

ooRexx

Documentation 1.0.0

ooSQLite Reference

Friday, December 23, 2022 svn revision 12583



ooRexx Documentation 1.0.0 ooSQLite Reference

Friday, December 23, 2022 svn revision 12583

Edition 2022.12.22

Author	Open Object Rexx™
Author	W. David Ashley
Author	Rony G. Flatscher
Author	Rick McGuire
Author	Mark Miesfeld
Author	Lee Peedin
Author	Oliver Sims
Author	Jon Wolfers

Copyright © 2005-2022 Rexx Language Association. All rights reserved.

Portions Copyright © 1995, 2004 IBM Corporation and others. All rights reserved.

This documentation and accompanying materials are made available under the terms of the Common Public License v1.0 which accompanies this distribution. A copy is also available as an appendix to this document and at the following address: <http://www.oorexx.org/license.html>.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Rexx Language Association nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Preface	x
1. Document Conventions	x
1.1. Typographic Conventions	x
1.2. Notes and Warnings	x
2. Who Should Use This Book	xi
3. How This Book is Structured	xi
4. Related Information	xii
5. How to Read the Syntax Diagrams	xii
6. Getting Help and Submitting Feedback	xiii
6.1. The Open Object REXX SourceForge Site	xiii
6.2. The REXX Language Association Mailing List	xiv
6.3. comp.lang.rexx Newsgroup	xv
1. Brief Overview	1
1.1. Getting Started	1
1.1.1. Command Line Shell	1
1.2. Definition of Terms	2
1.2.1. Database Name	3
1.2.2. Handle	3
1.2.3. Threading Mode	3
1.3. Common Concepts	4
1.3.1. Embedded	4
1.3.2. Database Engine	4
1.4. SQLite Features	4
2. The Object Orientated Interface to SQLite	8
3. Primary ooSQLite Classes	9
3.1. The ooSQLite Class	9
3.1.1. Method Table	9
3.1.2. null (Attribute)	10
3.1.3. recordFormat (Attribute)	11
3.1.4. compileOptionGet	13
3.1.5. compileOptionUsed	14
3.1.6. complete	14
3.1.7. enquote	15
3.1.8. errStr	17
3.1.9. libVersion	18
3.1.10. libVersionNumber	19
3.1.11. memoryHighWater	19
3.1.12. memoryUsed	20
3.1.13. releaseMemory	20
3.1.14. softHeapLimit64	21
3.1.15. sourceID	22
3.1.16. sqlite3Version	22
3.1.17. status	23
3.1.18. threadSafe	25
3.1.19. version	25
3.2. The ooSQLiteBackup Class	27
3.2.1. Method Table	27
3.2.2. new (Class method)	28
3.2.3. finished (Attribute)	30
3.2.4. initCode (Attribute)	30
3.2.5. lastErrCode (Attribute)	31
3.2.6. lastErrMsg (Attribute)	32

3.2.7. pageCount (Attribute)	32
3.2.8. remaining (Attribute)	33
3.2.9. saveDestConn (Attribute)	34
3.2.10. finish	35
3.2.11. getDestConn	36
3.2.12. step	36
3.3. The ooSQLiteConnection Class	37
3.3.1. Method Table	38
3.3.2. new (Class method)	39
3.3.3. backupDestination (Attribute)	41
3.3.4. closed (Attribute)	41
3.3.5. fileName (Attribute)	42
3.3.6. initCode (Attribute)	42
3.3.7. lastErrCode (Attribute)	43
3.3.8. lastErrMsg (Attribute)	44
3.3.9. null (Attribute)	45
3.3.10. recordFormat (Attribute)	46
3.3.11. busyHandler	46
3.3.12. busyTimeOut	48
3.3.13. changes	49
3.3.14. close	49
3.3.15. commitHook	50
3.3.16. createCollation	52
3.3.17. createFunction	52
3.3.18. dbFileName	53
3.3.19. dbMutex	53
3.3.20. dbReadOnly	54
3.3.21. dbReleaseMemory	55
3.3.22. dbStatus	55
3.3.23. errCode	57
3.3.24. errMsg	57
3.3.25. exec	58
3.3.26. extendedErrCode	61
3.3.27. extendedResultCodes	62
3.3.28. getAutocommit	62
3.3.29. interrupt	63
3.3.30. lastInsertRowID	63
3.3.31. limit	64
3.3.32. nextStmt	65
3.3.33. profile	65
3.3.34. progressHandler	67
3.3.35. pragma	69
3.3.36. rollbackHook	80
3.3.37. setAuthorizer	81
3.3.38. tableColumnMetadata	85
3.3.39. totalChanges	86
3.3.40. trace	87
3.3.41. updateHook	88
3.4. The ooSQLiteMutex Class	90
3.4.1. Method Table	90
3.4.2. new (Class method)	91
3.4.3. closed (Attribute)	91
3.4.4.isNull (Attribute)	92
3.4.5. enter	92

3.4.6. free	93
3.4.7. leave	93
3.4.8. try	94
3.5. The ooSQLiteStmt Class	94
3.5.1. Method Table	94
3.5.2. new (Class method)	95
3.5.3. initCode (Attribute)	95
3.5.4. finalized (Attribute)	96
3.5.5. lastErrCode (Attribute)	96
3.5.6. lastErrMsg (Attribute)	97
3.5.7. null (Attribute)	98
3.5.8. recordFormat (Attribute)	99
3.5.9. bindBlob	99
3.5.10. bindDouble	100
3.5.11. bindInt	100
3.5.12. bindInt64	101
3.5.13. bindNull	101
3.5.14. bindParameterCount	102
3.5.15. bindParameterIndex	102
3.5.16. bindParameterName	103
3.5.17. bindText	103
3.5.18. bindValue	104
3.5.19. bindZeroBlob	104
3.5.20. clearBindings	105
3.5.21. columnBlob	105
3.5.22. columnBytes	106
3.5.23. columnCount	106
3.5.24. columnDataBaseName	107
3.5.25. columnDeclType	107
3.5.26. columnDouble	108
3.5.27. columnIndex	108
3.5.28. columnInt	109
3.5.29. columnInt64	109
3.5.30. columnName	110
3.5.31. columnOriginName	110
3.5.32. columnTableName	111
3.5.33. columnText	111
3.5.34. columnType	112
3.5.35. columnValue	112
3.5.36. dataCount	113
3.5.37. dbHandle	113
3.5.38. finalize	114
3.5.39. reset	114
3.5.40. step	115
3.5.41. stmtBusy	115
3.5.42. stmt_READONLY	116
3.5.43. stmtStatus	116
3.5.44. value	117
4. User Defined Extension Classes	118
4.1. The ooSQLCollation Class	118
4.1.1. new (Class method)	118
4.2. The ooSQLCollationNeeded Class	119
4.2.1. new (Class method)	119

