continuent

Sequoia

Controller/Driver Protocol

Specifications

Version history:

Version	Date	Author	Status	Notes
0.1	2005-09-12	Gilles Rayrat	Draft	Document creation
0.2	2005-10-27	Gilles Rayrat	Public	Fixes following review by Emmanuel Cecchet Document committed in public CVS
0.3	2005-11-03	Gilles Rayrat	Public	Updated from Emic to Continuent Misc. typos corrections
0.4	2005-11-18	Gilles Rayrat	Public	Updated to protocol 24/sequoia version 2.4: connection establishment line separator added (and removed from queries), execXXXrequest to StatementExecuteXXX, ExecXXXStoredProc to CallableStatementXXX, XXXExecute functions added
0.5	2005-11-24	Gilles Rayrat	Public	Updated to protocol version 27: Field serialization in ResultSet now sends fiel label after field name
0.6	2005-12-12	Gilles Rayrat	Public	Updated to protocol version 29: added request ids in Execute functions results retrieval + RetreiveExecuteXXXResult commands
0.7	2006-01-06	Gilles Rayrat	Public	Updated to protocol version 35: Added PreparedStatementGetMetaData and RetrieveExecuteUpdateWithKeysResult commands, string serialization is now real UTF8, removed controllerNeedSkeleton, added persistent connection support, modified RestoreConnectionState

2000 Powell Street, Suite 1090, Emeryville, CA 94608
Tel. 408-998-7878 Fax 408-668-1009

0.8	2006-01-23	Gilles Rayrat	Public	Updated to protocol version 38. Added vdbFound at connection establishment, RetrieveExecute* now just provide the unique request id, added RetrieveCommit and RetrieveRollback commands, fixed Commit typo, added writeExecutedInTransaction for RestoreConnectionState Removed useless prefixes before diagram names Removed sequoia version in filenames

2000 Powell Street, Suite 1090, Emeryville, CA 94608
Tel. 408-998-7878 Fax 408-668-1009

TABLE OF CONTENT

2000 Powell Street, Suite 1090, Emeryville, CA 94608
Tel. 408-998-7878 Fax 408-668-1009

2000 Powell Street, Suite 1090, Emeryville, CA 94608
Tel. 408-998-7878 Fax 408-668-1009

2000 Powell Street, Suite 1090, Emeryville, CA 94608
Tel. 408-998-7878 Fax 408-668-1009

1. About this document

This document describes the specifications of the communication protocol between Sequoia controller and the external driver.

This document has been build upon and applies to Sequoia 2.6

2. Related documents

3. Overview

This document approach is from the client (equally called driver) side. It is mandatory to keep in mind that all communications are viewed from the client point of view, *sending* commands *to* and *receiving* data *from* the controller.

Knowing this, the protocol can now be demistified by saying that the communication between driver and controller is actually an exchange of basic data types (integers, booleans, bytes, ...) over a regular network socket. For example, what is called a 'command', is in fact a dedicated integer sent to the controller. But for apparent lisibility reasons, an abstraction is provided in the whole protocol, so complex types like ResultSets or exceptions serialization can be easily described.

This document follows this abstraction by first describing, at the higher level, the set of commands accepted by the controller. Next sections will present the way to serialize complex types. Finally, basic data type exchange will be detailed in the last section.

4. Commands

All data exchange in the protocol start by sending commands. These commands are defined in the source file `org/continuent/sequoia/common/protocol/Commands.java`, which is fully commented out (see javadoc of this class) and on which this document is based.

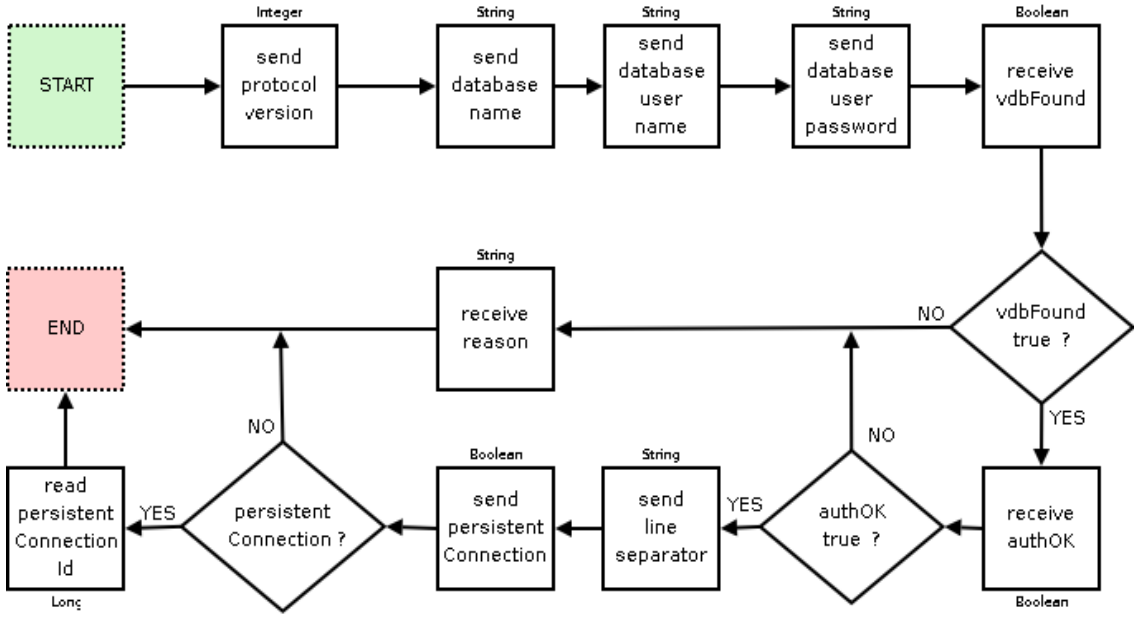
The following command description consists in graphsets that give a quick view of the command protocol, followed by an exhaustive parameter description for full details on the command parameters and data returned back by the controller.

4.1. CONNECTION MANAGEMENT COMMANDS

4.1.1 CONNECTION ESTABLISHMENT

The connection creation is not a *real* command: in order to establish the connection, the client must send a **Protocol Version** to the controller. This allows to check that driver and controller are compatible with each other.

4.1.1.1 Diagram



4.1.1.2 Parameters description

- [out] **protocol version**: Integer aimed to check compatibility between driver and controller. Sequoia 2.6 current protocol version is 38
- [out] **database name**: String name of the virtual database to connect to
- [out] **database user name**: String user name to access the DB
- [out] **database user password**: String user password to log on to the DB
- [in] **vdbFound**: Boolean sent by the controller to confirm that the given virtual database is available on the controller. `true` if the vdb was found, `false` otherwise

  (only if **vdbFound** `false`)

  - [in] **reason**: String giving the reason of failure

  (only if **vdbFound** `true`)

- [in] **authOK**: Boolean sent by the controller to notify whether the authentification succeeded. `true` if connection ok, `false` if an error occured

  (only if **authOK** `false`)

  - [in] **reason**: String giving the reason of connection failure

  (only if **authOK** `true`)

  - [out] **line separator**: String line separator that will be used in the (eventual) queries for this connection (typically "\n", "\r", "\r\n" or "\n\r")
  - [out] **persistent connection**: Boolean `true` if the connection must be persisted on all cluster backends even when autoCommit=true

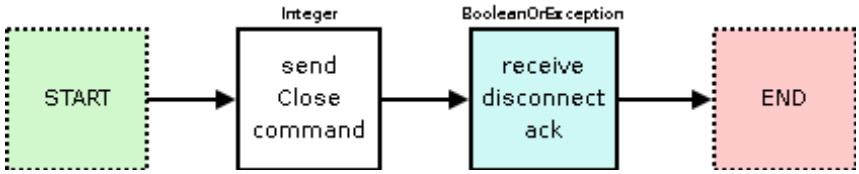
    (only if **persistent connection** `true`)

    - [in] **persistent connection id**: Long identifier of the connection

4.1.2 CLOSE

When receiving a Close command, the controller replies `true` upon successfull disconnection. This response should be used to wait for the controller to close the connection himself before closing the socket at the client side.

4.1.2.1 Diagram



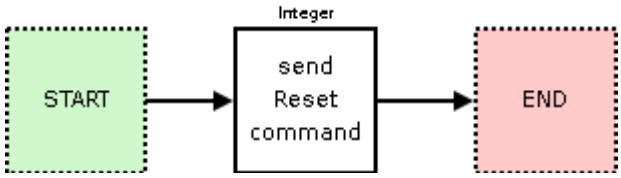
4.1.2.2 Parameters description

- [out] **Close command**: Integer value = 30
- [in] **ack**: BooleanOrException `true` if the connection was successfully closed, `false` otherwise

4.1.3 RESET

The reset command is used for closing the connection but keeping it opened to transparently pool it.

4.1.3.1 Diagram



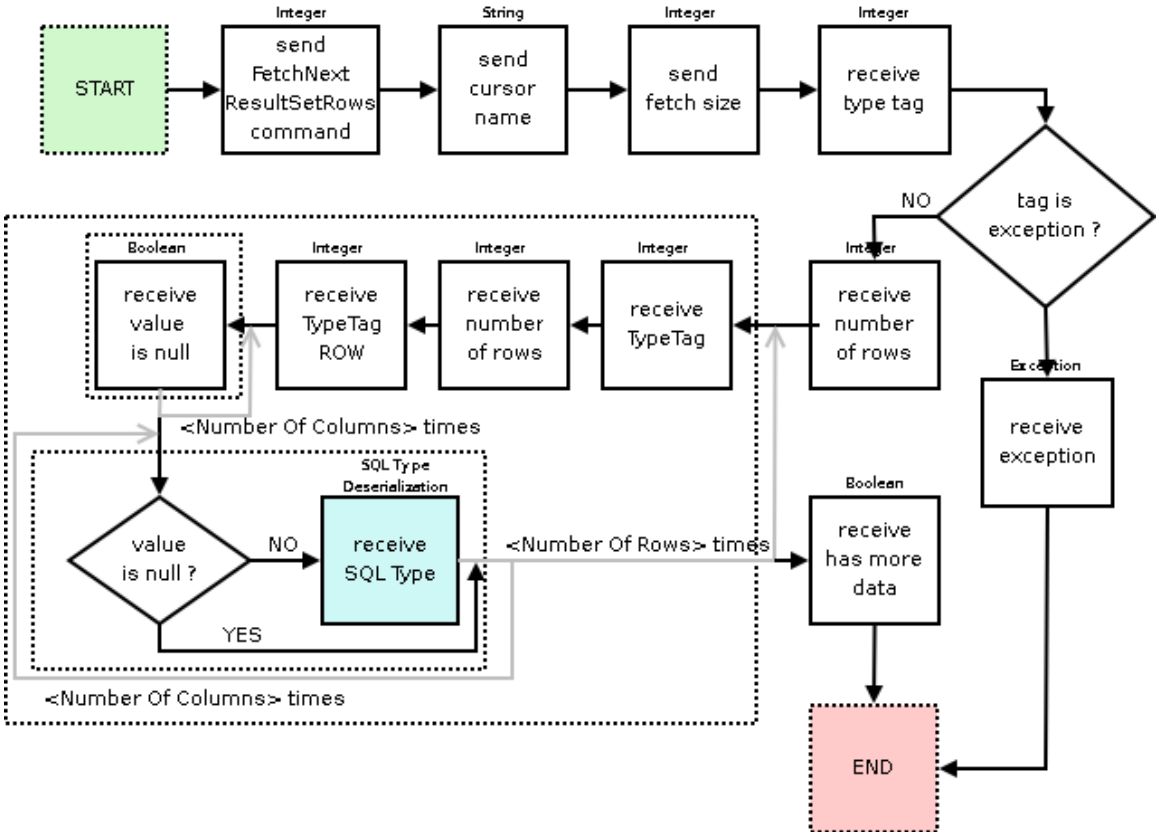
4.1.3.2 Parameters description

- [out] **Reset command**: Integer value = 31

4.1.4 FETCH NEXT RESULTSET ROWS

Fetches next rows of data for ResultSet streaming.

4.1.4.1 Diagram



4.1.4.2 Parameters description

- [out] **FetchNextResultSetRows**: Integer value = 32
- [out] **cursor name**: String name of the result set cursor

- [out] **fetch size**: Integer representing the maximum number of results to be retrieved at a time
- [in] **type tag**: Integer `EXCEPTION` (19) or `NOT_EXCEPTION` (18)

  (only if **type tag** = `NOT_EXCEPTION`)

  - [in] **numberOfRows**: Integer

    Number of rows times:

    - [in] **TypeTag**: Integer possible values (`STRING=0, BIGDECIMAL=1, BOOLEAN=2, INTEGER=3, LONG=4, FLOAT=5, DOUBLE=6, BYTE_ARRAY=7, SQL_DATE=8, SQL_TIME=9, SQL_TIMESTAMP=10, CLOB=11, BLOB=12, JAVA_SERIALIZABLE=13`)
    - [in] **numberOfRows**: Integer
    - [in] **TypeTag**: Integer value ROW = 18

      number Of Rows times:

      number Of Columns times:

      - [in] **value is null**: Boolean

      number Of Columns times:

      - [in] **SQL type deserialization** (only if value is null `false`)
  - [in] **hasMoreData**: Boolean

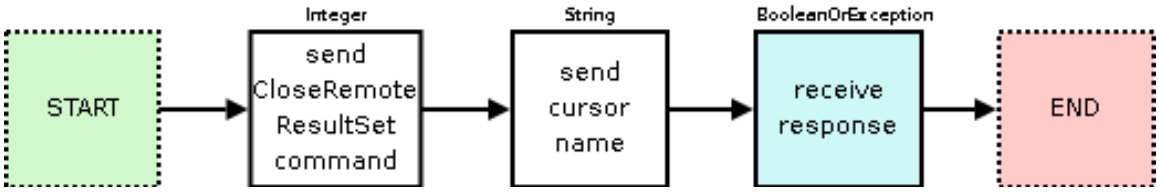
  (only if type tag = EXCEPTION)

  - [in] **exception**: Exception SQLException informing there were no valid ControllerResultSet to fetch data from

4.1.5 CLOSE REMOTE RESULTSET

Closes a remote ResultSet that was opened for streaming.

4.1.5.1 Diagram



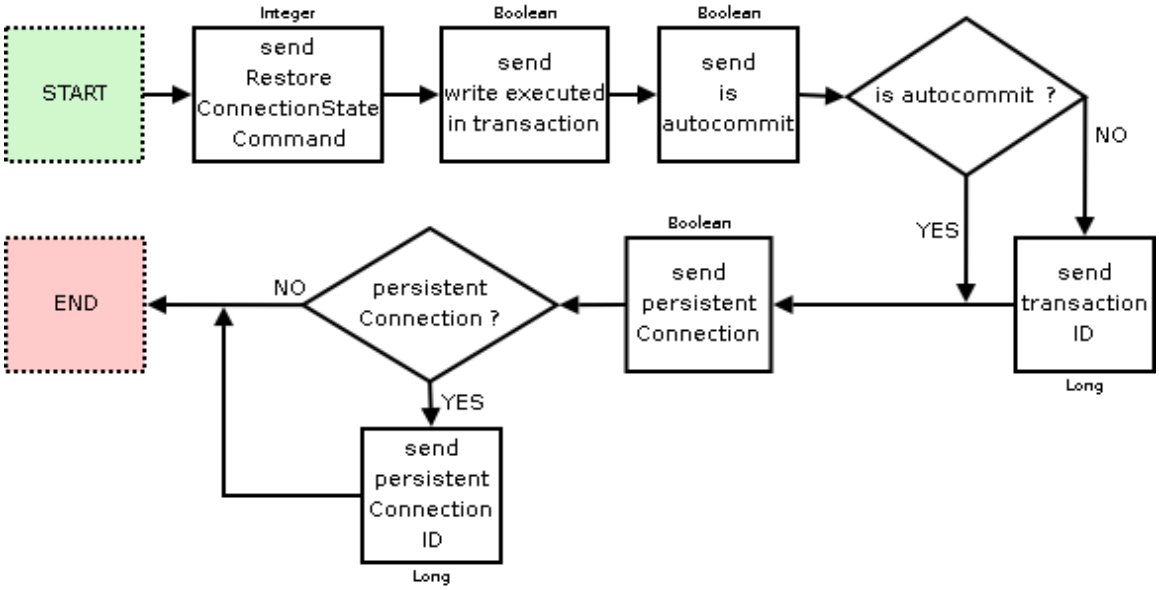
4.1.5.2 Parameters description

- [out] **CloseRemoteResultSet**: Integer value = 33
- [out] **cursor name**: String name of the result set cursor
- [in] **response**: BooleanOrException the controller response is always `true` if not an exception

4.1.6 RESTORE CONNECTION STATE

Restores a connection state after an automatic reconnection.

