PUBLIC

SAP HANA Platform 2.0 SPS 00
Document Version: 1.0 – 2016-11-30

SAP HANA SQL Command Network Protocol Reference



Content

1 Introduction. 5
1.1 Terminology. 5
1.2 Protocol Modification. 6

2 Message Format. 7
2.1 Message Header. 7
2.2 Segment Header. 8
Segment Kind. 9
Message Type. 9
Command Options. 10
Function Code. 10
2.3 Part Header. 11
Part Kind. 12
Part Attributes. 14
2.4 Part Support in Request Messages. 14
2.5 Part Support in Reply Messages. 16
2.6 Type Codes. 18
2.7 Part Data Format. 20
Option Part Format. 20
Multi-Line Option Part Format. 21
COMMAND. 21
RESULTSET. 22
ERROR. 22
STATEMENTID. 23
TRANSACTIONID. 23
ROWSAFFECTED. 23
RESULTSETID. 23
TOPOLOGYINFORMATION. 24
TABLELOCATION. 26
READLOBREQUEST. 26
READLOBREPLY. 27
ABAPISTREAM. 27
ABAPOSTREAM. 28
COMMANDINFO. 28
WRITELOBREQUEST. 28
WRITELOBREPLY. 29
PARAMETERS. 29

AUTHENTICATION. 30
SESSIONCONTEXT. 31
STATEMENTCONTEXT. 32
PARTITIONINFORMATION. 32
OUTPUTPARAMETERS. 34
CONNECTOPTIONS. 34
COMMITOPTIONS. 41
FETCHOPTIONS. 41
FETCHSIZE. 41
PARAMETERMETADATA. 41
RESULTSETMETADATA. 43
FINDLOBREQUEST. 44
FINDLOBREPLY. 44
ITABSHM. 44
CLIENTINFO. 45
STREAMDATA. 45
BATCHPREPARE. 45
BATCHEXECUTE. 45
TRANSACTIONFLAGS. 46
DBCONNECTINFO. 46
LOBFLAGS. 47
2.8 Input and Output Field Formats. 47
TINYINT. 47
SMALLINT. 48
INT. 48
BIGINT. 49
DECIMAL. 50
REAL. 50
DOUBLE. 51
STRING/NSTRING. 51
BINARY. 52
BLOB/CLOB/NCLOB. 52
DATE. 54
TIME. 54
TIMESTAMP. 54
ABAPITAB. 55
ABAPSTRUCT. 55
LONGDATE. 56
SECONDDATE. 56
DAYDATE. 56
SECONDTIME. 56

ST_POINT/ST_POINTZ/ST_GEOMETRY. 57
2.9 Usage Scenarios. 57
Communication Initialization. 57
Authentication and Connect. 57
ABAP Stream Handling. 66
Distributed Transaction Handling. 67
Statement Routing. 67

3 Glossary. 69

1 Introduction

This document describes the SQL Command Network Protocol of the SAP HANA database, the protocol used by SAP HANA database clients to communicate with the SAP HANA database.

This document describes the binary message format used in the communication, and explains the purpose of the various protocol elements. It also discusses usage scenarios for the various messages defined in the SQL Command Network Protocol and defines usage sequences to utilize database server functionality, such as:

- User authentication and connect process
- Execution of prepared statements
- Handling of large object data
- Usage of distributed transaction handling

1.1 Terminology

Within this document, specific terminology and abbreviations are used.

A **message** describes requests or replies exchanged between client and server. A **request** is always a message sent by the client, and a **reply** is always the message sent by the server.

The following abbreviations are used to describe data types in message formats:

Table 1:

Abbreviation	Data Type	C Data Type[1]
I1	1-byte integer value	`char`
UI1	1-byte unsigned integer value	`unsigned char`
B	1-byte unsigned integer value, or “one byte”	`unsigned char`
I2	2-byte integer in little-endian format	`short`
UI2	2-byte unsigned integer in little-endian format	`unsigned short`
BI2	2-byte integer in big-endian format	`unsigned short`
I4	4-byte integer in little-endian format	`int`
NI4	4-byte integer in client native (big/little-endian) format	`int`
UI4	4-byte unsigned integer in little-endian format	`unsigned int`
I8	8-byte integer in little-endian format	`long`
NUI8	8-byte unsigned integer in client native (big/little-endian) format	`unsigned long`
FLOAT	IEEE single precision floating point value in little-endian format	`float`

[1] For a Linux GCC in an x86-64 architecture environment.

Abbreviation	Data Type	C Data Type[1]
DOUBLE	IEEE double precision floating point value in little-endian format	`double`
X[n]	Array of n elements of data type X.	

1.2 Protocol Modification

Additions to the protocol require careful modification and extension of the exchanged connect options.

The SQL Command Network Protocol must be kept stable so that clients are able to communicate with recent version server software, and vice versa. A more recent client may have to degrade the usage of the protocol depending on the server version, and a more recent server may have to use only certain features if non-recent client software is detected.

Detection is performed during connection by exchanging connect options, which specify the required behavior of the client, and getting the supported feature set from the server. This detection does not perform global versioning, but enables or disables feature flags, or modifies certain functionality.

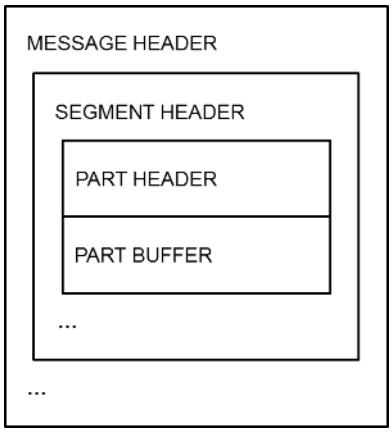
[1] For a Linux GCC in an x86-64 architecture environment.

2 Message Format

The communication between client and server is completely synchronous: the client can only send the next request once the reply to the previous request has been fully received.

A client is free to continue with its own processing, or communicate with other servers of a SAP HANA database system while waiting for the answer.

A message consists of a fixed part, called the **message header** and a variable length **message buffer**. The message buffer contains **message segments**, which, in turn, consist of a **segment header** and a **segment buffer**. The segment buffer contains parts, and the parts have a fixed length **part header** and a **variable length buffer**.



There is one exception to this format: during communication initialization, a different message pair is exchanged that is necessary to distinguish the current and former protocol variants.

2.1 Message Header

The message header is the fixed part of the message.

The message header is a 32 byte structure that consists of the following fields:

Table 2:

Field	Data Type	Description
SESSIONID	I8	A session identifier.
PACKETCOUNT	I4	A packet sequence number in this session. Packets with the same sequence number belong to one request/response pair.
VARPARTLENGTH	UI4	The amount of space used in the packet. The maximum size is 2G -1.
VARPARTSIZE	UI4	The total amount of space in the packet. The maximum size is 2G -1.

NOOFSEGM	I2	The number of segments in the packet.
PACKETOPTIONS	I1	Reserved. Do not use.
RESERVED	B[9]	Reserved. Do not use.

2.2 Segment Header

A segment header is a part of the message segment in the message buffer.

The segment header has a length of 24 bytes. There are different segment header structures for request and reply, but both have the same definition for first 13-bytes of the structure. They are structured as follows:

Table 3:

Request Segment Header					
SEGMEN-TLENGTH	SEGMENTOFS	NOOFPARTS	SEGMENTNO	SEGMENTKIND	...

Table 4:

Reply Segment Header					
SEGMEN-TLENGTH	SEGMENTOFS	NOOFPARTS	SEGMENTNO	SEGMENTKIND	...
...	RESERVED2	FUNCTIONCODE	RESERVED3		

Table 5:

Field	Data Type	Description
SEGMENTLENGTH	I4	The length of the segment, including the header.
SEGMENTOFS	I4	An offset of the segment within the message buffer.
NOOFPARTS	I2	The number of contained parts.
SEGMENTNO	I2	The number of segments within the packet.
SEGMENTKIND	I1	The segment kind, which further specifies the layout of the remaining segment header structure.
MESSAGETYPE	I1	The action requested from the database server..
COMMIT	I1	A value specifying whether the command is committed.
COMMANDOPTIONS	I1	A bit set that defines specific options for the sent message.
RESERVED1	B[8]	Reserved. Do not use.
RESERVED2	I1	Reserved. Do not use.
FUNCTIONCODE	I2	A value that defines the nature of the statement or functionality that has been prepared or executed.

RESERVED3	B[8]	Reserved. Do not use.

2.2.1 Segment Kind

The segment kind field is the last field of the segment header part, common to all types (kinds) of segments. It defines the segment kind, which further specifies the layout of the remaining segment header structure.

The following values are defined in the order listed below:

Table 6:

Value	Description
0	Reserved for invalid segments. Do not use.
1	A request segment.
2	A reply segment.
5	An error segment (reply segment containing error).

2.2.2 Message Type

The message type defines the action requested from the database server.

The following values are defined:

Table 7:

Value	Identifier	Description
0	NIL	Reserved for invalid messages. Do not use.
2	EXECUTEDIRECT	Directly executes an SQL statement.
3	PREPARE	Prepares an SQL statement.
4	ABAPSTREAM	Handles an ABAP stream parameter from a database procedure.
5	XA_START	Starts a distributed transaction.
6	XA_JOIN	Joins a distributed transaction.
13	EXECUTE	Executes a previously prepared SQL statement.
16	READLOB	Reads large object data.
17	WRITELOB	Writes large object data.
18	FINDLOB	Finds data in a large object.
25	PING	Reserved (was PING message).
65	AUTHENTICATE	Sends authentication data.
66	CONNECT	Connects to the database.
67	COMMIT	Commits the current transaction.

68	ROLLBACK	Rolls back the current transaction.
69	CLOSERESULTSET	Closes a result set.
70	DROPSTATEMENTID	Drops a prepared statement identifier.
71	FETCHNEXT	Fetches the next result from the result set.
72	FETCHABSOLUTE	Moves the cursor to the given row number and fetches the data.
73	FETCHRELATIVE	Moves the cursor a number of rows relative to the position, either positive or negative, and fetches the data.
74	FETCHFIRST	Moves the cursor to the first row and fetches the data.
75	FETCHLAST	Moves the cursor to the last row and fetches the data.
77	DISCONNECT	Disconnects the session.
78	EXECUTEITAB	Executes a command in Fast Data Access mode.
79	FETCHNEXTITAB	Fetches the next result for an ITAB object in Fast Data Access mode.
80	INSERTNEXTITAB	Inserts the next result for an ITAB object in Fast Data Access mode.
81	BATCHPREPARE	Reserved (was multiple statement preparation).
82	DBCONNECTINFO	Requests/receives database connect information.

2.2.3 Command Options

The command options field is a bit set that allows specific options for the sent message to be defined.

The options are as follows:

Table 8:

Bit	Identifier	Description
1	RESERVED_FIELD	Reserved field. Do not use.
3	HOLD_CURSORS_OVER_COMMIT	Keeps the result set created by this command over commit time.
4	EXECUTE_LOCALLY	Executes a command only on local partitions of affected partitioned table.
5	SCROLL_INSENSITIVE	Marks the result set created by this command as scroll insensitive.

2.2.4 Function Code

The function code identifies the nature of the statement or functionality that has been prepared or executed.

Table 9:

Value	Identifier	Description
0	NIL	Invalid command or function

1	DDL	DDL statement
2	INSERT	INSERT statement
3	UPDATE	UPDATE statement
4	DELETE	DELETE statement
6	SELECTFORUPDATE	SELECT...FOR UPDATE statement
7	EXPLAIN	EXPLAIN statement
8	DBPROCEDURECALL	CALL statement
9	DBPROCEDURECALLWITHRESULT	CALL statement returning one or more results
10	FETCH	FETCH message
11	COMMIT	COMMIT message or statement
12	ROLLBACK	ROLLBACK message or statement
13	SAVEPOINT	Reserved, do not use
14	CONNECT	CONNECT or AUTHENTICATION message
15	WRITELOB	WRITELOB message
16	READLOB	READLOB message
17	PING	Reserved--do not use
18	DISCONNECT	DISCONNECT message
19	CLOSECURSOR	CLOSECURSOR message
20	FINDLOB	FINDLOB message
21	ABAPSTREAM	ABAPSTREAM message
22	XASTART	XA_START message
23	XAJOIN	XA_JOIN message

2.3 Part Header

The part header is 16-bytes in length.

