The eXtended subsystem: SQLITE

This subsystem is intended to provide a native SQLITE support to TF PROTOCOL. Be aware that the implementation of the server is not compelled to implement any of the extended subsystems. The client can test if this subsystem is implemented by using one of its commands. If the server responses “UNKNOWN”, you will realize that the subsystem is not implemented.

XS_SQLITE

XS_SQLITE makes the server enter the subsystem. Once inside the subsystem, the server remains there until the client explicitly exits the module with the proper commands. If this module is not implemented by the server, it will return UNKNOWN. Otherwise the eXtended Subsystem SQLITE will start listening for commands.

Every command below will transmit before the payload a header indicating the size of the payload that is coming next. This header is a 64bit signed integer converted to network byte-order (big-endian).

OPEN “path/to/database”

OPEN opens a database in the path indicated by the first parameter. If the database already exist, then OPEN will open it. If the database does not exist, OPEN will create it. Note that there is a whitespace between OPEN and the first parameter. The return status for this command could be:

1 BD OPENED OK WITH ID “ID”
2 FAILED OPEN SQLITE

3 FAILED OPEN JAILDIR

If the command succeeds the ID returned in “ID” -without the double quotes- will represent that particular opened/created database. From now on, any of the possible operations will require that ID to identify the database among all opened databases.

If the first parameter instead of using a path, is set like :memory: the database will be created in memory. This kind of database is not persistent.

The DB-ID is a 64bit signed integer in its text representation. As this number may range from -(2^63)-1 up to 2^63-1, its text representation could be from 1 up to 20 bytes, every byte representing one decimal digit. 8 / 2 * 3 + 8. A buffer of 21 bytes should be enough to store all digits plus a possible sign (-).

EXEC “DB-ID” “SQL-QUERY”

EXEC executes an SQL-Query in the database represented by the specified handle in “DB-ID”. The parameters are separated by a whitespace and the double quotes must not be included. The return status for this command could be:

4 EXEC OK
5 FAILED EXEC BD DOES NOT EXIST
6 FAILED EXEC SQLITE

If EXEC returns OK -code 4- it means that the query has succeeded. May there be data to transmit or not. In any case when there is no more data to send, the header will have 0 has its value. This means that after the client reads 4 EXEC OK, it must continue reading from the network until the header sent previously to every payload is 0.

If there is no data to send, immediately after the client receives 4 EXEC OK will receive a header with value 0. If there is data to send, after the 4 EXEC OK, the server will send the names of the fields separated by two @@ like in the example below:

Name@@Last-Name@@etc

After the client receives the fields names, the server will send every row that the executed query has returned, formatted like the above example. This is, by separating the fields values by two @@:

John@@McDonald@@etc

EXECOF “path/to/file” “DB-ID” “SQL-QUERY”

EXEC executes an SQL-Query in the database represented by the specified handle in “DB-ID”. The result of the query is stored in the file indicated in the first parameter. If the file does not exist, then is created; otherwise, the resulting data from the query will be appended to the file. The format used to store the data is the same used by EXEC. The parameters are separated by a whitespace and the double quotes must not be included. The return status for this command could be:

7 EXECOF OK
8 FAILED EXECOF BD DOES NOT EXIST
9 FAILED EXECOF SQLITE

CLOSE “DB-ID”

CLOSE closes the database represented by the handle “DB-ID”. This frees that handle to be reused in next openings. The return status of this command could be:

10 CLOSE OK
11 FAILED CLOSE SQLITE
12 FAILED CLOSE BD DOES NOT <u>EXIST</u>

LASTROWID “DB-ID”

LASTROWID returns the last row id of the actual queried table in the database represented by the handle DB-ID. The return status of this command could be:

13 LASTROWID OK WITH ROWID [ROW_ID]
14 FAILED LASTROWID SQLITE
15 FAILED LASTROWID BD DOES NOT <u>EXIST</u>

If the command succeeds, the row id will be in “ROW_ID”. This is a 64bit signed integer in its text representation. The same login explained in the OPEN command applies.

SOFTHEAP “size”

SOFTHEAP sets the soft heap limit for the SQLITE engine. The “size” parameter should be a 64bit signed integer in its text representation. The return status of this command could be:

18 HARDHEAP OK

19 FAILED HARDHEAP SQLITE

HARDHEAP “size”

HARDHEAP sets the hard heap limit for the SQLITE engine. The “size” parameter should be a 64bit signed integer in its text representation. The return status of this command could be:

18 HARDHEAP OK
19 FAILED HARDHEAP SQLITE

BLOBIN “BD-ID” “BD-TABLE” “FILENAME” “FILEPATH”

BLOBIN stores in the database represented by the handle “BD-ID” a file read from “FILEPATH” which name will be “FILENAME” in the table “DB-TABLE”. The table indicated by the second parameter must have at least two fields named “Name” and “Data”. “Name” must be TEXT or VARCHAR and “Data” must be BLOB.

The “FILENAME” parameter will contain the desired name to store in the field “Name”. The value stored in the “Name” field is the one used later on to request the file. The “Data” field will contain the bytes of the specified file by the parameter “FILEPATH”.

The return status of this command could be:

20 BLOBIN OK
21 FAILED BLOBIN BD DOES NOT EXIST
22 FAILED BLOBIN FILESYSTEM
23 FAILED BLOBIN SQLITE

BLOBOUT “BD-ID” “BD-TABLE” “FILENAME” “FILEPATH”

BLOBOUT extracts from the database represented by the handle “BD-ID” a file which name is “FILENAME” from the table “DB-TABLE” to a file written to “FILEPATH”. The table indicated by the second parameter must have at least two fields named “Name” and “Data”. “Name” must be TEXT or VARCHAR and “Data” must be BLOB.

The “FILEPATH” parameter will contain the filename to write the output file. If the file already exists, the BLOBOUT command will truncate it to zero before writing it.

The return status of this command could be:

24 BLOBOUT OK
25 FAILED BLOBIN BD DOES NOT EXIST
26 FAILED BLOBIN FILESYSTEM
27 FAILED BLOBIN SQLITE

EXIT

EXIT exits the XS_SQLITE module without freeing the previously allocated resources. This is, every opened database will remain as such and the returned handles, valid. However, you are now at the main command interface of the TF PROTOCOL. If you want enter again the SQLITE subsystem, you can use the XS_SQLITE command again.

TERMINATE

TERMINATE exits the XS_SQLITE module, unlike EXIT command, freeing the previously allocated resources. This is, every opened database will be closed and every handle unallocated. Now you are in the main command interface of the TF PROTOCOL. You may enter the subsystem again by using the XS_SQLITE command.