4.1.6.1 Diagram



4.1.6.2 Parameters description

- [out] **RestoreConnectionState**: Integer value = 34
- [out] **write executed in transaction**: Boolean telling whether or not a write request been executed in the current transaction
- [out] **is autocommit**: Boolean telling whether or not this request has been sent in autocommit mode

  (only if **is autocommit** is `false`)

  - [out] **transaction ID**: Long ID of the transaction before restore
- [out] **persistent connection**: Boolean `true` if the connection must be persisted on all cluster backends even when autoCommit=true

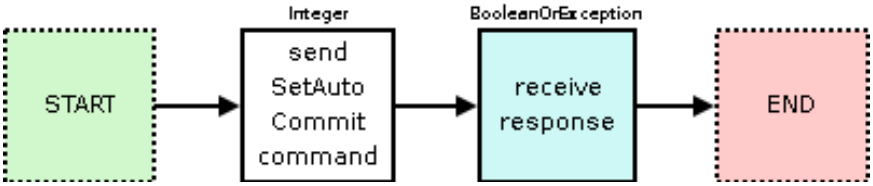
(only if **persistent connection** `true`)

- [out] **persistent connection id**: Long identifier of the connection

4.1.7 SET AUTOCOMMIT

Command to change the autocommit value from `false` to `true`. We want to commit the current transaction but we don't want to start a new one.

4.1.7.1 Diagram



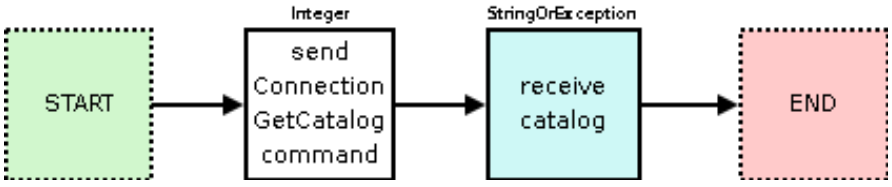
4.1.7.2 Parameters description

- [out] **SetAutoCommit**: Integer value = 35
- [in] **response**: BooleanOrException the controller response is always `true` if not an exception

4.1.8 CONNECTION GET CATALOG

Retrieves the catalog (database) we are connected to.

4.1.8.1 Diagram



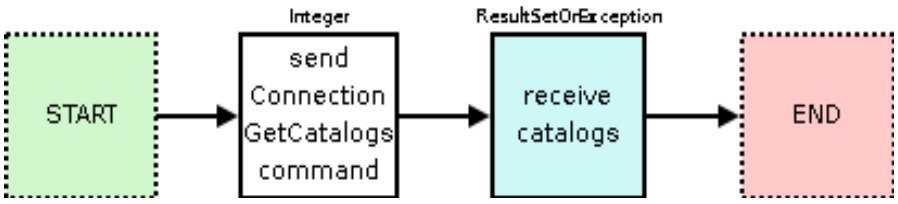
4.1.8.2 Parameters description

- [out] **ConnectionGetCatalog**: Integer value = 36
- [in] **catalog**: StringOrException the virtual database name to be used by the client

4.1.9 CONNECTION GET CATALOGS

Retrieves the list of available catalogs (databases).

4.1.9.1 Diagram



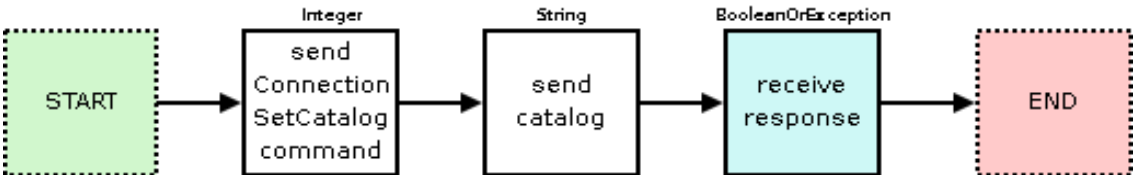
4.1.9.2 Parameters description

- [out] **ConnectionGetCatalogs**: Integer value = 37
- [in] **catalogs**: ResultSetOrException. Possible exceptions types are `ControllerCoreException, BackendDriverException`.

4.1.10 CONNECTION SET CATALOG

Connects to another catalog/database (as the same user).

4.1.10.1 Diagram



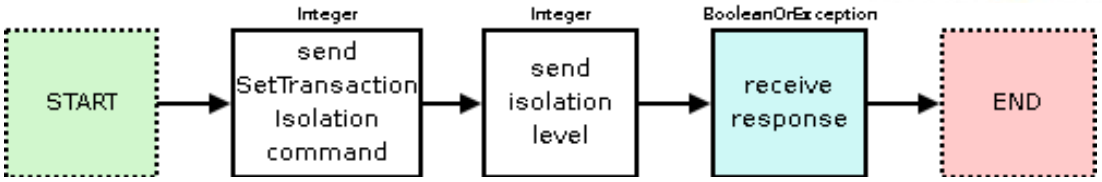
4.1.10.2 Parameters description

- [out] **ConnectionSetCatalog**: Integer value = 38
- [out] **catalog**: String catalog as a String
- [in] **response**: BooleanOrException `true` if the catalog could be set, `false` if the corresponding Virtual Database does not exists

4.1.11 SET TRANSACTION ISOLATION

Sets the new transaction isolation level to use for this connection.

4.1.11.1 Diagram



4.1.11.2 Parameters description

- [out] **SetTransactionIsolation**: Integer value = 39
- [out] **isolation level**: Integer possible values

```
TRANSACTION_READ_UNCOMMITTED = 1,
TRANSACTION_READ_COMMITTED = 2,
TRANSACTION_REPEATABLE_READ = 4,
TRANSACTION_SERIALIZABLE = 8
```

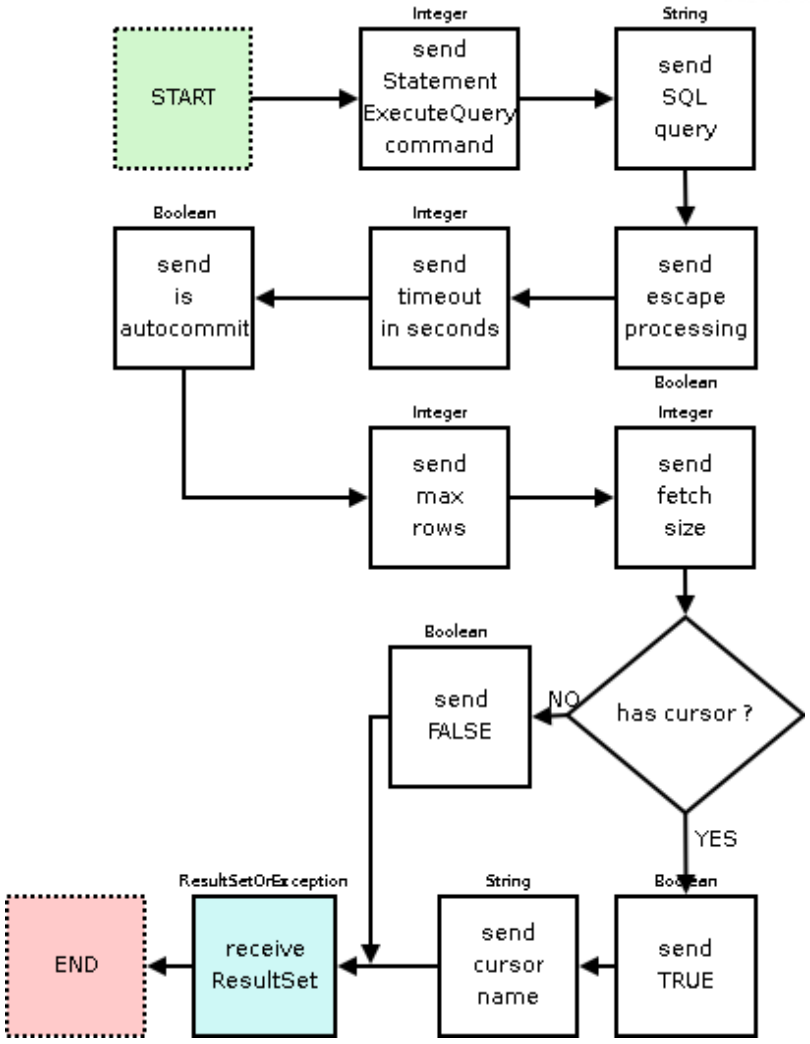
- [in] **response**: BooleanOrException alway `true` if not an exception. Exception can only be of type `SQLException` and can occur when a transaction is running while trying to change level, or because the database does not support transactions

4.2. SQL REQUESTS HANDLING

4.2.1 STATEMENT EXECUTE QUERY

Performs a read request and receives the data read.

4.2.1.1 Diagram



4.2.1.2 Parameters description

- [out] **StatementExecuteQuery**: Integer value = 0
- [out] **SQL query**: String containing the query in SQL
- [out] **escape processing**: Boolean indicating if the backend driver should escape processing before sending to the database
- [out] **timeout in seconds**: Integer representing the number of seconds for this request time out (0 = no timeout)
- [out] **is autocommit**: Boolean telling whether or not this request must execute in autocommit mode
- [out] **max rows**: Integer representing the maximum number of rows the result set can contain
- [out] **fetch size**: Integer representing the maximum number of results to be retrieved at a time

- [out] **has cursor**: Boolean telling whether or not we have a cursor name

  (only if **has cursor** is `true`)

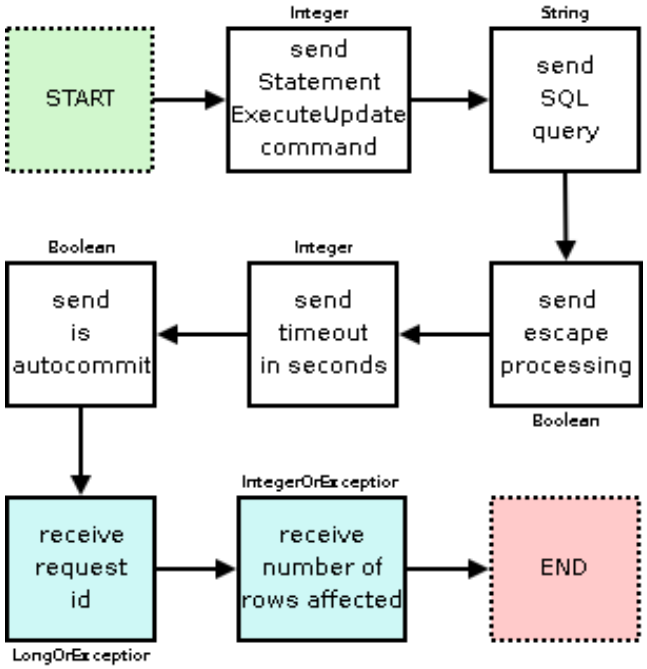
  - [out] **cursor name**: Boolean the name of the cursor to be used

- [in] **result**: ResultSetOrException. Possible exceptions types are `ControllerCoreException, NoMoreBackendException, IOException, BackendDriverException, NotImplementedException`

4.2.2 STATEMENT EXECUTE UPDATE

Performs a write request and receives the number of rows affected.

4.2.2.1 Diagram



4.2.2.2 Parameters description

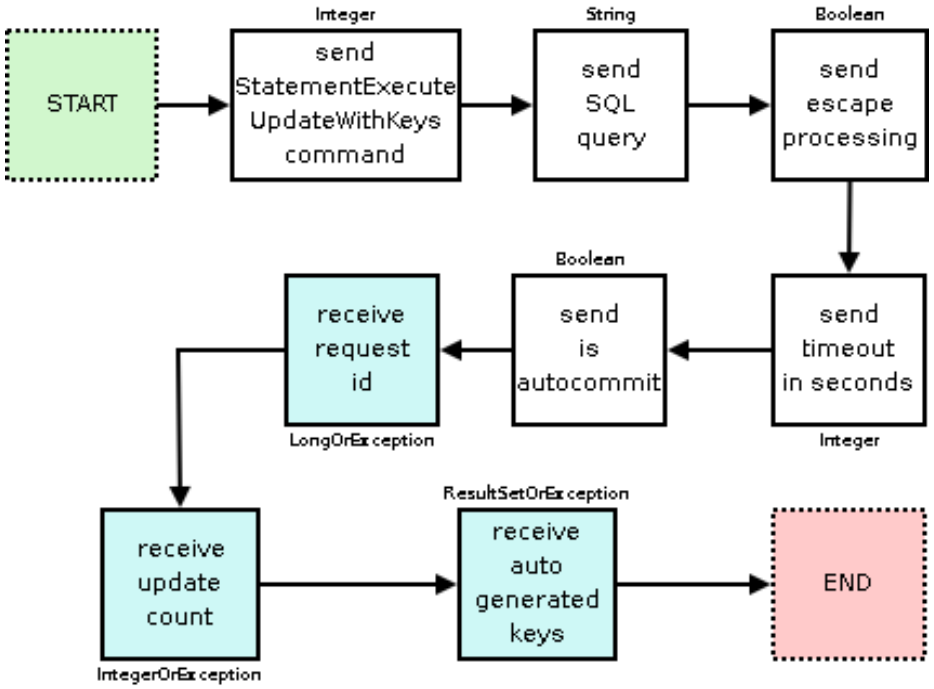
- [out] **StatementExecuteUpdate**: Integer value = 1
- [out] **SQL query**: String containing the query in SQL
- [out] **escape processing**: Boolean for knowing if the backend driver should escape processing before sending to the database
- [out] **timeout in seconds**: Integer representing the number of seconds for this request time out (0 = no timeout)

- [out] **is autocommit**: Boolean telling whether or not this request has been sent in autocommit mode
- [in] **request id**: LongOrException. Request identifier used for failover purposes
- [in] **number of rows affected**: IntegerOrException the number of rows affected by the write request

4.2.3 STATEMENT EXECUTE UPDATE WITH KEYS

Performs a write request and receives the auto generated keys.