The header fields are defined as follows:

Table 10:

Field	Data Type	Description
PARTKIND	I1	Specifies nature of part data.
PARTATTRIBUTES	I1	Specifies further attributes of part.
ARGUMENTCOUNT	I2	Specifies the argument count, number of elements in part data.

BIGARGUMENTCOUNT	I4	Specifies the argument count, number of elements in part data (only for some part kinds).
BUFFERLENGTH	I4	Specifies the length of the part buffer in bytes (used space).
BUFFERSIZE	I4	Specifies the length remaining in the packet.

Related Information

Part Kind [page 12]
Part Attributes [page 14]
Part Kind [page 12]
Part Attributes [page 14]

2.3.1 Part Kind

The part kind field is the first section of the part header.

The following values are defined for the PARTKIND field:

Table 11:

Value	Identifier	Description
0	NIL	Reserved for invalid part, do not use
3	COMMAND	SQL Command Data
5	RESULTSET	Tabular result set data
6	ERROR	Error information
10	STATEMENTID	Prepared statement identifier
11	TRANSACTIONID	Transaction identifier
12	ROWSAFFECTED	Number of affected rows of DML statement
13	RESULTSETID	Identifier of result set
15	TOPOLOGYINFORMATION	Topology information
16	TABLELOCATION	Location of table data
17	READLOBREQUEST	Request data of READLOB message
18	READLOBREPLY	Reply data of READLOB message
25	ABAPISTREAM	ABAP input stream identifier
26	ABAPOSTREAM	ABAP output stream identifier
27	COMMANDINFO	Command information
28	WRITELOBREQUEST	Request data of WRITELOB message

30	WRITELOBREPLY	Reply data of WRITELOB message
32	PARAMETERS	Parameter data
33	AUTHENTICATION	Authentication data
34	SESSIONCONTEXT	Session context information
39	STATEMENTCONTEXT	Statement visibility context
40	PARTITIONINFORMATION	Table partitioning information
41	OUTPUTPARAMETERS	Output parameter data
42	CONNECTOPTIONS	Connect options
43	COMMITOPTIONS	Commit options
44	FETCHOPTIONS	Fetch options
45	FETCHSIZE	Number of rows to fetch
47	PARAMETERMETADATA	Parameter metadata (type and length information)
48	RESULTSETMETADATA	Result set metadata (type, length, and name information)
49	FINDLOBREQUEST	Request data of FINDLOB message
50	FINDLOBREPLY	Reply data of FINDLOB message
51	ITABSHM	Information on shared memory segment used for ITAB transfer
53	ITABCHUNKMETADATA	Reserved, do not use
55	ITABMETADATA	Information on ABAP ITAB structure for ITAB transfer
56	ITABRESULTCHUNK	ABAP ITAB data chunk
57	CLIENTINFO	Client information values
58	STREAMDATA	ABAP stream data
59	OSTREAMRESULT	ABAP output stream result information
60	FDAREQUESTMETADATA	Information on memory and request details for FDA request
61	FDAREPLYMETADATA	Information on memory and request details for FDA reply
62	BATCHPREPARE	Reserved, do not use
63	BATCHEXECUTE	Reserved, do not use
64	TRANSACTIONFLAGS	Transaction handling flags
65	ROWDATAPARTMETADATA	Reserved, do not use
66	COLDATAPARTMETADATA	Reserved, do not use
67	DBCONNECTINFO	Reserved, do not use
68	LOBFLAGS	LOB flags
69	RESULTSETOPTIONS	Additional context data for result sets
70	XATRANSACTIONINFO	XA transaction information

2.3.2 Part Attributes

The part attributes make up a bit set to indicate special features of a certain part.

The following values are defined:

Table 12:

Bit	Identifier	Description
0	LASTPACKET	The last part in a sequence of parts (FETCH, array command EXECUTE).
1	NEXTPACKET	A part in a sequence of parts.
2	FIRSTPACKET	The first part in a sequence of parts.
3	ROWNOTFOUND	An empty part, which is caused by a row not found error.
4	RESULTSETCLOSED	The result set that produced this part is closed
5	RESERVED5	Reserved. Do not use.
6	RESERVED6	Reserved. Do not use.
7	RESERVED7	Reserved. Do not use.

2.4 Part Support in Request Messages

Parts can be used in a variety of request messages.

In the following table, letters denote the relationship between the part and the message:

- X The default relationship between the part and the message.
- A This part can always be sent if distributed transaction handling requires it.
- B Supported when the prepared statement has input parameters.
- C Supported when the client application set information flags.

PART KIND \ MESSAGE TYPE	EXECUTE DIRECT	PREPARE	ABAP STREAM	XA_START	XA_JOIN	EXECUTE	WRITELOB	READLOB	FINDLOB	AUTHENTICATE	CONNECT	COMMIT	ROLLBACK	FETCHNEXT	DISCONNECT	CLOSERESULTSET	DROPSTATEMENTID	EXECUTEITAB	FETCHNEXTITAB	INSERTNEXTITAB
NIL																				
COMMAND	X	X																		
RESULTSET																				
ERROR																				
STATEMENTID						X											X			
TRANSACTIONID					X															
ROWSAFFECTED																				
RESULTSETID														X		X				
TOPOLOGYINFORMATION																				
TABLELOCATION																				
READLOBREQUEST								X												
READLOBREPLY																				
ABAPISTREAM			X																	
ABAPOSTREAM			X																	
COMMANDINFO	X	X				X														
WRITELOBREQUEST							X													
WRITELOBREPLY																				
PARAMETERS						B														
AUTHENTICATION										X	X									
SESSIONCONTEXT	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
STATEMENTCONTEXT	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
PARTITIONINFORMATION																				
OUTPUTPARAMETERS																				
CONNECTOPTIONS											X									
COMMITOPTIONS												X	X							
FETCHOPTIONS														X						
FETCHSIZE														X						
PARAMETERMETADATA																				
RESULTSETMETADATA																				
FINDLOBREQUEST									X											
FINDLOBREPLY																				
ITABSHM																				
ITABCHUNKMETADATA																				
ITABMETADATA																				
ITABRESULTCHUNK																				
CLIENTINFO	C	C				C								C						
STREAMDATA			X																	
OSTREAMRESULT			X																	
FDAREQUESTMETADATA																				
FDAREPLYMETADATA																				
BATCHPREPARE																				
BATCHEXECUTE																				
TRANSACTIONFLAGS																				

2.5 Part Support in Reply Messages

Parts can be used in a variety of reply messages.

In the following table, the letters denote the relationship between the part and the message:

- X The default relationship between the part and the message.
- A Supported when the message yields one or more result sets.
- B Always as a reply in case of errors while processing the statement.
- C Supported when the statement generates a row count of affected rows.
- D Always if the statement caused an action in transaction handling (commit, rollback, or start of a new write transaction)
- E Available if the server has information useful to applying statement routing (see the corresponding Usage Scenario Statement Routing).
- F Supported when the statement has input and/or output parameters.
- G Supported when the statement has output parameters.
- H Indicates a possible change of visible version.
- I Supported when the parameter metadata has changed due to schema changes.

MESSAGE TYPE / PART KIND	EXECUTEDIRECT	PREPARE	ABAPSTREAM	XA_START	XA_JOIN	EXECUTE	WRITELOB	READLOB	FINDLOB	AUTHENTICATE	CONNECT	COMMIT	ROLLBACK	FETCHNEXT	DISCONNECT	CLOSERESULTSET	DROPSTATEMENTID	EXECUTEITAB	FETCHNEXTITAB	INSERTNEXTITAB
NIL																				
COMMAND																				
RESULTSET	A					A														
ERROR	B	B	B	B	B	B	B	B	B											
STATEMENTID		X																		
TRANSACTIONID				X																
ROWSAFFECTED	C					C														
RESULTSETID	A					A														
TOPOLOGYINFORMATION																				
TABLELOCATION		E																		
READLOBREQUEST																				
READLOBREPLY								X												
ABAPISTREAM			X																	
ABAPOSTREAM			X																	
COMMANDINFO																				
WRITELOBREQUEST																				
WRITELOBREPLY							X													
PARAMETERS																				
AUTHENTICATION										X										
SESSIONCONTEXT																				
STATEMENTCONTEXT	H	H	H	H	H	H	H	H	H											
PARTITIONINFORMATION		E																		
OUTPUTPARAMETERS						G														
CONNECTOPTIONS																				
COMMITOPTIONS																				
FETCHOPTIONS																				
FETCHSIZE																				
PARAMETERMETADATA		F				I														
RESULTSETMETADATA	A	A				A														
FINDLOBREQUEST																				
FINDLOBREPLY									X											
ITABSHM																				
ITABCHUNKMETADATA																				
ITABMETADATA																				
ITABRESULTCHUNK																				
CLIENTINFO																				
STREAMDATA			X																	
OSTREAMRESULT			X																	
FDAREQUESTMETADATA																				
FDAREPLYMETADATA																				
BATCHPREPARE																				
BATCHEXECUTE																				
TRANSACTIONFLAGS	D	D	D	D	D	D	D	D												
LOBFLAGS						X														

2.6 Type Codes

Type codes identify the type of a field transferred from or to the database.

Not all type codes known and processed internally are used in the protocol, and clients may support a different level of type support depending on their version. The protocol mechanism (connect options) for connecting to the database ensures that the client sends the server data it is able to process, and that the client receives only data it can process from the server.

Type codes are in the range of 0-127, as the most significant bit is used on input to signal a NULL value of a certain type.

Table 13:

Value	Identifier	Description	Support Level
0	NULL	NULL value	-
1	TINYINT	TINYINT	1
2	SMALLINT	SMALLINT	1
3	INT	INTEGER	1
4	BIGINT	BIGINT	1
5	DECIMAL	DECIMAL, and DECIMAL(p,s)	1
6	REAL	REAL	1
7	DOUBLE	DOUBLE	1
8	CHAR	CHAR	1
9	VARCHAR	VARCHAR	1
10	NCHAR	NCHAR (Unicode character type)	1
11	NVARCHAR	NVARCHAR (Unicode character type)	1
12	BINARY	BINARY	1
13	VARBINARY	VARBINARY	1
14	DATE	DATE (deprecated type)	1 (deprecated with 3)
15	TIME	TIME (deprecated type)	1 (deprecated with 3)
16	TIMESTAMP	TIMESTAMP (millisecond precision)	1 (deprecated with 3)
17	TIME_TZ	Reserved, do not use	-
18	TIME_LTZ	Reserved, do not use	-
19	TIMESTAMP_TZ	Reserved, do not use	-
20	TIMESTAMP_LTZ	Reserved, do not use	-
21	INTERVAL_YM	Reserved, do not use	-
22	INTERVAL_DS	Reserved, do not use	-
23	ROWID	Reserved, do not use	-
24	UROWID	Reserved, do not use	-

25	CLOB	Character Large Object	1
26	NCLOB	Unicode Character Large Object	1
27	BLOB	Binary Large Object	1
28	BOOLEAN	BOOLEAN	7
29	STRING	Character string	1
30	NSTRING	Unicode character string	1
31	BLOCATOR	Binary locator	1
32	NLOCATOR	Unicode character locator	1
33	BSTRING	Binary string	1
34	DECIMAL_DIGIT_ARRAY	Reserved, do not use	-
35	VARCHAR2	VARCHAR	-
36	VARCHAR3	VARCHAR	-
37	NVARCHAR3	NVARCHAR	-
38	VARBINARY3	VARBINARY	-
39	VARGROUP	Reserved, do not use	-
40	TINYINT_NOTNULL	Reserved, do not use	-
41	SMALLINT_NOTNULL	Reserved, do not use	-
42	INT_NOTNULL	Reserved, do not use	-
43	BIGINT_NOTNULL	Reserved, do not use	-
44	ARGUMENT	Reserved, do not use	-
45	TABLE	Reserved, do not use	-
46	CURSOR	Reserved, do not use	-
47	SMALLDECIMAL	SMALLDECIMAL data type	-
48	ABAPITAB	ABAPSTREAM procedure parameter	1
49	ABAPSTRUCT	ABAP structure procedure parameter	1
50	ARRAY	ARRAY	7
51	TEXT	TEXT data type	3
52	SHORTTEXT	SHORTTEXT data type	3
53	FIXEDSTRING	Reserved, do not use	-
54	FIXEDPOINTDECIMAL	Reserved, do not use	-
55	ALPHANUM	ALPHANUM data type	3
56	TLOCATOR	Reserved, do not use	-
61	LONGDATE	TIMESTAMP data type	3
62	SECONDDATE	TIMESTAMP type with second precision	3
63	DAYDATE	DATE data type	3
64	SECONDTIME	TIME data type	3

65	CSDATE	Reserved, do not use	-
66	CSTIME	Reserved, do not use	-
71	BLOB_DISK	Reserved, do not use	-
72	CLOB_DISK	Reserved, do not use	-
73	NCLOB_DISK	Reserved, do not use	-
74	GEOMETRY	Reserved, do not use	-
75	POINT	Reserved, do not use	-
76	FIXED16	Reserved, do not use	-
77	BLOB_HYBRID	Reserved, do not use	-
78	CLOB_HYBRID	Reserved, do not use	-
79	NCLOB_HYBRID	Reserved, do not use	-
80	POINTZ	Reserved, do not use	-

2.7 Part Data Format

This section describes the format of parts transmitted in messages between the client and the database server.

