SQLite Windows GMS Extension

Table of Contents

[Summary 4](#_Toc472105853)

[Complete Examples 5](#_Toc472105854)

[Open Database, Print SQLite Version 5](#_Toc472105855)

[Example 5](#_Toc472105856)

[Execute a query to fetch the tables that are stored inside the DB 5](#_Toc472105857)

[Example 5](#_Toc472105858)

[Execute a query and fetch all results 6](#_Toc472105859)

[Example 6](#_Toc472105860)

[Execute an insert statement 7](#_Toc472105861)

[Example 7](#_Toc472105862)

[Execute an update statement 7](#_Toc472105863)

[Example 7](#_Toc472105864)

[Execute a delete statement 7](#_Toc472105865)

[Example 7](#_Toc472105866)

[Execute generic statements using execute SQL 8](#_Toc472105867)

[Example 8](#_Toc472105868)

[Method Specifications 9](#_Toc472105869)

[SQLiteWinOpenDB 9](#_Toc472105870)

[Example 9](#_Toc472105871)

[Parameters 9](#_Toc472105872)

[Returns 9](#_Toc472105873)

[SQLiteWinGetLibVersion 10](#_Toc472105874)

[Example 10](#_Toc472105875)

[Output 10](#_Toc472105876)

[Parameters 10](#_Toc472105877)

[Returns 10](#_Toc472105878)

[SQLiteWinCloseDB 11](#_Toc472105879)

[Example 11](#_Toc472105880)

[Parameters 11](#_Toc472105881)

[Returns 11](#_Toc472105882)

[SQLiteWinQueryOpen 12](#_Toc472105883)

[Example 12](#_Toc472105884)

[Parameters 12](#_Toc472105885)

[Returns 12](#_Toc472105886)

[SQLiteWinQueryGetNext 13](#_Toc472105887)

[Example 13](#_Toc472105888)

[Parameters 13](#_Toc472105889)

[Returns 13](#_Toc472105890)

[SQLiteWinQueryGetColumnCount 14](#_Toc472105891)

[Example 14](#_Toc472105892)

[Parameters 14](#_Toc472105893)

[Returns 14](#_Toc472105894)

[SQLiteWinQueryGetData 15](#_Toc472105895)

[Example 15](#_Toc472105896)

[Parameters 15](#_Toc472105897)

[Returns 15](#_Toc472105898)

[SQLiteWinQueryClose 16](#_Toc472105899)

[Example 16](#_Toc472105900)

[Parameters 16](#_Toc472105901)

[Returns 16](#_Toc472105902)

[SQLiteWinExecuteInsert 17](#_Toc472105903)

[Example 17](#_Toc472105904)

[Parameters 17](#_Toc472105905)

[Returns 17](#_Toc472105906)

[SQLiteWinExecuteUpdate 18](#_Toc472105907)

[Example 18](#_Toc472105908)

[Parameters 18](#_Toc472105909)

[Returns 18](#_Toc472105910)

[SQLiteWinExecuteDelete 19](#_Toc472105911)

[Example 19](#_Toc472105912)

[Parameters 19](#_Toc472105913)

[Returns 19](#_Toc472105914)

[SQLiteWinExecuteSQL 20](#_Toc472105915)

[Example 20](#_Toc472105916)

[Parameters 20](#_Toc472105917)

[Returns 20](#_Toc472105918)

# Summary

SQLite Windows GMS Extension provides easy access to an SQLite database, making use of standard’s based use of SQL and RDBMS.

With this extension you can port your SQLite database file with the distribution of your game to Windows platforms.

This extension provides the most common functionality of an SQLite DataBase (e.g. insert, update, delete, query, create/alter/drop tables/indexes).

Just add your SQLite DataBase file inside Game Maker's Project. Database files are imported into a project using the "Create Included Files" functionality. All database files must reside inside the the group named "databases".

If there is anything more than you need to do with an SQLite DataBase in Windows, do not hesitate to contact me so that I enrich this extension further.

Special care has been taken to provide fine grained Exception Handling. All SQLite’s Return Codes are captured and forwarded to Game Maker Studio, so that the game always knows what has happened with the execution of every method. Error Handling messages return either the expected result, or in case of an exception a String with the following Template

**Exception String :**

{MethodName}: ERROR:{ReturnCode}: {ErrorMessageForCode]

Feel free to contact me if you have any questions or need something more from this library.