4.2.3.1 Diagram



4.2.3.2 Parameters description

- [out] **StatementExecuteUpdateWithKeys**: Integer value = 2
- [out] **SQL query**: String containing the query in SQL
- [out] **escape processing**: Boolean for knowing if the backend driver should escape processing before sending to the database
- [out] **timeout in seconds**: Integer representing the number of seconds for this request time out (0 = no timeout)

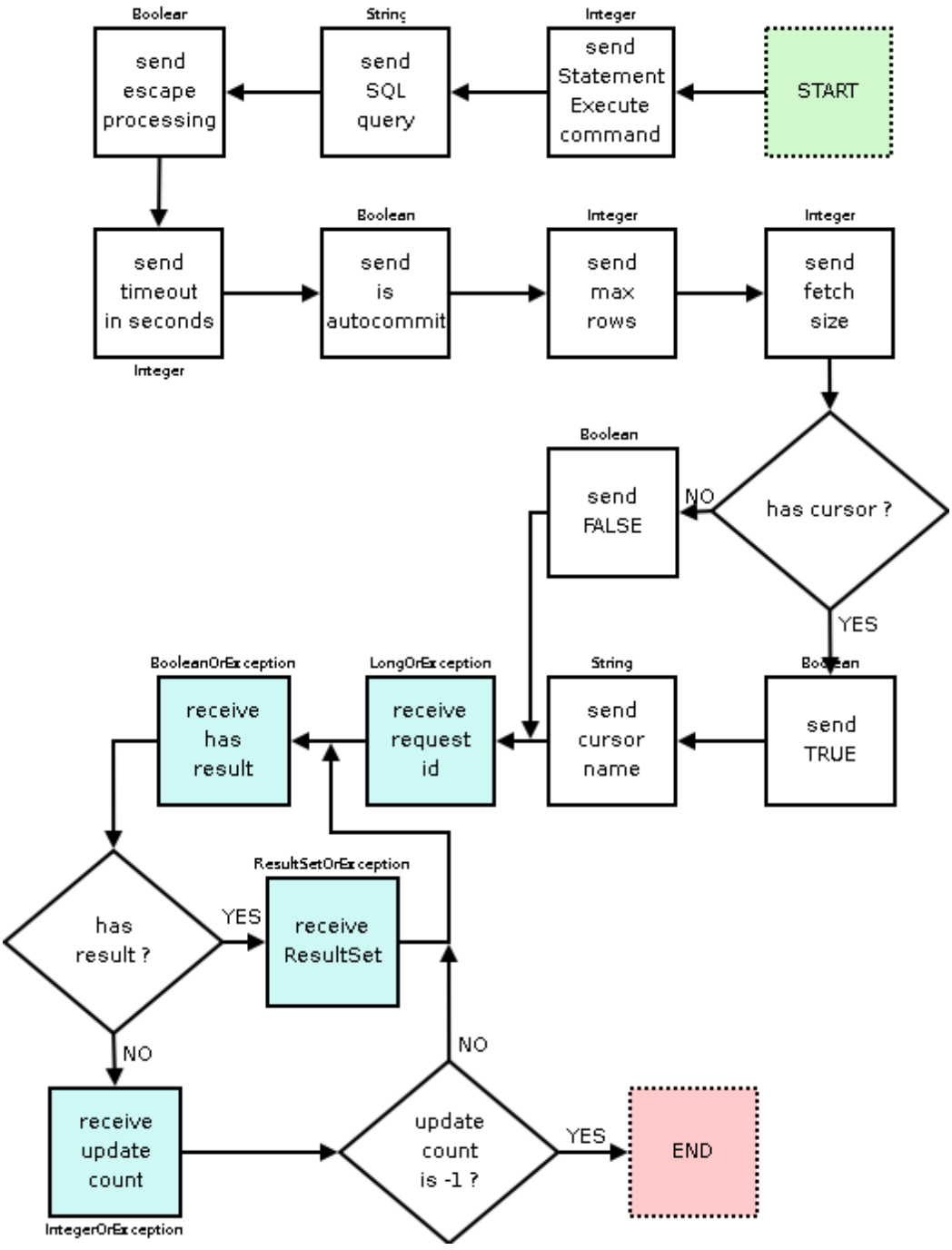
- [out] **is autocommit**: Boolean telling whether or not this request has been sent in autocommit mode
- [in] **request id**: LongOrException. Request identifier used for failover purposes
- [in] **update count**: IntegerOrException the number of rows affected by the write request
- [in] **auto generated keys**: ResultSetOrException. Possible exceptions types are `ControllerCoreException, NoMoreBackendException, IOException, BackendDriverException, NotImplementedException`

4.2.4 STATEMENT EXECUTE

Execute a request and returns an arbitrary number of update counts and/or ResultSets

4.2.4.1 Diagram

2000 Powell Street, Suite 1090, Emeryville, CA 94608
Tel. 408-998-7878 Fax 408-668-1009



4.2.4.2 Parameters description

- [out] **StatementExecute**: Integer value = 6

- [out] **SQL query**: String containing the query in SQL
- [out] **escape processing**: Boolean for knowing if the backend driver should escape processing before sending to the database
- [out] **timeout in seconds**: Integer representing the number of seconds for this request time out (0 = no timeout)
- [out] **is autocommit**: Boolean telling whether or not this request has been sent in autocommit mode
- [out] **max rows**: Integer representing the maximum number of rows the result set can contain
- [out] **fetch size**: Integer representing the maximum number of results to be retrieved at a time
- [out] **has cursor**: Boolean telling whether or not we have a cursor name

  (only if **has cursor** is `true`)

  - [out] **cursor name**: Boolean the name of the cursor to be used
- [in] **request id**: LongOrException. Request identifier used for failover purposes

Repeat:

- [in] **has result**: BooleanOrException `true` if the data to follow is a **ResultSet**, `false` for an **update count**

  (only if **has result** is `true`)

  - [in] **result**: ResultSetOrException. Possible exceptions types are `ControllerCoreException, BackendDriverException`.

  (only if **has result** is `false`)

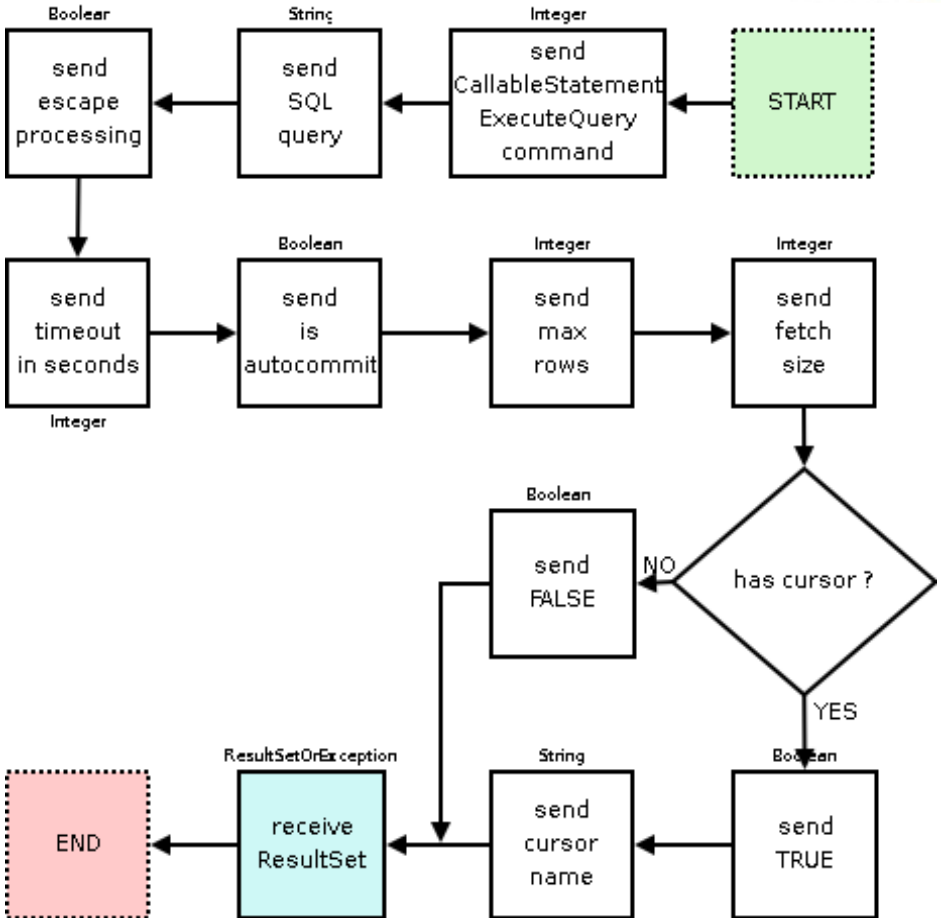
  - [in] **update count**: IntegerOrException number of rows affected by the update or `-1` for end-of-list

*until **has result** is `false` and **update count** is `-1`*

4.2.5 CALLABLE STATEMENT EXECUTE QUERY

Calls a stored procedure and receives the data read (ResultSet).

4.2.5.1 Diagram



4.2.5.2 Parameters description

- [out] **CallableStatementExecuteQuery**: Integer value = 3
- [out] **SQL query**: String containing the query in SQL
- [out] **escape processing**: Boolean for knowing if the backend driver should escape processing before sending to the database
- [out] **timeout in seconds**: Integer representing the number of seconds for this request time out (0 = no timeout)
- [out] **is autocommit**: Boolean telling whether or not this request has been sent in autocommit mode
- [out] **max rows**: Integer representing the maximum number of rows the result set can contain
- [out] **fetch size**: Integer representing the maximum number of results to be retrieved at a time
- [out] **has cursor**: Boolean telling whether or not we have a cursor name

(only if **has cursor** is `true`)

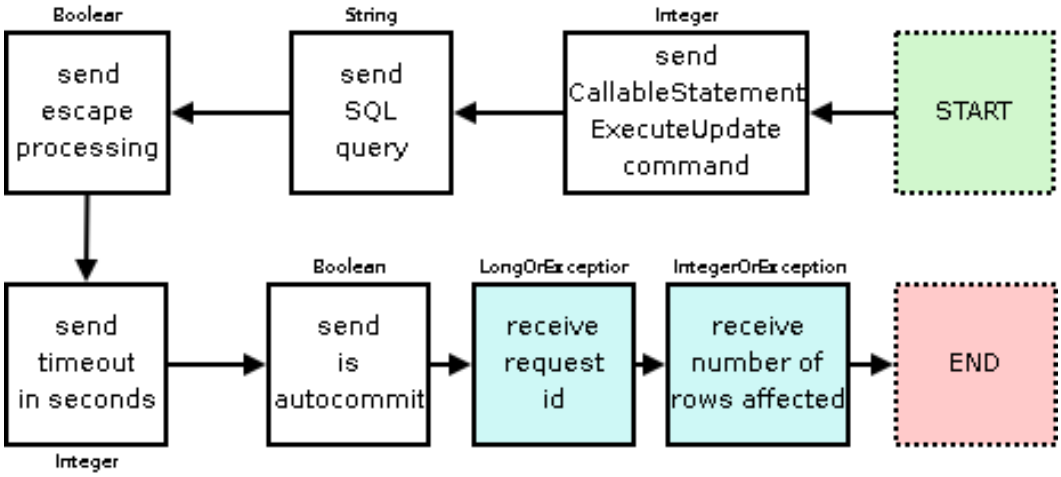
- [out] **cursor name**: Boolean the name of the cursor to be used

- [in] **result**: ResultSetOrException. Possible exceptions types are `ControllerCoreException, BackendDriverException`.

4.2.6 CALLABLE STATEMENT EXECUTE UPDATE

Calls a stored procedure and receives the number of rows affected by the write query.

4.2.6.1 Diagram



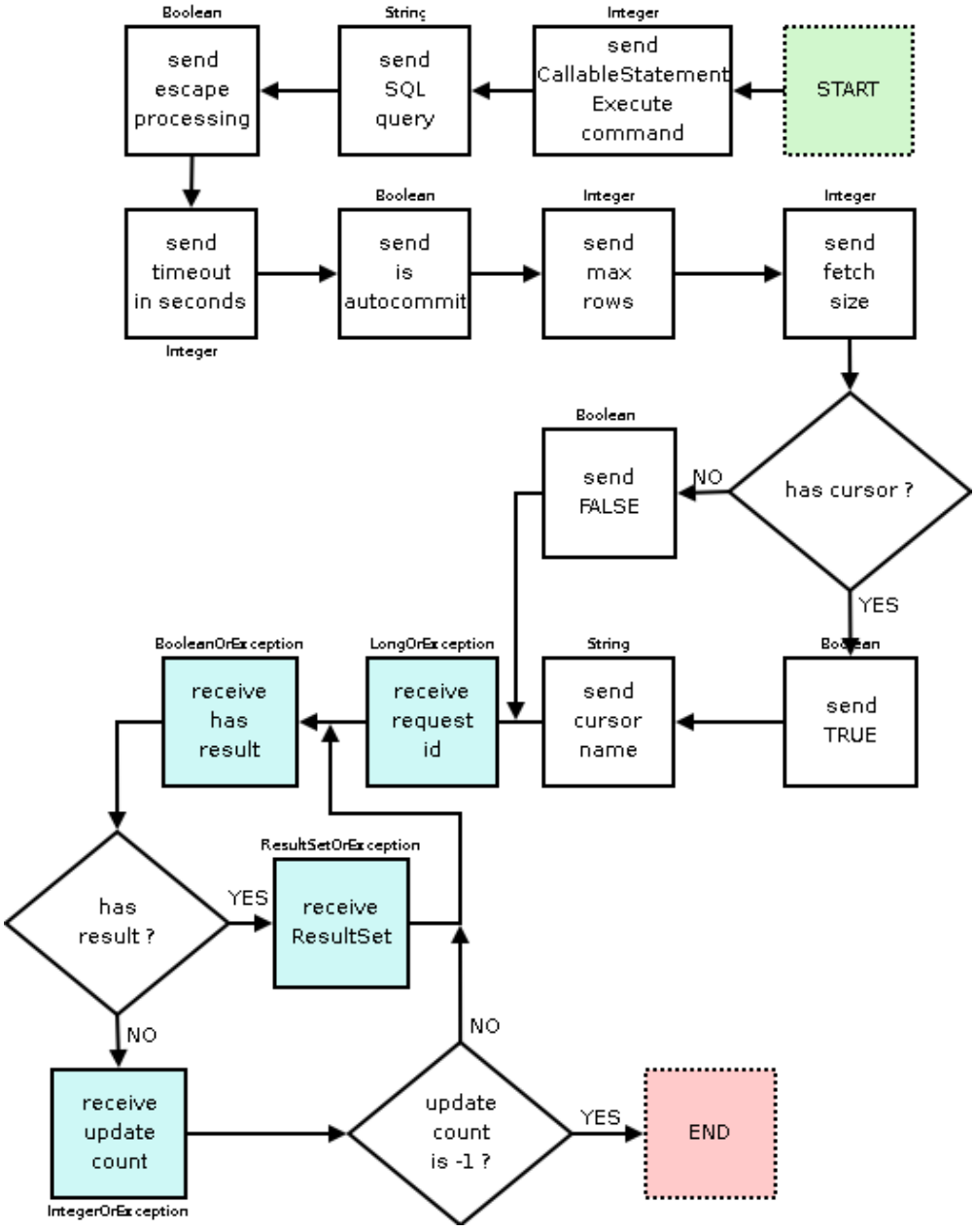
4.2.6.2 Parameters description

- [out] **CallableStatementExecuteUpdate**: Integer value = 4
- [out] **SQL query**: String containing the query in SQL
- [out] **escape processing**: Boolean for knowing if the backend driver should escape processing before sending to the database
- [out] **timeout in seconds**: Integer representing the number of seconds for this request time out (0 = no timeout)
- [out] **is autocommit**: Boolean telling whether or not this request has been sent in autocommit mode
- [in] **request id**: LongOrException. Request identifier used for failover purposes
- [in] **number of rows affected**: IntegerOrException number of rows affected by the update

4.2.7 CALLABLE STATEMENT EXECUTE

Calls a stored procedure using CallableStatement.execute(). This returns an arbitrary number of update counts and/or ResultSets.

4.2.7.1 Diagram



4.2.7.2 Parameters description

- [out] **CallableStatementExecute**: Integer value = 5

- [out] **SQL query**: String containing the query in SQL
- [out] **escape processing**: Boolean for knowing if the backend driver should escape processing before sending to the database
- [out] **timeout in seconds**: Integer representing the number of seconds for this request time out (0 = no timeout)
- [out] **is autocommit**: Boolean telling whether or not this request has been sent in autocommit mode
- [out] **max rows**: Integer representing the maximum number of rows the result set can contain
- [out] **fetch size**: Integer representing the maximum number of results to be retrieved at a time
- [out] **has cursor**: Boolean telling whether or not we have a cursor name

  (only if **has cursor** is `true`)

  - [out] **cursor name**: Boolean the name of the cursor to be used
- [in] **request id**: LongOrException. Request identifier used for failover purposes

Repeat:

- [in] **has result**: BooleanOrException `true` if the data to follow is a **ResultSet**, `false` for an **update count**

  (only if **has result** is `true`)

  - [in] **result**: ResultSetOrException. Possible exceptions types are `ControllerCoreException, BackendDriverException`.