The format of the data transmitted is uniquely identified by the part kind, some part kinds share the same format as the contained data.

2.7.1 Option Part Format

The option part format is a common format to transmit options, that is, typed key-value pairs.

An option part contains elements having the following structure:



The fields are defined as follows:

Table 14:

Field	Data Type	Description
OPTIONNAME	I1	Option key to identify the option
OPTIONTYPE	I1	Type code of option value
OPTIONVALUE	B[...]	Option data

Only some type codes are understood for the option type field, and the data format of the option data is as follows:

Table 15:

Type Code	Value Format
BOOLEAN	1-byte, zero for *false*, non-zero for *true* value
INT	4-byte signed integer (int), little-endian format
BIGINT	8-byte signed integer (long), little-endian format
STRING	2-byte signed integer (short) length information, little-endian format, followed by the string in CESU-8 encoding (number of bytes is specified by length information)
BSTRING	2-byte signed integer (short) length information, little-endian format, followed by the binary string (number of bytes is specified by length information)
DOUBLE	IEEE double precision value, little-endian

2.7.2 Multi-Line Option Part Format

The multi-line option part format is a common format to transmit collections of options (typed key-value pairs).

An option part contains rows having the following structure. The number of rows is the part argument count value.



The fields are defined as follows:

Table 16:

Field	Data Type	Description
ARGCOUNT	I2	Number of options in a row
OPTION		Option element, as described in option part format

Different rows can have a different argument count, so the format is not completely tabular.

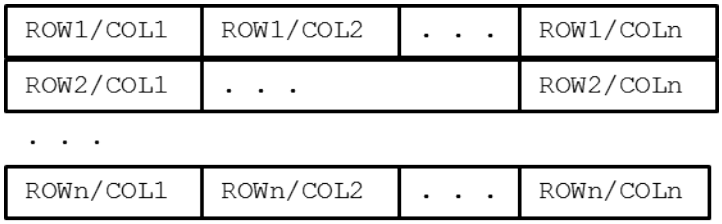
2.7.3 COMMAND

The COMMAND part contains the text of an SQL command. The text is always encoded in CESU-8 encoding.

2.7.4 RESULTSET

A result set part contains row data of a result set.

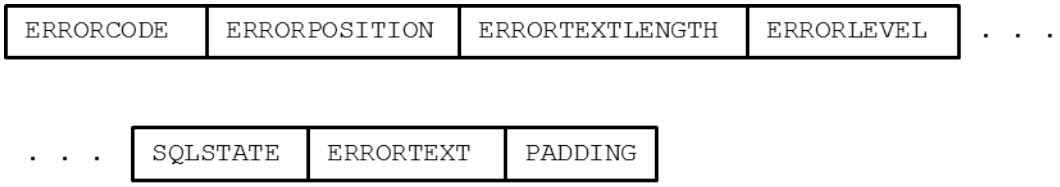
The fields of the individual rows are formatted as defined in the output field format section. One result set part contains as many rows as the argument count field indicates:



Note that the field values may have variable lengths, depending on the data type. Therefore, all the values in the current row must be scanned before moving to the next row.

2.7.5 ERROR

An ERROR part contains one or more errors returned by the database server to the client. The error part contains elements having the following structure:



The fields are defined as follows:

Table 17:

Field	Data Type	Description
ERRORCODE	I4	Error code as sent from the database server
ERRORPOSITION	I4	Position of the error
ERRORTEXTLENGTH	I4	Length of error text
ERRORLEVEL	I1	Error level (see the table below)
SQLSTATE	B[5]	SQLSTATE as defined by the SQL standard for this message
ERRORTEXT	B[ERRORTEXTLENGTH]	Error text/message in CESU-8 encoding
PADDING	B[0..7]	Padding so the length of structure is aligned to 8

The error level defines the level of the message – warning, error, or fatal session-terminating error -- using the following values:

Table 18:

Value	Identifier	Description

0	WARNING	Warning message
1	ERROR	Error message, this statement (or part of it) has failed
2	FATALERROR	Fatal error message, session is in an unusable state now

2.7.6 STATEMENTID

A STATEMENTID part contains a statement identifier, which is a byte array of 8-bytes in length. The statement identifier is opaque to the client program.

2.7.7 TRANSACTIONID

A TRANSACTIONID part contains the identifier of a transaction, which is a byte array of variable length, and stored in the same way as a VARBINARY output value. The transaction identifier is opaque to the client program.

2.7.8 ROWSAFFECTED

A ROWSAFFECTED part contains one or multiple (as many as the part's ARGUMENTCOUNT field defines) 32-bit integers, which indicate the number of rows affected by a statement, or a row of an array command.

The following special values are defined to indicate certain situations:

Table 19:

Value	Identifier	Description
-2	SUCCESS_NO_INFO	Statement/row has been processed but number of affected rows cannot be determined.
-3	EXECUTION_FAILED	Execution of statement or processing of row has failed.

2.7.9 RESULTSETID

A RESULTSETID part contains the identifier of a result set, which is a byte array of 8-bytes in length. The result set identifier is opaque to the client program.

2.7.10 TOPOLOGYINFORMATION

A TOPOLOGYINFORMATION part contains information about the server topology.

This is a multi-line option part, with the following options defined:

Table 20:

Value	Identifier	Data Type	Description
1	HOSTNAME	STRING	Host name B
2	HOSTPORT-NUMBER	INT	SQL port number
3	TENANT-NAME	STRING	Reserved
4	LOADFAC-TOR	DOUBLE	Load factor in round-robin scheduling
5	VOLUMEID	INT	Data volume ID
6	ISMASTER	BOOLEAN	Master server flag
7	ISCURRENT-SESSION	BOOLEAN	Current session flag
8	SERVICE-TYPE	INT	Service type
9	NETWORK-DOMAIN	STRING	Network domain of server A
10	ISSTANDBY	BOOLEAN	Standby server flag
11	ALLIPAD-DRESSES	STRING	All TCP/IP addresses assigned to the node A
12	ALLHOST-NAMES	STRING	All host names assigned to the node A

A Deprecated, not sent by server starting with SAP HANA SPS 06

B Externally-visible IP address/name in SAP HANA SPS 06, internal host name in previous SAP HANA revisions

The options are used as follows:

HOSTNAME

This field contains the host name of the server, without any domain part, for SAP HANA revisions before SPS 06.

Beginning with SAP HANA SPS 06, this is the externally visible TCP/IP address of the server.

HOSTPORTNUMBER

This field contains the port number of the SQL port of the server.

TENANTNAME

This field is reserved and not currently used.

LOADFACTOR

This field contains a factor used when distributing workload among servers. A load factor of 1.0 is the default value.

VOLUMEID

This field contains the volume ID of the data volume managed by this server, or 0 if there is no data volume attached yet (as in case with a standby server).

ISMASTER

This field contains a 1 (true), if the server is the master server of the SAP HANA database system.

ISCURRENTSESSION

This field contains a 1 (true), if the server that manages the session has returned this topology information.

NETWORKDOMAIN

This field contains the network domain of the server, if a non-empty domain is defined. Beginning with SAP HANA SPS 06, this field is not sent to the client.

ISSTANDBY

This field contains a 1 (true), if the server is a standby server.

ALLIPADDRESSES

This field contains a comma-separated list of the TCP/IP addresses assigned to this node. Beginning with SAP HANA SPS 06, this field is not sent to the client.

ALLHOSTNAMES

This field contains a comma-separated list of all host names assigned to this node. Beginning with SAP HANA SPS 06, this field is not sent to the client.

2.7.11 TABLELOCATION

A table location part contains one or multiple 4-byte integers in little-endian format, which identify the data volume IDs.

The number of elements is the argument count of the part. The returned volume IDs signal preferred execution locations for the statement, and depend on the:

- Nature of the SQL command that is prepared (some commands can only be executed on a certain server)
- Number of partitions of the tables affected by the statement
- Location of the tables affected by the statement

2.7.12 READLOBREQUEST

A read LOB request part is sent by the client to request BLOB/CLOB/NCLOB data to be read.

It contains a structure which has the following layout:

LOCATORID	READOFFSET	READLENGTH	FILLER

The fields are defined as follows:

Table 21:

Field	Data Type	Description
LOCATORID	B[8]	Identifier of BLOB/CLOB/NCLOB locator

READOFFSET	I8	Offset within large object for reading, starting with 1
READLENGTH	I4	Length to read
FILLER	B[4]	Reserved, do not use

For byte-based large object types, the READOFFSET and READLENGTH define byte positions and length values, for character-based large object types, these fields define character positions and length values within the large object.

2.7.13 READLOBREPLY

The READLOBREPLY part is sent from the database as answer to a read LOB request of the client.

The READLOBREPLY part has the following structure:



The fields are defined as follows:

Table 22:

Field	Data Type	Description
LOCATORID	B[8]	Identifier of BLOB/CLOB/NCLOB locator
OPTIONS	I1	Result options (see the table below)
CHUNKLENGTH	I4	Length of data in part
FILLER	B[3]	Reserved, do not use
DATA	B[CHUNKLENGTH]	Large object data

The OPTIONS field is a bit set which has the following defined values:

Table 23:

Bit	Identifier	Description
0	NULLINDICATOR	The large object value is NULL (not used for READLOBREPLY).
1	DATAINCLUDED	Data is included (not used for READLOBREPLY).
2	LASTDATA	There is no additional data remaining in the large object.

2.7.14 ABAPISTREAM

An ABAPISTREAM part is sent from the database requesting an ABAP input stream for C++ procedure processing, and as a response from the client when sending the stream data.

The part content is structured as follows:



The fields are defined as follows:

Table 24:

Field	Data Type	Description
ABAPTABID	I4	Identifier of ABAP table
MASK	B[...]	Bit field to mask entries of the ABAP table do not transfer them

The MASK field may be omitted completely. In this case, all columns of the ABAP table are transferred. Otherwise, only the marked values are transferred, enabling a projection of the ABAP table in the server processing. The number of rows requested is supplied in the argument count of the ABAPISTREAM part. In the case that the client sends this part to identify stream data transmitted to the database server, only the ABAPTABID field is used.

2.7.15 ABAPOSTREAM

The ABAPOSTREAM part contains one 4-byte integer value identifying the ABAPTABID of the table, whose data is sent in the corresponding STREAMDATA part.

2.7.16 COMMANDINFO

A COMMANDINFO part is an options part.

The COMMANDINFO part supports the following option values:

Table 25:

Value	Identifier	Data Type	Description
1	LINENUMBER	INT	Line number in source
2	SOURCEMODULE	STRING	Name of source module

This part is used to identify the location of a statement in the source program for debugging and analysis purposes.