# Complete Examples

## Open Database, Print SQLite Version

|  |
| --- |
| Example |
| //Open db with the specified name  var rslt = SQLiteWinOpenDB( 'databases\win.db' );  show\_debug\_message("Opened database = " + rslt);  //Print SQLite library version  rslt = SQLiteWinGetLibVersion();  show\_debug\_message("SQLite version = " + rslt); |

## Execute a query to fetch the tables that are stored inside the DB

|  |
| --- |
| Example |
| //Open-Prepare Query  var rslt = SQLiteWinQueryOpen("SELECT name FROM sqlite\_master WHERE type='table';");  show\_debug\_message("Query open result = " + rslt);  //If query was opened successfully  if ( rslt == "0" ) {  var i = 0;  //Goto First/Next row  rslt = SQLiteWinQueryGetNext();  while ( rslt == "0" ) {  //Get column data for current row  var tbl\_name = SQLiteWinQueryGetData("0");  show\_debug\_message("Table:"+ tbl\_name);  i++;  rslt = SQLiteWinQueryGetNext();  show\_debug\_message("Query get next result = " + rslt );  }  show\_debug\_message("Query end result = " + rslt );  rslt = SQLiteWinQueryClose();  show\_debug\_message("Query close result = " + rslt);  } |

## Execute a query and fetch all results

|  |
| --- |
| Example |
| //Open-Prepare Query  var rslt = SQLiteWinQueryOpen("SELECT \* FROM users;");  show\_debug\_message("Query open result = " + rslt);  //If query was opened successfully  if ( rslt == "0" ) {  //Get column count  var col\_count = SQLiteWinQueryGetColumnCount();  show\_debug\_message("Column count = " + col\_count);  var i = 0;  //Goto First/Next row  while ( (rslt = SQLiteWinQueryGetNext()) == "0" ) {  for(var j=0; j<real(col\_count); j++){  //Get current column data for current row  var cur\_col = SQLiteWinQueryGetData(string(j));  show\_debug\_message(  "["+string(i)+","+string(j)+"]:"+cur\_col);  }  i++;  show\_debug\_message("Query get next result = " + rslt );  }  show\_debug\_message("Query end result = " + rslt );  //Close Query when Done  rslt = SQLiteWinQueryClose();  show\_debug\_message("Query close result = " + rslt);  } |

## Execute an insert statement

|  |
| --- |
| Example |
| var sql\_insert = "INSERT INTO users (username, password) VALUES ('user1','a\_simple\_pass');";  var rslt = SQLiteWinExecuteInsert( sql\_insert );  show\_debug\_message("Inserted row id = " + rslt); |

## Execute an update statement

|  |
| --- |
| Example |
| var sql\_update = "UPDATE users SET password = 'a\_more\_secure\_apss' WHERE username = 'user1';";  var rslt = SQLiteWinExecuteUpdate( sql\_update );  show\_debug\_message("Updated rows = " + rslt); |

## Execute a delete statement

|  |
| --- |
| Example |
| var sql\_delete = "DELETE FROM users WHERE username = 'user1';";  var rslt = SQLiteWinExecuteDelete( sql\_delete );  show\_debug\_message("Deleted rows = " + rslt); |

## Execute generic statements using execute SQL

|  |
| --- |
| Example |
| //Execute an insert statement  var insert\_stmt = "INSERT INTO users (username, password) VALUES ('user1','a\_simple\_pass');";  var rslt = SQLiteWinExecuteSQL ( insert\_stmt );  show\_debug\_message("Insert statement result code = " + rslt);  //Execute an update statement  var update\_stmt = "UPDATE users SET password = 'a\_more\_secure\_apss' WHERE username = 'user1';";  rslt = SQLiteWinExecuteSQL ( update\_stmt );  show\_debug\_message("Update statement result code = " + rslt);  //Execute a delete statement  var delete\_stmt = "DELETE FROM users WHERE username = 'user1';";  rslt = SQLiteWinExecuteSQL ( delete\_stmt );  show\_debug\_message("Delete statement result code = " + rslt);  //Create a new table  var create\_table\_stmt = "CREATE TABLE upgrades "+  "(version INT, label TEXT);";  rslt = SQLiteWinExecuteSQL ( create\_table\_stmt );  show\_debug\_message("Create table result code = " + rslt);  //Create a new index  var create\_index\_stmt = "CREATE INDEX IDX\_upgrades\_label "+  "ON upgrades (label);";  rslt = SQLiteWinExecuteSQL ( create\_index\_stmt );  show\_debug\_message("Create Index result code = " + rslt); |

# Method Specifications

## SQLiteWinOpenDB

**String** SQLiteWinOpenDB(**String** db\_name)

This method sets the database filename and opens the database.

|  |
| --- |
| Example |
| var rslt = SQLiteWinOpenDB("databases/win.db");  if ( rslt == "0" )  show\_debug\_message("SQLiteWinOpenDB was successful!");  else  show\_debug\_message("SQLiteWinOpenDB error="+rslt); |

|  |  |
| --- | --- |
| Parameters | |
| **db\_name** | **[String]:** The filename of the database that was included into Game Maker Studio Project (Included Files\databases). If the file does not exist, SQLite will create a new Empty DataBase with the provided filename. |

|  |  |
| --- | --- |
| Returns | |
| **String** | Returns "0" in case of no errors.  In case of errors, it returns the Exception String |

## SQLiteWinGetLibVersion

**String** SQLiteWinGetLibVersion()

This method prints the SQLite library version that was compiled with this version of the Extension.