  (only if **has result** is `false`)

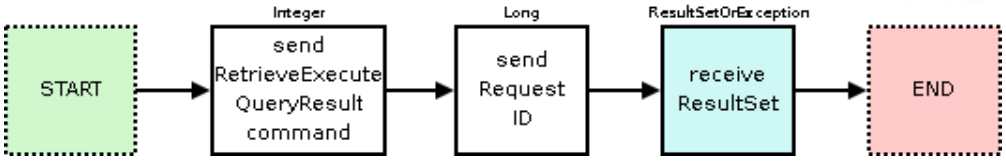
  - [in] **update count**: IntegerOrException number of rows affected by the update or `-1` for end-of-list

*until **has result** is `false` and **update count** is `-1`*

4.2.8 RETRIEVE EXECUTE QUERY RESULT

Tries to retrieve the result of a previouly executed query using executeQuery().

4.2.8.1 Diagram



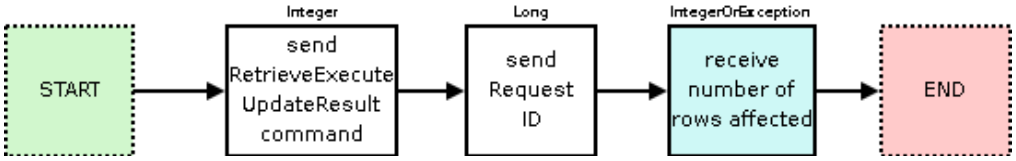
4.2.8.2 Parameters description

- [out] **RetrieveExecuteQueryResult**: Integer value = 10
- [out] **request ID**: Long request unique identifier
- [in] **result**: ResultSetOrException. Possible exceptions types are `ControllerCoreException, NoMoreBackendException, IOException, BackendDriverException, NotImplementedException`

4.2.9 RETRIEVE EXECUTE UPDATE RESULT

Check if the given query has already been executed or not on the controller we are currently connected to. If so, retrieve the result of this update.

4.2.9.1 Diagram



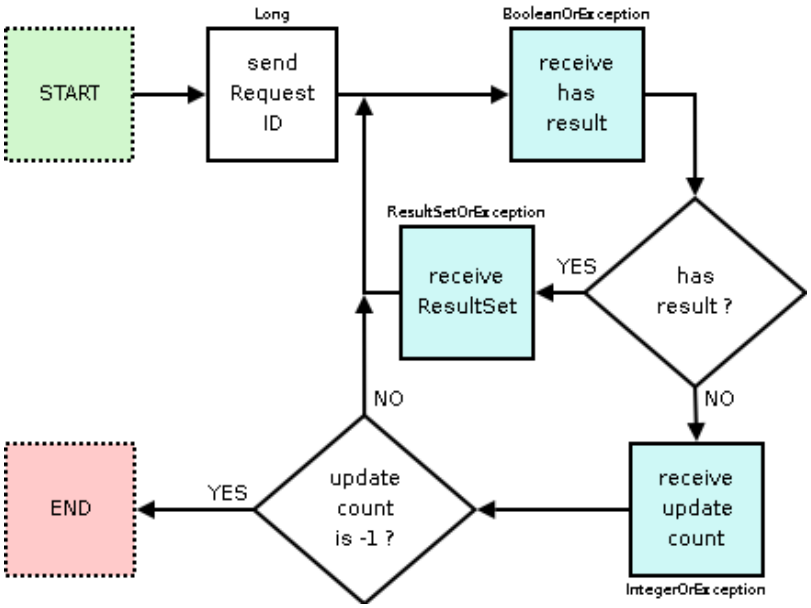
4.2.9.2 Parameters description

- [out] **RetrieveExecuteUpdateResult**: Integer value = 11
- [out] **request ID**: Long request unique identifier
- [in] **number of rows affected**: IntegerOrException the number of rows affected by the write request

4.2.10 RETRIEVE EXECUTE RESULT

Checks if the given query has already been executed or not on the controller we are currently connected to. If so, retreive the result of this Execute query.

4.2.10.1 Diagram



4.2.10.2 Parameters description

- [out] **RetrieveExecuteResult**: Integer value = 12
- [out] **request ID**: Long request unique identifier

Repeat:

- [in] **has result**: BooleanOrException `true` if the data to follow is a **ResultSet**, `false` for an **update count**

  (only if **has result** is `true`)

  - [in] **result**: ResultSetOrException. Possible exceptions types are `ControllerCoreException, BackendDriverException`.

  (only if **has result** is `false`)

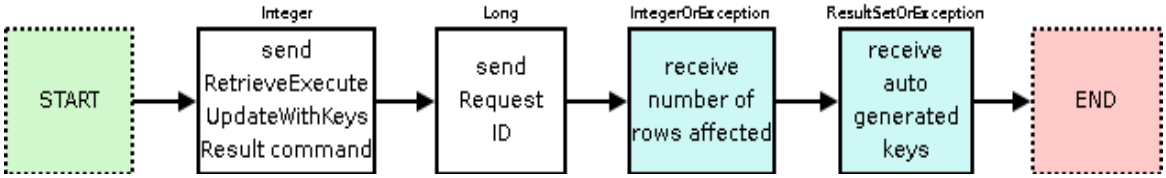
  - [in] **update count**: IntegerOrException number of rows affected by the update or `-1` for end-of-list

*until **has result** is `false` and **update count** is `-1`*

4.2.11 RETRIEVE EXECUTE UPDATE WITH KEYS RESULT

Check if the given query already executed or not on the controller we are currently connected to. If so, retreive the result of this ExecuteUpdateWithKeys query.

4.2.11.1 Diagram



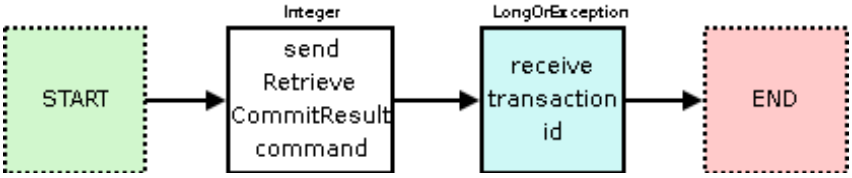
4.2.11.2 Parameters description

- [out] **RetrieveExecuteUpdateWithKeysResult**: Integer value = 13
- [out] **request ID**: Long request unique identifier
- [in] **number of rows affected**: IntegerOrException the number of rows affected by the write request
- [in] **auto generated keys**: ResultSetOrException. Possible exceptions types are `ControllerCoreException, NoMoreBackendException, IOException, BackendDriverException, NotImplementedException`

4.2.12 RETRIEVE COMMIT RESULT

Try to retrieve the result of a previouly executed commit. If the commit was not executed before, the controller will execute it and return the result which is the transaction id acknowledgement.

4.2.12.1 Diagram



4.2.12.2 Parameters description

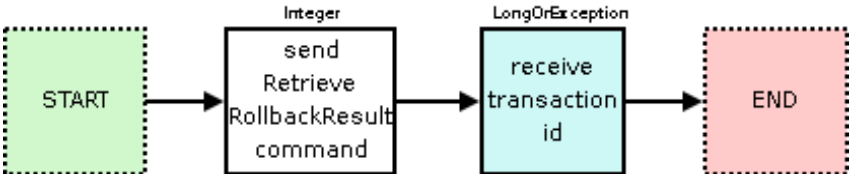
- [out] **RetrieveCommitResult**: Integer value = 14
- [in] **transactionId**: LongOrException id of commited transaction

2000 Powell Street, Suite 1090, Emeryville, CA 94608
Tel. 408-998-7878 Fax 408-668-1009

4.2.13 RETRIEVE ROLLBACK RESULT

Try to retrieve the result of a previouly executed rollback. If the rollback was not executed before, the controller will execute it and return the result which is the transaction id acknowledgement.

4.2.13.1 Diagram



4.2.13.2 Parameters description

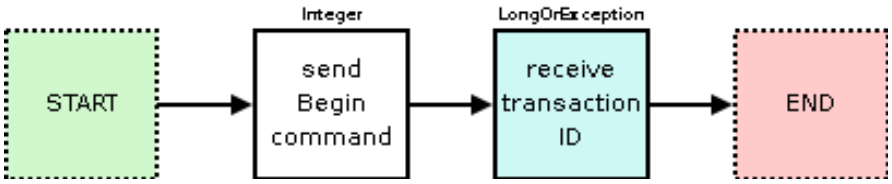
- [out] **RetrieveRollbackResult**: Integer value = 15
- [in] **transactionId**: LongOrException id of rollbacked transaction

4.3. TRANSACTION MANAGEMENT COMMANDS

4.3.1 BEGIN

Begins a new transaction and receives the corresponding transaction identifier. This method is called from the client when is autocommit mode is off.

4.3.1.1 Diagram



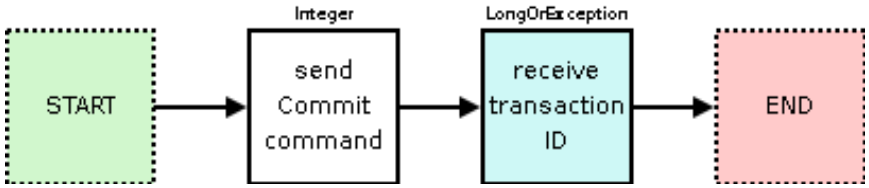
4.3.1.2 Parameters description

- [out] **Begin**: Integer value = 20
- [in] **transaction ID**: LongOrException the id for this transaction

4.3.2 COMMIT

Commits the current transaction.

4.3.2.1 Diagram



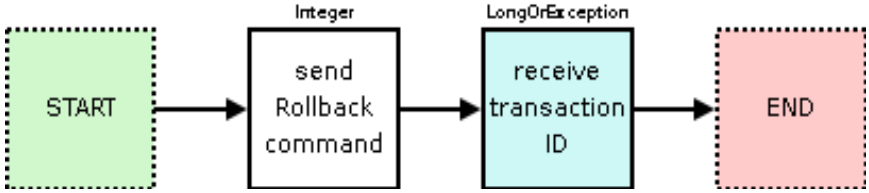
4.3.2.2 Parameters description

- [out] **Commit**: Integer value = 21
- [in] **transaction ID**: LongOrException id of the commited transaction

4.3.3 ROLLBACK

Rollbacks the current transaction.

4.3.3.1 Diagram



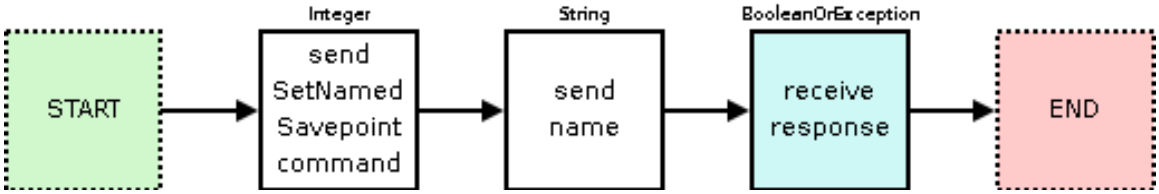
4.3.3.2 Parameters description

- [out] **Rollback**: Integer value = 22
- [in] **transaction ID**: LongOrException the id for this transaction

4.3.4 SET NAMED SAVEPOINT

Sets a named savepoint to a transaction given its id.

4.3.4.1 Diagram



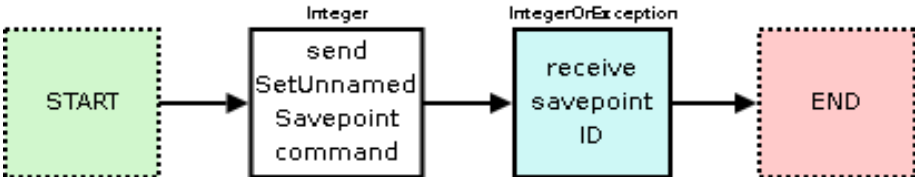
4.3.4.2 Parameters description

- [out] **SetNamedSavepoint**: Integer value = 23
- [out] **name**: String name of the savepoint
- [in] **response**: BooleanOrException the controller response. Always `true` if it is not an exception

4.3.5 SET UNNAMED SAVEPOINT

Sets a unnamed savepoint to a transaction given its id.

4.3.5.1 Diagram



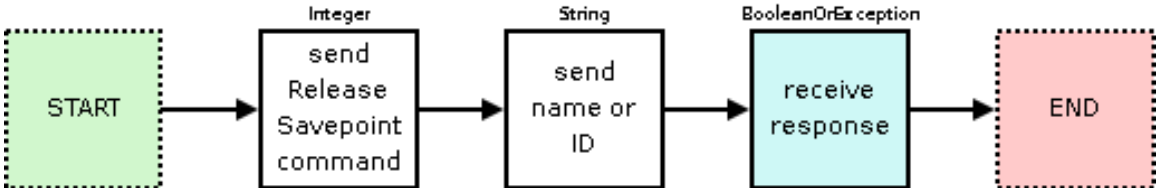
4.3.5.2 Parameters description

- [out] **SetUnnamedSavepoint**: Integer value = 24
- [in] **savepoint ID**: IntegerOrException the generated ID for this savepoint

4.3.6 RELEASE SAVEPOINT

Releases a savepoint from a transaction given its name or id

4.3.6.1 Diagram



4.3.6.2 Parameters description

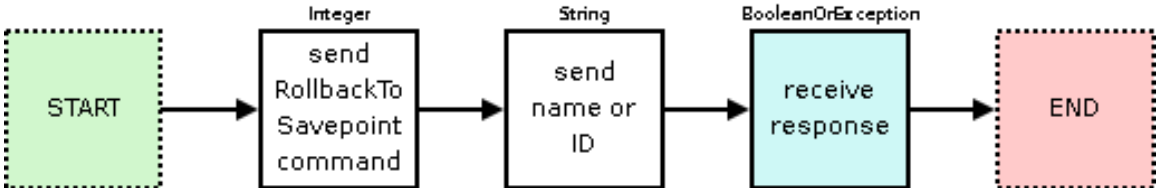
- [out] **ReleaseSavepoint**: Integer value = 25
- [out] **name or ID**: String name of the savepoint, or ID as a string if it is an unnamed savepoint

- [in] **response**: BooleanOrException the controller response. Always `true` if it is not an exception

4.3.7 ROLLBACK TO SAVEPOINT

Rollbacks the current transaction back to the given savepoint

4.3.7.1 Diagram



4.3.7.2 Parameters description

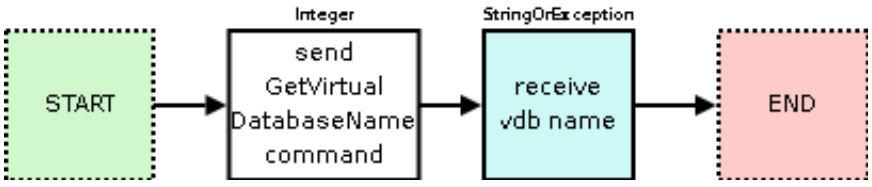
- [out] **RollbackToSavepoint**: Integer value = 26
- [out] **name or ID**: String name of the savepoint, or ID as a string if it is an unnamed savepoint
- [in] **response**: BooleanOrException the controller response. Always `true` if it is not an exception

4.4. METADATA FUNCTIONS

4.4.1 GET VIRTUAL DATABASE NAME

Gets the virtual database name to be used by the client.

