_style TYPE zcl\_rm\_spreadsheetml*->*ty\_style.

# 

# ABAP AMDP Procedure Output Parameter Type must be Structured

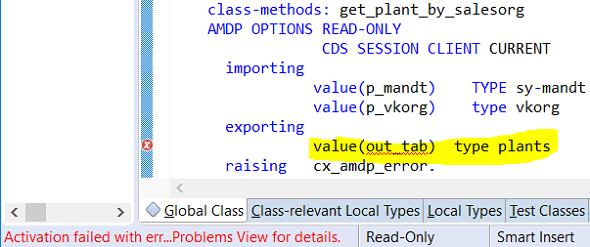
[Содержание](#Содержание)

If method contains a database procedure

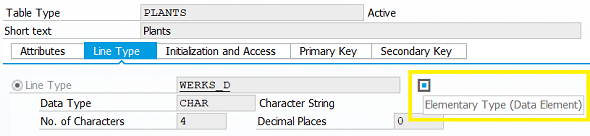
* *row type* of output parameters must be structured;
* *all components* of the row type must be elementary.

ABAP programmers developing AMDP procedures must convert output table parameters' row types to structured row types.

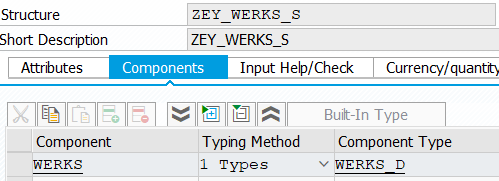
When I try to activate AMDP procedure on SAP HANA Studio, I see following error message -  
*The method "GET\_PLANT\_BY\_SALESORG" contains a database procedure, which means that the row type of "OUT\_TAB" must be structured. All components of the row type must be elementary*.



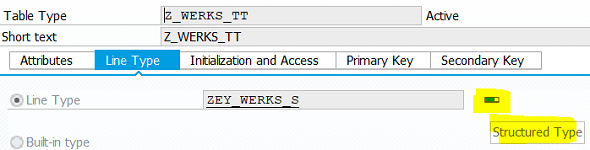
The AMDP activation error is because the *PLANTS* table type is based on an elementary type WERKS\_D data element.



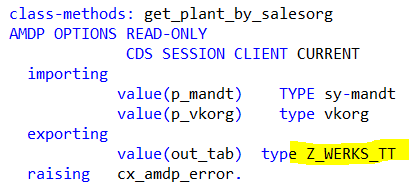
But for AMDP output parameters, the row type of the table must be structured not elementary.  
To resolve this issue, I created a structure which includes only the WERKS field of WERKS\_D data element as seen in below.



Additionall I created a new ABAP Data Dictionary object as Table Type using above structure as its row type.



When I modify AMDP class code for the procedure method definition by replacing the PLANTS table type with new table type, the AMDP activation is succeeded immediately.



# Examples Of Working With Other ABAP Data Types

<https://www.saptraininghq.com/examples-of-working-with-other-abap-data-types/>

[**ABAP for BW**](https://abap4bw.wordpress.com/)

<https://abap4bw.wordpress.com/2018/08/03/bw-stepout-all/>

There are two types of objects

* SAP delivered - ***/BI0/***
* Custom created - ***/BIC/***

Fact table, dimension table, SID table, master data (attr, text, hier)

* P — /BIC/P (IO tech name) – time independent master data attr table
* Q — /BIC/Q (IO tech name) – time dependent master data attr table
* X — /BIC/X (IO tech name) – SID table for time independent nav attr
* Y — /BIC/Y (IO tech name) – SID table for time dependent nav attr
* S — /BIC/S (IO tech name) — SID table for char
* T — /BIC/T (IO tech name) – text table
* J — /BIC/J (IO tech name) – hier interval table
* H – Hierarchy table
* K – Hierarchy SID table
* I – SID hierarchy structure

# ABAP Select data from SAP table /BIC/CCTA0000016489 into internal table

Below is a number of ABAP code snippets to demonstrate how to select data from SAP **/BIC/CCTA0000016489** table and store it within an internal table, including using the newer @DATA inline declaration methods. It also shows you various ways to process this data using ABAP work area, inline declaration or field symbols including executing all the relevant **CONVERSION\_EXIT** routines specific to /BIC/CCTA0000016489. See here for more generic [Select statement tips](https://www.trailsap.com/dev/abap/?topic=tips_select).

Sometimes data within SAP is stored within the database table in a different format to what it is displayed to the user. These input/output conversation FM routines are what translates the data between the two formats.

There is also a full declaration of the /BIC/CCTA0000016489 table where each field has a char/string type for you to simply copy and paste. This allows you to use processing that is only available to these field types such as the CONCATENATE statement.

DATA: IT\_/BIC/CCTA0000016489 TYPE STANDARD TABLE OF /BIC/CCTA0000016489,

WA\_/BIC/CCTA0000016489 TYPE /BIC/CCTA0000016489,

GD\_STR TYPE STRING.

DATA: lo\_typedescr type REF TO cl\_abap\_typedescr.

DATA: lv\_fieldname type fieldname.

FIELD-SYMBOLS: <FIELD> TYPE any.

FIELD-SYMBOLS: </BIC/CCTA0000016489> TYPE /BIC/CCTA0000016489.

\*Process all fields in table header/work area as string values

PERFORM process\_as\_string\_field\_values CHANGING wa\_/BIC/CCTA0000016489.

SELECT \*

\*restrict ABAP select to first 10 rows

UP TO 10 ROWS

FROM /BIC/CCTA0000016489

INTO TABLE IT\_/BIC/CCTA0000016489.

\*Select data and declare internal table using [in-line method @DATA](https://www.trailsap.com/dev/abap/?topic=abap-inline-declarations)

\*SELECT \*

\* FROM /BIC/CCTA0000016489

\* INTO TABLE @DATA(IT\_/BIC/CCTA00000164892).

\*--Further methods of using ABAP code to [select data from SAP database tables](https://www.trailsap.com/dev/abap/?topic=tips_select)

\*You can also declare the header/work area using the [in-line DATA declaration method](https://www.trailsap.com/dev/abap/?topic=abap-inline-declarations)

READ TABLE IT\_/BIC/CCTA0000016489 INDEX 1 INTO DATA(WA\_/BIC/CCTA00000164892).

\*Demonstrate how to loop at an internal table and update values using a FIELD-SYMBOL

LOOP AT IT\_/BIC/CCTA0000016489 ASSIGNING </BIC/CCTA0000016489>.

\*To update a field value using a field symbol simply change the value via the field symbol pointer  
</BIC/CCTA0000016489>-GUID = 1.  
</BIC/CCTA0000016489>-LANGU = 1.  
</BIC/CCTA0000016489>-TEXT1 = 1.

ENDLOOP.

LOOP AT IT\_/BIC/CCTA0000016489 INTO WA\_/BIC/CCTA0000016489.

\*Write horizonal line to screen report.

WRITE:/ sy-uline.

\*Write selected data to screen/report before conversion.

WRITE:/ sy-vline, ENDLOOP.

\*Add any further fields from structure WA\_/BIC/CCTA0000016489 you want to display...

WRITE:/ sy-uline.

\* Aternatively use generic code to Write field values (and NAME) to screen report

DO.

ASSIGN COMPONENT sy-index OF STRUCTURE wa\_/BIC/CCTA0000016489 TO <field>.

IF sy-subrc <> 0. EXIT. ENDIF.

WRITE:/ 'Field Value', <field>, sy-vline.

gd\_str = <field> .

lo\_typedescr ?= CL\_ABAP\_DATADESCR=>DESCRIBE\_BY\_DATA( <field> ).

lv\_fieldname = lo\_typedescr->GET\_RELATIVE\_NAME( ).

WRITE:/ 'Field Name', lv\_fieldname.

ENDDO.

\*Redo loop but convert all fields from internal to out value

LOOP AT IT\_/BIC/CCTA0000016489 INTO WA\_/BIC/CCTA0000016489.

\*Write horizonal line to screen report.

WRITE:/ sy-uline.