2.7.17 WRITELOBREQUEST

The WRITELOBREQUEST part is sent from the client to write data piece-wise into a large object.

It contains elements having the following structure:

LOCATORID	OPTIONS	WRITEOFFSET	CHUNKLENGTH	DATA

The fields are defined as follows:

Table 26:

Field	Data Type	Description
LOCATORID	B[8]	Identifier of BLOB/CLOB/NCLOB locator
OPTIONS	I1	Request options
WRITEOFFSET	I8	Offset within the large object data, an offset of -1 request appending the data to the already existing large object data
CHUNKLENGTH	I4	Length of data that follows
DATA	B[CHUNKLENGTH]	Large object data

A number of large objects may be written using this request. The argument count of the part describes how many elements are contained in the part.

The OPTIONS field is a bit set which has the following defined values:

Table 27:

Bit	Identifier	Description
0	NULLINDICATOR	The large object value is NULL (not used for WRITELOBREQUEST).
1	DATAINCLUDED	Data is included.
2	LASTDATA	There is no more data remaining.

2.7.18 WRITELOBREPLY

The WRITELOBREPLY part is sent as reply from the server to a WRITELOBREQUEST part.

It contains the LOCATORID identifiers for all large object locators which can still receive data (if LASTDATA has not been set). The argument count defines the number of locator identifiers contained in the data.

2.7.19 PARAMETERS

A PARAMETERS part contains input parameters.

The parameters are densely packed, and use the input field format. The argument count of the part defines how many rows of parameters are included.

Related Information

Input and Output Field Formats [page 47]
Input and Output Field Formats [page 47]

2.7.20 AUTHENTICATION

This part contains information exchanged to perform authentication of clients.

It has the following general structure:

FIELDCOUNT	FIELD1	FIELD2	. . .

This structure is defined as follows:

Table 28:

Field	Data Type	Description
FIELDCOUNT	BI2	Number of fields that follow
FIELD		Field data (see the tables below)

Field data has the following structure, depending on its length:

Less than or Equal to 250-Bytes

FIELDLENGTH	FIELDDATA

Table 29:

Field	Data Type	Description
FIELDLENGTH	I1	Length of data
FIELDDATA	B[FIELDLENGTH]	Field data

Larger than 250-Bytes

FIELDINDICATOR	FIELDLENGTH	FIELDDATA

Table 30:

Field	Data Type	Description
FIELDINDICATOR	I1	Always 0xFF
FIELDLENGTH	BI2	Field length
FIELDDATA	B[FIELDLENGTH]	Field data

Initial Request

The initial request contains the database user name, and field pairs (method name/value) for each authentication method requested by the client.

Table 31:

Method Name	Description
PLAINPASSWORD	Obsolete, unused
SCRAMMD5	Obsolete, unused
SCRAMSHA256	Password-based authentication
GSS	GSS/Kerberos authentication
SAML	SAML authentication

Additional Requests

Requests, following the initial request, contain a number of field pairs for authentication methods, which are performed by exchanging messages.

2.7.21 SESSIONCONTEXT

The session context part is an option part.

The session context part has the following options defined:

Table 32:

Value	Identifier	Data Type	Description
1	PRIMARYCONNECTIONID	INT	ID of primary connection
2	PRIMARYHOSTNAME	STRING	Host name of primary connection host
3	PRIMARYHOSTPORTNUMBER	INT	Port number of SQL port for primary connection
4	MASTERCONNECTIONID	INT	Connection ID of transaction master
5	MASTERHOSTNAME	STRING	Host name of transaction master connection host
6	MASTERHOSTPORTNUMBER	INT	Port number of SQL port for transaction master connection

The **primary** connection is the first connection opened by the client program to the system in a distributed scenario. The **master** connection is the connection that started the current distributed transaction.

2.7.22 STATEMENTCONTEXT

The statement context part is an option part containing the following:

Table 33:

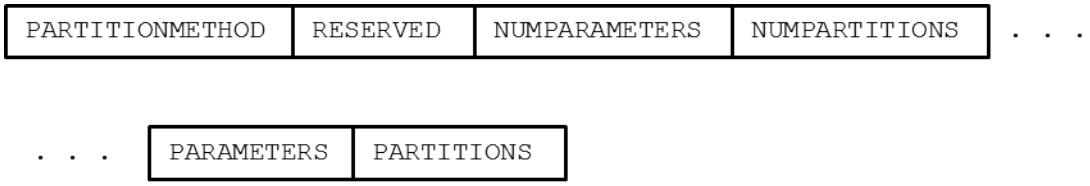
Value	Identifier	Data Type	Description
1	STATEMENTSEQUENCEINFO	BINARY	Information on the statement sequence within the transaction
2	SERVERPROCESSINGTIME	BIGINT	Server processing time in microseconds
3	SCHEMANAME	STRING	Current schema after schema change

This part is populated from previously received statement context information. The binary option content is opaque to the client application.

2.7.23 PARTITIONINFORMATION

The PARTITIONINFORMATION part is returned by the server to allow a more specific pruning of input data, as it pertains to the partitioning of data in the database server.

The PARTITIONINFORMATION part is structured as follows:



The fields are defined as follows:

Table 34:

Field	Data Type	Description
PARTITIONMETHOD	I1	Method of partitioning used
RESERVED	B[7]	Reserved, do not use
NUMPARAMETERS	I4	Number of subsequent parameter descriptors
NUMPARTITIONS	I4	Number of subsequent partition descriptors

The PARTITIONMETHOD field supports the following values:

Table 35:

Value	Identifier	Description
0	INVALID	Invalid method, do not use
1	ROUNDROBIN	Partitions are assigned using a round-robin specification.
2	HASH	Partitions are assigned using a hash specification.

The PARAMETERS field contains NUMPARAMETERS elements having the following structure:

PARAMETERINDEX	PARAMETERFUNCTION	ATTRIBUTETYPE	RESERVED

The fields of this structure are defined as follows:

Table 36:

Field	Data Type	Description
PARAMETERINDEX	I4	Index of parameter in statement
PARAMETERFUNCTION	I1	Function to apply to parameter before hash
ATTRIBUTETYPE	I1	Storage type to be used for hash calculation
RESERVED	B[2]	Reserved, do not use

The PARAMETERFUNCTION field supports the following values:

Table 37:

Value	Identifier	Description
0	INVALID	Invalid method, do not use
1	YEAR	Extracted year part of DATE
2	MONTH	Extracted month part of DATE

The ATTRIBUTETYPE field supports the following values:

Table 38:

Value	Identifier	Description	SQL Data Type[2]
0	INVALID	Invalid type	
65	ALPHANUM	A character string consisting of numbers only	ALPHANUM
83	STRING	A character string in CESU-8 format	NVARCHAR
67	FIXEDSTRING	A character string in CESU-8 format	NCHAR
86	TEXT		SHORTTEXT
73	INT	32-bit integer value	INTEGER
70	FLOAT	IEEE float type	
68	DATE		
64	LONGDATE	LONGDATE value	LONGDATE
116	TIME		
84	TEXT_OLD		
66	FIXED	DECIMAL value	DECIMAL
100	DOUBLE	IEEE double value	DOUBLE

[2] This reflects the default mapping of SQL data types to column store types.

Value	Identifier	Description	SQL Data Type[2]
69	UNITDECFLOAT		
82	RAW	Binary string	VARBINARY
77	DECIMAL_FLOAT		
76	SDFLOAT		
115	SECONDDATE	SECONDDATE value	SECONDDATE
101	DAYDATE	DAYDATE value	DAYDATE
117	SECONDTIME	SECONDTIME value	SECONDTIME

2.7.24 OUTPUTPARAMETERS

The format of the OUTPUTPARAMETERS part is the same as the format of the RESULTSET part.

The part contains the scalar output parameter values returned by the statement.[3] The argument count is always 1.

2.7.25 CONNECTOPTIONS

The connect option part includes the following:

Table 39:

Value	Identifier	Data Type	Description
1	CONNECTIONID	INT	ID of connection
2	COMPLETEARRAY EXECUTION	BOOLEAN	Complete array execution feature
3	CLIENTLOCALE	STRING	Locale set by the application
4	SUPPORTSLARGE BULKOPERATIONS	BOOLEAN	Supports a large bulk of operations
5	DATAFORMATVERSION2	INTEGER	Sends spatial types ST_POINT, ST_POINTZ, and ST_GEOMETRY to client. No change in behavior for applications, unless connect option SPATIALTYPES is set. If it is set, the types are used in data type description in ParameterMetadata and ResultMetadata.

[2] This reflects the default mapping of SQL data types to column store types.

[3] An empty output parameter part can be sent if there are no scalar output parameters in the procedure definition, but non-scalar output parameters (tables) instead.

Value	Identifier	Data Type	Description
6			Reserved, do not use
7			Reserved, do not use
8			Reserved, do not use
9			Reserved, do not use
10	LARGENUMBEROF PARAMETERSSUPPORT	BOOLEAN	Supports a large number of parameters
11	SYSTEMID	STRING	System ID of the system
12			Reserved, do not use
13	ABAPVARCHARMODE	BOOLEAN	ABAP VARCHAR mode flag
14	SELECTFORUPDATE SUPPORTED	BOOLEAN	SELECT FOR UPDATE function code support flag
15	CLIENTDISTRIBUTIONMODE	INT	Mode of distribution in the client
16	ENGINEDATA FORMATVERSION	INT	Level of data type format support
17	DISTRIBUTIONPROTOCOL VERSION	INT	Level of distribution protocol support
18	SPLITBATCHCOMMANDS	BOOLEAN	Splitting of batch commands allowed
19	USETRANSACTION FLAGSONLY	BOOLEAN	Usage of transaction flags
20	ROWANDCOLUMN OPTIMIZEDFORMAT	BOOLEAN	Reserved, do not use
21	IGNOREUNKNOWNPARTS	BOOLEAN	Server ignores unknown parts
22	TABLEOUTPUTPARAMETER	BOOLEAN	Table type as output parameter type supported
23	DATAFORMATVERSION2	INTEGER	Version of data format
24	ITABPARAMETER	BOOLEAN	Support of ABAP ITAB parameter in statements
25	DESCRIBETABLE OUTPUTPARAMETER	BOOLEAN	Request table output parameter metadata in the session
26	COLUMNARRESULTSET	BITVECTOR	List of types for which columnar result set format is supported
27	SCROLLABLERESULTSET	INTEGER	Scrollable result sets supported.

Value	Identifier	Data Type	Description
28	CLIENTINFONULLVALUESSUP-PORTED	BOOLEAN	Server can handle null values in client info
29	ASSOCIATEDCONNECTIONID	INTEGER	The connection ID of the associated connection. Used only in dynamic tier-ing scenarios.
30	NONTRANSACTIONALPREPARE	BOOLEAN	Server can handle and uses non-trans-actional prepare
31	FDAENABLED	BOOLEAN	Fast Data Access (FDA) is enabled
32	OSUSER	STRING	The user name of the current user on the client operating system
33	ROWSLOTIMAGERESULT	BITVECTOR	List of types for which row-slot image result set format is supported
34	ENDIANNESS	INTEGER	Endianness (big or little endian) of the server
35			Reserved, do not use
36			Reserved, do not use
37	IMPLICITLOBSTREAMING	BOOLEAN	Support implicit LOB streaming with auto-commit on

The options are used as follows:

CONNECTIONID

This field contains the connection ID. It is filled by the server when the connection is established. This number can be used in DISCONNECT/KILL commands for command or session cancellation.

COMPLETEARRAYEXECUTION

This field is set if array commands continue to process remaining input when detecting an error in an input row. Always set for current client and server.

CLIENTLOCALE

The session locale can be set by the client. The locale is used in language-dependent handling within the SAP HANA database calculation engine.

SUPPORTSLARGEBULKOPERATIONS

This field is set by the server to process array commands.

LARGENUMBEROFPARAMETERSSUPPORT

This field contains the host name of the server, without any domain part. It is filled by the server with the host name it resolves, so that it does not contain an alias name of the database server.