4.4.1.1 Diagram



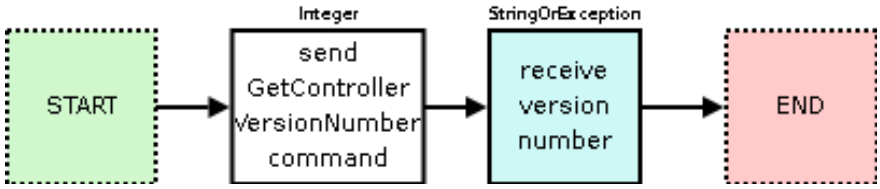
4.4.1.2 Parameters description

- [out] **GetVirtualDatabaseName**: Integer value = 50
- [in] **vdb name**: StringOrException virtual database name

4.4.2 GET CONTROLLER VERSION NUMBER

Gets the controller version number.

4.4.2.1 Diagram



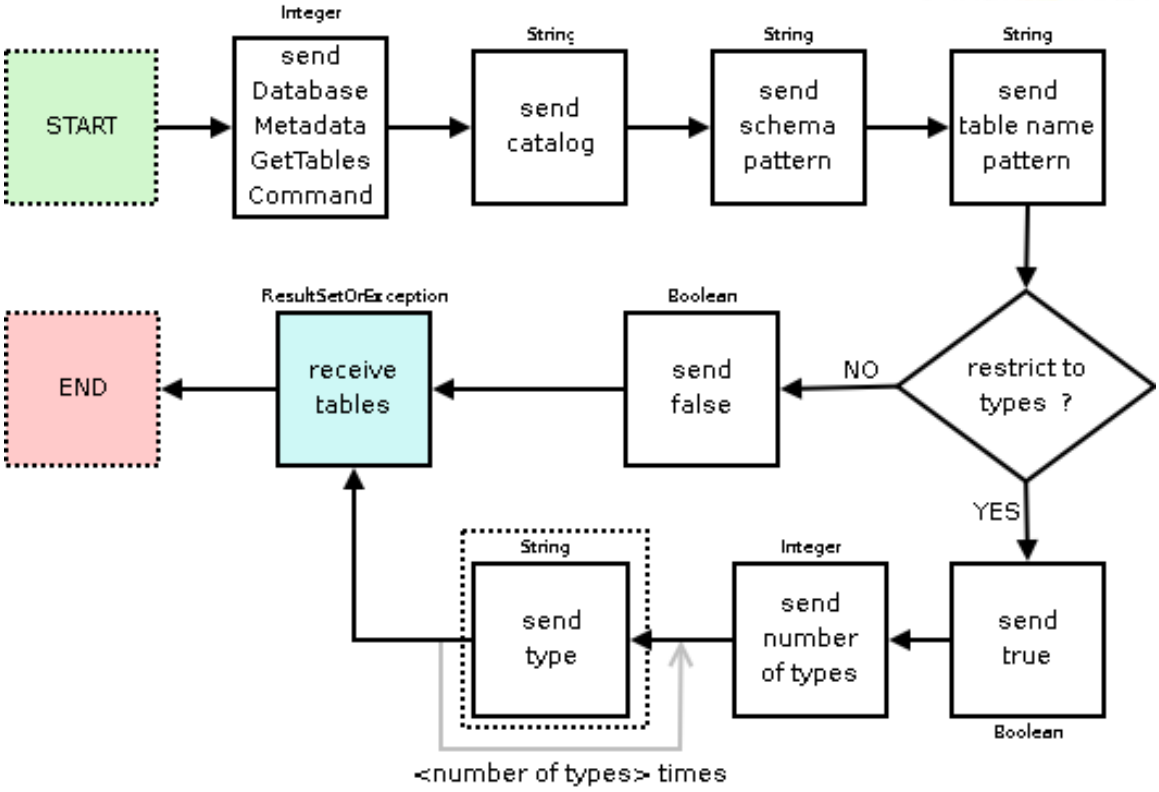
4.4.2.2 Parameters description

- [out] **GetControllerVersionNumber**: Integer value = 51
- [in] **version number**: StringOrException controller version number as a string

4.4.3 DATABASE METADATA GET TABLES

Retrieves a description of the tables available in the given catalog.

4.4.3.1 Diagram



4.4.3.2 Parameters description

- [out] **DatabaseMetaDataGetTables**: Integer value = 53
- [out] **catalog**: String name of the catalog (database)
- [out] **schema pattern**: String a schema name pattern
- [out] **table name pattern**: String a table name pattern
- [out] **restrict to types**: Boolean `false` if we want all types of tables. `true` if only the following types are asked

  (only if **types not null** is `true`)

  - [out] **number of types**: Integer number of types to follow

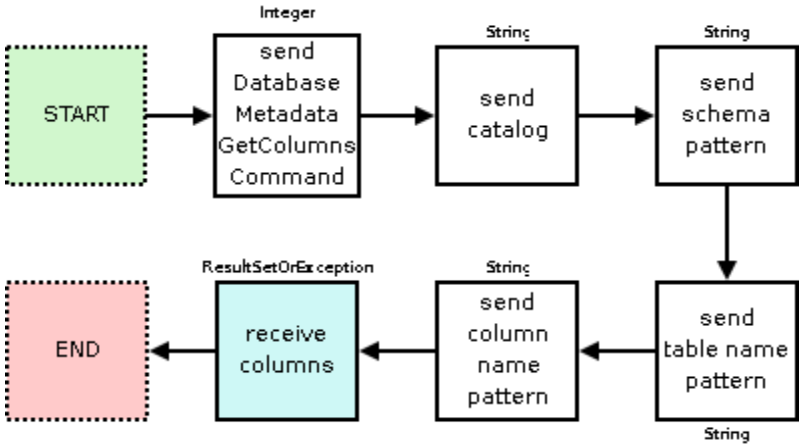
    number of types times:

    - **type**: String table type to include
- [in] **tables**: ResultSetOrException. Database tables as a ResultSet. Each row is a table description. Exception of type `SQLException` if a database access error occurs or if no database schema could be found (which probably means that no backend is enabled on the controller)

4.4.4 DATABASE METADATA GET COLUMNS

Retrieves a description of table columns available in the specified catalog.

4.4.4.1 Diagram



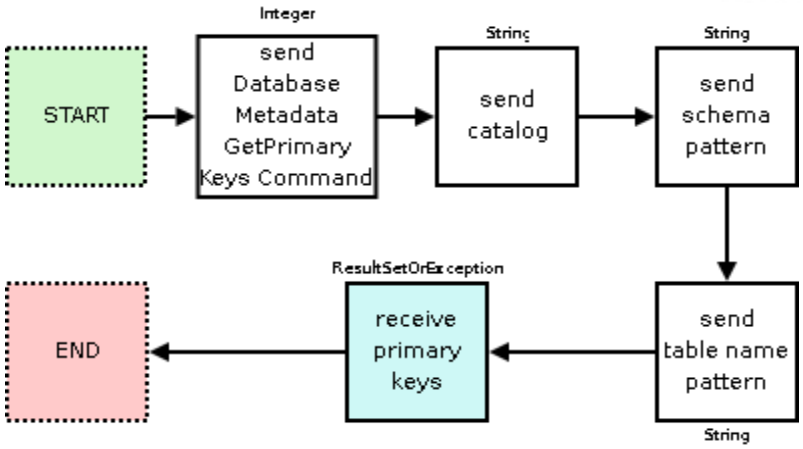
4.4.4.2 Parameters description

- [out] **DatabaseMetaDataGetColumns**: Integer value = 53
- [out] **catalog**: String name of the catalog (database)
- [out] **schema pattern**: String a schema name pattern
- [out] **table name pattern**: String a table name pattern
- [out] **column name pattern**: String a column name pattern
- [in] **columns**: ResultSetOrException. Table columns as a ResultSet. Each row is a column description. Exception of type `SQLException` if a database access error occurs

4.4.5 DATABASE METADATA GET PRIMARY KEYS

Gets a description of a table's primary key columns.

4.4.5.1 Diagram



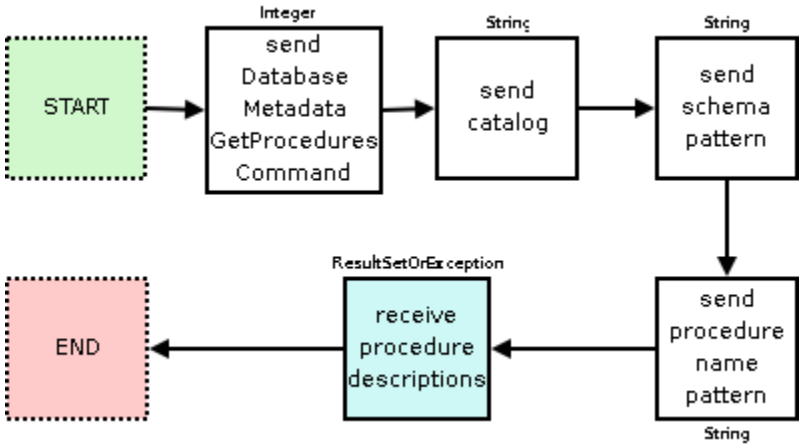
4.4.5.2 Parameters description

- [out] **DatabaseMetaDataGetPrimaryKeys**: Integer value = 54
- [out] **catalog**: String name of the catalog (database)
- [out] **schema pattern**: String a schema name pattern
- [out] **table name pattern**: String a table name pattern
- [in] **primary keys**: ResultSetOrException. Table primary keys as a ResultSet. Each row is a primary key column description. Exception of type `SQLException` if a database access error occurs

4.4.6 DATABASE METADATA GET PROCEDURES

Retrieves a description of the stored procedures available in the given catalog.

4.4.6.1 Diagram



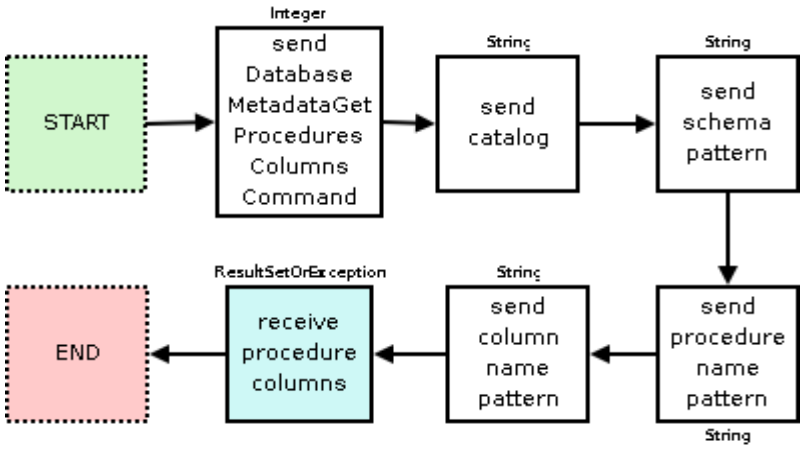
4.4.6.2 Parameters description

- [out] **DatabaseMetaDataGetProcedures**: Integer value = 55
- [out] **catalog**: String name of the catalog (database)
- [out] **schema pattern**: String a schema name pattern
- [out] **procedure name pattern**: String a procedure name pattern
- [in] **procedure descriptions**: ResultSetOrException. Table stored procedures as a ResultSet. Each row is a procedure description. Exception of type `SQLException` if a database access error occurs

4.4.7 DATABASE METADATA GET PROCEDURE COLUMNS

Retrieves a description of the given catalog stored procedure parameter and result columns.

4.4.7.1 Diagram



4.4.7.2 Parameters description

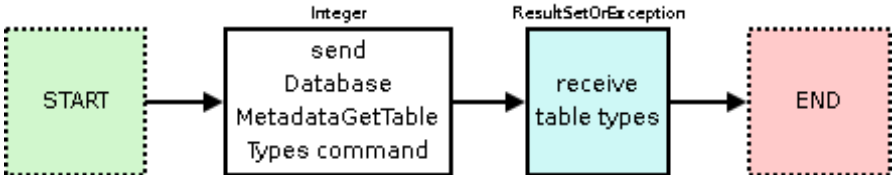
- [out] **DatabaseMetaDataGetProcedureColumns**: Integer value = 56
- [out] **catalog**: String name of the catalog (database)
- [out] **schema pattern**: String a schema name pattern
- [out] **procedure name pattern**: String a procedure name pattern
- [out] **column name pattern**: String a column name pattern

- [in] **procedure columns**: ResultSetOrException. Table stored procedure columns as a ResultSet. Each row describes a stored procedure parameter or column. Exception of type `SQLException` if if a database access error occurs

4.4.8 DATABASE METADATA GET TABLE TYPES

Gets the table types available in this database. The results are ordered by table type.

4.4.8.1 Diagram



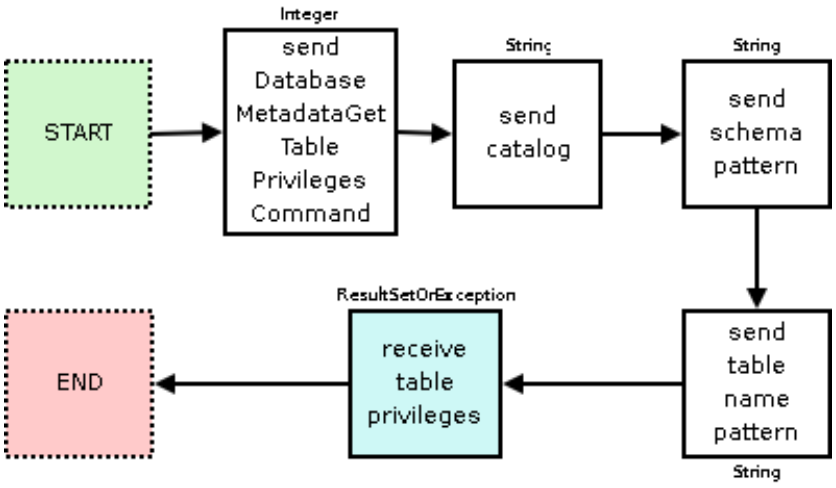
4.4.8.2 Parameters description

- [out] **DatabaseMetaDataGetTableTypes**: Integer value = 58
- [in] **table types**: ResultSetOrException. Table types as a ResultSet. Each row in the ResultSet will have a single String column that is a catalog name. Exception of type `SQLException` if a database error occured

4.4.9 DATABASE METADATA GET TABLE PRIVILEGES

Gets a description of the access rights for each table available in a catalog

4.4.9.1 Diagram



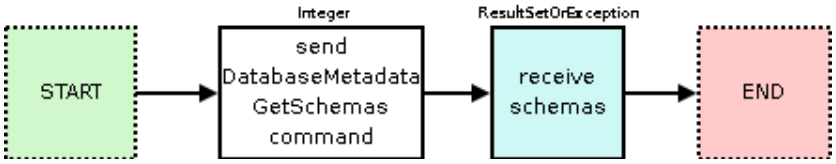
4.4.9.2 Parameters description

- [out] **DatabaseMetaDataGetTablePrivileges**: Integer value = 59
- [out] **catalog**: String name of the catalog (database)
- [out] **schema pattern**: String a schema name pattern
- [out] **table name pattern**: String a table name pattern
- [in] **table privileges**: ResultSetOrException. Table priviledges as a ResultSet. Each row is a table privilege description. Exception of type `SQLException` if a database error occured

4.4.10 DATABASE METADATA GET SCHEMAS

Retrieves the schema names available in this database.

4.4.10.1 Diagram



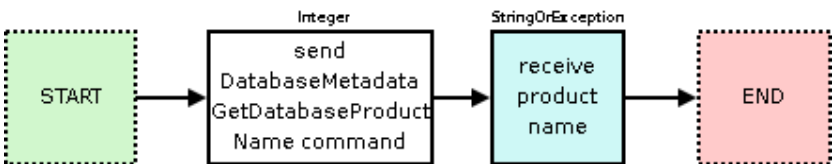
4.4.10.2 Parameters description

- [out] **DatabaseMetaDataGetSchemas**: Integer value = 60
- [in] **schemas**: ResultSetOrException. Schemas as a ResultSet. Each row is a schema decription. Exception of type `SQLException` if a database error occured

4.4.11 DATABASE METADATA GET DATABASE PRODUCT NAME

Retrieves a comma separated list of database engine names connected to the controller.