The extension contains the latest available sqlite version at the time of the publication.

Please contact me to provide a newer version of the extension, if a newer version of sqlite library is needed.

|  |
| --- |
| Example |
| var rslt = SQLiteWinGetLibVersion();  show\_debug\_message( rslt ); |

|  |
| --- |
| Output |
| SQLiteWinGetLibVersion:3.16.2:2017-01-06 16:32:41 a65a62893ca8319e89e48b8a38cf8a59c69a8209:3016002 |

|  |
| --- |
| Parameters |
| No parameters |

|  |  |
| --- | --- |
| Returns | |
| **String** | Returns (delimeted by “:”) the sqlite version, the date of the sqlite publication and the version as a number:  "SQLiteWinGetLibVersion:3.16.2:2017-01-06 16:32:41 a65a62893ca8319e89e48b8a38cf8a59c69a8209:3016002" |

## SQLiteWinCloseDB

**String** SQLiteWinCloseDB()

This method closes the SQLite database.

It should be called :

* Just before exiting the Game Maker Application
* After all statement on the DB have finished and after closing last executed query

|  |
| --- |
| Example |
| var rslt = SQLiteWinCloseDB();  if ( rslt == "0" )  show\_debug\_message("SQLiteWinCloseDB was successful!");  else  show\_debug\_message("SQLiteWinCloseDB error="+rslt); |

|  |
| --- |
| Parameters |
| No parameters |

|  |  |
| --- | --- |
| Returns | |
| **String** | Returns "0" in case of no errors.  In case of errors, it returns the Exception String |

## SQLiteWinQueryOpen

**String** SQLiteWinQueryOpen(**String** query)

This method opens/prepares a query.

One can iterate through the results using the functions *SQLiteWinQueryGetNext, SQLiteWinQueryGetColumnCount* and *SQLiteWinQueryGetData*.

|  |
| --- |
| Example |
| var rslt = SQLiteWinQueryOpen("SELECT \* FROM users");  if (string\_pos("SQLiteWinQueryOpen", rslt) != 1)  show\_debug\_message("SQLiteWinQueryOpen was successful!");  else  show\_debug\_message("SQLiteWinQueryOpen error="+rslt); |

|  |  |
| --- | --- |
| Parameters | |
| **query** | **[String]:** The query |

|  |  |
| --- | --- |
| Returns | |
| **String** | Returns "0" in case of no errors.  In case of errors, it returns the Exception String |

## SQLiteWinQueryGetNext

**String** SQLiteWinQueryGetNext()

This method fetches the first or the next row available in the last query opened.

* If there are more rows and it successfully goes to the next row, it returns “0”
* If the query fetches no more rows or there are no rows to fetch it returns “-1”

|  |
| --- |
| Example |
| var rslt = SQLiteWinQueryGetNext();  if ( rslt == "0")  show\_debug\_message("SQLiteWinQueryGetNext has more rows!");  else if ( rslt == "-1")  show\_debug\_message("SQLiteWinQueryGetNext has **NO** more rows!");  else  show\_debug\_message("SQLiteWinQueryGetNext error="+rslt); |

|  |
| --- |
| Parameters |
| No Parameters |

|  |  |
| --- | --- |
| Returns | |
| **String** | Returns "0" in case there are more rows.  Returns "-1" in case there **NO** more rows.  In case of errors, it returns the Exception String |

## SQLiteWinQueryGetColumnCount

**String** SQLiteWinQueryGetColumnCount()

This method fetches the number of wolumns in the currently opened query.

This is a helper method, to fetch all columns with a for loop.

|  |
| --- |
| Example |
| var rslt = SQLiteWinQueryGetColumnCount();  show\_debug\_message("Column count is :" + rslt); |

|  |
| --- |
| Parameters |
| No Parameters |

|  |  |
| --- | --- |
| Returns | |
| **String** | Returns the number of columns in the currently open query.  In case of errors, it returns “0”. |

## SQLiteWinQueryGetData

**String** SQLiteWinQueryGetData(**String** column)

This method fetches the data of the specified column for the currently open query and the currently fetched row (see complete examples).