SYSTEMID

This option is set by the server and filled with the SAPSYSTEMNAME of the connected instance for tracing and supportability purposes.

DATAFORMATVERSION2

The client indicates this set of understood type codes and field formats. The server then defines the value according to its own capabilities, and sends it back. The following values are supported:

Table 40:

Value	Description
1	Baseline data type support (SAP HANA SPS 02)
3	Extended data type support: ALPHANUM, TEXT, SHORTTEXT, LONGDATE, SECONDDATE, DAYDATE, SECONDTIME supported without translation. Deprecated, do not use
4	Support for ALPHANUM, TEXT, SHORTTEXT, LONGDATE, SECONDDATE, DAYDATE, and SECONDTIME. Baseline data type support format for SAP HANA SPS 06.
6	Send data type BINTEXT to client.

ABAPVARCHARMODE

This field is set by the client to indicate that the connection should honor the ABAP character handling, that is:

- Trailing space of character parameters and column values is not significant.
- Trailing space in character literals is not relevant. For example, the character literal '' is identical to the character literal' '.

SELECTFORUPDATESUPPORTED

This field is set by the client to indicate that the client is able to handle the special function code for SELECT ... FOR UPDATE commands.

CLIENTDISTRIBUTIONMODE

This field is set by the client to indicate the mode for handling statement routing and client distribution. The server sets this field to the appropriate support level depending on the client value and its own configuration. The following values are supported:

Table 41:

Value	Description
0	OFF, no routing or distributed transaction handling is done.
1	CONNECTION, client can connect to any (master/slave) server in the topology, and connections are enabled, such that the connection load on the nodes is balanced.
2	STATEMENT, server returns information about which node is preferred for executing the statement, clients execute on that node, if possible.
3	STATEMENT_CONNECTION, both STATEMENT and CONNECTION level

ENGINEDATAFORMATVERSION

The server sets this field to the maximum version it is able to support. The possible values correspond to the DATAFORMATVERSION flag.

DISTRIBUTIONPROTOCOLVERSION

This field is set by the client and indicates the support level in the protocol for distribution features. The server may choose to disable distribution if the support level is not sufficient for the handling.

Table 42:

Value	Description
0	Baseline version
1	Client handles statement sequence number information (statement context part handling).

CLIENTDISTRIBUTIONMODE is OFF if a value less than one (<1) is returned by the server.

SPLITBATCHCOMMANDS

This field is sent by the client and returned by the server if configuration allows splitting batch (array) commands for parallel execution.

USETRANSACTIONFLAGSONLY

This field is sent by the server to indicate the client should gather the state of the current transaction only from the TRANSACTIONFLAGS command, not from the nature of the command (DDL, UPDATE, and so on).

IGNOREUNKNOWNPARTS

This field is sent by the server to indicate it ignores unknown parts of the communication protocol instead of raising a fatal error.

TABLEOUTPUTPARAMETER

This field is sent by the client to indicate that it understands output parameters described by type code TABLE in result sets.

ITABPARAMETER

This field is sent by the server to signal it understands ABAP ITAB parameters of SQL statements (For-All-Entries Optimization).

DESCRIBETABLEOUTPUTPARAMETER

This field is sent by the client to request that table output parameter metadata is included in the parameter metadata of a CALL statement.[4] The returned type of the table output parameter is either STRING or TABLE, depending on the TABLEOUTPUTPARAMETER connect option.

The following table further illustrates the use of the connect options. An option can depend on:

- Client parameter (set in client to change server behavior)

[4] This is the same behavior that can be achieved by setting the parameter `omit_table_output_parameter_metadata` to false, except that by using the connect option, the setting is only valid for the current session, not globally for the database instance.

- Server parameter (set in server configuration to enable/disable)
- Server and client version (if a feature was introduced which needs to be in sync between a client and server)

Of course, more than one of these statements can be true for any option.

Table 43:

Option	Client Parameter	Server Parameter	Version Dependent
CONNECTIONID	-	-	-
COMPLETEARRAYEXECUTION	-	-	X
CLIENTLOCALE	X	-	-
SUPPORTSLARGEBULKOPERATIONS	-	-	X
LARGENUMBEROFPARAMETERSSUPPORT	-	X	X
SYSTEMID	-	-	-
DATAFORMATVERSION	X	-	X
ABAPVARCHARMODE	X	-	X
SELECTFORUPDATESUPPORTED	-	-	X
CLIENTDISTRIBUTIONMODE	X	X	X
ENGINEDATAFORMATVERSION	-	-	X
DISTRIBUTIONPROTOCOLVERSION	-	-	X
SPLITBATCHCOMMANDS	X	X	X
USETRANSACTIONFLAGSONLY	-	-	X
IGNOREUNKNOWNPARTS	-	-	X
TABLEOUTPUTPARAMETER	X	-	X
ITABPARAMETER	-	X	X
DESCRIBETABLEOUTPUTPARAMETER	X	-	X

IMPLICITLOBSTREAMING

This field is sent by the client and indicates whether the server supports implicit LOB streaming even though auto-commit is on instead of raising an error.

2.7.26 COMMITOPTIONS

The commit option part is an option part.

The commit option part uses the following:

Table 44:

Value	Identifier	Data Type	Description
1	HOLDCURSORSOVERCOMMIT	BOOLEAN	Hold cursors

If HOLDCURSORSOVERCOMMIT is set by the client on commit, not only cursors marked explicitly as HOLD, but all cursors, are held.[5]

2.7.27 FETCHOPTIONS

The FETCHOPTIONS part is an option part.

The FETCHOPTIONS part uses the following:

Table 45:

Value	Identifier	Data Type	Description
1	RESULTSETPOS	INT	Position for FETCH

The RESULTSETPOS field can be used to skip over entries when fetching.[6]

2.7.28 FETCHSIZE

The FETCHSIZE part contains one 4-byte integer value (little-endian) defining the number of rows the application wants to fetch.

2.7.29 PARAMETERMETADATA

This part contains metadata (type information) of parameters, that is, input parameters of prepared statements, and additional output parameters of stored procedure call statements.

A PARAMETERMETADATA part starts with an array of entries having the following structure:

PARAMETEROPTIONS	DATETYPE	MODE	FILLER1	NAMEOFFSET	LENGTH	FRACTION	FILLER2

[5] Currently not used by any client interface implementation.
[6] Currently not used by any client interface implementation.

The fields of this structure are defined as follows:

Table 46:

Field	Data Type	Description
PARAMETEROPTIONS	I1	Options that further refine parameter details
DATATYPE	I1	Data type of parameter (type code)
MODE	I1	Whether the parameter is input or output
FILLER1	I1	Reserved, do not use
NAMEOFFSET	UI4	Offset of parameter name in part, set to 0xFFFFFFFF to signal no name
LENGTH	I2	Length/Precision of the parameter
FRACTION	I2	Scale of the parameter
FILLER2	I4	Reserved, do not use

The PARAMETEROPTIONS field is a bit set, which has the following defined values:

Table 47:

Bit	Identifier	Description
0	MANDATORY	Parameter is not nullable (must not be set to NULL).
1	OPTIONAL	Parameter is nullable (can be set to NULL).
2	DEFAULT	Parameter has a defined DEFAULT value.
3		Reserved, do not use
4		Reserved, do not use
5		Reserved, do not use

The MODE field is a bit set that defines the direction of a parameter:

Table 48:

Bit	Identifier	Description
0	IN	Parameter direction is IN.
1	INOUT	Parameter direction is INOUT.
2	OUT	Parameter direction is OUT.

The array of parameter descriptions may be followed by the names of the parameters. Here, each name is written in the following format:

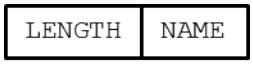


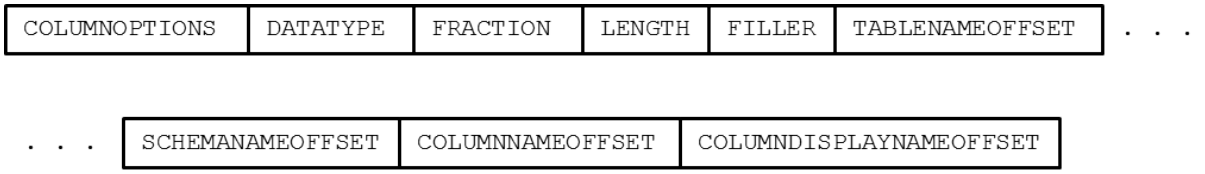
Table 49:

Field	Data Type	Description
LENGTH	UI1	Length of name in bytes
NAME	B[LENGTH]	Name

2.7.30 RESULTSETMETADATA

This part contains metadata (type information) of result columns.

A RESULTSETMETADATA part starts with an array of entries having the following structure:



The fields of this structure are defined as follows:

Table 50:

Field	Data Type	Description
COLUMNOPTIONS	I1	Options that further refine column details
DATATYPE	I1	Data type of parameter (type code)
FRACTION	I2	Scale of the column
LENGTH	I2	Length/Precision of the column
FILLER	I2	Reserved, do not use
TABLENAMEOFFSET	UI4	Offset of table name in part. Set to 0xFFFFFFFF to signal no available name.
SCHEMANAMEOFFSET	UI4	Offset of schema name in part. Set to 0xFFFFFFFF to signal no available name.
COLUMNNAMEOFFSET	UI4	Offset of column name in part. Set to 0xFFFFFFFF to signal no available name.
COLUMNDISPLAYNAMEOFFSET	UI4	Offset of column display name (label) in part. Set to 0xFFFFFFFF to signal no available name.

The COLUMNOPTIONS field is a bit set which has the following defined values:

Table 51:

Bit	Identifier	Description
0	MANDATORY	Column is defined as NOT NULL.
1	OPTIONAL	Column can be NULL value.
2		Reserved, do not use
3		Reserved, do not use
4		Reserved, do not use
5		Reserved, do not use

The array of column descriptions may be followed by the individual schema names, table names, column names, and column display names. Here, each name is written in the following format:

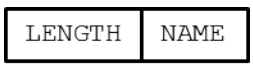


Table 52:

Field	Data Type	Description
LENGTH	UI1	Length of name in bytes
NAME	B[LENGTH]	Name

2.7.31 FINDLOBREQUEST

The FINDLOBREQUEST part is sent from the client to search for a substring of a BLOB, CLOB, NCLOB, or TEXT value.

The FINDLOBREQUEST part has the following structure:

LOCATORID	STARTOFFSET	PATTERNLENGTH	PATTERN

The fields are defined as follows:

Table 53:

Field	Data Type	Description
LOCATORID	B[8]	Identifier of BLOB/CLOB/NCLOB/TEXT locator
STARTOFFSET	I8	Start offset for search. The offset is a character or byte position dependent on the data type of locator
PATTERNLENGTH	I4	Length of pattern in bytes
PATTERN	B[PATTERNLENGTH]	Pattern data, which is binary data or character data, depending on data type of locator

Currently, the length of the pattern must not exceed 256-bytes.

2.7.32 FINDLOBREPLY

The FINDLOBREPLY part is sent from the database server to the client in response to a FINDLOBREQUEST part.

It contains one 8-byte integer value (little-endian) defining the position within the locator of the search pattern. The value is -1 to indicate that the pattern has not been found.

2.7.33 ITABSHM

This part describes how the memory used for ITAB transfer is sent to the SAP HANA database.

An ITABSHM part has the following structure:

TRANSPORTTYPE	SHMID	OFFSET	LENGTH

The fields of this structure are defined as follows:

Table 54:

Field	Data Type	Description
TRANSPORTTYPE	NI4	Transport mechanism used
SHMID	NI4	Reserved, do not use
OFFSET	NUI8	Reserved, do not use
LENGTH	NUI8	Reserved, do not use

The following values are defined for the TRANSPORTTYPE field:

Table 55:

Value	Identifier	Description
0		Reserved, do not use
1	SOCKET	ITAB is sent over socket connection.

2.7.34 CLIENTINFO

The client info part contains key/value pairs, sent by the client for additional information.

The fields are formatted as variable length strings (like an NSTRING value in result set part). The argument count is the number of strings in the part. Because these are key/value pairs, the argument count is always an even number.

2.7.35 STREAMDATA

The STREAMDATA part contains stream data read or written by a C++ database procedure.

Its structure depends on the field information described by the corresponding ABAPITAB parameter.

2.7.36 BATCHPREPARE

This part is deprecated and not used.

2.7.37 BATCHEXECUTE

This part is deprecated and not used.

2.7.38 TRANSACTIONFLAGS

The TRANSACTIONFLAGS part is an option part.

The TRANSACTIONFLAGS part can return one or more of the following defined options:

Table 56:

Value	Identifier	Data Type	Description
0	ROLLEDBACK	BOOL	The transaction is rolled back.
1	COMMITED	BOOL	The transaction is committed.
2	NEWISOLATIONLEVEL	INT	The transaction isolation level has changed.
3	DDLCOMMITMODE CHANGED	BOOL	The DDL auto-commit mode has been changed.
4	WRITETRANSACTION STARTED	BOOL	A write transaction has been started.
5	NOWRITETRANSACTION STARTED	BOOL	No write transaction has been started.
6	SESSIONCLOSING TRANSACTIONERROR	BOOL	An error happened that implies the session must be terminated.

The part is sent from the server to signal changes of the current transaction status (committed, rolled back, start of a write transaction) and changes of the general session state, that is, whether the transaction isolation level has been changed, or whether DDL statements are automatically committed or not. Also, the server can signal it has detected a state that makes it impossible to continue processing the session.

2.7.39 DBCONNECTINFO

The DBCONNECTINFO part is an option part.

The DBCONNECTINFO part has the following defined options:

Table 57:

Value	Identifier	Data Type	Description
1	DATABASENAME	STRING	Database name, sent from client to server
2	HOST	STRING	Replied host name of the database (master)
3	PORT	INT4	Replied SQL port number of master index server
4	ISCONNECTED	BOOL	Replied status, if the database is connected

In the request message, only the database name is sent. The replied message includes the ISCONNECTED value, and may include HOST, and PORT values. The DATABASENAME value is not sent back.

2.7.40 LOBFLAGS

The LOBFLAGS part is an option part.

The LOBFLAGS part can return one or more of the following defined options:

Table 58:

Value	Identifier	Data Type	Description
0	IMPLICITSTREAMING	BOOL	The implicit streaming has been started.

The part is sent from the client to signal whether the implicit LOB streaming is started so that the server does not commit the current transaction even with auto-commit on while LOB streaming.

2.8 Input and Output Field Formats

Input fields generally consist of type code information, followed possibly by field data, if the value is not the NULL value.



The value is left blank if the type code indicates a NULL value (the MSB of the type code field is set). The following sections only describe the VALUE format.

Output fields contain no special type code field. The type information is supplied in the respective PARAMETERDATA (for output parameters) or RESULTSETMETADATA parts. All output data is densely packed, that is, there are no gaps between individual values. Thus, some values may not be aligned in memory as required for the native type.

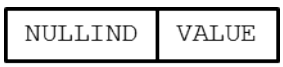
2.8.1 TINYINT

Input TINYINT

A TINYINT is sent as a 1-byte unsigned integer.

Output TINYINT

A TINYINT value is formatted as follows:



The fields are defined as follows:

Table 59:

Field	Data Type	Description
NULLIND	I1	NULL value indicator. The value is NULL if this is 0.
VALUE	UI1	TINYINT value, only present if NULLIND is not 0.

The field has a length of 2-bytes if the value is not NULL, and 1-byte if the value is NULL.

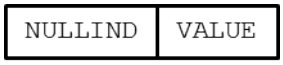
2.8.2 SMALLINT

Input SMALLINT

A SMALLINT is sent as a 2-byte signed integer in little-endian format.

Output SMALLINT

A SMALLINT value is formatted as follows:



The fields are defined as follows:

Table 60:

Field	Data Type	Description
NULLIND	I1	NULL value indicator. The value is NULL if this is 0.
VALUE	I2	SMALLINT value, only present if NULLIND is not 0.

The field has a length of 3-bytes if the value is not NULL, and 1-byte if the value is NULL.

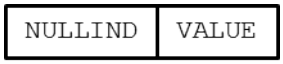
2.8.3 INT

Input INT

An INT is sent as a 4-byte signed integer in little-endian format.

Output INT

An INT value is formatted as follows:



The fields are defined as follows:

Table 61:

Field	Data Type	Description
NULLIND	I1	NULL value indicator. The value is NULL if this is 0.
VALUE	I4	INT value, only present if NULLIND is not 0.

The field has a length of 5-bytes if the value is not NULL, and 1-byte if the value is NULL.

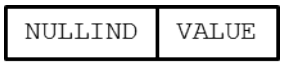
2.8.4 BIGINT

Input BIGINT

A BIGINT is sent as 8-byte signed integer in little-endian format.

Output BIGINT

A BIGINT value is formatted as follows:



The fields are defined as follows:

Table 62:

Field	Data Type	Description
NULLIND	I1	NULL value indicator. The value is NULL if this is 0.
VALUE	I8	BIGINT value, only present if NULLIND is not 0.

The field has a length of 9-bytes if the value is not NULL, and 1-byte if the value is NULL.

2.8.5 DECIMAL

Input DECIMAL

A decimal value is a 128-bit (16-byte) value formatted as follows:



Table 63:

Field	Length	Description
SIGN	1-bit	Sign: 0 is positive, 1 is negative
EXPONENT	14-bit	Exponent, biased with 6176, leading to a range -6143 to +6144
MANTISSA	113-bit	Integer mantissa

The number represented is (10^EXPONENT)*MANTISSA. It is expected that MANTISSA is not a multiple of 10.

Output DECIMAL

A DECIMAL value is formatted similarly as the input format, with the NULL value indicated by having bits 4, 5, and, 6 set to 1 in the last byte.

2.8.6 REAL

Input REAL

A REAL value is sent as float value (IEEE single precision floating point), in little-endian format.

Output REAL

A REAL value is sent as float value (IEEE single precision floating point), in little-endian format. The NULL value is indicated by all bit sets in the value sent from the server (equal to 0xFFFFFFFF as unsigned int).

2.8.7 DOUBLE

Input DOUBLE

A REAL value is sent as double value (IEEE double precision floating point), in little-endian format.

Output DOUBLE

A DOUBLE value is sent as double value (IEEE double precision floating point), in little-endian format. The NULL value is indicated by all bit sets in the value sent from the server (equal to 0xFFFFFFFFFFFFFFFFul as unsigned long).

2.8.8 STRING/NSTRING

Input STRING/NSTRING

A STRING/NSTRING input value is formatted as follows:

LENGTHINDICATOR	VALUE

LENGTHINDICATOR is a field up to 5-bytes, containing the following:

- The length in bytes of VALUE, if the VALUE length is less than or equal to 245
- 246, followed by a 2-byte integer (little-endian), which contains the actual length of VALUE
- 247, followed by a 4-byte integer (little-endian), which contains the actual length of VALUE

The VALUE field contains the input string in CESU-8 encoding.

Output STRING/NSTRING

The formatting is similar to the input value, with a LENGTHINDICATOR value of 255 indicating a NULL value.

2.8.9 BINARY

Input BINARY

A BINARY input value is formatted similarly as the STRING/NSTRING value, with respect to a LENGTHINDICATOR and a VALUE part. The VALUE contains the binary data.

Output BINARY

The formatting is similar to the input value, with a LENGTHINDICATOR value of 255 indicating a NULL value.

2.8.10 BLOB/CLOB/NCLOB

Input BLOB/CLOB/NCLOB

A BLOB/CLOB/NCLOB field is indicated using an input LOB descriptor, followed by the LOB data after the end of the record.



Table 64:

Field	Data Type	Description
OPTIONS	I1	Options that further refine the descriptor
LENGTH	I4	Length in bytes of data that follows
POSITION	I4	Position (1-based) of the data in the part
DATA	B[LENGTH]	BLOB/CLOB/NCLOB data

The DATA field does not immediately follow the descriptor. First, all parameters of a row are transferred, and then the BLOB/CLOB/NCLOB data follows.

The OPTIONS field is a bit set defined as follows:

Table 65:

Bit	Identifier	Description
0	NULLINDICATOR	The large object value is NULL (not used in input).

Bit	Identifier	Description
1	DATAINCLUDED	Data is included.
2	LASTDATA	There is no more data remaining.

Output BLOB/CLOB/NCLOB

A BLOB/CLOB/NCLOB field is formatted using an output LOB descriptor, followed by the LOB data.



Table 66:

Field	Data Type	Description
TYPE	I1	Type of data
OPTIONS	I1	Options that further refine the descriptor
LENGTH	I4	Length in bytes of data that follows
LOCATORID	I8	Locator identifier
DATALENGTH	I4	Length of data that follows
DATA	B[DATALENGTH]	BLOB/CLOB/NCLOB data

The TYPE field further refines the LOB source type code received in the result set metadata.

Table 67:

Value	Data Type
0	undefined
1	BLOB
2	CLOB
3	NCLOB

The OPTIONS field is a bit set defined as follows:

Table 68:

Bit	Identifier	Description
0	NULLINDICATOR	The large object value is NULL.
1	DATAINCLUDED	Data is included.
2	LASTDATA	There is no more data remaining.

2.8.11 DATE

A DATE field is formatted as follows:[7]



Table 69:

Field	Data Type	Description
YEAR	I2	Year
MONTH	I1	Month
DAY	I2	Day

A NULL value is indicated by the MSB (0x8000) set in the YEAR field.

2.8.12 TIME

A TIME value is formatted as follows:[8]



Table 70:

Field	Data Type	Description
HOUR	I1	Hour
MINUTE	I1	Minute
MILLISEC	UI2	Milliseconds

A NULL value is indicated by setting the MSB (0x80) in the HOUR field.

2.8.13 TIMESTAMP

Input TIMESTAMP

A TIMESTAMP is formatted as a DATE value followed by a TIME value.[9]

[7] This format is retained for purpose of compatibility, DAYDATE is used where possible and the endorsed format.
[8] This format is retained for compatibility purposes. SECONDTIME is used where possible and is the endorsed format.

Output TIMESTAMP

A TIMESTAMP is formatted as a DATE value followed by a TIME value. A NULL value is indicated by setting the NULL value in both components. For more information about the DATE and TIME formats, see Related Information.

Related Information

DATE [page 54]
TIME [page 54]
DATE [page 54]
TIME [page 54]

2.8.14 ABAPITAB

This value does not occur in the input or output data.

2.8.15 ABAPSTRUCT

Input ABAPSTRUCT

An ABAPSTRUCT is formatted similarly as a BINARY value. The layout of the structure depends on meta-information known only to the processing liveCache C++ Procedure and the ABAP client.

Output ABAPSTRUCT

The value is formatted similarly as a BINARY value.

[9] This format is retained for compatibility purposes. The LONGDATE or SECONDDATE formats are used where possible and are the endorsed formats.

2.8.16 LONGDATE

A LONGDATE is a 64-bit integer value computed as follows:

LONGDATE = (DAYDATE-1) * DAYFACTOR + TIMEVALUE + 1

The variables are computed as follows:

- DAYDATE – The date value as a DAYDATE data type
- DAYFACTOR – 1000000 * 60 * 60* 24
- TIMEVALUE – ((((Hours * 24) + Minutes) * 60 + Seconds) * 60 + Nanoseconds /100) * 1000000

The value 3155380704000000001 is the NULL value.

2.8.17 SECONDDATE

A SECONDDATE is a 64-bit integer value computed as follows:

SECONDDATE = (DAYDATE-1) * DAYFACTOR + TIMEVALUE + 1

The variables are computed as follows:

- DAYDATE – The date value as a DAYDATE data type
- DAYFACTOR – 60 * 60* 24
- TIMEVALUE – (((Hours * 24) + Minutes) * 60 + Seconds) * 60

The value 315538070401 is the NULL value.

2.8.18 DAYDATE

A DAYDATE is computed by taking the Julian Day Number of the specified date and subtracting 1721423.

The value 3652062 is the NULL value.

2.8.19 SECONDTIME

A SECONDTIME is a 32-bit integer value, which is computed as follows:

(((Hours * 24) + Minutes) * 60 + Seconds) * 60

The value 86401 is the NULL value.

2.8.20 ST_POINT/ST_POINTZ/ST_GEOMETRY

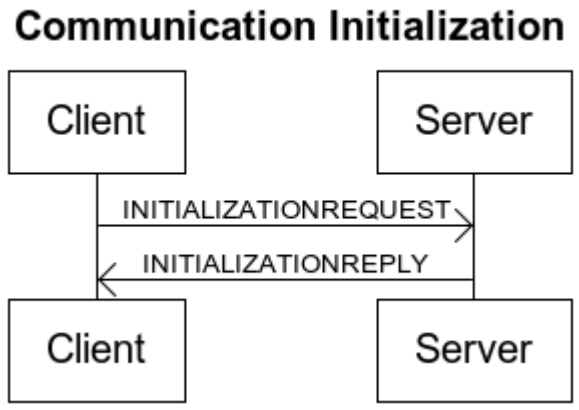
Input ST_POINT/ST_POINTZ/ST_GEOMETRY

ST_POINT/ST_POINTZ/ST_GEOMETRY are sent as strings.

2.9 Usage Scenarios

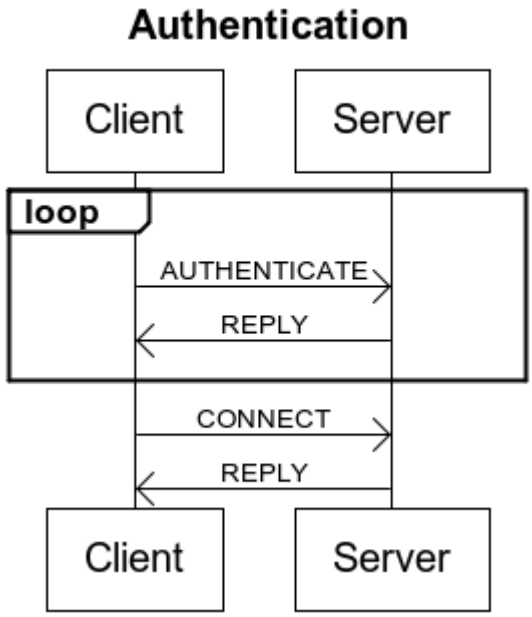
The following section depicts common scenarios and the parameters that apply.

2.9.1 Communication Initialization



2.9.2 Authentication and Connect

As laid out in the sequence diagram below, the authentication and connection process consist of possibly a number of AUTHENTICATE messages, finally concluded with the exchanging of a CONNECT message.



The AUTHENTICATE message fulfills the purpose of performing a handshake on supported authentication methods, and also is used repeatedly in case the authentication requires multiple messages to be exchanged until completion (currently only necessary for GSS authentication).

For several fields in the authentication request a LENGTHINDICATOR is used. It can be up to 5-bytes:

- The length in bytes of the following value, if the length is less 250
- 250, followed by a 2-byte integer (little-endian) which contains the actual length of the value

The following sections explain, in detail, the processing of the various authentication methods.

2.9.2.1 SCRAMSHA256 Authentication

Following the SCRAMSHA256 authentication mechanism two server roundtrips are necessary, an initial request and a final request.

Initial Authentication Request

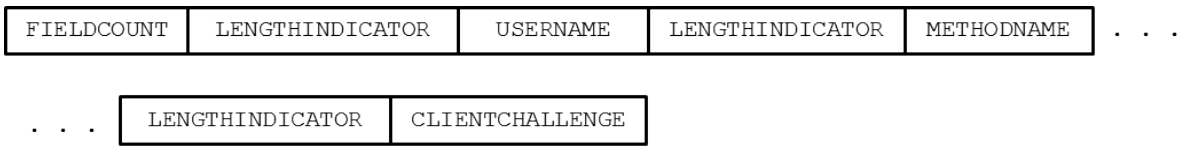


Table 71:

Field	Data Type	Description

FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field
USERNAME	B[DATALENGTH]	User name
LENGTHINDICATOR	B1-5	Length of the following field
METHODNAME	B[DATALENGTH]	Method name
LENGTHINDICATOR	B1	Length of the following field (always 64-bytes)
CLIENTCHALLENGE	B[64]	Client challenge

Initial Authentication Reply



Table 72:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field
METHODNAME	B[DATALENGTH]	Method name
LENGTHINDICATOR	B1	Length of the following field (always 68-bytes)
SERVERCHALLENGEDATA	B[68]	Server challenge

SERVERCHALLENGEDATA has the following format:

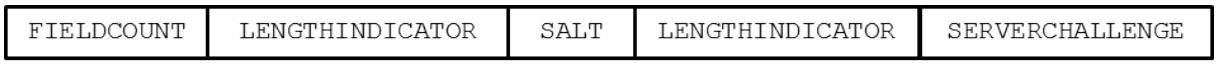


Table 73:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field
SALT	B[DATALENGTH]	Password salt
LENGTHINDICATOR	B1	Length of the following field
SERVERCHALLENGE	B[DATALENGTH]	Server challenge

Final Authentication Request

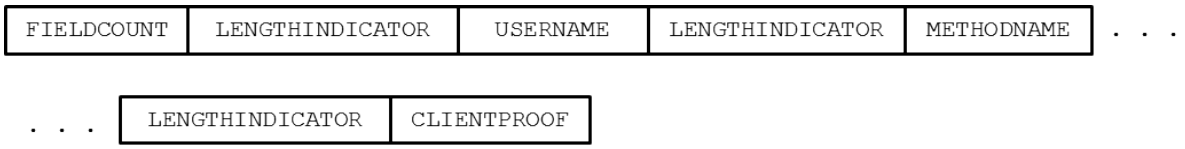


Table 74:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field
USERNAME	B[DATALENGTH]	User name
LENGTHINDICATOR	B1-5	Length of the following field
METHODNAME	B[DATALENGTH]	Method name
LENGTHINDICATOR	B1	Length of the following field (always 35-bytes)
CLIENTPROOF	B[35]	Client proof

Final Authentication Reply



Table 75:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field
METHODNAME	B[DATALENGTH]	Method name
LENGTHINDICATOR	B1	Length of the following field (until now always 0-bytes)
SERVERPROOF	B[DATALENGTH]	Server proof

2.9.2.2 GSS Authentication

Initial Authentication Request

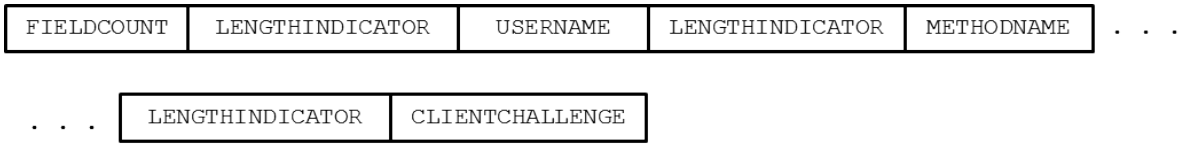


Table 76:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field
USERNAME	B[DATALENGTH]	User name
LENGTHINDICATOR	B1-5	Length of the following field
METHODNAME	B[DATALENGTH]	Method name
LENGTHINDICATOR	B1	Length of the following field
CLIENTCHALLENGE	B[DATALENGTH]	Client challenge

CLIENTCHALLENGE has the following format:

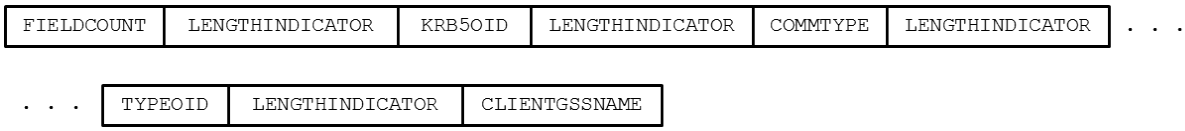


Table 77:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field
KRB5OID	B[DATALENGTH]	KRB5 object ID
LENGTHINDICATOR	B1	Length of the following field
COMMTYPE	B1	Communication type
LENGTHINDICATOR	B1	Length of the following field
TYPEOID	B[DATALENGTH]	Type object ID
LENGTHINDICATOR	B1	Length of the following field
CLIENTGSSNAME	B[DATALENGTH]	Client GSS Name

Authentication Reply



Table 78:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field
METHODNAME	B[DATALENGTH]	Method name
LENGTHINDICATOR	B1	Length of the following field
SERVERTOKEN	B[DATALENGTH]	Server-specific Kerberos tokens

Follow-Up Authentication Reply

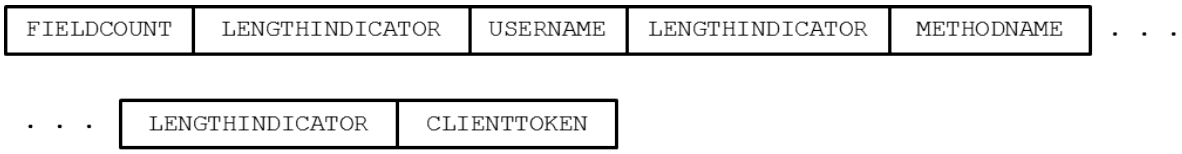


Table 79:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field
USERNAME	B[DATALENGTH]	User name
LENGTHINDICATOR	B1-5	Length of the following field
METHODNAME	B[DATALENGTH]	Method name
LENGTHINDICATOR	B1	Length of the following field
CLIENTOKEN	B[DATALENGTH]	Client specific Kerberos tokens

2.9.2.3 SAML Authentication

The SAML authentication mechanism does two server roundtrips, an initial request, and a final request.

Initial Authentication Request

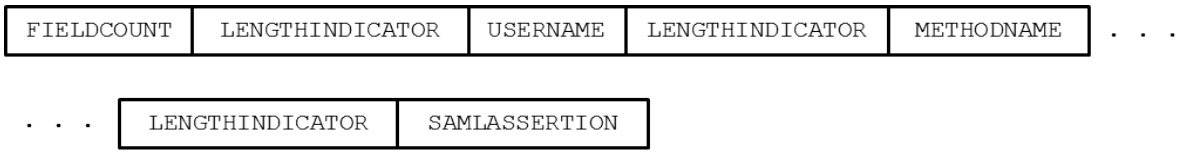


Table 80:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field (always zero)
USERNAME	B[0]	User name (always empty user name)
LENGTHINDICATOR	B1-5	Length of the following field
METHODNAME	B[DATALENGTH]	Method name
LENGTHINDICATOR	B1-5	Length of the following field
SAMLASSERTION	B[DATALENGTH]	SAML assertion

Initial Authentication Reply



Table 81:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field
METHODNAME	B[DATALENGTH]	Method name
LENGTHINDICATOR	B1	Length of the following field (always zero)
SAMLUSER	B[0]	User name associated with the SAML assertion

Final Authentication Request

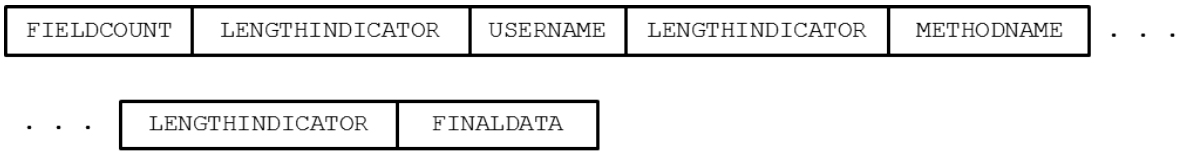


Table 82:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field
USERNAME	B[DATALENGTH]	User name

LENGTHINDICATOR	B1-5	Length of the following field
METHODNAME	B[DATALENGTH]	Method name
LENGTHINDICATOR	B1	Length of the following field (always zero)
FINALDATA	B[0]	Final data (currently empty)

Final Authentication Reply

FIELDCOUNT	LENGTHINDICATOR	METHODNAME	LENGTHINDICATOR	SESSIONCOOKIE

Table 83:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field
METHODNAME	B[DATALENGTH]	Method name
LENGTHINDICATOR	B1	Length of the following field (always zero)
SESSIONCOOKIE	B[0]	Session cookie used for reconnect

2.9.2.4 Session Cookie Authentication

Session cookies can only be used in case of a reconnection. The cookie is obtained from a previous connection. It makes two server roundtrips.