4.4.11.1 Diagram



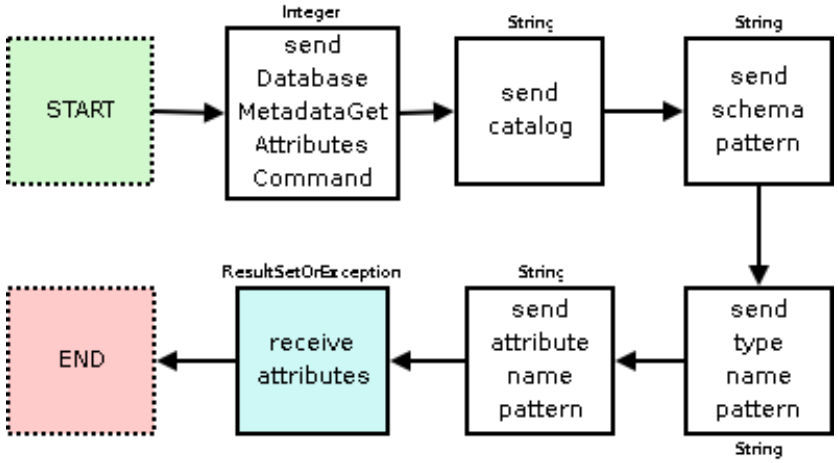
4.4.11.2 Parameters description

- [out] **DatabaseMetaDataGetDatabaseProductName**: Integer value = 61
- [in] **product name**: StringOrException comma separated list of database product names or exception if a database error occurs

4.4.12 DATABASE METADATA GET ATTRIBUTES

Retrieves a description of the given attribute of the given type for a user-defined type (UDT) that is available in the given schema and catalog.

4.4.12.1 Diagram



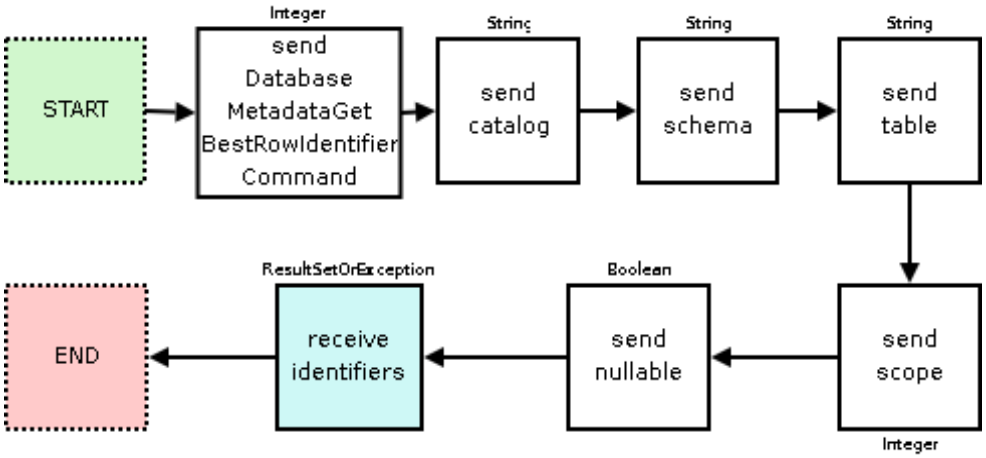
4.4.12.2 Parameters description

- [out] **DatabaseMetaDataGetAttributes**: Integer value = 62
- [out] **catalog**: String name of the catalog (database)
- [out] **schema pattern**: String a schema name pattern
- [out] **type name pattern**: String a type name pattern
- [out] **attribute name pattern**: String an attribute name pattern
- [in] **attributes**: ResultSetOrException. Attributes as a ResultSet. Each row is an attribute description. Exception of type `SQLException` if a database access error occurs

4.4.13 DATABASE METADATA GET BEST ROW IDENTIFIER

Retrieves a description of a table's optimal set of columns that uniquely identifies a row.

4.4.13.1 Diagram



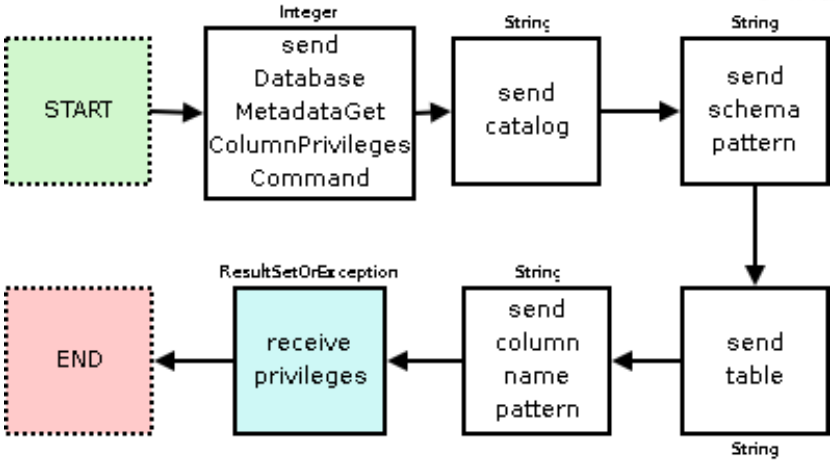
4.4.13.2 Parameters description

- [out] **DatabaseMetaDataGetBestRowIdentifier**: Integer value = 63
- [out] **catalog**: String name of the catalog (database)
- [out] **schema**: String a schema name
- [out] **table**: String a table name
- [out] **scope**: Integer scope the scope of interest
- [out] **nullable**: Boolean include columns that are nullable ?
- [in] **identifiers**: ResultSetOrException. Identifiers as a ResultSet. Each row is a column description. Exception of type `SQLException` if a database access error occurs

4.4.14 DATABASE METADATA GET COLUMN PRIVILEGES

Retrieves a description of the access rights for a table's columns.

4.4.14.1 Diagram



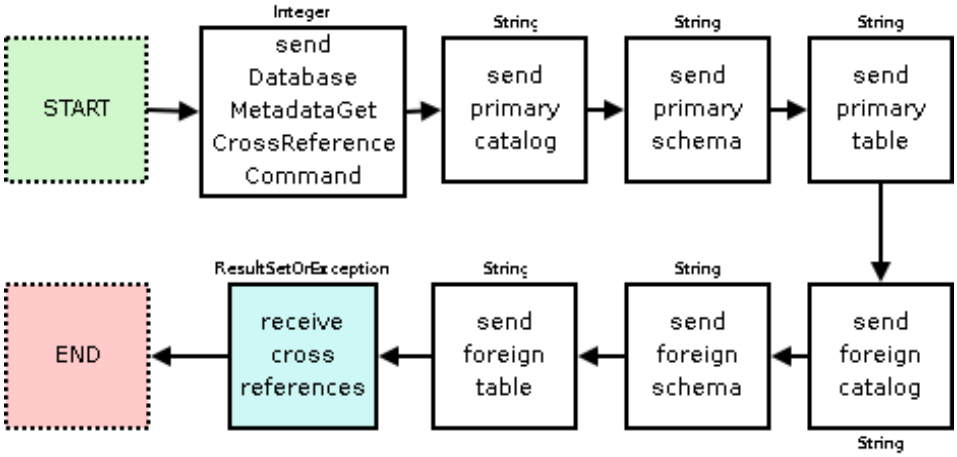
4.4.14.2 Parameters description

- [out] **DatabaseMetaDataGetColumnPrivileges**: Integer value = 64
- [out] **catalog**: String name of the catalog (database)
- [out] **schema pattern**: String a schema name pattern
- [out] **table**: String a table name
- [out] **column name pattern**: String a column name pattern
- [in] **privileges**: ResultSetOrException. Privileges as a ResultSet. Each row is a column privilege description. Exception of type `SQLException` if a database access error occurs

4.4.15 DATABASE METADATA GET CROSSREFERENCE

Retrieves a description of the foreign key columns in the given foreign key table that reference the primary key columns of the given primary key table (describe how one table imports another's key).

4.4.15.1 Diagram



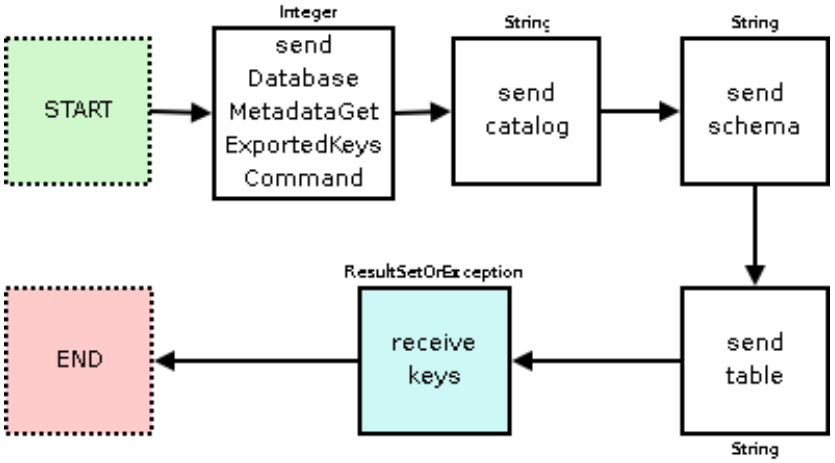
4.4.15.2 Parameters description

- [out] **DatabaseMetaDataGetCrossReference**: Integer value = 65
- [out] **primary catalog**: String a name of catalog (database)
- [out] **primary schema**: String a schema name
- [out] **primary table**: String name of the table that exports the key
- [out] **foreign catalog**: String a name of catalog (database)
- [out] **foreign schema**: String a schema name
- [out] **foreign table**: String name of the table that imports the key
- [in] **cross references**: ResultSetOrException. Cross references as a ResultSet. Each row is a foreign key column description. Exception of type `SQLException` if a database access error occurs

4.4.16 DATABASE METADATA GET EXPORTED KEYS

Retrieves a description of the foreign key columns that reference the given table's primary key columns (the foreign keys exported by a table).

4.4.16.1 Diagram



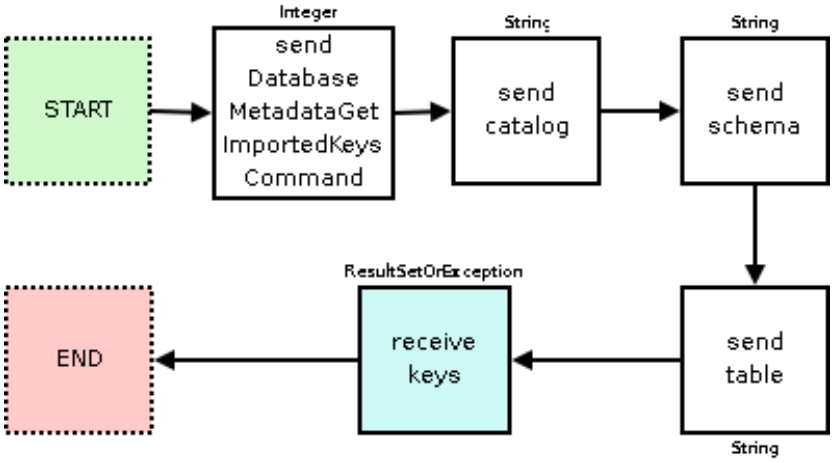
4.4.16.2 Parameters description

- [out] **DatabaseMetaDataGetExportedKeys**: Integer value = 66
- [out] **catalog**: String a name of catalog (database)
- [out] **schema**: String a schema name
- [out] **table**: String name of the table that exports the key
- [in] **keys**: ResultSetOrException. Exported keys as a ResultSet. Each row is a foreign key column description. Exception of type `SQLException` if a database access error occurs

4.4.17 DATABASE METADATA GET IMPORTED KEYS

Retrieves a description of the primary key columns that are referenced by a table's foreign key columns (the primary keys imported by a table).

4.4.17.1 Diagram



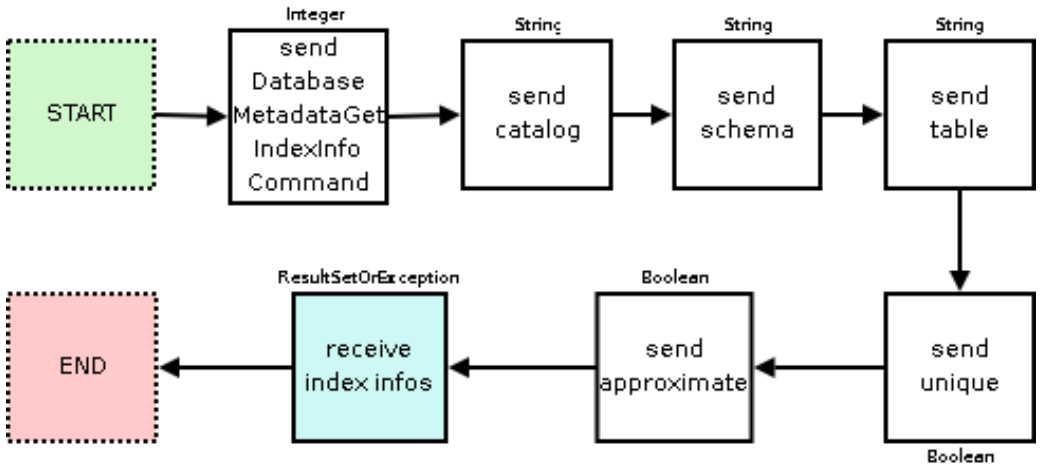
4.4.17.2 Parameters description

- [out] **DatabaseMetaDataGetImportedKeys**: Integer value = 67
- [out] **catalog**: String a name of catalog (database)
- [out] **schema**: String a schema name
- [out] **table**: String a table name
- [in] **keys**: ResultSetOrException. Imported keys as a ResultSet. Each row is a primary key column description. Exception of type `SQLException` if a database access error occurs

4.4.18 DATABASE METADATA GET INDEX INFO

Retrieves a description of the given table's indices and statistics.

4.4.18.1 Diagram



4.4.18.2 Parameters description

- [out] **DatabaseMetaDataGetIndexInfo**: Integer value = 68
- [out] **catalog**: String name of the catalog (database)
- [out] **schema**: String a schema name
- [out] **table**: String a table name
- [out] **unique**: Boolean when true, return only indices for unique values; when false, return indices regardless of whether unique or not

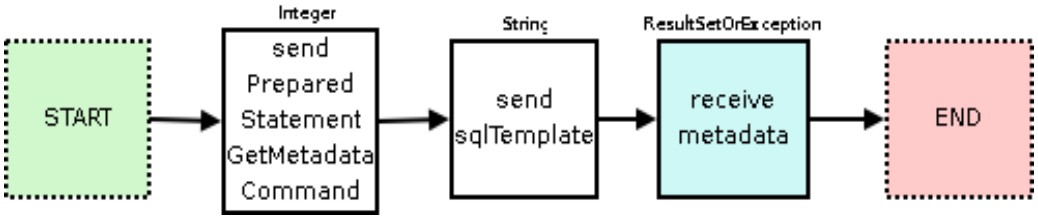
- [out] **approximate**: Boolean when true, result is allowed to reflect approximate or out of data values; when false, results are requested to be accurate

- [in] **index infos**: ResultSetOrException. Index infos as a ResultSet. Each row is an index column description. Exception of type `SQLException` if a database access error occurs

4.4.19 PREPARED STATEMENT GET METADATA

Retrieves the metadata of the given prepared statement.