\*Convert all fields to display/output versions using conversion routines

PERFORM convert\_all\_field\_values CHANGING wa\_EKKO.

ENDLOOP.

\*&---------------------------------------------------------------------\*

\*& Form convert\_all\_field\_values

\*&---------------------------------------------------------------------\*

FORM convert\_all\_field\_values CHANGING p\_EKKO LIKE wa\_EKKO.

DATA: ld\_input(1000) TYPE c, ld\_output(1000) TYPE C.  
ENDFORM.

\*&---------------------------------------------------------------------\*

\*& Form process\_as\_string\_field\_values

\*&---------------------------------------------------------------------\*

FORM process\_as\_string\_field\_values CHANGING p\_EKKO LIKE wa\_EKKO.

TYPES: BEGIN OF T\_/BIC/CCTA0000016489\_STR,   
GUID TYPE STRING,  
LANGU TYPE STRING,  
TEXT1 TYPE STRING,END OF T\_EKKO\_STR.

DATA: WA\_/BIC/CCTA0000016489\_STR type T\_EKKO\_STR.

DATA: ld\_text TYPE string.

LOOP AT IT\_EKKO INTO WA\_EKKO.

MOVE-CORRESPONDING wa\_EKKO TO WA\_EKKO\_STR.

CONCATENATE: sy-vline  
WA\_/BIC/CCTA0000016489\_STR-GUID sy-vline   
WA\_/BIC/CCTA0000016489\_STR-LANGU sy-vline   
WA\_/BIC/CCTA0000016489\_STR-TEXT1 sy-vline INTO ld\_text SEPARATED BY SPACE.

\*Add any further fields from structure WA\_EKKO\_STR you want to CONCATENATE...

ENDLOOP.

ENDFORM.

report this ad

select t1-IDIN, t1-RECORDMODE, GDS-/BIC/BGDSCURR, 0 as /BIC/BCCHK000,

SLSDOC-/BIC/BAZSOPER, t1-/BIC/BSLST000, SLSDOC-/BIC/BDOCSTAT,

sum(GDS-/BIC/BSPACK00\*t1-/BIC/BSREC000) as /BIC/BSPACK00,

sum(GDS-/BIC/BSEXCTOT\*t1-/BIC/BSREC000) as /BIC/BSEXCTOT,

t1-/BIC/BCOM0000, NULL as /BIC/BPAYT000, t1-/BIC/BCHKNUM0,

null as /BIC/BSPRCSLS,

sum(t1-/BIC/BCREC000) as /BIC/BCREC000,

sum(t1-/BIC/BSREC000) as /BIC/BSREC000,

t1-UNIT,

sum(GDS-/BIC/BSPRCAC0\*t1-/BIC/BSREC000) as /BIC/BSAC0000,

sum(GDS-/BIC/BSEXC000\*t1-/BIC/BSREC000) as /BIC/BSEXC000,

sum(GDS-/BIC/BSVAT000\*t1-/BIC/BSREC000) as /BIC/BSVAT000,

sum(GDS-/BIC/BSPRCPVD\*t1-/BIC/BSREC000) as /BIC/BSPVD000,

sum(GDS-/BIC/BSVATEXC\*t1-/BIC/BSREC000) as /BIC/BSVATEXC,

sum(GDS-/BIC/BSVATPVD\*t1-/BIC/BSREC000) as /BIC/BSVATPVD,

sum(GDS-/BIC/BSVTEXCT\*t1-/BIC/BSREC000) as /BIC/BSVTEXCT,

sum(GDS-/BIC/BSEXCCAT\*t1-/BIC/BSREC000) as /BIC/BSEXCCAT,

sum(GDS-/BIC/BSSLSTAX\*t1-/BIC/BSREC000) as /BIC/BSSLSTAX,

sum(GDS-/BIC/BSPRCMNF\*t1-/BIC/BSREC000) as /BIC/BSMNF000,

t1-/BIC/BAZS0000,

t1-/BIC/BEMIT000,

CASE WHEN MONTH(t1-DATE0) < 7 THEN 1 ELSE 2 END as HALFYEAR1,

t1-DATE0 as DATE0,

RIGHT(CONCAT('0', SUBSTRING(t1-TIME,3,2)),2) as TCTHOURSLT,

WEEKDAY(t1-DATE0) as WEEKDAY1,

t1."TIME" as "TIME",

CONCAT(SUBSTRING(t1."DATE0", 1, 4), CONCAT(RIGHT(CONCAT('0', MONTH(t1."DATE0")),2), RIGHT(CONCAT('0', DAYOFMONTH(t1."DATE0")),2))) as "CALDAY",

DAYOFMONTH(t1."DATE0") as "/BIC/BCALDAYM",

CONCAT(SUBSTRING(t1."DATE0", 1, 4),RIGHT(CONCAT('0', MONTH(t1."DATE0")),2)) as "CALMONTH",

RIGHT(CONCAT('0', MONTH(t1."DATE0")),2) as "CALMONTH2",

SUBSTRING(QUARTER(t1."DATE0"),7,1) as "CALQUART1",

CONCAT(SUBSTRING(t1."DATE0", 1, 4),SUBSTRING(QUARTER(t1."DATE0"),7,1)) as "CALQUARTER",

CASE WHEN RIGHT(CONCAT('0',WEEK(t1."DATE0")),2) = '53' THEN CONCAT(SUBSTRING(t1."DATE0", 1, 4), '52') ELSE CONCAT(SUBSTRING(t1."DATE0", 1, 4), RIGHT(CONCAT('0',WEEK(t1."DATE0")),2)) END as "CALWEEK",

SUBSTRING(t1."DATE0", 1, 4) as "CALYEAR",

SLSDOC."CURRENCY",

GDS."/BIC/BNOM0000",

sum(GDS."/BIC/BPEXC000") as "/BIC/BPEXC000",

sum(GDS."/BIC/BPVAT000") as "/BIC/BPVAT000",

GDS."/BIC/BCOMT000",

GDS."/BIC/BPRT0000",

null as "/BIC/BCRDNUM0",

0 as RECORD,

0 as SQL\_\_PROCEDURE\_\_SOURCE\_\_RECORD

from /BIC/ABDINGDS02 as t1

inner join /BIC/ABDGDS0002 as GDS on t1-/BIC/BCOM0000 = GDS-/BIC/BCOM0000

left join /BIC/ABDSLSDOC2 as SLSDOC on t1-/BIC/BEMIT000 = SLSDOC-/BIC/BEMIT000 and t1-/BIC/BCHKNUM0 = SLSDOC-/BIC/BCHKNUM0 and t1-/BIC/BAZS0000 = SLSDOC-/BIC/BAZS0000

where t1-/BIC/BEMIT000 = 600 and

t1-/BIC/BCOM0000 in ('01600100446927200000', '01600100445723100000', '01600100447731400000') and

t1-RECORDMODE = 'N'

group by t1-CURRENCY,

GDS-/BIC/BGDSCURR,

t1-IDIN,

t1-/BIC/BCOM0000,

t1-/BIC/BCHKNUM0,

t1-DATE0,

t1-TIME,

t1-/BIC/BEMIT000,

t1-/BIC/BAZS0000,

SLSDOC-/BIC/BDOCSTAT,

t1-/BIC/BSLST000,

GDS-/BIC/BNOM0000,

GDS-/BIC/BCOMT000,

GDS-/BIC/BPRT0000,

t1-RECORDMODE,

SLSDOC-/BIC/BAZSOPER,

SLSDOC-CURRENCY,

t1-UNIT

t1~idin, t1~RECORDMODE, GDS~/BIC/BGDSCURR, 0 as /BIC/BCCHK000,

doc~/BIC/BAZSOPER, t1~/BIC/BSLST000, doc~/BIC/BDOCSTAT,

sum( gds~/BIC/BSPACK00 \* t1~/BIC/BSREC000 ) as /BIC/BSPACK00,

sum( gds~/BIC/BSEXCTOT \* t1~/BIC/BSREC000 ) as /BIC/BSEXCTOT,

t1~/BIC/BCOM0000, NULL as /BIC/BPAYT000, t1~/BIC/BCHKNUM0,

null as /BIC/BSPRCSLS,

sum( t1~/BIC/BCREC000 ) as /BIC/BCREC000,

sum( t1~/BIC/BSREC000 ) as /BIC/BSREC000,

t1~UNIT as UNIT,

sum( gds~/BIC/BSPRCAC0 \* t1~/BIC/BSREC000 ) as /BIC/BSAC0000,

sum( gds~/BIC/BSEXC000 \* t1~/BIC/BSREC000 ) as /BIC/BSEXC000,

sum( gds~/BIC/BSVAT000 \* t1~/BIC/BSREC000 ) as /BIC/BSVAT000,

sum( gds~/BIC/BSPRCPVD \* t1~/BIC/BSREC000 ) as /BIC/BSPVD000,

sum( gds~/BIC/BSVATEXC \* t1~/BIC/BSREC000 ) as /BIC/BSVATEXC,

sum( gds~/BIC/BSVATPVD \* t1~/BIC/BSREC000 ) as /BIC/BSVATPVD,

sum( gds~/BIC/BSVTEXCT \* t1~/BIC/BSREC000 ) as /BIC/BSVTEXCT,

sum( gds~/BIC/BSEXCCAT \* t1~/BIC/BSREC000 ) as /BIC/BSEXCCAT,

sum( gds~/BIC/BSSLSTAX \* t1~/BIC/BSREC000 ) as /BIC/BSSLSTAX,

sum( gds~/BIC/BSPRCMNF \* t1~/BIC/BSREC000 ) as /BIC/BSMNF000,

t1~/BIC/BAZS0000 as /BIC/BAZS0000,

t1~/BIC/BEMIT000 as /BIC/BEMIT000

**Ошибки**

***using @ to escape host variables***

[Содержание](#Содержание)

lf the new Open SQL syntax is used; it must be used throughout. This includes *using @ to escape host variables*.

data: *x\_data* type standard table of /BIC/ABDIN00002.

select t1~idin, t1~RECORDMODE, GDS~/BIC/BGDSCURR, …

into corresponding fields of table *@x\_data*

from /BIC/ABDINGDS02 as t1

***Reduce the precision of any subexpressions***

[Содержание](#Содержание)

The maximum possible number of places in this expression is 35 with 4 decimal places. Decimal expressions, however, can have no more than 31 places and 14 decimal places. *Reduce the precision of any subexpressions*.

## Literals

<https://blogs.sap.com/2023/02/08/literals-in-abap/>

|  |  |  |
| --- | --- | --- |
|  | **abap** | **abap cds** |
| ***Decimal floating point literal*** | **DECFLOAT16`…`** | abap.decfloat16’…’ |
| **D16N`…`** | abap.d16n’…’ |
| **DF16\_RAW`…`** |  |
| **D16R`…`** |  |
| **DECFLOAT34`…`** | abap.decfloat34’…’ |
| **D34N`…`** | abap.d34n’…’ |
| **DF34\_RAW`…`** |  |
| **D34R`…`** |  |
| **DF16\_DEC`…`** |  |
| **D16D`…`** |  |
| **DF34\_DEC`…`** |  |
| **D34D`…`** |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

The ***literal operator &***can be used to concatenate literals. It is available in ABAP and ABAP SQL, but not in ABAP CDS.

**CAST | #CAST**

[Содержание](#Содержание)

... CAST type( dobj ) ... - performs a *down or an up cast* for the argument *dobj* and creates a reference variable of the static type *type* as a result.

Example - two equally valid down casts with the two possible casting operators ***?=*** and ***CAST***.

Вариант 1 - Приведение типа с помощью оператора *CAST*

CLASS **c1** DEFINITION. ENDCLASS.

CLASS **c2** DEFINITION *INHERITING FROM* ***c1***. ENDCLASS.

DATA: *oref1* TYPE REF TO c1, *oref2* TYPE REF TO c2.

TRY.

CAST c2( oref1 ). *" oref1 TYPE REF TO c2.*

CATCH cx\_sy\_move\_cast\_error. RETURN.

ENDTRY.

Вариант 2 - Приведение типа с помощью оператора *?=*

oref2 ?= oref1. *" oref1 -> oref2*

oref2 = *CAST #*( oref1 ).

Example

TYPES: BEGIN OF **t\_struc**,

col1 TYPE i, col2 TYPE i,

END OF t\_struc.

DATA *dref* TYPE REF TO data.

DATA *struc* TYPE t\_struc.

dref = NEW t\_struc( ).

struc = CAST t\_struc( dref )->\*.

struc-col1 = CAST t\_struc( dref )->col1.

struc-col2 = CAST t\_struc( dref )->col2.

CAST t\_struc( dref )->\* = struc.

CAST t\_struc( dref )->col1 = struc-col1.

CAST t\_struc( dref )->col2 = struc-col2.

Example - RTTS often requires a helper variable to perform a down cast of a type description object to a specialized class. These examples show how helper variables can be omitted by using the operator CAST.

DATA(attributes) =  
CAST cl\_abap\_classdescr(  
cl\_abap\_classdescr=>describe\_by\_name( 'CL\_ABAP\_FORMAT' )  
)->attributes.  
  
DATA(components) =  
CAST cl\_abap\_structdescr(  
cl\_abap\_typedescr=>describe\_by\_name( 'T100' )  
)->components.  
  
DATA(no\_of\_components) =  
lines( CAST cl\_abap\_structdescr(  
cl\_abap\_typedescr=>describe\_by\_name( 'SYST' )  
)->get\_components( ) ).

**sql\_exp - sql\_cast**

<https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-US/abensql_cast.htm>

Example - splitting a [*time stam*p in a packed number](javascript:call_link('abentime_stamps_packed.htm')) into its *date* part and its time part.

FINAL(*timestamp*) = cl\_abap\_tstmp=>utclong2tstmp\_short( utclong\_current( ) ).

DELETE FROM demo\_expressions.

INSERT demo\_expressions FROM @( VALUE #( id = 'X' timestamp1 = timestamp ) ).

SELECT SINGLE FROM demo\_expressions

FIELDS CAST( CAST( *div*( timestamp1, 1000000 ) AS CHAR ) AS DATS ) AS date,

              CAST( *substring*( CAST( timestamp1 AS CHAR ), 9, 6 ) AS TIMS ) AS time

INTO @FINAL(wa).

cl\_demo\_output=>display( wa ).

sum( gds~/BIC/BSPACK00 \* t1~/BIC/BSREC000 ) as /BIC/BSPACK00,

*Error*

abap sql The maximum possible number of places in this expression is 35 with 4 decimal places. *Decimal expressions*, however, can have no more than 31 places and 14 decimal places. Reduce the precision of any subexpressions.

Причина

/BIC/BSPACK00 CURR(17,2) \* /BIC/BSREC000 QUAN (17,3) = DEC(35,4)

В [sql\_exp - sql\_arith](https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-us/index.htm?file=abensql_arith.htm)

The operator / is not allowed in decimal expressions.

The result has the type DEC with the length 31 and a maximum of 14 decimal places. Using the associated ***assignment rule***, it can be assigned to all numeric ABAP types whose value range is large enough, except for decimal floating point numbers.

**Decimal Floating Point Numbers**

The data types for [decimal floating point numbers](javascript:call_link('abenbinfloat_glosry.htm')) are [***decfloat16***](javascript:call_link('abenbuiltin_types_numeric.htm')) and [***decfloat34***](javascript:call_link('abenbuiltin_types_numeric.htm')). The maximum precision is 16 places or 34 places, respectively.

[***Arithmetic expressions***](javascript:call_link('abenarithmetic_expression_glosry.htm'))with decimal floating point numbers always have the [calculation type](javascript:call_link('abencalculation_type_glosry.htm')) *decfloat34*.

Hana ***QUAN*** (m,n) -> abap ***p*** length m/2+1 decimals n

Hana ***CURR*** (m,n) -> abap ***p*** length m/2+1 decimals n

types: quan type ***p*** length *9* decimals *2*,

dec type ***decfloat34***.

# SAP CONSTRUCTOR EXPRESSION CAST

<https://www.trailsap.com/sap-help/?topic=constructor_expression_cast>