Initial Authentication Request

FIELDCOUNT	LENGTHINDICATOR	USERNAME	LENGTHINDICATOR	METHODNAME	. . .

. . . | LENGTHINDICATOR | SESSIONCOOKIE |

Table 84:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field

USERNAME	B[DATALENGTH]	User name
LENGTHINDICATOR	B1-5	Length of the following field
METHODNAME	B[DATALENGTH]	Method name
LENGTHINDICATOR	B1-5	Length of the following field
SESSIONCOOKIE	B[DATALENGTH]	Session cookie, process ID, and host-name

Initial Authentication Reply



Table 85:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field
METHODNAME	B[DATALENGTH]	Method name
LENGTHINDICATOR	B1	Length of the following field (always zero)
SERVERREPLY	B[0]	Server reply (currently empty)

Final Authentication Request

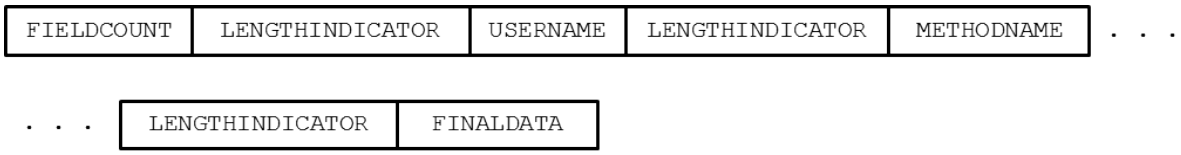


Table 86:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field
USERNAME	B[DATALENGTH]	User name
LENGTHINDICATOR	B1-5	Length of the following field
METHODNAME	B[DATALENGTH]	Method name
LENGTHINDICATOR	B1	Length of the following field (always zero)

FINALDATA	B[0]	Final data (currently empty)

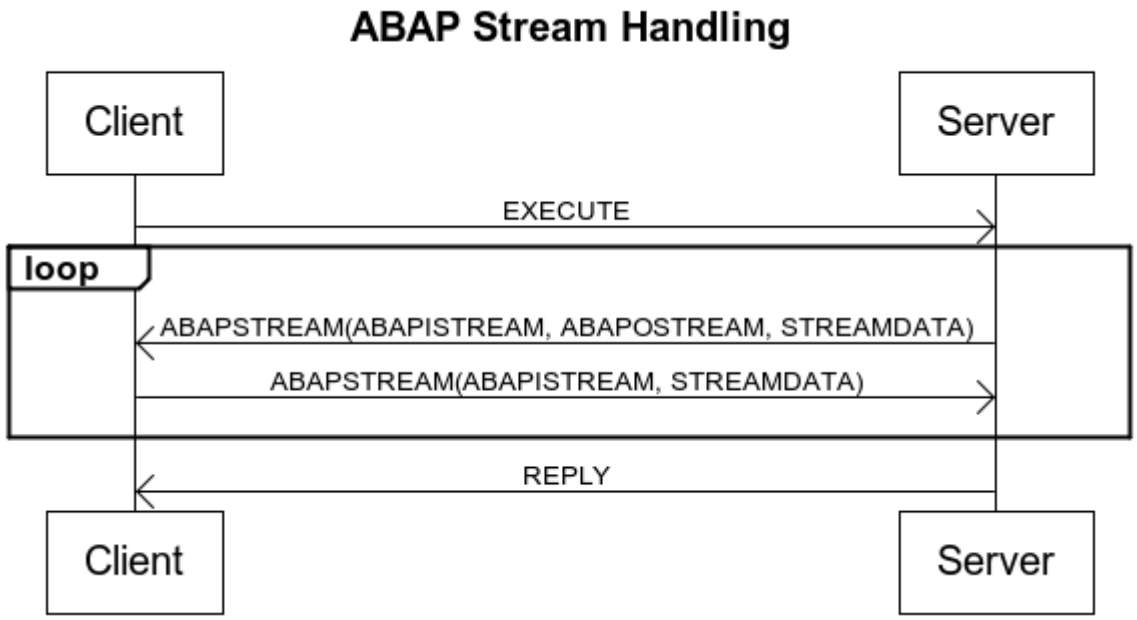
Final Authentication Reply

FIELDCOUNT	LENGTHINDICATOR	METHODNAME	LENGTHINDICATOR	FINALDATA

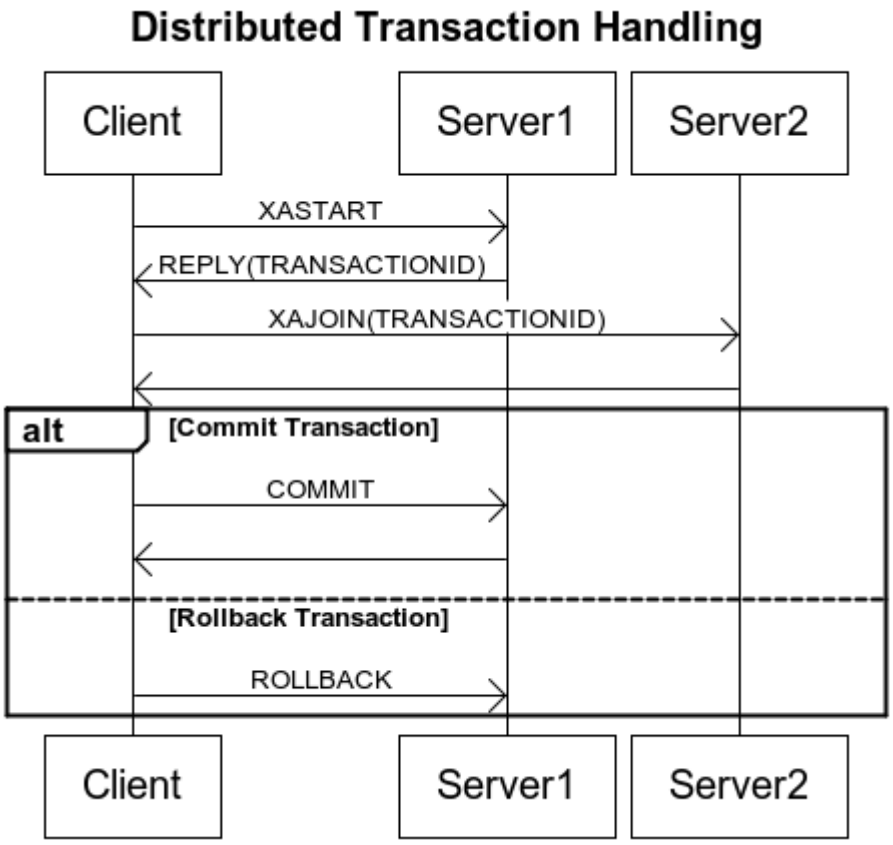
Table 87:

Field	Data Type	Description
FIELDCOUNT	I2	Number of fields within this request
LENGTHINDICATOR	B1	Length of the following field
METHODNAME	B[DATALENGTH]	Method name
LENGTHINDICATOR	B1	Length of the following field (always zero)
FINALDATA	B[0]	Final data (currently empty)

2.9.3 ABAP Stream Handling



2.9.4 Distributed Transaction Handling



2.9.5 Statement Routing

Statement routing is the method of evaluating the correct server node of a distributed system before statement execution. It reduces the overhead in server processing and reduces communication between server nodes.

For SQLDBC-based clients, such as ODBC, ODBO, ADO.NET, Python DB API, and DBSL, the client library makes statement routing decisions to reduce server-side inter-node routing. To utilize this feature, the optimized_routing_for_partition_table property must be set to "on". This is done by default.

Table 88: Utilization of Table Partitioning Information for Statement Routing Optimization

Utilization	JDBC	SQLDBC
Hash partitioning	Yes	Yes
Hash partitioning during split batch insert	Split batch not supported	Yes
Range partitioning	No	Yes

Utilization	JDBC	SQLDBC
Range partitioning during split batch insert	Split batch not supported	Yes

> **Note**
>
> Active/Active (Read Enabled) hint-based routing uses the same mechanism as statement routing.

Preconditions

The server decides which statements are eligible for statement routing. When preparing a statement, the server returns a TABLELOCATION part or a PARTITIONINFORMATION part to describe the preferred nodes in detail.

Table 89: JDBC and SQLDBC Statement Routing Settings

Server setting	Client Connection Property	Must Be Set To
indexserver.ini->distribution->client_distribution_mode	client_distribution_mode	"Statement" (default) or "All"

Table 90: SQLDBC Split Batch Client-Side Routing Settings

Server setting	Client Connection Property	Must Be Set To
indexserver.ini->distribution->split_batch_commands	SPLITBATCHCOMMANDS	"1" (default)

3 Glossary

CESU-8 Compatibility Encoding Scheme for UTF-16: 8-Bit

DDL Data Definition Language

DML Data Modification Language

FAE For All Entries. A specific ABAP Language construct, where a client-side table is joined with a server-side table.

FDA Fast Data Access. A method to submit data for INSERT in the format used by the SAP ABAP Application Server (ABAP Table) to the server or retrieve SELECT results in the same format, to avoid field-wise copying and data conversion.

MSB Most Significant Bit. The highest bit in an integer value, for example, Bit 7 in a byte.

SAML Structured Authentication Markup Language

Important Disclaimer for Features in SAP HANA Platform, Options and Capabilities

SAP HANA server software and tools can be used for several SAP HANA platform and options scenarios as well as the respective capabilities used in these scenarios. The availability of these is based on the available SAP HANA licenses and the SAP HANA landscape, including the type and version of the back-end systems the SAP HANA administration and development tools are connected to. There are several types of licenses available for SAP HANA. Depending on your SAP HANA installation license type, some of the features and tools described in the SAP HANA platform documentation may only be available in the SAP HANA options and capabilities, which may be released independently of an SAP HANA Platform Support Package Stack (SPS). Although various features included in SAP HANA options and capabilities are cited in the SAP HANA platform documentation, each SAP HANA edition governs the options and capabilities available. Based on this, customers do not necessarily have the right to use features included in SAP HANA options and capabilities. For customers to whom these license restrictions apply, the use of features included in SAP HANA options and capabilities in a production system requires purchasing the corresponding software license(s) from SAP. The documentation for the SAP HANA optional components is available in SAP Help Portal at http://help.sap.com/hana_options. If you have additional questions about what your particular license provides, or wish to discuss licensing features available in SAP HANA options, please contact your SAP account team representative.

Important Disclaimers and Legal Information

Coding Samples

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, unless damages were caused by SAP intentionally or by SAP's gross negligence.

Accessibility

The information contained in the SAP documentation represents SAP's current view of accessibility criteria as of the date of publication; it is in no way intended to be a binding guideline on how to ensure accessibility of software products. SAP in particular disclaims any liability in relation to this document. This disclaimer, however, does not apply in cases of willful misconduct or gross negligence of SAP. Furthermore, this document does not result in any direct or indirect contractual obligations of SAP.

Gender-Neutral Language

As far as possible, SAP documentation is gender neutral. Depending on the context, the reader is addressed directly with "you", or a gender-neutral noun (such as "sales person" or "working days") is used. If when referring to members of both sexes, however, the third-person singular cannot be avoided or a gender-neutral noun does not exist, SAP reserves the right to use the masculine form of the noun and pronoun. This is to ensure that the documentation remains comprehensible.

Internet Hyperlinks

The SAP documentation may contain hyperlinks to the Internet. These hyperlinks are intended to serve as a hint about where to find related information. SAP does not warrant the availability and correctness of this related information or the ability of this information to serve a particular purpose. SAP shall not be liable for any damages caused by the use of related information unless damages have been caused by SAP's gross negligence or willful misconduct. All links are categorized for transparency (see: http://help.sap.com/disclaimer).



go.sap.com/registration/contact.html

© 2016 SAP SE or an SAP affiliate company. All rights reserved.
No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.
Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.
These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.
SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.
Please see http://www.sap.com/corporate-en/legal/copyright/index.epx for additional trademark information and notices.