4.4.19.1 Diagram



4.4.19.2 Parameters description

- [out] **PreparedStatementGetMetaData**: Integer value = 81

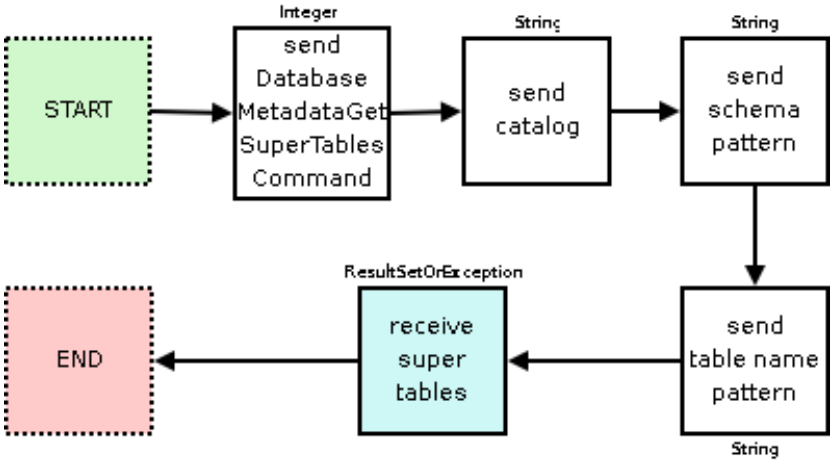
- [out] **sqlTemplate**: String name of the catalog (database)

- [in] **metadata**: ResultSetOrException ResultSet which metadata contains the given prepared statement metadata. Exception of type `SQLException` if a database access error occurs

4.4.20 DATABASE METADATA GET SUPER TABLES

Retrieves a description of the table hierarchies defined in a particular schema in this database.

4.4.20.1 Diagram



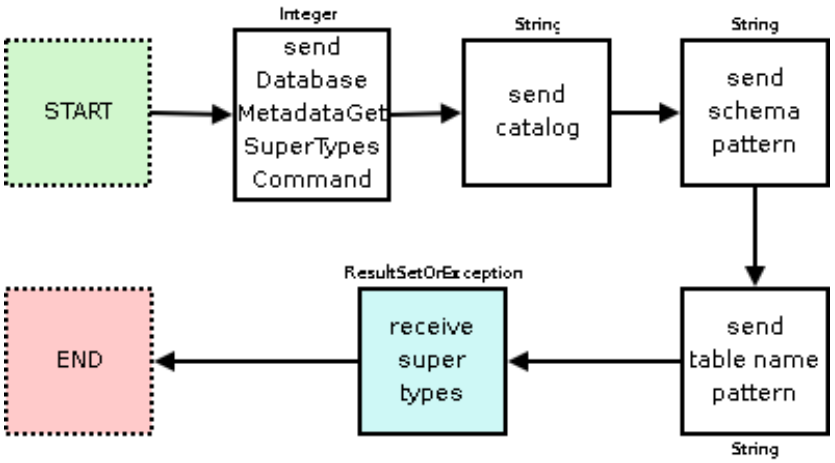
4.4.20.2 Parameters description

- [out] **DatabaseMetaDataGetSuperTables**: Integer value = 69
- [out] **catalog**: String name of the catalog (database)
- [out] **schema pattern**: String a schema name pattern
- [out] **table name pattern**: String a table name pattern
- [in] **super tables**: ResultSetOrException. Super tables as a ResultSet. Each row is a type description. Exception of type `SQLException` if a database access error occurs

4.4.21 DATABASE METADATA GET SUPER TYPES

Retrieves a description of the user-defined type (UDT) hierarchies defined in a particular schema in this database.

4.4.21.1 Diagram



2000 Powell Street, Suite 1090, Emeryville, CA 94608
Tel. 408-998-7878 Fax 408-668-1009

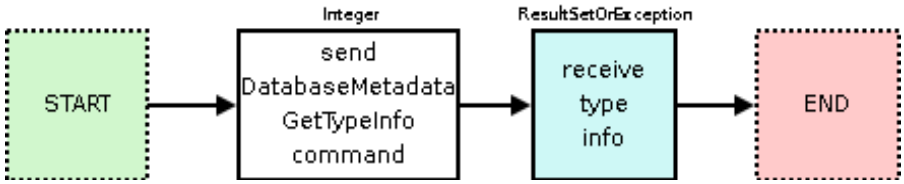
4.4.21.2 Parameters description

- [out] **DatabaseMetaDataGetSuperTypes**: Integer value = 70
- [out] **catalog**: String name of the catalog (database)
- [out] **schema pattern**: String a schema name pattern
- [out] **table name pattern**: String a table name pattern
- [in] **super types**: ResultSetOrException. Super types as a ResultSet. Each row gives information about the designated UDT. Exception of type `SQLException` if a database access error occurs

4.4.22 DATABASE METADATA GET TYPE INFO

Retrieves a description of all the standard SQL types supported by this database.

4.4.22.1 Diagram



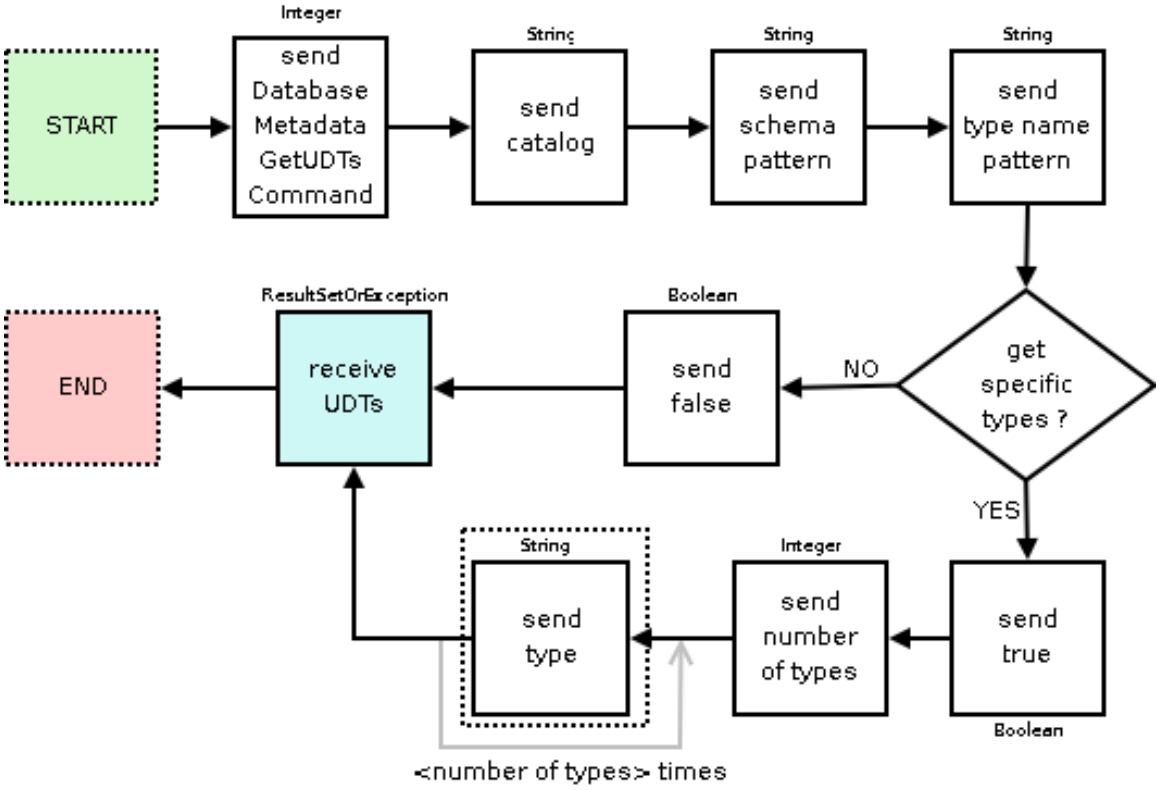
4.4.22.2 Parameters description

- [out] **DatabaseMetaDataGetTypeInfo**: Integer value = 71
- [in] **type info**: ResultSetOrException. Type infos as a ResultSet. Each row is an SQL type description. Exception of type `SQLException` if a database access error occurs

4.4.23 DATABASE METADATA GET UDTS

Retrieves a description of the user-defined types (UDTs) defined in a particular schema.

4.4.23.1 Diagram



4.4.23.2 Parameters description

- [out] **DatabaseMetaDataGetUDTs**: Integer value = 72
- [out] **catalog**: String name of the catalog (database)
- [out] **schema pattern**: String a schema name pattern
- [out] **type name pattern**: String a type name pattern
- [out] **get specific types**: Boolean if `false`, gets all types

  (only if get specific type = `true`)

  - [out] **number of types**: Integer the number of types to follow

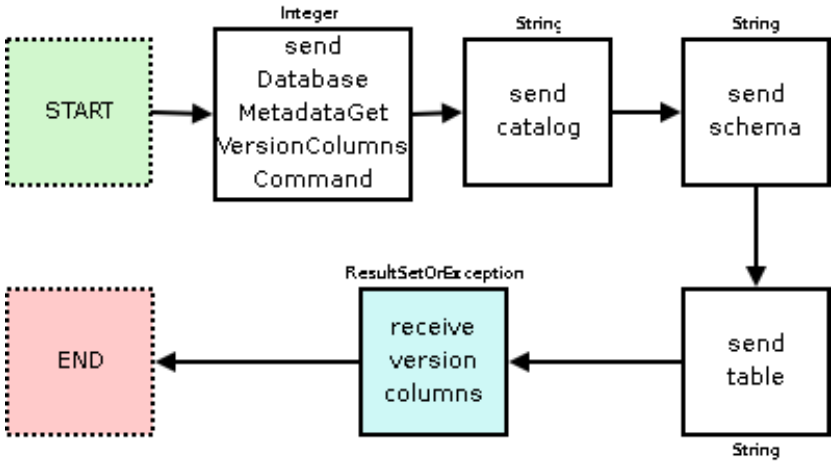
    number of types times:

    - [out] **type**: Integer user-defined type
- [in] **UDTs**: ResultSetOrException. UDTs as a ResultSet. Each row describes a UDT. Exception of type `SQLException` if a database access error occurs

4.4.24 DATABASE METADATA GET VERSION COLUMNS

Retrieves a description of a table's columns that are automatically updated when any value in a row is updated.

4.4.24.1 Diagram



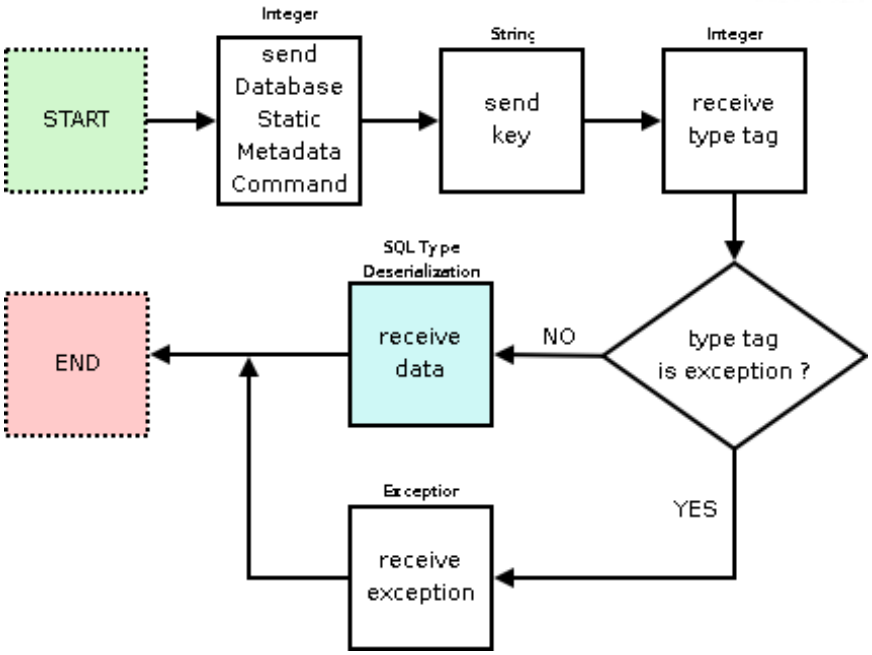
4.4.24.2 Parameters description

- [out] **DatabaseMetaDataGetVersionColumns**: Integer value = 73
- [out] **catalog**: String name of the catalog (database)
- [out] **schema**: String a schema name
- [out] **table**: String a table name
- [in] **version columns**: ResultSetOrException. Version columns as a ResultSet. Each row is a column description. Exception of type `SQLException` if a database access error occurs

4.4.25 DATABASE STATIC METADATA

Retrieve a static metadata from the controller.

4.4.25.1 Diagram



4.4.25.2 Parmeters description

- [out] **DatabaseStaticMetadata**: Integer value = 80
- [out] **key**: String hashkey of the metadata to be retrieved
- [in] **type tag**: Integer `EXCEPTION` (19) or `NOT_EXCEPTION` (18)

  (only if **type tag** = `EXCEPTION`)

  - [in] **exception**: Exception controller error of type `SQLException` if no metadata is available (probably means that no backend is enabled on that controller)

  (only if **type tag** = `NOT_EXCEPTION`)

  - [in] **data**: **SQL type deserialization** the data

5. Data types exchange

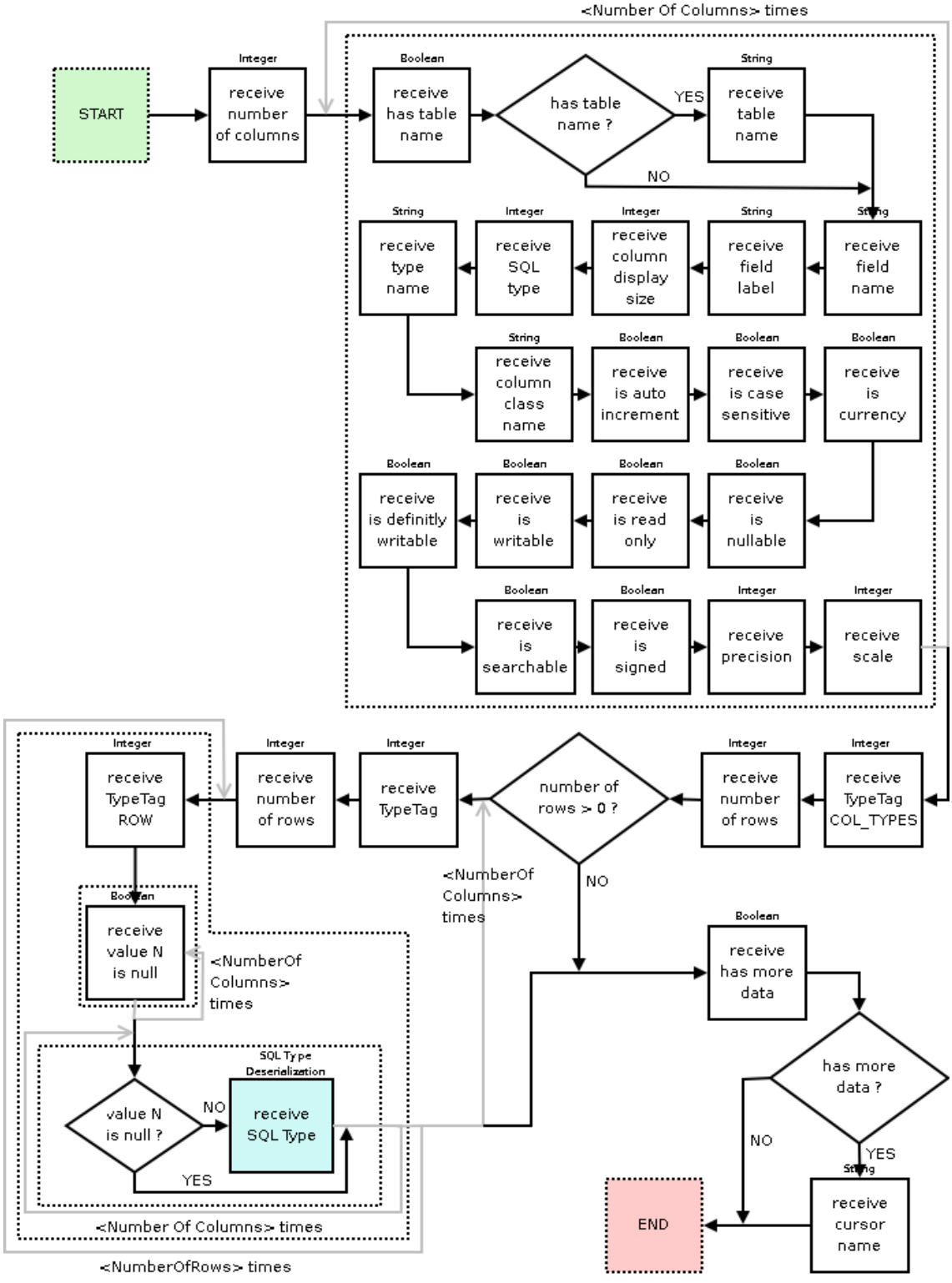
5.1. RESULT SETS

Some commands like the Read request imply the reception of a result set. The section below describes how this complex type is received by the driver. Furthermore, ResultSets contain data from the underlying database. This data is actually made of SQL datatypes that must be read a special way, detailed in the second section of this chapter.