The columns start from 0 (0 based indexing).

|  |
| --- |
| Example |
| var rslt = SQLiteWinQueryGetData( "0" );  if (string\_pos("SQLiteWinQueryGetData", rslt) != 1){  show\_debug\_message("SQLiteWinQueryGetData was successful!");  show\_debug\_message("Current column data :"+rslt);  } else  show\_debug\_message("SQLiteWinQueryGetData error="+rslt); |

|  |  |
| --- | --- |
| Parameters | |
| **column** | **[String]:** The column in the query resultset (0 based indexing). |

|  |  |
| --- | --- |
| Returns | |
| **String** | Returns the data that reside on the current row and the given column for this query.  In case of errors, it returns the Exception String |

## SQLiteWinQueryClose

**String** SQLiteWinQueryClose()

This method closes the currently opened query. This means that data will no longer be available, unless we re-open a new query. Therefore *SQLiteWinQueryGetNext* and *SQLiteWinQueryGetData* will result in error until the next successful opening of the next query.

|  |
| --- |
| Example |
| var rslt = SQLiteWinQueryClose();  if ( rslt == "0" )  show\_debug\_message("SQLiteWinQueryClose was successful!");  else  show\_debug\_message("SQLiteWinQueryClose error="+rslt); |

|  |
| --- |
| Parameters |
| No Parameters |

|  |  |
| --- | --- |
| Returns | |
| **String** | Returns "0" in case of no errors.  In case of errors, it returns the Exception String |

## SQLiteWinExecuteInsert

**String** SQLiteWinExecuteInsert(**String** sql\_insert)

This method executes an insert statement.

If the insert is successful the method returns the id of the newly inserted row.

|  |
| --- |
| Example |
| var rslt = SQLiteWinExecuteInsert( sql\_insert );  if (string\_pos("SQLiteWinExecuteInsert", rslt) != 1){  show\_debug\_message("SQLiteWinExecuteInsert was successful!");  show\_debug\_message("Newly created row’s id is :"+rslt);  } else  show\_debug\_message("SQLiteWinExecuteInsert error="+rslt); |

|  |  |
| --- | --- |
| Parameters | |
| **sql\_insert** | **[String]:** The INSERT statement |

|  |  |
| --- | --- |
| Returns | |
| **String** | Returns the id of the newly created row.  In case of errors, it returns the Exception String |

## SQLiteWinExecuteUpdate

**String** SQLiteWinExecuteUpdate(**String** sql\_update)

This method executes an update statement.

If the update is successful the method returns the number of the rows that were updated.

|  |
| --- |
| Example |
| var rslt = SQLiteWinExecuteUpdate( sql\_update );  if (string\_pos("SQLiteWinExecuteUpdate", rslt) != 1){  show\_debug\_message("SQLiteWinExecuteUpdate was successful!");  show\_debug\_message("Count of updated rows is :"+rslt);  } else  show\_debug\_message("SQLiteWinExecuteUpdate error="+rslt); |

|  |  |
| --- | --- |
| Parameters | |
| **sql\_update** | **[String]:** The UPDATE statement |

|  |  |
| --- | --- |
| Returns | |
| **String** | Returns the number of the rows that were updated.  In case of errors, it returns the Exception String |

## SQLiteWinExecuteDelete

**String** SQLiteWinExecuteDelete(**String** sql\_delete)

This method executes a delete statement.

If the delete is successful the method returns the number of the rows that were deleted.

|  |
| --- |
| Example |
| var rslt = SQLiteWinExecuteDelete( sql\_delete );  if (string\_pos("SQLiteWinExecuteDelete ", rslt) != 1){  show\_debug\_message("SQLiteWinExecuteDelete was successful!");  show\_debug\_message("Count of deleted rows is :"+rslt);  } else  show\_debug\_message("SQLiteWinExecuteDelete error="+rslt); |

|  |  |
| --- | --- |
| Parameters | |
| **sql\_delete** | **[String]:** The DELETE statement. |

|  |  |
| --- | --- |
| Returns | |
| **String** | Returns the number of the rows that were deleted.  In case of errors, it returns the Exception String |

## SQLiteWinExecuteSQL

**String** SQLiteWinExecuteSQL(**String** statement)

This method executes any kind of statement (except for queries) :

* INSERT
* UPDATE
* DELETE
* CREATE
* ALTER
* DROP

Although it does not provide feedback regarding the result of the statement (e.g. the row id of the inserted row), it does provide feedback in regards of the success or error of the execution of the statement.

It is a very convenient way to do batch jobs or upgrade the database schema (e.g. add a column to a table).

|  |
| --- |
| Example |
| var rslt = SQLiteWinExecuteSQL( statement );  if ( rslt == "0" )  show\_debug\_message("SQLiteWinExecuteSQL was successful!");  else  show\_debug\_message("SQLiteWinExecuteSQL error="+rslt); |

|  |  |
| --- | --- |
| Parameters | |
| **statement** | **[String]:** The statement to be executed. |

|  |  |
| --- | --- |
| Returns | |
| **String** | Returns "0" in case of no errors.  In case of errors, it returns the Exception String |