Virtual DDBMS Network Protocol
Draft v0.8.0.0

Note: In this draft (*major* version is zero), compatibility between protocol versions are not guaranteed.

General Protocol Format

<magic_numbers> <data_size> <protocol_version> <connection_id> <serial_number> <compression> <res_1> <res_2> <res_3> <res_a> <res_b> <message_body>

magic_numbers : four characters, which can be :
VDCM : message from client to server.
VDSM : message from server to client.

data_size : a 32-bit unsigned integer containing the total message size (the length of the *message_body* + 32).

protocol_version : a 32-bit unsigned integer containing the protocol version.

MSB			LSB
8-bit	8-bit	8-bit	8-bit
major	*minor*	*revision*	*fix*

connection_id : a 32-bit unsigned integer containing the connection ID (ignored for some messages).

serial_number : a 32-bit unsigned integer containing the message serial number (a sequence number generated by the client).

compression : an 8-bit unsigned integer defining the compression type of the message body.

res_1, res_2, res_3 : an 8-bit unsigned integer, reserved for future expansion.

res_a, res_b : a 32-bit unsigned integer, reserved for future expansion.

message_body : variable length bytes containing the real message (can be compressed and/or symmetrically encrypted).

Note:
- Change in *major* version indicates major revision on the protocol which is not backward compatible.
- Change in *minor* version indicates minor addition to the protocol string(s) which is backward compatible. In case the newly added protocol string(s) are not used, the system would still function normally.
- Change in *revision* version indicates minor extension in the current protocol string(s) which is backward compatible. For example: when new compression/encryption mode is added.
- Change in *fix* version indicates bug-fixes in the server/client software. This number can be safely ignore in most of the case. Users, however, may use this number if need to track special bugs or changes. The server and client can set different number for this version number. Generally, server/client with higher *fix* version would has less bug and more compliance to the protocol standard.

Upon opening TCP connection to the server, the client must send a valid *Request for Connection* followed by a valid *Request for Login* in a reasonable amount of time (or the server will disconnect automatically). The same TCP connection (socket) must always be used for all the time (from the start of the connection until disconnection).

Message Body Protocol Format

(C) : identifies client's message (request).
(S) : identifies server's message (response).

The numbers postfix, like **(S1)**, **(S2)**, etc. identify alternative message which may be sent.

In secure connection, except explicitly stated, the *message_body* must be encrypted with the server key.

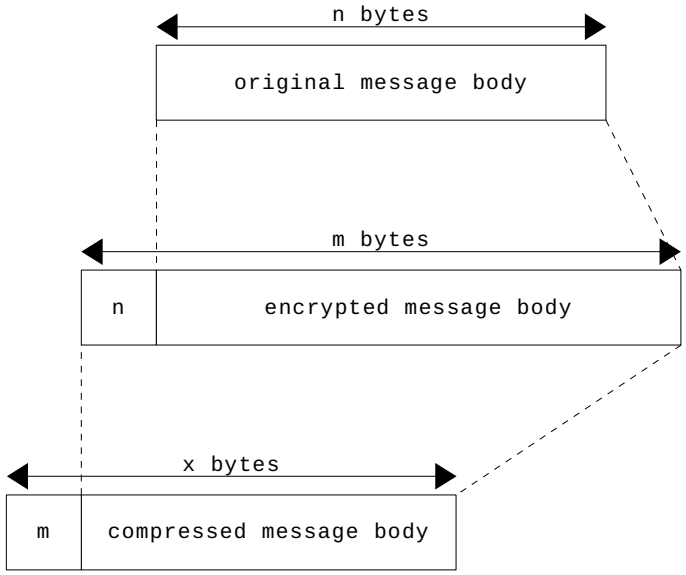
All the strings, except explicitly stated, are non null-terminated strings.

All numbers must be represented in network byte order (Big Endian).

In secure connection, the *message_body* must be always encrypted (except explicitly stated). However, compressed and non-compressed request/response may come interleaved.

In case both compression and encryption are used, the *message_body* must be encrypted first before compressed.

Encrypted and/or compressed message body will have/has additional header(s) as shown by this example picture:



n = a 32-bit unsigned integer containing the size of the original message body.

m = a 32-bit unsigned integer containing the size of the encrypted message body plus the the size of *n*.

x = a 32-bit unsigned integer containing the size of the compressed message body plus the the size of *m*.

Hence, in the header, the *data_size* must be set with (*x* + 32).

Request for Server Version

(C) : version

Note: - The *protocol_version* when sending the request must be set with the highest protocol version supported/requested by the client.
- The *connection_id* when sending the request is ignored.
- If this request is sent prior any *Request for Connection*, the *message_body* must not be encrypted.
- After successful *Request for Connection*, the *message_body* of this request can be encrypted or not encrypted.

(S1) : *error <message>*

message : variable length bytes containing the error message.

Note: - The server will send *error* if there is error or the server does not support the protocol version requested by the client.
- The server will close the connection after sending the response if there is no prior successful *Request for Connection*.

(S2) : *ok <highest_protocol_version> <message>*

highest_protocol_version : a 32-bit unsigned integer containing the highest protocol version supported by the server.
message : variable length bytes containing informational message from the server.

Request for Connection

(C) : connect <*secure*> [<*pk_size*> <*pk_data*>]

secure : an 8-bit unsigned integer, if non zero means the client wants secure connection.
pk_size : a 32-bit unsigned integer containing the size of the client's public key (currently the size of the key can be 256, 512, 1024, or 2048 bits).
pk_data : the client's RSA public key in 256/512/1024/2048 bits multi-precision unsigned integer (32/64/128/256 bytes).

Note: - The *connection_id* when sending the request is ignored.
- The *message_body* must not be encrypted.

(S1) : *error <message>*

message : variable length bytes containing the error message.

Note: - Upon rejection, the server will close the connection.

(S2) : *ok <new_connection_id> [<server_key_len> <server_key>]*

new_connection_id : a 32-bit unsigned integer containing the ID of the newly established connection (must be used for subsequent requests with the same).
server_key_len : a 32-bit unsigned integer containing the decrypted length of the server's key (currently the size of the key is always 128 bits).

server_key : the server key encrypted with the client's public key (hence the length will become 32/64/128/256 bytes).

Note: - The *server_key* is to be used for symmetrical encryption (AES) of the *message_body* on subsequent requests.
- *Request for Connection* must be followed by a successful *Request for Login* in a reasonable amount of time or the server will close the connection automatically (currently the limit is set to 90 seconds).

Request for Login

(C) : login <user_len> <*user_name*> <*password*>

user_*len* : an 8-bit unsigned integer containing the length of the user name.
user_name : variable length bytes containing the user name.
password : variable length bytes containing the user password.

Note: - The *user_name* and *password* are sent in CRC-32 encoded form.

(S1) : *error* <*message*>

message : variable length bytes containing the error message.

(S2) : *ok*

Note: - The server will translate the given *user_name* and *password* to the appropriate 4 base-64 characters ID needed by the DDBMS engine, complete with the ACL for the user.

Request for Compression

(C) : compression <*code*>

code : an 8-bit unsigned integer containing the requested compression mode to be enabled, supported values:

0 : no compression
1 : gzip

Note: - For new connection, compression is disabled by default.
- Even the client has requested to enable compression, may not all the sever messages will be compressed. The server will decide when to compress or not to compress the messages.

(S1) : *error* <*message*>

message : variable length bytes containing the error message.

(S2) : *ok*

Request for Disconnection

(C) : *disconnect*

(S1) : *error <message>*

message : variable length bytes containing the error message.

(S2) : *ok*

Request for Ping (Keep-Alive)

(C) : *ping*

Note: - This request is necessary to keep the connection alive if the client will be idle for some time (currently the limit is set to 10 minutes).

(S1) : *error <message>*

message : variable length bytes containing the error message.

(S2) : *ok*

Request for SQL Evaluation

(C) : *eval_sql <sql_statement>*

(S1) : *error <message>*

message : variable length bytes containing the error message.

(S2) : *ok*

Request for SQL Execution

(C) : *exec_sql <sql_statement>*

(S1) : *error <message>*

message : variable length bytes containing the error message.

(S2) : ok

Request for SQL Preparation

(C) : prep_*sql <sql_statement>*

(S1) : *error <message>*

message : variable length bytes containing the error message.

(S2) : ok

Request for Execution of the Prepared SQL

(C) : prep_*exec* <*total*> [<*id*> <*len*> <*value*>
[<*id*> <*len*> <*value*>]...]

total : an 8-bit unsigned integer containing the number of parameters that will be set.
id : an 8-bit unsigned integer containing ID of the parameter.
len : a 32-bit unsigned integer containing length of the parameter.
value : variable length bytes containing the value.

(S1) : *error* <*message*>

message : variable length bytes containing the error message.

(S2) : ok

Request for Results (Rows)

(C) : *fetch_row* <*max_num_of_row*>

max_num_of_row : an 8-bit unsigned integer containing the maximum number of rows which shall be returned.

(S1) : *error* <*message*>

message : variable length bytes containing the error message.

(S2) : *ok* <*num_of_row*> <*num_of_col*> [<*col_len*>... <*col_data*>...
[<*col_len*>... <*col_data*>...]...]

num_of_row : an 8-bit unsigned integer containing the number of rows returned.
num_of_col : an 8-bit unsigned integer containing the number of columns in each row.
col_len : 32-bit unsigned integers containing the length of the columns' data.
col_data : variable length bytes containing the columns' data.

Note: - A *col_len* of zero means that the corresponding column contains SQL NULL.

Request for Discarding Results (Rows)

(C) : discard_*row* <*num_of_row*>

num_of_row : a 32-bit unsigned integer containing the number of rows which shall be discarded.

(S1) : *error* <*message*>

message : variable length bytes containing the error message.

(S2) : *ok*

Request for Discarding All Results (Rows)

(C) : discard_*all_rows*

(S1) : *error <message>*

message : variable length bytes containing the error message.

(S2) : *ok*

Request for Adding A Resource

(C) : *add_resource <type_len> <host_len> <user_len> <password_len> <type> <host> <port> <user> <password> <description>*

type_len : an 8-bit unsigned integer containing the length of the resource type string.
host_len : an 8-bit unsigned integer containing the length of the host name string.
user_len : an 8-bit unsigned integer containing the length of the user name string.
password_len : an 8-bit unsigned integer containing the length of the user password string.
type : variable length bytes containing the resource type.
host : variable length bytes containing the host name.
port : a 32-bit unsigned integer containing the port number.
user : variable length bytes containing the user name.
password : variable length bytes containing the password.
description : variable length bytes containing the description.

(S1) : *error <message>*

message : variable length bytes containing the error message.

(S2) : *ok <new_resource_id>*

new_resource_id : four base-64 characters of the newly added resource.

Request for Removing A Resource

(C) : *remove_resource <resource_id>*

resource_id : four base-64 characters containing the resource ID.

(S1) : *error <message>*

message : variable length bytes containing the error message.

(S2) : *ok*

Request for Adding A Resource - Extended Version 1

(C) : ***add_resource_ex <type_len> <host_len> <dbname_len>***
<user_len> <password_len>
<type> <host> <port> <dbname>
<user> <password> <description>

dbname_len : an 8-bit unsigned integer containing the length of the database name string.
dbname : variable length bytes containing the database name.

Note:

- This protocol is available starting from protocol version 0.6.1.0.
- Please refer to the original *add_resource* protocol string for the full field explanations.

Request for Adding A Resource - Extended Version 2

(C) : ***add_resource_ex2 <type_len> <host_len> <dbname_len>***
<user_len> <password_len> <odbcopts_len>
<type> <host> <port> <dbname>
<user> <password> <odbcopts> <description>

odbcopts_len : an 8-bit unsigned integer containing the length of the ODBC options
odbcopts : variable length bytes containing the ODBC options.

Note:

- This protocol is available starting from protocol version 0.6.2.0.
- Please refer to *add_resource_ex* protocol string for the full field explanations.