5.1.1 RESULTSETS SERIALIZATION

ResultSets are not received directly by the driver. If an exception occurs during the request that should have sent the ResultSet, this exception will be sent **instead of** the ResultSet. ReceiveResultSetOrException shows how exceptions are handled in the protocol.

5.1.1.1 Diagram



5.1.1.2 Parameters Description

- [in] **Number of columns**: Integer the number of column that will be fetched this time

Number of columns times:

- [in] **has table name**: Boolean `false` if the table name is null

  (only if **has table name** is `true`)

  - [in] **table name**: String designated column's table name
- [in] **field name**: String designated column's name
- [in] **field label**: String designated column's suggested title for use in printouts and displays.
- [in] **columnDisplaySize**: Integer indicating the designated column's normal maximum width in characters
- [in] **sqlType**: Integer designated column's SQL type (from java.sql.Types)
- [in] **typeName**: String designated column's database-specific type name
- [in] **columnClassName**: String fully-qualified name of the Java class whose instances are manufactured if the method `ResultSet.getObject` is called to retrieve a value from the column
- [in] **isAutoIncrement**: Boolean whether the designated column is automatically numbered, thus read-only
- [in] **isCaseSensitive**: Boolean whether a column's case matters
- [in] **isCurrency**: Boolean whether the designated column is a cash value
- [in] **isNullable**: Integer nullability status of the given column; one of `columnNoNulls`, `columnNullable` or `columnNullableUnknown`
- [in] **isReadOnly**: Boolean whether the designated column is definitely not writable
- [in] **isWritable**: Boolean whether it is possible for a write on the designated column to succeed
- [in] **isDefinitelyWritable**: Boolean whether a write on the designated column will definitely succeed
- [in] **isSearchable**: Boolean whether the designated column can be used in a where clause
- [in] **isSigned**: Boolean whether values in the designated column are signed numbers
- [in] **precision**: Integer designated column's number of decimal digits

    - [in] **scale**: Integer designated column's number of digits to right of the decimal point

- [in] **TypeTag**: Integer value COL_TYPES 17
- [in] **numberOfRows**: Integer whether or not rows will follow. This number must be considered as a boolean, if equals to zero, it means that no rows have been retrieved

  only if **Number of rows** > 0

  numberOfColumns times:

  - [in] **TypeTag**: Integer possible values `STRING=0, BIGDECIMAL=1, BOOLEAN=2, INTEGER=3, LONG=4, FLOAT=5, DOUBLE=6, BYTE_ARRAY=7, SQL_DATE=8, SQL_TIME=9, SQL_TIMESTAMP=10, CLOB=11, BLOB=12, JAVA_SERIALIZABLE=13`
  - [in] **numberOfRows**: Integer number of rows to follow

  numberOfRows times:

  - [in] **TypeTag**: Integer value ROW = 18

    numberOfColumns times:

    - [in] **value N is null**: Boolean whether the values to follow are null. All of these are receive at a time and must be kept for the following

    numberOfColumns times:

    only if **value N is null =** `false`

    - [in] **SQL type deserialization** the data

- [in] **hasMoreData**: Boolean

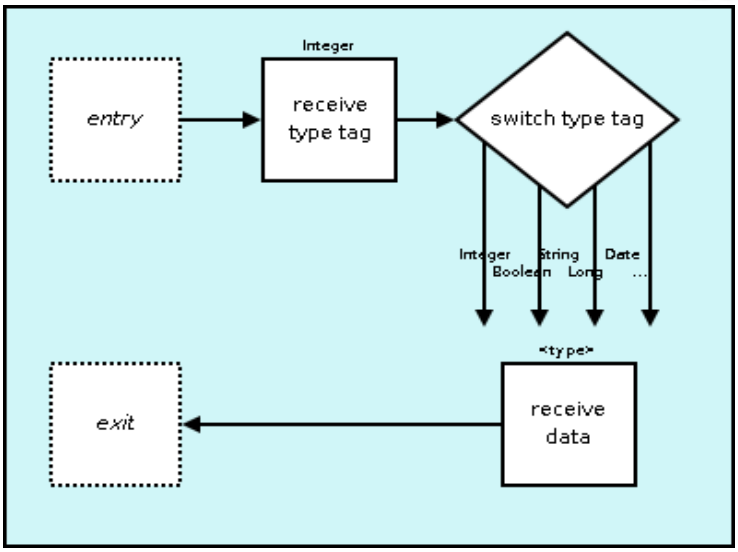
  (only if **hasMoreData** = `true`)

  - [in] **cursorName**: String name of the cursor that will be used

Note that when the result set is incomplete (ie. hasMoreData is `true`), the remaining data must be retrieved by the FetchNextResultSetRows command.

5.1.2 SQL TYPES

SQL types comming from the database are received by dedicated functions called after having receive a TypeTag that indicates the type of incoming data. Also, receiving a SQL type can be specified as follows:



- [in] **type tag**: Integer type of the data to follow
- [in] **data**: <type> the data

Depending on the **type tag** value, the <type> will be:

1 STRING (type tag = 0)

(De)Serialized as java Strings:

5.1.2.1 BIGDECIMAL (type tag = 1)

A BigDecimal is composed of one BigInteger called 'value' and a scale (integer). The 'value' is first send, then the scale.

To be serialized, the value is converted to a byte array (using BigInteger.toByteArray()). Then this bytes are serialized whithin integers (for perfomance purpose). As described in the javadoc, the array of bytes is the two's-complement representation of the BigInteger, in big-endian byte-order: the most significant byte is the zeroth element. It is serialized as-is, modulo head padding alignment bytes. For example, the byte array `[AA BB CC DD EE]` (hexadecimal values) will require two integers to carry its five bytes: `integer1`, the first integer, will be head-padded until byte array tail is aligned. So, `integer1` will be equal to `[00 00 00 AA]` while `integer2` will be `[BB CC DD EE]`.

The reading of a big integer will be done as follows:

- [in] **value length**: Integer number of bytes in the *byte array* representation of the value

  *if padding (ie. "**value length** modulo 4" is positive)*

  - [in] **first word**: Integer first word of data. The number of significant bytes is equal to "**value length** modulo 4". So padded bytes are the

most significant bytes of **first word** (set to zero). First significant byte will be the (4 – value length modulo 4)iest byte

*(**value length** / 4) times:*

- [in] **word**: Integer containing 4 bytes of data (two's-complement - big-endian)

- [in] **scale**: Integer scale to apply to the value

The BigDecimal value will be:

$$\frac{value}{10^{scale}}$$

5.1.2.2 BOOLEAN (type tag = 2)

(De)Serialized as java Booleans (see below)

5.1.2.3 INTEGER (type tag = 3)

(De)Serialized as java Integers (see below)

5.1.2.4 LONG (type tag = 4)

(De)Serialized as java longs (see below)

5.1.2.5 FLOAT (type tag = 5)

(De)Serialized as java floats (see below)

5.1.2.6 DOUBLE (type tag = 6)

(De)Serialized as java doubles (see below)

5.1.2.7 BYTE_ARRAY (type tag = 7)

Byte arrays are simply (de)serialized by sending/receiving the length of data (in bytes) to follow, then the data.

[in] **length**: Integer

length times:

[in] **byte n**: Byte

2 SQL_DATE (type tag = 8)

Sql date (de)serialization is done by sending/receiving the number of milliseconds ellapsed since January 1st 1970 0:00:00 GMT

[in] **millis**: Long

5.1.2.8 SQL_TIME (type tag = 9)

Sql time (de)serialization is the same as sql date, except that the milliseconds value is given as an int

[in] **millis**: Integer

5.1.2.9 SQL_TIMESTAMP (type tag = 10)

Time stamps are serialized like sql date (time in milliseconds as a long) plus the fractional seconds component called nanos, as Integer. Deserialization looks like this:

[in] **millis**: Long

[in] **nanos**: Integer

5.1.2.10 CLOB (type tag = 11)

NOT IMPLEMENTED

5.1.2.11 BLOB= (type tag = 12)

Blobs are (de)serialized as byte arrays, sending the length of the blobs as an integer, then the raw bytes on the stream

5.1.2.12 JAVA_SERIALIZABLE (type tag = 13)

Unknown-but-serializable types are (de)serialized as bytes arrays.

5.2. EXCEPTION-READY ROUTINES

When the controller catches an exception, it sometimes has to notify the driver about the error. This is done by serializing the exception over the stream. To do so, the driver must know that it can receive either the ”normal result” or an exception. Thus, what can be called ”exception-ready routines” have been made available for 5 distinct types: Strings, Booleans, Integers, Longs and ResultSets. They are called Receive<Type>OrException.

These routines basically work on the same scheme (except for ResultSets, see bellow): instead of directly receiving their type, they first receive a ”TypeTag”, indicating wether what follows is an exception or not. Then, if there is no exception (TypeTag `NOT_EXCEPTION` = 18), the ”normal” type is received. Otherwise (TypeTag = 19), the special routine of exception serialization is launched.

For ResultSets, the reception is very similar, but a third type, `NULL_RESULTSET`, has been introduced. If there is no exception, TypeTags can be `RESULTSET` (TypeTag = 14) or `NULL_RESULTSET` (TypeTag = 15). In case of null ResultSet, note that there will be no other incoming data (consider that the request succeed but the result was null). In case of `EXCEPTION` (TypeTag = 19), the behavior described above will apply.

In the rest of the document, these routines will be represented as follows:

5.2.1 EXCEPTIONS

An exception is defined by its type, zero or more exceptions called ”cause” and a stack trace.

Cause exceptions are serialized recursively on the stream.

Formally, the BNF-like definition of exception serialization is:

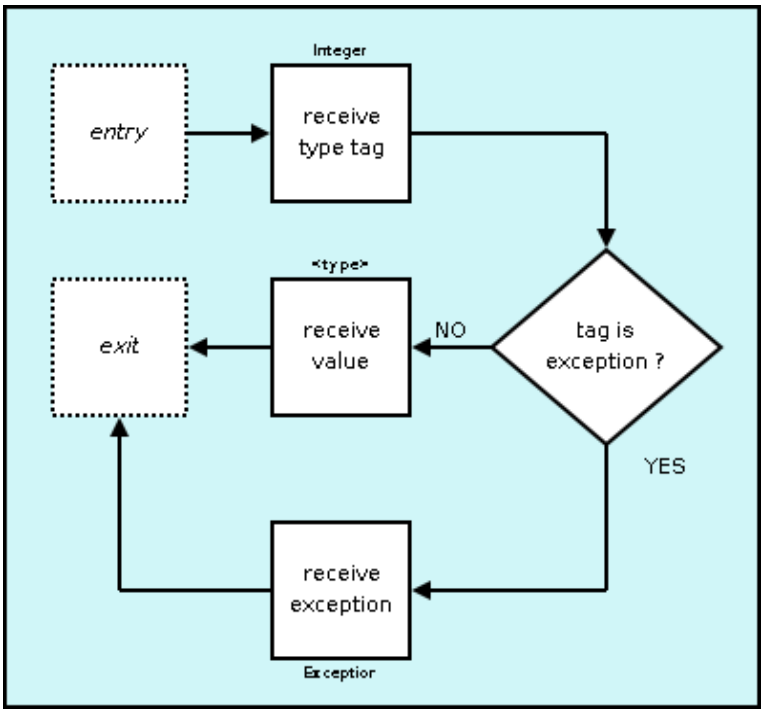
```
exception           = exception-type exception-core
exception-type      = INTEGER
exception-core      = message [cause-exception] stack-trace
message             = STRING
cause-exception     = false/(true exception-core)
stack-trace         = depth (depth*stack-trace-element)
depth               = INTEGER
stack-trace-element = declaring-class method-name
                      filename line-number
declaring-class     = STRING
method-name         = STRING
filename            = STRING
line-number         = INTEGER
```

As an example, let's take the exception E1 with type 1 and message ”I am E1”, which has two cause exceptions E2 and E3 (with message ”I am ...” and without stack trace) plus a stack trace of two elements. On the stream, we will receive chronologically:

```
1                       ;type of E1
“I am E1”               ;E1 message
true                    ;there is a cause exception
“I am E2”               ;E2 message
true                    ;there is a cause exception
“I am E3”               ;E3 message
false                   ;there is no cause exception
0                       ;E3 has no stack trace
0                       ;E2 has no stack trace
2                       ;E1 has 2 stack traces
“C1” “M1” “F1” 1        ;E1 first stack trace
“C2” “M2” “F2” 2        ;E1 second stack trace
```

5.2.2 RECEIVE<TYPE>OREXCEPTION

As told before, to serialize a type, the controller first sends a type tag to indicate the type of the following data. Then, if the type tag is 18, the type is serialized the usual way (see previous chapters). If the type tag is 19 this means that an exception will come, the way described above.



5.3. BASIC DATA TYPES EXCHANGE

As the controller is written in Java, the major part of the exchanged data types is actually Java types. Appart from Strings which is a particular case, all data types are serialized using the com.java.io.DataOutputStream from Sun's JDK.

Here is a list of these types and how they are transmitted over the socket.

5.3.1 BOOLEANS

Booleans are serialized as 32bits integers (see below). `0` for `false`, `1` for `true`.

5.3.2 CHARACTERS

Characters as they appear in Java are send as 16 bits integers, high byte first.

5.3.3 DOUBLES

Doubles are 64bits values send as a Long after having been converted by java.lang.Double.doubleToLongBits(v). See http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Double.html#doubleToLongBits(double) for more info.

5.3.4 FLOATS

Floats are 32bits values written by the `writeInt` function, after having been converted by java.lang.Float.floatToIntBits(float value) function. For more detailed information, see http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Float.html#floatToIntBits(float)

5.3.5 INTEGERS

Java integers are 32bits-long and are send byte after byte, high byte first.

If `i` is the integer to be sent, here is the algorithm to send `i`:

```
1. send ((i >> 24) & 0xFF)
2. send ((i >> 16) & 0xFF)
3. send ((i >>  8) & 0xFF)
4. send ((i >>  0) & 0xFF)
```

5.3.6 LONGS

64 bits are sent high byte first, like other decimals:

```
1. send ((l >> 56) & 0xFF)
2. send ((l >> 48) & 0xFF)
3. send ((l >> 40) & 0xFF)
4. send ((l >> 32) & 0xFF)
5. send ((l >> 24) & 0xFF)
6. send ((l >> 16) & 0xFF)
7. send ((l >>  8) & 0xFF)
8. send ((l >>  0) & 0xFF)
```

5.3.7 SHORTS

16 bits shorts are send high byte first (then low byte)

5.3.8 STRINGS

To avoid handling big strings all at once and being a memory hog, strings are send by chunks of 65535 bytes maximum. To do so, the length of the full string is sent first, then the string itself, chunk-by-chunk. Chunks are composed of first a short, representing the chunck length, then the UTF string.

In order to handle null strings, a boolean is first sent, `false` if the string is null, `true` otherwise.

Here is a summary of ”how a string s is send/retrieved”: