Order Interface 7.2-7.4

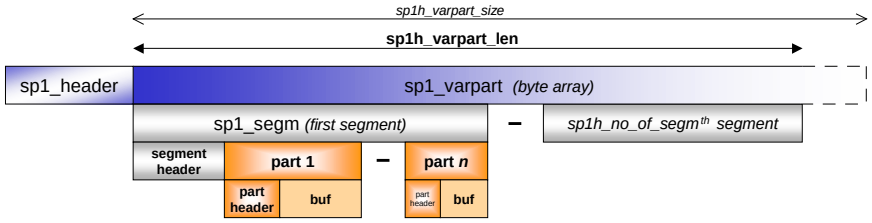
Kernel-Quicky 2001/05/10

SQL Packet

```
tsp1_packet = RECORD
   sp1_header : tsp1_packet_header;
   CASE integer OF
      1:    (sp1_varpart  : tsp_moveobj);

      2:    (sp1_segm     : tsp1_segment);
   END;
```

```
tsp1_packet_header = RECORD
   sp1h_mess_code      : tsp_code_type;
   sp1h_mess_swap      : tsp_swap_kind;
   sp1h_filler1        : tsp_int2;
   sp1h_appl_version   : tsp_c5;
   sp1h_application    : tsp_c3;
   sp1h_varpart_size   : tsp_int4;
   sp1h_varpart_len    : tsp_int4;
   sp1h_filler2        : tsp_int2;
   sp1h_no_of_segm     : tsp_int2;
   sp1h_filler3        : tsp_c8
END;
```



More than 1 segment/command can be send to the kernel at once.

For each input-segment one output-segment is send back (in the same ordering)

Output-segments are BEHIND the input-segments in the same SQL-packet

(to give clients the chance to reuse parts of the input; as far as I know: not used by clients;

useful for kernel, because no intermediate storing of the following segments/parts is necessary)

The whole SQL-packet has to have at least 16KB and not more than 128KB (_PACKET_SIZE in cserv.pcf)

To be able to return at least a ‚not enough space in order interface to answer correctly‘ a minimal number

of bytes are reserved in the order interface which may only be used for answers kernel **à** client

(_MINREPLY_SIZE in cserv.pcf).

The number of bytes available for the communication client **à** kernel are

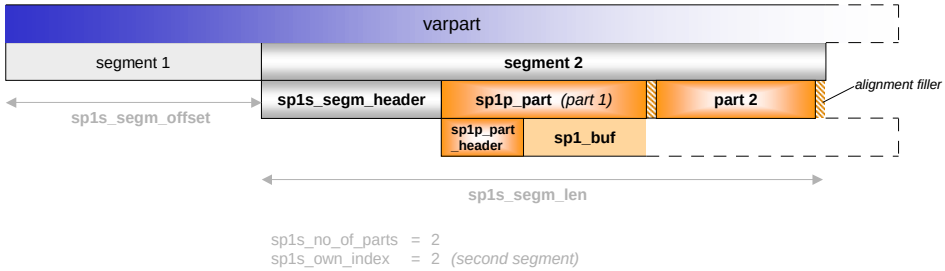
_PACKET_SIZE - _MINREPLY_SIZE - some overhead for the OS (depending on the OS)

SQL Packet: Segment

```
tsp1_segment = RECORD
   CASE integer OF
      1: (  sp1s_segm_header    : tsp1_segment_header;
            sp1p_part           : tsp1_part);

      2: (  sp1s_space1         : tsp1_segment_header;
            sp1p_part_header    : tsp1_part_header;
            sp1p_buf            : tsp_moveobj);
   END;
```

```
tsp1_segment_header = RECORD
    sp1s_segm_len      : tsp_int4;
    sp1s_segm_offset   : tsp_int4;
    sp1s_no_of_parts   : tsp_int2;
    sp1s_own_index     : tsp_int2;
    sp1s_segm_kind     : tsp1_segment_kind;
    sp1c_mess_type     : tsp1_cmd_mess_type;
    sp1c_sqlmode       : tsp1_sqlmode;
    ...
    sp1c_mass_cmd   : boolean;
    ...
END;
```



sp1s_no_of_parts = 2
sp1s_own_index = 2 *(second segment)*

One segment consists of 0 (return-segment with no output and no error) to n parts.

In the segment header it is given if this is a cmd- (client **à** kernel) or return-segment (kernel **à** client)

In the segment header it is specified (sp1c_mess_type) what kind of command it is, for example:

dbs (parse SQL-statements, prepare internal messbuffer and execute at once, no data given)

parse (parse SQL-statements and prepare internal messbuffer)

execute (execute messbuffer prepared earlier for SQL-statement using given data)

incopy (pages are send to the kernel during restore and so on)

utility (parse and execute utility-commands like RESTART, SAVE…)

In the segment header the sqlmode can be specified, usually as 'session_sqlmode', but sometimes with a

special sqlmode for just this single statement

In the segment header it can be specified during parse that more than one dataportion (more than one row

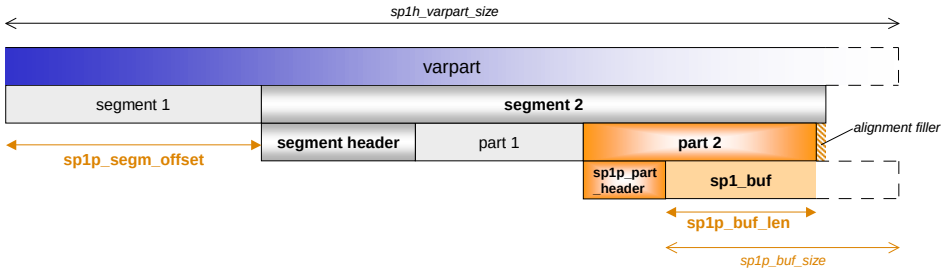
in an insert for example) will be given during one execute call for this SQL-statement

Each part in a segment and each segment is aligned to be able to use pointers to those segments/parts. These pointers will point to pascal-records/C-structs.

SQL Packet: Part

```
tsp1_part = RECORD
    sp1p_part_header  : tsp1_part_header;
    sp1p_buf          : tsp_moveobj
END;
```

```
tsp1_part_header = RECORD
    sp1p_part_kind      : tsp1_part_kind;
    sp1p_attributes     : tsp1_part_attr;
    sp1p_arg_count      : tsp_int2;
    sp1p_segm_offset    : tsp_int4;
    sp1p_buf_len        : tsp_int4;
    sp1p_buf_size       : tsp_int4
END;
```



In the part header is specified, what part kind this part is of (dbs, parse, execute, utility, ...), that is: how to interpret the buffer behind the header

In one segment, 0 or 1 (not more) parts of one part kind may exist.

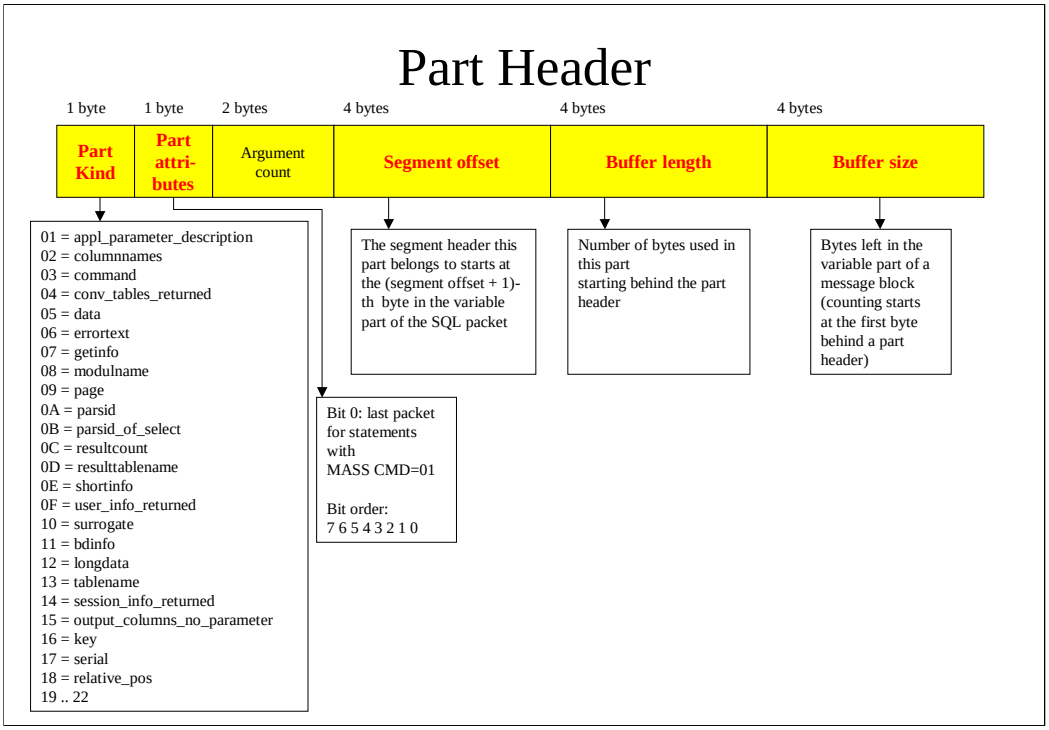
In the part header is specified how many arguments belong to this part. Usually this is 1.

With data and masscommand this may be > 1.

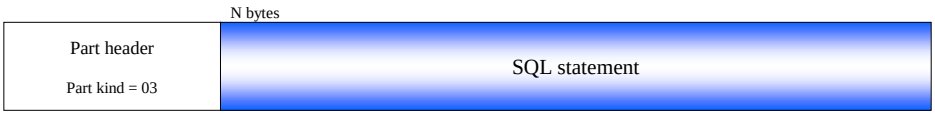
With shortinfo and/or columnnames usually it is > 1.

Only few other partkinds have more than 1 argument.

Part Header


1 byte
1 byte
2 bytes
4 bytes
4 bytes
4 bytes
Part Kind
Part attri-butes
Argument count
Segment offset
Buffer length
Buffer size
01 = appl_parameter_description
02 = columnnames
03 = command
04 = conv_tables_returned
05 = data
06 = errortext
07 = getinfo
08 = modulname
09 = page
0A = parsid
0B = parsid_of_select
0C = resultcount
0D = resulttablename
0E = shortinfo
0F = user_info_returned
10 = surrogate
11 = bdinfo
12 = longdata
13 = tablename
14 = session_info_returned
15 = output_columns_no_parameter
16 = key
17 = serial
18 = relative_pos
19 .. 22
Bit 0: last packet for statements with MASS CMD=01
Bit order:
7 6 5 4 3 2 1 0
The segment header this part belongs to starts at the (segment offset + 1)-th byte in the variable part of the SQL packet
Number of bytes used in this part starting behind the part header
Bytes left in the variable part of a message block (counting starts at the first byte behind a part header)

Part kind: command

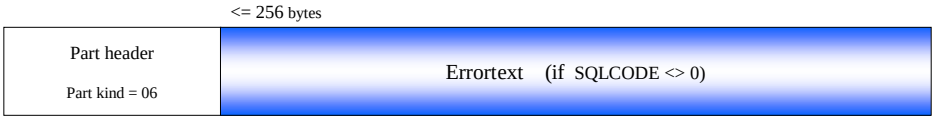


• **requests** with the following MESSAGE TYPEs: *DBS*
PARSE
GETPARSE
SYNTAX

The readable SQL statement is written there

Part kind: errortext

(used for results)



The errortext is written there.

The errornumber will be returned in the return-segment-header

Example 1

```
Input with writing error near ‚WHERE‘ (WHRE)

REQUEST: ascii, full_swap, 70205-XCI     (1 segment, len: 120)
  dbs SEGMENT 1   (1 part, len: 120)
       session_sqlmode, user_cmd
       with_info
  command PART   (1 argument, size: 28136)
       buf(57):
       'SELECT EMPNAME, EMPNO FROM EMP WHRE FIRST_NAME = 'Martin''


RECEIVE: ascii, full_swap, 70205-XCI     (1 segment, len: 88)
  *** -3014 / RETURN SEGMENT 1   (1 part, len: 88)
       select_fc, errpos: 37, sqlstate: '42000'
  errortext PART   (1 argument, size: 32080)
       buf(28): 'Invalid end of SQL statement'
```

1 segment with one part is send to the kernel.

This single part consists of the readable SQL-statement.

Because this SQL-statement can not be handled, an error message is send back, which consists of

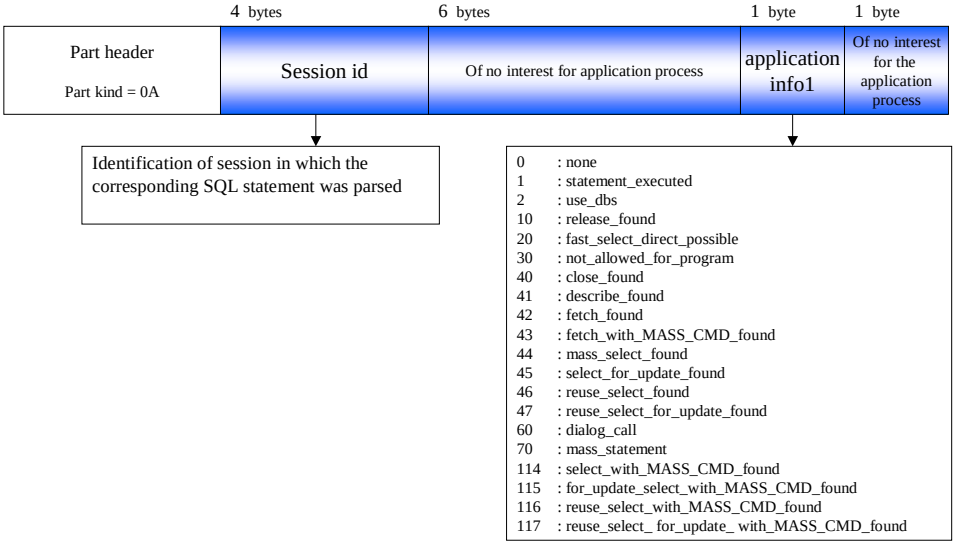
one part with the errortext.

The errorno (-3014) is send back in the segment header, together with

- the function_code (..._fc, to distinguish commit, insert, select, create... from each other. The client shall notparse the statement to get this info, which is necessary at least for OCI)
- the errorposition in the statement
- the sqlstate (the ANSI-standard-‘errorno‘)

Part kind: parseid

(used for requests and results)



With session-release the parseid is of no use any more. Every client should throw away parseids (somewhere

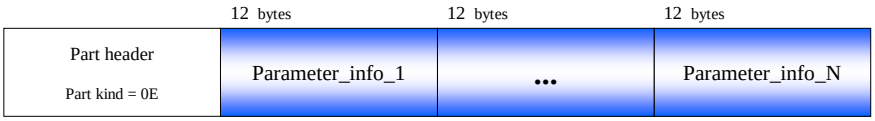
stored in his memory) belonging to the old session.

The last but one byte existed before the function code in the return-segment-header was created. Sometimes

(MASS_CMD... or use_dbs (no parsing for this statement allowed)) a little bit more info is in this byte

than in the function_code of the return segment header

Part kind: shortinfo, pic. 1



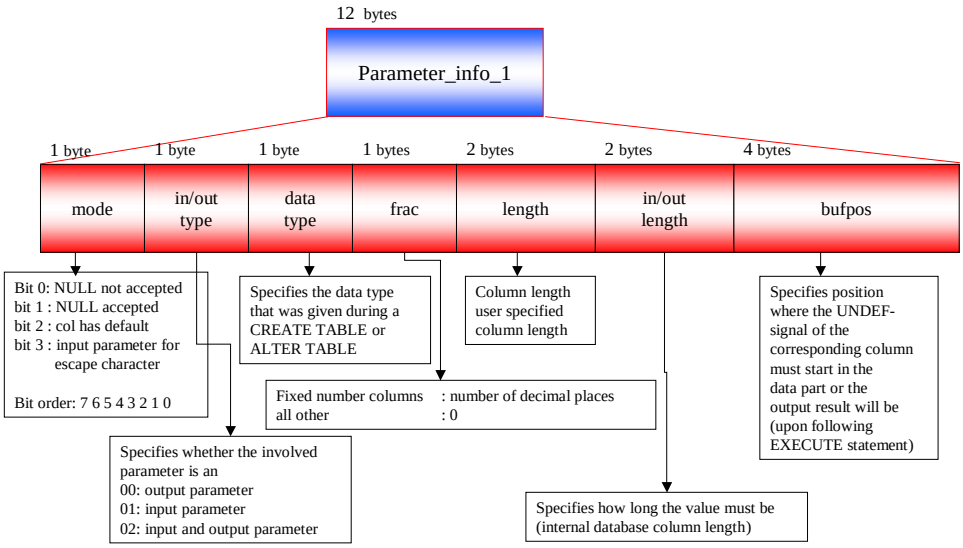
- **Returned part** for requests with MESSAGE TYPE: *PARSE* or *DBS*
- this part contains a description of all input and output parameters used in the statement

Parameter are

-:<name>

-?

Part kind: shortinfo, pic. 2



Length and in/out length correspond. Frac is of no use for calculating the in/out length.

in/out length

LONG-col (length = 0) : 41

FIXED (n[,m]), FLOAT (n) : (n+1) DIV 2 + 2

CHAR (n) : n+1
(no unicode order interface)

CHAR (n) : (2*n) + 1 (
unicode order interface)

CHAR (n) UNICODE : (2*n) + 1

.....

Every data has fixed length and an undef-byte in front of it.

The undef-byte shows x‘FF‘ if the NULL-value was specified, x’00‘, x’20‘, x’40‘, x’01‘ otherwise, depending on the data type

Example 2.1

Input to be just prepared, not executed, with input- and output-variables

```
REQUEST: ascii, full_swap, 70205-XCI     (1 segment, len: 120)
  parse SEGMENT 1   (1 part, len: 120)
       session_sqlmode, user_cmd
       with_info
  command PART   (1 argument, size: 28136)
       buf(64):
       'SELECT NUMBVAL, TI INTO :NUMB, :TIME FROM T WHERE KEYCOL = :KEYC'

RECEIVE: ascii, full_swap, 70205-XCI     (1 segment, len: 128)
  ok / RETURN SEGMENT 1   (2 parts, len: 128)
       select_into, sqlstate: '00000'
  shortinfo PART   (3 arguments, size: 32080)
        1.  pos:    1  io_len:   8 output fixed(12)       optional
        2.  pos:    9  io_len:   9 output time(8)         optional
        3.  pos:    1  io_len:   9 input  char_asc(8)     optional
  parsid PART   (1 argument, size: 32024)
       mess_type: select
       buf(12):
       00000019 00000601 3C000000
```

This is a vtrace-output:

REQUEST:

way from client to kernel

RECEIVE:

way back from kernel to client

The example was prepared with the client-tool XCI (not distributed) of the version 7.02.05.

Part kind: data

(used for requests and results)

- **Received part**
  - requests with MESSAGE TYPES: *DBS* or *DBSINFO* or *EXECUTE* or *GETEXECUTE*
  - request with SELECTs whose first results are returned during the execution of SELECT
- **Request part**
  - SQL statements with MESSAGE TYPES: *PARSE* or *GETPARSE* which include parameters. The data for those parameters must be specified during the corresponding EXECUTE or GETEXECUTE (the data must be given as described in a part of part kind shortinfo which will be returned during PARSE).
  - SQL statements ALTER PASSWORD, CONNECT, CREATE USER, SET LANGUAGE, SET FORMAT, USAGE ... are specified and a SQL statement with MESSAGE TYPE DBS contains parameters. The data, corresponding to the parameters must be given in a part of part kind data in the same segment.

Data is filled in this part according to the part kind shortinfo, starting at position bufpos (in the buffer behind the

part header).

Data is filled close to each other. It is NOT aligned. Even unicode-data (sometimes handled as int) are NOT aligned.

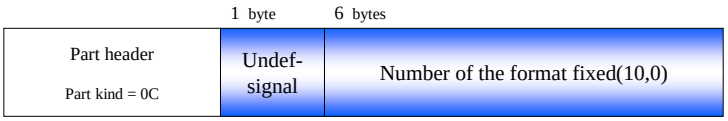
Each data has ist undef-byte with him and is of fixed length. If neccesary it is filled with x'00', or blank.

Numbers (FIXED, FLOAT) are stored in a way to be comparable byte-wise, that is:

one byte characteristic, up to 19 bytes (maximum of 38 significant digits) mantissa, one digit per half-byte

Part kind: resultcount

(used for requests and results)



- **Returned part** for requests for all MESSAGE TYPEs except: ***SYNTAX*** and the following SQL statements: INSERT, UPDATE, DELETE, SELECT, DECLARE CURSOR, FETCH, RFETCH
  - specifies the number of rows inserted, updated, deleted, found or fetched
- **Request part** for SQL statements FETCH with MASS CMD or RFETCH
  - specifies the number of wanted result rows (by the application)

Number –1 is returned in case of ‚the kernel does not know by now‘ because the result set was not copied, that

means, the kernel does not know how many rows will fulfil the conditions.

Number –1 is returned in case of ‚the kernel does not know because he did not count‘ the number of deleted rows

if the whole table was deleted

Number 0 (usually in Oracle mode) is returned after a SELECT / DECLARE CURSOR was executed and means: no result rows are fetched by now

Example 2.2

Execute for the SELECT .. INTO :NUMB, :TIME FROM T WHERE KEYCOL = :KEYC'

```
REQUEST: ascii, full_swap, 70205-XCI     (1 segment, len: 104)
  execute SEGMENT 1   (2 parts, len: 104)
       session_sqlmode, user_cmd
       with_info
  parsid PART   (1 argument, size: 28136)
       mess_type: select
       buf(12):
       00000019 00000601 3C000000
  data PART   (1 argument, size: 28104)
       buf(9): ' keyval1 '

RECEIVE: ascii, full_swap, 70205-XCI     (1 segment, len: 104)
  ok / RETURN SEGMENT 1   (2 parts, len: 104)
       select_into, sqlstate: '00000'
       external WARNING   0: warning_exists
       external WARNING   8: table_scan
  resultcount PART   (1 argument, size: 32096)
       result_count: 1
  data PART   (1 argument, size: 32072)
       buf(17):
       00C44711 00000000 20303031 34333934 34 '..G..... 00143944'
```

Remember:

every data has undef-byte in front

numbers start with the mantissa (C4 means : 10^4), $0.4711 * 10^4$

Advantage of this Layout

The shown layout has the advantage of being very flexible:

- a SQL packet may consist of one or more segments each with one command
- a segment may consist of n (n of nearly any number, up to the number of different part kinds) parts
- if a new part kind has to be created, this is easy possible.
  Every client and the kernel have to ignore unknown part kinds to be backward and forward compatible with older/newer releases

More Info

The first 3 pages are taken from http://pwww/Kern/v62/Internals62.ppt. Page 3 to 5.

The next pages are taken (and a little bit modified) from
//P26326/InetPub/wwwroot/SAP_DB_80/Foren/parts_header.ppt

A word description of the order interface can be found:
Version 6.2: http://pwww/Kern/v62/xorder.htm
Version 7.*: http://pwww/Kern/v72/xorder7.doc