4.3. The ooSQLExtensions Class	119
4.3.1. Method Table	119
4.3.2. lastErrCode (Attribute)	120
4.3.3. lastErrMsg (Attribute)	121
4.3.4. autoBuiltin (Class method)	121
4.3.5. autoCollationNeeded (Class method)	122
4.3.6. autoCollation	122
4.3.7. autoFunction	123
4.3.8. autoPackage	123
4.3.9. cancelAutoBuiltin	124
4.3.10. getLibrary	124
4.3.11. getPackage	125
4.3.12. listBuiltins	125
4.3.13. loadLibrary	126
4.3.14. loadPackage	126
4.3.15. resetAutoBuiltin	127
4.3.16. registerBuiltin	127
4.4. The ooSQLFunction Class	128
4.4.1. new (Class method)	128
4.5. The ooSQLLibrary Class	128
4.5.1. Method Table	129
4.5.2. new (Class method)	129
4.5.3. lastErrCode (Attribute)	129
4.5.4. lastErrMsg (Attribute)	130
4.6. The ooSQLPackage Class	131
4.6.1. Method Table	131
4.6.2. new (Class method)	131
4.6.3. lastErrCode (Attribute)	132
4.6.4. lastErrMsg (Attribute)	133
4.6.5. getCollation	133
4.6.6. getCollationNeeded	134
4.6.7. getFunction	134
4.6.8. register	135
4.7. The ooSQLResult Class	135
4.7.1. Method Table	135
4.7.2. blob (Class method)	136
4.8. The ooSQLValue Class	136
4.8.1. Method Table	136
4.8.2. blob (Class method)	136
5. The ooSQLite Constants	138
5.1. All Constants Table	138
5.2. Compile Time Version Constants	148
5.3. ooSQLite Specific Constants	148
5.4. ooSQLite Specific Result Code Constants	149
5.5. Result Code Constants	149
5.6. File Open Constants	151
5.7. Authorizer Action Constants	152
5.8. Authorizer Return Code Constants	153
5.9. xAccess VFS Method Constants	153
5.10. Checkpoint Operation Parameter Constants	153
5.11. Configuration Option Constants	154
5.12. DB Connection Configuration Constants	154
5.13. DB Status Parameter Constants	155

5.14. File Control Opcode Constants	156
5.15. Fundamental Datatype Constants	156
5.16. Device Characteristic Constants	157
5.17. Run-Time Limit Constants	157
5.18. File Locking Constants	158
5.19. Mutex Type Constants	158
5.20. xShmLock VFS Constants	158
5.21. Destructor Behavior Constants	159
5.22. Status Parameter Constants	159
5.23. Status Parameter (stmt) Constants	159
5.24. Synchronization Constants	160
5.25. Text Encoding Constants	160
5.26. Virtual Table Config Option Constants	161
5.27. merge (Class method)	161
6. The Classic Rexx Interface to SQLite	162
6.1. Online Backup Feature	164
7. ooSQLite Specific Functions	165
7.1. ooSQLiteEnquote	165
7.2. ooSQLiteMerge	166
7.3. ooSQLiteRegisterBuiltin	167
7.4. ooSQLiteVersion	167
8. ooSQLite Functions A - F	169
8.1. oosqlAutoExtension	169
8.2. oosqlBackupFinish	169
8.3. oosqlBackupInit	170
8.4. oosqlBackupPageCount	171
8.5. oosqlBackupRemaining	172
8.6. oosqlBackupStep	173
8.7. oosqlBindBlob	174
8.8. oosqlBindDouble	174
8.9. oosqlBindInt	175
8.10. oosqlBindInt64	175
8.11. oosqlBindNull	176
8.12. oosqlBindParameterCount	176
8.13. oosqlBindParameterIndex	177
8.14. oosqlBindParameterName	177
8.15. oosqlBindText	178
8.16. oosqlBindValue	178
8.17. oosqlBindZeroBlob	179
8.18. oosqlBusyHandler	179
8.19. oosqlBusyTimeOut	180
8.20. oosqlCancelAutoBuiltin	180
8.21. oosqlChanges	181
8.22. oosqlClearBindings	181
8.23. oosqlClose	182
8.24. oosqlColumnBlob	182
8.25. oosqlColumnBytes	183
8.26. oosqlColumnCount	183
8.27. oosqlColumnNameDatabaseName	184
8.28. oosqlColumnDeclType	184
8.29. oosqlColumnDouble	185
8.30. oosqlColumnIndex	185

8.31. oosqlColumnInt	186
8.32. oosqlColumnInt64	187
8.33. oosqlColumnName	187
8.34. oosqlColumnOriginName	188
8.35. oosqlColumnTableName	188
8.36. oosqlColumnText	189
8.37. oosqlColumnType	189
8.38. oosqlColumnValue	190
8.39. oosqlCollationNeeded	190
8.40. oosqlCommitHook	191
8.41. oosqlCompileOptionGet	191
8.42. oosqlCompileOptionUsed	192
8.43. oosqlComplete	192
8.44. oosqlCreateCollation	193
8.45. oosqlCreateFunction	193
8.46. oosqlDataCount	194
8.47. oosqlDbFileName	194
8.48. oosqlDbHandle	195
8.49. oosqlDbMutex	195
8.50. oosqlDbReadOnly	196
8.51. oosqlDbReleaseMemory	196
8.52. oosqlDbStatus	197
8.53. oosqlEnableLoadExtension	197
8.54. oosqlErrCode	198
8.55. oosqlErrMsg	198
8.56. oosqlErrStr	199
8.57. oosqlExec	200
8.58. oosqlExtendedErrCode	200
8.59. oosqlExtendedResultCodes	201
8.60. oosqlFinalize	201
9. ooSQLite Functions G - R	203
9.1. oosqlGetAutocommit	203
9.2. oosqlInterrupt	203
9.3. oosqlIsHandleNull	204
9.4. oosqlLastInsertRowID	204
9.5. oosqlLibVersion	205
9.6. oosqlLibVersionNumber	205
9.7. oosqlLimit	206
9.8. oosqlLoadExtension	206
9.9. oosqlMemoryHighWater	207
9.10. oosqlMemoryUsed	207
9.11. oosqlMutexAlloc	208
9.12. oosqlMutexEnter	208
9.13. oosqlMutexFree	209
9.14. oosqlMutexLeave	209
9.15. oosqlMutexTry	210
9.16. oosqlNextStmt	210
9.17. oosqlOpen	211
9.18. oosqlPrepare	212
9.19. oosqlProfile	214
9.20. oosqlProgressHandler	214
9.21. oosqlReleaseMemory	215
9.22. oosqlReset	215

9.23. oosqlResetAutoExtension	216
9.24. oosqlResultBlob	216
9.25. oosqlResultDouble	217
9.26. oosqlResultError	217
9.27. oosqlResultErrorCode	218
9.28. oosqlResultErrorNoMem	218
9.29. oosqlResultErrorTooBig	219
9.30. oosqlResultInt	219
9.31. oosqlResultInt64	220
9.32. oosqlResultNull	220
9.33. oosqlResultText	221
9.34. oosqlResultValue	221
9.35. oosqlResultZeroBlob	222
9.36. oosqlRollbackHook	222
10. ooSQLite Functions S - Z	224
10.1. oosqlSetAuthorizer	224
10.2. oosqlSoftHeapLimit64	224
10.3. oosqlSourceID	225
10.4. oosqlSql	225
10.5. oosqlStatus	226
10.6. oosqlStep	226
10.7. oosqlStmtBusy	227
10.8. oosqlStmtReadonly	227
10.9. oosqlStmtStatus	228
10.10. oosqlStrGlob	228
10.11. oosqlTableColumnMetadata	229
10.12. oosqlThreadSafe	229
10.13. oosqlTotalChanges	230
10.14. oosqlTrace	230
10.15. oosqlUpdateHook	231
10.16. oosqlValueBlob	231
10.17. oosqlValueBytes	232
10.18. oosqlValueDouble	232
10.19. oosqlValueInt	233
10.20. oosqlValueNumericType	233
10.21. oosqlValueText	234
10.22. oosqlValueType	234
10.23. oosqlVersion	235
A. Notices	236
A.1. Trademarks	236
A.2. Source Code For This Document	237
B. Common Public License Version 1.0	238
B.1. Definitions	238
B.2. Grant of Rights	238
B.3. Requirements	239
B.4. Commercial Distribution	239
B.5. No Warranty	240
B.6. Disclaimer of Liability	240
B.7. General	240
C. Revision History	242
Index	243

Preface

This book describes the ooSQLite framework, which is implemented as an external library package for ooRexx. External library packages are often called *extensions*, or *native extensions*. The library package extends the capability of the Rexx interpreter by adding functionality or features not present in the base interpreter.

The ooSQLite library package gives programmers access to the SQLite database engine directly from their Rexx programs. This book describes the Classes, Methods, and Functions in ooSQLite that allow that access and describes how to use them.

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

1.1. Typographic Conventions

Typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

Mono-spaced Bold is used to highlight literal strings, class names, or inline code examples. For example:

The **Class** class comparison methods return **.true** or **.false**, the result of performing the comparison operation.

This method is exactly equivalent to **subWord(n, 1)**.

Mono-spaced Normal denotes method names or source code in program listings set off as separate examples.

This method has no effect on the action of any `hasEntry`, `hasIndex`, `items`, `remove`, or `supplier` message sent to the collection.

```
-- reverse an array
a = .Array-of("one", "two", "three", "four", "five")

-- five, four, three, two, one
aReverse = .CircularQueue~new(a~size)~appendAll(a)~makeArray("lifo")
```

Proportional Italic is used for method and function variables and arguments.

A supplier loop specifies one or two control variables, *index*, and *item*, which receive a different value on each repetition of the loop.

Returns a string of length *length* with *string* centered in it and with *pad* characters added as necessary to make up length.

1.2. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.



Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.



Important

Important boxes detail things that are easily missed, like mandatory initialization. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.



Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. Who Should Use This Book

This book is intended for Open Object REXX programmers who are using or intend to use ooSQLite in their REXX programs. The book does not teach SQL nor relational database design or management. It provides a basic introduction to using SQLite through the documentation of the classes and functions available in ooSQLite. In depth documentation of SQLite is provided by the developers of SQLite at their [website](#)¹, not in this book.

3. How This Book is Structured

This book is primarily a reference to the classes and functions in ooSQLite. ooSQLite provides an object-orientated interface and a classic REXX interface to SQLite. The book is broken into two main sections. The first section describes the object-orientated interface and the second section describes the classic REXX section. Preceding the object-orientated section are the two chapters, *About This Book* and *Brief Overview*. They contain a small amount of overall information. A detailed index follows the classic REXX section.

The object-orientated section contains a reference for each class in ooSQLite. Each class reference contains a single section for every class, instance, and attribute method of the class. The first section in a class reference is a table listing every method with a brief description and a link to the detailed documentation for the method.

Each method section starts with a [syntax](#) diagram as a synopsis of how to invoke the method. Text follows describing the method, listing each argument to the method, and the return from the method. Most method sections have a *remarks* component discussing the method in more detail. Most method sections also have a *details* component pointing to the SQLite documentation that is most relevant

¹ <http://www.sqlite.org/index.html>

to the method. To fully understand the use of any single method, the reader should also consult the SQLite documentation relevant to that method. Finally, many, but not all, of the method sections end with a short code snippet as an example of using the method.

The classic Rexx section starts with a table that lists every function available for the classic Rexx programmer. This table is similar to the method table that begins each class reference. The table contains a brief description of the function and a link to the detailed documentation for the function. Each function is fully documented in a single section.

The reference for each single function is very similar to the reference to each single method. It starts with a syntax diagram, describes the function, lists the arguments and return. It usually has a remarks component. The details component points the way to the authoritative SQLite documentation. Some, but not all function references contain an example.

Each method and function reference is meant to be stand alone. All the details to use any method or function are included within the section for that method or function. Each method and function is listed in the index. The reader can look up any function or method in the index or the table of contents and go straight to the reference for the function or method.

Within the reference for a single class, each method appears in alphabetical order. Likewise the reference for each function appears in this reference manual in alphabetical order.

4. Related Information

See also: *Open Object Rexx: Reference*

There is a wealth of information on SQLite and how to use SQLite on the Web. This information is directly applicable to using ooSQLite. The SQLite [home](#)² page is the authoritative answer to any SQLite question. In addition there is a SQLiter users mailing [list](#)³ that can be subscribed to.

5. How to Read the Syntax Diagrams

Throughout this book, syntax is described using the structure defined below. This is similar to, but slightly different than, the IBM syntax diagrams used in other ooRexx reference documentation. The author is calling these diagrams *simplified railroad tracks*. It primarily strives to limit all diagrams to 2 lines, and does away with much of the complexity of true IBM railroad tracks. The body of text following the syntax diagrams will resolve any ambiguities in the diagram.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

The >>--- symbol indicates the beginning of a statement.

The ---> symbol indicates that the statement syntax is continued on the next line. In most cases statement continuation is avoided.

The >--- symbol indicates that a statement is continued from the previous line.

The --->< symbol indicates the end of a statement.

Diagrams of syntactical units other than complete statements start with the >--- symbol and end with the ---> symbol.

² <http://www.sqlite.org/index.html>

³ <http://sqlite.org:8080/cgi-bin/mailman/listinfo/sqlite-users>

- Required items appear on the horizontal line (the main path).

```
>>-STATEMENT--required_item-----><
```

- Optional items appear below the main path.

```
>>-STATEMENT -+-----><
    +-optional_item-+
```

In this reference, the syntax being presented is the syntax for method invocations and function calls. The diagrams show the method, or function, names and their arguments. For method invocations, the section the method is included in, and / or the text itself, make it clear as to which class the method belongs to. The arguments to the method or function are enclosed in parenthesis and separated by commas. The arguments are represented by appropriate variable names and these variable names are then described in the text for the method. When arguments are optional, the default value and or behavior if the argument is omitted is also described in the text. In all cases, the text rather than the syntax diagram should be considered definitive.

The following example shows the described syntax:

```
>>--rollbackHook(--callBackObj---+-----+---+---+--><
    +-,-mthName--+  +-,-userData--+
```

6. Getting Help and Submitting Feedback

The Open Object Rexx Project has a number of methods to obtain help and submit feedback for ooRexx and the extension packages that are part of ooRexx. These methods, in no particular order of preference, are listed below.

6.1. The Open Object Rexx SourceForge Site

The [Open Object Rexx Project](#)⁴ utilizes [SourceForge](#)⁵ to house the [ooRexx Project](#)⁶ source repositories, mailing lists and other project features. Over time it has become apparent that the Developer and User mailing lists are better tools for carrying on discussions concerning ooRexx and that the Forums provided by SourceForge are cumbersome to use. The ooRexx user is most likely to get timely replies from one of the mailing lists.

Here is a list of some of the most useful facilities provided by SourceForge.

The Developer Mailing List

You can subscribe to the oorexx-devel mailing list at [ooRexx Mailing List Subscriptions](#)⁷ page. This list is for discussing ooRexx project development activities and future interpreter enhancements. It also supports a historical archive of past messages.

The Users Mailing List

You can subscribe to the oorexx-users mailing list at [ooRexx Mailing List Subscriptions](#)⁸ page. This list is for discussing using ooRexx. It also supports a historical archive of past messages.

⁴ <http://www.oorexx.org/>

⁵ <http://sourceforge.net/>

⁶ <http://sourceforge.net/projects/oorexx>

⁷ http://sourceforge.net/mail/?group_id=119701

⁸ http://sourceforge.net/mail/?group_id=119701

The Announcements Mailing List

You can subscribe to the oorexx-announce mailing list at [ooRexx Mailing List Subscriptions](#)⁹ page. This list is only used to announce significant ooRexx project events.

The Bug Mailing List

You can subscribe to the oorexx-bugs mailing list at [ooRexx Mailing List Subscriptions](#)¹⁰ page. This list is only used for monitoring changes to the ooRexx bug tracking system.

Bug Reports

You can create a bug report at [ooRexx Bug Report](#)¹¹ page. Please try to provide as much information in the bug report as possible so that the developers can determine the problem as quickly as possible. Sample programs that can reproduce your problem will make it easier to debug reported problems.

Documentation Feedback

You can submit feedback for, or report errors in, the documentation at [ooRexx Documentation Report](#)¹² page. Please try to provide as much information in a documentation report as possible. In addition to listing the document and section the report concerns, direct quotes of the text will help the developers locate the text in the source code for the document. (Section numbers are generated when the document is produced and are not available in the source code itself.) Suggestions as to how to reword or fix the existing text should also be included.

Request For Enhancement

You can suggest ooRexx features at the [ooRexx Feature Requests](#)¹³ page.

Patch Reports

If you create an enhancement patch for ooRexx please post the patch using the [ooRexx Patch Report](#)¹⁴ page. Please provide as much information in the patch report as possible so that the developers can evaluate the enhancement as quickly as possible.

Please do not post bug fix patches here, instead you should open a bug report and attach the patch to it.

The ooRexx Forums

The ooRexx project maintains a set of forums that anyone may contribute to or monitor. They are located on the [ooRexx Forums](#)¹⁵ page. There are currently three forums available: Help, Developers and Open Discussion. In addition, you can monitor the forums via email.

6.2. The Rexx Language Association Mailing List

The [Rexx Language Association](#)¹⁶ maintains a mailing list for its members. This mailing list is only available to RexxLA members thus you will need to join RexxLA in order to get on the list. The dues for RexxLA membership are small and are charged on a yearly basis. For details on joining RexxLA please refer to the [RexxLA Home Page](#)¹⁷ or the [RexxLA Membership Application](#)¹⁸ page.

⁹ http://sourceforge.net/mail/?group_id=119701

¹⁰ http://sourceforge.net/mail/?group_id=119701

¹¹ http://sourceforge.net/tracker/?group_id=119701&atid=684730

¹² http://sourceforge.net/tracker/?group_id=119701&atid=1001880

¹³ http://sourceforge.net/tracker/?group_id=119701&atid=684733

¹⁴ http://sourceforge.net/tracker/?group_id=119701&atid=684732

¹⁵ http://sourceforge.net/forum/?group_id=119701

¹⁶ <http://www.rexxla.org/>

¹⁷ <http://rexxla.org/>

¹⁸ <http://www.rexxla.org/rexxla/join.html>

6.3. comp.lang.rexx Newsgroup

The [*comp.lang.rexx*](#)¹⁹ newsgroup is a good place to obtain help from many individuals within the Rexx community. You can obtain help on Open Object Rexx or on any number of other Rexx interpreters and tools.

¹⁹ <http://groups.google.com/group/comp.lang.rexx/topics?hl=en>

Brief Overview

ooSQLite is a direct interface to SQLite. SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database [engine](#).

SQLite is an [embedded](#) SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures.

ooSQLite provides both an object-orientated [interface](#) and a classic Rexx [interface](#) to SQLite. Some people refer to this as a *binding*. The binding allows a programmer to write applications using the Rexx programming language that access and use databases through the SQLite database engine.

1.1. Getting Started

If the user of ooSQLite is unfamiliar with SQLite itself, the best way to get started is to browse the SQLite [website](#)¹ to get a basic feel for what SQLite is.

To actually use the ooSQLite package in an ooRexx program, simply *require* the **ooSQLite.cls** in the program and ensure that **ooSQLite.cls** is in a location where the operating system can find it:

```
::requires 'ooSQLite.cls'
```

This is no different than using any other external package in ooRexx, such as **ooDialog.cls**, **rxftp.cls**, **cvsStream.cls**, etc..

Next, the user should familiarize herself with the example programs provided by the ooSQLite package. In the installation location of ooSQLite are the **examples** and **testing** subdirectories. These directories contain examples meant to be run from the command line. The user should run the examples after reading through the source code. The examples are meant to be played with. That is, the user should make small changes to the samples and observe what happens. As in any activity, experience is the best teacher.

Using the examples should be done in conjunction with reading this reference manual. The best advice would be to read the source code in an example and look up each class and method in the example in the reference. Or each function call in the example if the reader is more interested in the classic Rexx interface.

If things are unclear, then post a question to the ooRexx user's list. The list is easy to [subscribe](#)² to. In addition the [getting help](#) section of this reference lists a number of other resources for the programmer with questions about ooSQLite.

1.1.1. Command Line Shell

ooSQLite comes with a command line shell executable. **ooSQLite3.exe** on Windows and **oosqlite3** on Unix-like systems. This documentation will refer to the command line shell as

¹ <http://www.sqlite.org/index.html>

² http://sourceforge.net/mail/?group_id=119701

ooSQLite3.exe as opposed to *oosqlite3* because it is easy to distinguish that *ooSQLite3.exe* is an executable. In all cases, *ooSQLite3.exe* refers to both the Windows *and* Unix-like command line shell.

The command line shell is a simple program to use and will read any SQLite database file. It is extremely useful when first starting out with *ooSQLite*. Indeed, it is probably very useful at any time for anyone working with SQLite databases.

The executable is installed alongside the **ooSQLite.cls** file and the implementation library. The installation directory is normally added to the path, putting the command line shell also in the path. As long as it is in the path the command line shell can be executed from anywhere on the system.

Note that the *ooSQLite* command line shell is the same as the command line shell provided by SQLite. The shell source code is simply compiled as part of the *ooSQLite* build process and renamed. The source code is virtually unchanged, so *ooSQLite3.exe* behaves exactly the same as *sqlite3.exe*. At this time, the only change in *ooSQLite3.exe* is a few of the startup defaults.

Typically a command line shell is started up and presents some type of a prompt. The user then enters commands, the shell executes the command, and then returns to the prompt to await the next command. *ooSQLite3.exe* is no different. To start it type *ooSQLite3*, or *oosqlite3* at a command prompt:

```
C:\>ooSQLite3
SQLite version 3.7.13 2012-06-11 02:05:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"

sqlite>
```

The *.help* command gives help on how to use the shell. Typically, the shell is started with the name of a database and the shell opens that database:

```
C:\>oosQLite3 phoneBook.rdbx
SQLite version 3.7.13 2012-06-11 02:05:22
Enter ".help" for instructions
Enter SQL statements terminated with a ;;
sqlite> .tables
addr_type    contacts      inet_addr     phone_addr    street_addr
sqlite>
```

To exit the shell type *.exit*, or just *.*

```
sqlite> .
C:\>
```

As noted, *ooSQLite3.exe* is the same as the *sqlite3.exe* command line shell that can be downloaded from [SQLite³](#). There is a wealth of information on the Internet regarding the SQLite command shell program. Anything said about *sqlite3.exe* applies to *ooSQLite3.exe*. Have fun with *ooSQLite3.exe*, it is an useful tool.

1.2. Definition of Terms

A collection of definitions and explanations for terms used in the *ooSQLite* documentation. These terms may not be familiar to the average Rexx programmer.

³ <http://www.sqlite.org/download.html>

1.2.1. Database Name

The SQLite database [engine](#) tracks the databases opened on a database connection by internal names. The main database file opened on a connection has the name "main" and the name "temp" refers to the database used for temporary tables. However, SQLite also has the concept of *attaching* another database file to the current database connection. Attaching a database uses a SQL statement with the format of **ATTACH DATABASE expr AS database-name**. The expression after the AS keyword is the name used internally by the database engine for the attached database.

Many of the ooSQLite methods and functions have as an argument the database *name*. In these methods and functions, the *name* refers to the internal name used by SQLite, not the *file* name of the database. This documentation attempts to distinguish between a database *file* name and the internal database *name* by always using *file name* when referring to the actual database file.

1.2.2. Handle

A unique reference to a system resource managed by the operating system. In ooSQLite, most handles are references to data structures assigned by the SQLite database engine to manage its databases. These can be references to a database connection, a prepared statement, an online backup, etc.. Other handles not related to the database engine are also used by ooSQLite.

Handles are mostly used in the [classic](#) Rexx interface and are required as arguments for certain functions. Some handles are also used in the object orientated interface. A handle is an opaque type, the Rexx programmer need not be aware of the specific format of a handle. For the classical interface, the ooSQLite package provides functions that return handles. When a function requires a handle as an argument, the Rexx programmer needs to obtain the handle from one of those provided functions. In the object orientated interface the situation is similar, ooSQLite provides methods that return handles and methods that require handles as arguments. To obtain a handle required for a method argument, the Rexx programmer needs to obtain the handle from one of the provided methods.

As a general rule, handles obtained from functions in the classic Rexx interface should not be used as handle arguments to methods, and vice versa. However, as ooSQLite is enhanced, this may not be an absolute rule. The ooSQLite documentation will point out exceptions to this rule if there are any.

1.2.3. Threading Mode

SQLite itself supports three different threading modes:

Single-thread:

In this mode, all mutexes are disabled and SQLite is unsafe to use in more than a single thread at once.

Multi-thread:

In this mode, SQLite can be safely used by multiple threads provided that no single database connection is used simultaneously in two or more threads.

Serialized:

In serialized mode, SQLite can be safely used by multiple threads with no restriction.

SQLite allows the threading mode to be determined at compile-time, start-time, and run-time. However, the ooSQLite framework is compiled in serialized threading mode and, currently, provides no way to change this at start-time. Therefore, for all intents and purposes, ooSQLite uses the serialized threading mode.

It is possible to change the threading mode to multi-thread for individual database connections when [instantiating](#) a new connection object or [opening](#) the connection using the classic Rexx interface.

However, this is strongly advised against. The ooSQLite implementation is done using the assumption that the threading mode is always serialized.

Consult the SQLite documentation on the [threading mode](#)⁴ for more information.

1.3. Common Concepts

There are some common concepts shared throughout ooSQLite whether the programmer is using the object-orientated [interface](#) or the classic Rexx [interface](#). Some of these concepts are briefly discussed here.

1.3.1. Embedded

The ooSQLite package *embeds* the SQLite *database engine* within the package. *Embedding* indicates that the SQLite software code is compiled directly into the ooSQLite library. This in turn means that ooSQLite does not need any external piece to work. There is no need for a SQLite library to be installed on the system. There is no reason why a SQLite library can not be installed on the system, but if there is one, ooSQLite will not access it.

1.3.2. Database Engine

Database engine refers to the code that carries out the actual database function. For example, MySQL comes with a database engine, but it also comes with components that allow a server to serve up connections to the database engine, allow a client to contact the server, etc.. These components are not a part of the database engine of MySQL. ooSQLite works by making calls into the SQLite database engine, (which is embedded within ooSQLite.) Within this documentation, a reference to *the database engine* means the SQLite database engine. In places in the documentation, *the SQLite* database engine is stated explicitly. In places where the SQLite part is left off, the SQLite part is implicit.

1.4. SQLite Features

SQLite has some characteristics and features that are distinct from other database engines. Many of these features seem suitable for the types of applications ooRexx programmers want to write.

Zero-Configuration:

SQLite does not need to be "installed" before it is used. There is no "setup" procedure. There is no server process that needs to be started, stopped, or configured. There is no need for an administrator to create a new database instance or assign access permissions to users. SQLite uses no configuration files. Nothing needs to be done to tell the system that SQLite is running. No actions are required to recover after a system crash or power failure. There is nothing to troubleshoot.

SQLite just works.

Other more familiar database engines run great once you get them going. But doing the initial installation and configuration can be intimidatingly complex.

Serverless:

Most SQL database engines are implemented as a separate server process. Programs that want to access the database communicate with the server using some kind of interprocess

⁴ <http://www.sqlite.org/threadsafe.html>

communication (typically TCP/IP) to send requests to the server and to receive back results. SQLite does not work this way. With SQLite, the process that wants to access the database reads and writes directly from the database files on disk. There is no intermediary server process.

There are advantages and disadvantages to being serverless. The main advantage is that there is no separate server process to install, setup, configure, initialize, manage, and troubleshoot. This is one reason why SQLite is a "zero-configuration" database engine. Programs that use SQLite require no administrative support for setting up the database engine before they are run. Any program that is able to access the disk is able to use an SQLite database.

On the other hand, a database engine that uses a server can provide better protection from bugs in the client application - stray pointers in a client cannot corrupt memory on the server. And because a server is a single persistent process, it is able to control database access with more precision, allowing for finer grain locking and better concurrency.

Most SQL database engines are client/server based. Of those that are serverless, SQLite is probably the only one that allows multiple applications to access the same database at the same time.

Single Database File:

A SQLite database is a single ordinary disk file that can be located anywhere in the directory hierarchy. If SQLite can read the disk file then it can read anything in the database. If the disk file and its directory are writable, then SQLite can change anything in the database. Database files can easily be copied onto a USB memory stick or emailed for sharing.

Other SQL database engines tend to store data as a large collection of files. Often these files are in a standard location that only the database engine itself can access. This makes the data more secure, but also makes it harder to access. Some SQL database engines provide the option of writing directly to disk and bypassing the filesystem all together. This provides added performance, but at the cost of considerable setup and maintenance complexity.

Cross-Platform Database File:

The SQLite file format is cross-platform. A database file written on one machine can be copied to and used on a different machine with a different architecture. Big-endian or little-endian, 32-bit or 64-bit does not matter. All machines use the same file format. Furthermore, the developers have pledged to keep the file format stable and backwards compatible, so newer versions of SQLite can read and write older database files.

Most other SQL database engines require you to dump and restore the database when moving from one platform to another and often when upgrading to a newer version of the software.

Compact:

When optimized for size, the whole SQLite library with everything enabled is [footprint | less than 350KiB in size] (as measured on an ix86 using the "size" utility from the GNU compiler suite.) Unneeded features can be disabled at compile-time to further reduce the size of the library to under 190KiB if desired.

Most other SQL database engines are much larger than this. IBM boasts that its recently released CloudScape database engine is "only" a 2MiB jar file - an order of magnitude larger than SQLite even after it is compressed! Firebird boasts that its client-side library is only 350KiB. That's as big as SQLite and does not even contain the database engine. The Berkeley DB library from Oracle is 450KiB and it omits SQL support, providing the programmer with only simple key/value pairs.

Manifest typing:

Most SQL database engines use static typing. A datatype is associated with each column in a table and only values of that particular datatype are allowed to be stored in that column. SQLite

relaxes this restriction by using manifest typing. In manifest typing, the datatype is a property of the value itself, not of the column in which the value is stored. SQLite thus allows the user to store any value of any datatype into any column regardless of the declared type of that column. (There are some exceptions to this rule: An INTEGER PRIMARY KEY column may only store integers. And SQLite attempts to coerce values into the declared datatype of the column when it can.)

As far as the authors of SQLite can tell, the SQL language specification allows the use of manifest typing. Nevertheless, most other SQL database engines are statically typed and so some people feel that the use of manifest typing is a bug in SQLite. But the authors of SQLite feel very strongly that this is a feature. The use of manifest typing in SQLite is a deliberate design decision which has proven in practice to make SQLite more reliable and easier to use, especially when used in combination with dynamically typed programming languages such as Tcl and Python. This feature should go very well with ooRexx.

Variable-length records:

Most other SQL database engines allocated a fixed amount of disk space for each row in most tables. They play special tricks for handling BLOBS and CLOBS which can be of wildly varying length. But for most tables, if you declare a column to be a VARCHAR(100) then the database engine will allocate 100 bytes of disk space regardless of how much information you actually store in that column.

SQLite, in contrast, uses only the amount of disk space actually needed to store the information in a row. If you store a single character in a VARCHAR(100) column, then only a single byte of disk space is consumed. (Actually two bytes - there is some overhead at the beginning of each column to record its datatype and length.)

The use of variable-length records by SQLite has a number of advantages. It results in smaller database files, obviously. It also makes the database run faster, since there is less information to move to and from disk. And, the use of variable-length records makes it possible for SQLite to employ manifest typing instead of static typing.

Public domain

The source code for SQLite is in the public domain. No claim of copyright is made on any part of the core source code. (The documentation and test code is a different matter - some sections of documentation and test logic are governed by open-source licenses.) All contributors to the SQLite core software have signed affidavits specifically disavowing any copyright interest in the code. This means that anybody is able to legally do anything they want with the SQLite source code. This allows ooRexx to directly embed the database engine in the ooSQLite package.

There are other SQL database engines with liberal licenses that allow the code to be broadly and freely used. But those other engines are still governed by copyright law. SQLite is different in that copyright law simply does not apply.

The source code files for other SQL database engines typically begin with a comment describing your license rights to view and copy that file. The SQLite source code contains no license since it is not governed by copyright. Instead of a license, the SQLite source code offers a blessing:

May you do good and not evil

May you find forgiveness for yourself and forgive others

May you share freely, never taking more than you give.

SQL language extensions:

SQLite provides a number of enhancements to the SQL language not normally found in other database engines. The EXPLAIN keyword and manifest typing have already been mentioned

above. SQLite also provides statements such as REPLACE and the ON CONFLICT clause that allow for added control over the resolution of constraint conflicts. SQLite supports ATTACH and DETACH commands that allow multiple independent databases to be used together in the same query. And SQLite defines APIs that allows the user to add new SQL functions and collating sequences.

The Object Orientated Interface to SQLite

The object-orientated interface to SQLite provides a number of classes whose methods are used to work with SQLite databases.

The intent is for the object-orientated interface to allow access to the complete functionality and feature set of SQLite. The first release of ooSQLite will not meet, and is not intended to meet, that goal. Lesser used functionality will be added over time.

The object-orientated and *classic* Rexx interfaces are developed in tandem. As each new feature or functionality of SQLite is added to ooSQLite, access to the feature is added to both interfaces at the same time. There is no SQLite functionality in the object-orientated interface that can not be accessed through the classic Rexx interface. And, vice versa.

The following table lists the classes used in the object orientated interface of the ooSQLite package:

Table 2.1. ooSQLite Class Listing

Class	Description
The ooSQLite Class	The ooSQLite class provieds a number of <i>class</i> methods that are generally useful in working with SQLite databases. Many of the methods are used to query or set values in the database <i>engine</i> rather than an individual database.
The ooSQLiteBackup Class	An ooSQLiteBackup object provides methods for copying the content of one database into another. It is useful either for creating backups of databases or for copying in-memory databases to or from persistent files.
The ooSQLiteConnection Class	Each ooSQLiteConnection object represents an open connection to a SQLite database. Mulitple connections to the same database are allowed by instantiating multiple ooSQLiteConnection objects.
The ooSQLiteConstants Class	The ooSQLiteConstants class provides a CONSTANT value for each SQLite defined constant, and a small number of ooSQLite specific constants.
The ooSQLiteMutex Class	An ooSQLiteMutex object provides methods for allocating and using a mutex. In general a mutex is a synchronization object used in multi-threading programs to prevent different threads from accessing a shared resource at the same time.
The ooSQLiteStmt Class	Each ooSQLiteStmt object represents a single SQL statement, which is often referred to as a <i>prepared</i> statement.

Primary ooSQLite Classes

Most of the work to use the SQLite database engine is done through the primary ooSQLite classes documented in this chapter. SQLite also allows extensions to the database engine to be written by the user of the engine. An SQLite extension is typically a shared library or DLL and written in C / C++ code. ooSQLite provides full support for the loading of these extensions with a number of classes. Those classes are documented in their own [chapter](#).

As a note, support for user defined extensions written in Rexx are also part of ooSQLite. This support is done through the [createCollation](#) and [createFunction](#). In the future support for user defined virtual tables is intended to be added to ooSQLite through a [createModule](#) method.

The following table lists the primary classes used in the object orientated interface of the ooSQLite package that are documented in this chapter:

Table 3.1. ooSQLite Class Listing

Class	Description
The ooSQLite Class	The ooSQLite class provides a number of <i>class</i> methods that are generally useful in working with SQLite databases. Many of the methods are used to query or set values in the database <i>engine</i> rather than an individual database.
The ooSQLiteBackup Class	An ooSQLiteBackup object provides methods for copying the content of one database into another. It is useful either for creating backups of databases or for copying in-memory databases to or from persistent files.
The ooSQLiteConnection Class	Each ooSQLiteConnection object represents an open connection to a SQLite database. Multiple connections to the same database are allowed by instantiating multiple ooSQLiteConnection objects.
The ooSQLiteMutex Class	An ooSQLiteMutex object provides methods for allocating and using a mutex. In general a mutex is a synchronization object used in multi-threading programs to prevent different threads from accessing a shared resource at the same time.
The ooSQLiteStmt Class	Each ooSQLiteStmt object represents a single SQL statement, which is often referred to as a <i>prepared</i> statement.

3.1. The ooSQLite Class

The methods of the **ooSQLite** class consist entirely of class methods. These are utility methods that primarily deal with application-wide or process-wide settings, and the database engine itself, rather than specific databases.

3.1.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with the database engine using the **ooSQLite** class

Table 3.2. ooSQLite Methods and Attributes

Method	Documentation
Class Attribute Methods	
recordFormat	Sets or queries the default, process-wide format for returned result sets.

Method	Documentation
<code>null</code>	Reflects the default representation for the SQL NULL value that is returned by the interpreter for database values that are NULL.
Class Methods	
<code>compileOptionGet</code>	Returns the name of the <i>nth</i> compile time option if the option was used during the build of ooSQLite.
<code>compileOptionUsed</code>	Determines if the named compile-time option was used during the build of ooSQLite.
<code>complete</code>	Determines if the specified text seems to be a complete SQL statement.
<code>enquote</code>	Converts the supplied Rexx value(s) into SQL literals.
<code>enquote</code>	Retrieves the English language descriptive string for a result code.
<code>libVersion</code>	Returns the embedded SQLite library version as a string.
<code>libVersionNumber</code>	Returns the embedded SQLite library version as a number..
<code>memoryHighWater</code>	Returns the maximum value of the number of bytes of memory malloced but not freed (memory in use) of the database engine since the high-water mark was last reset.
<code>memoryUsed</code>	Returns the number of bytes of memory currently in use by the database engine.
<code>releaseMemory</code>	Causes the database engine to attempt to free the specified number of bytes of heap memory by deallocating non-essential memory.
<code>softHeapLimit64</code>	Imposes a limit on the heap size, or queries the current size.
<code>sourceID</code>	Returns the embedded SQLite library source ID string.
<code>sqlite3Version</code>	Returns the value of the SQLite C code <code>sqlite3_version[]</code> string constant.
<code>status</code>	Retrieves runtime status information about the performance of the database engine, and optionally resets various highwater marks.
<code>threadSafe</code>	Tests to see if the currently running database engine is using a library that is thread safe.
<code>version</code>	Produces a version string in the format specified.

3.1.2. null (Attribute)

```
>>--null-----><
>>--null = nullobj-----><
```

Reflects the default representation for the SQL NULL value that is returned by the interpreter for database values that are NULL.

null get:

Returns the current default object the interpreter uses for the SQL NULL value. If the programmer has not changed this attribute, its value is the `.nil` object.

null set:

Set this attribute to either the `.nil` object, or some alternative *string* value.

Remarks:

By default, ooSQLite uses the `.nil` object to represent the SQL NULL value. Queries for values stored in the database will return the `.nil` object for any value that is SQL NULL. However, by changing the value of the `null` attribute, the Rexx programmer can change the value the interpreter returns for NULL. Typically this would be done when the returned values are going to be displayed as text and the programmer would prefer to work with a string directly. Perhaps the value `NULL`, or `no value` would be assigned.

The `null` attribute of the **ooSQLite** class is the default value used for the entire application.

The `null` attribute of the **ooSQLiteConnection** object can be used to change the value on a database connection only. Likewise the `null` attribute of the **ooSQLiteStmt** can be used to change the value for a single statement only.

Note that this attribute does not affect the value the programmer must use to assign a SQL NULL to the database. The programmer must use the `.nil` object for that.

Details

Raises syntax errors when incorrect usage is detected.

This attribute is provided by ooSQLite, there is no similar feature provided by SQLite.

Example:

This example sets the `null` attribute of the **ooSQLite** class to `empty`. This allows the application to invoke the `left` method on the returned value without having to check that the return is the `.nil` object. Note that invoking the `left` method on the `.nil` object will raise a syntax condition:

```
.ooSQLite~null = 'empty'

dbConn = .ooSQLiteConnection~new(dbName, .ooSQLite~OPEN_READWRITE)

sql = 'SELECT * FROM foods ORDER BY name;'
stmt = .ooSQLiteStmt~new(dbConn, sql)

say stmt~columnName(1)~left(25) || stmt~columnName(2)~left(25) ||
stmt~columnName(3)~left(25)
say '='~copies(80)

do while stmt~step == stmt~ROW
    say stmt~columnText(1)~left(25) || stmt~columnText(2)~left(25) ||
stmt~columnText(3)~left(25)
end
```

3.1.3. recordFormat (Attribute)

```
>>--recordFormat----->
>>--recordFormat--varName----->
```

Sets or queries the default format for any result set produced by ooSQLite. The default format on startup is [OO_ARRAY_OF_DIRECTORIES](#).

recordFormat get:

The value of this attribute is one of the ooSQLite Result Set Format [Constants](#) that define how a result set is formatted. The default format can be over-ridden at the database [connection](#), [statement](#), or method level. If not over-ridden, all returned result sets for the entire process will have the format specified by this attribute.

recordFormat set:

The default format of a record set can be changed by setting this attribute to one of the ooSQLite Result Set Format [Constants](#).

Remarks:

There are currently 4 formats defined:

- OO_ARRAY_OF_DIRECTORIES - The result set is returned in an **Array** object. Each row (each record) in the result set is an item in that array. The rows 1 through N in the result set are the items 1 through N in the array. Each row (each record) is a **Directory** object. The directory object has a single item for each column in the row, where the index of the item is the name of the column and the value at that index is the value of the column.
- OO_ARRAY_OF_ARRAYS - The result set is returned in an **Array** object. Each row (each record) in the result set is an item in that array. Each item in the array is also an array. The first item in the array is an array of the column names for a row, with column name 1 through column name N as item 1 through N in the array. Each row in the result set is an array of the column values for the row, in column order. The first row in the result set will be the second item in the array. Thus the rows 1 through R in the result set will have indexes 2 through R + 1 in the array.
- OO_STEM_OF_STEMS - The result set is returned in a **Stem** object. Each row (each record) in the result set is also a stem. The stem for a row has a single tail for each column in the row, where the tail is the name of the column and the value of that tail is the value of the column. The returned stem has tails 0 through N, N equal to the number of rows in the result set. The value at tail 0 is the count of rows in the result set. The values at tail 1 through N are the stems for row 1 through N in the result set..
- OO_CLASSIC_STEM - The result set is returned in a **Stem** object. The returned stem has tails 0 through N, N equal to the number of rows in the result set. The value at tail 0 is the count of rows in the result set. Each tail 1 through N are compound tails that represent the rows in the result set. Each compound tail is the row number combined with a column name in the row. The value at that compound tail is the value of the named column in that row. E.g., for a result set that contains the 3 columns, **id**, **name**, and **type_id**, the values for row 1 in the result set would be at the stem variables **stm.1.id**, **stm.1.name**, and **stm.1.type_id**.

The format for a result set can also be set for an individual database connection. Use the **ooSQLiteConnection** object's [recordFormat](#) attribute, or the *defFormat* argument in the [new](#) method.

Details

This attribute is provided by ooSQLite, there is no similar feature provided by SQLite.

Example:

This example changes the default format for a result set to be an array of records where each record in the array is an array of the column values. The first item in the array is an array of the column names, and the records are the second through nth items, where n is the number of records:

```
-- Set the result set format to an array of arrays:  
.ooSQLite~recordFormat = .ooSQLite~OO_ARRAY_OF_ARRAYS  
  
dbName = 'ooFoods.rdbx'  
  
dbConn = .ooSQLiteConnection~new(dbName, .oosSQLite~OPEN_READWRITE)  
  
sql = 'SELECT * FROM foods ORDER BY name;'  
resultSet = dbConn~exec(sql, .true)
```

3.1.4. compileOptionGet

```
>> -compileOptionGet( --nth-- )-----><
```

Returns the name of the *nth* compile time option if the option was used during the build of ooSQLite.
Returns the string *NULL* if the *nth* compile time option was not used.

Arguments:

The single arguments is:

nth [required]

nth is the index of the compile time option. The database engine has a list of options that were set at compile time, 1 through N. This argument specifies which N is being queried.

Return value:

The name of the *nth* compile time option, or *NULL* if *nth* is out of range.

Remarks:

The *compileOptionGet* method allows one to iterate over the compile-time options that were defined during the build of the SQLite database engine. This is done by starting with index 1 and incrementing the index at each iteration until the string *NULL* is returned. This produces a list of the compile-time options that were specified at build time.

For most purposes, SQLite can be built just fine using the default compilation options. However, if required, compile-time options can be used to omit SQLite features (resulting in a smaller compiled library size) or to change the default values of some parameters. ooSQLite is built using just a few compile-time options.

Details

The functionality of the *compileOptionGet* method is similar to that of the SQLite [*sqlite3_compileoption_get*](#)¹ API.

Example:

This example shows how to iterate over the compile-time options:

```
opt = ''
do i = 1 while opt \== 'NULL'
    opt = .ooSQLite~compileOptionGet(i)
    say 'Compile option:' opt
end
say

/* Output might be:

Compile option: CURDIR
Compile option: ENABLE_COLUMN_METADATA
Compile option: ENABLE_MEMORY_MANAGEMENT
Compile option: TEMP_STORE=1
Compile option: THREADSAFE=1
Compile option: NULL

*/
```

¹ http://www.sqlite.org/c3ref/compileoption_get.html

3.1.5. compileOptionUsed

```
>>--compileOptionUsed(--name--)-----><
```

Determines if the named compile-time option was used during the build of ooSQLite.

Arguments:

The single argument is:

name [required]

The name of the SQLite compile-time option to check. The *SQLITE_* prefix may be omitted. I.e., *SQLITE_THREADSAFE* and *THREADSAFE* will produce the same result.

Return value:

Returns true if the name option was used, otherwise false.

Remarks:

For most purposes, SQLite can be built just fine using the default compilation options. However, if required, compile-time options can be used to omit SQLite features (resulting in a smaller compiled library size) or to change the default values of some parameters. ooSQLite is built using just a few compile-time options.

Details

The functionality of the *compileOptionUsed* method is similar to that of the SQLite *sqlite3_compileoption_used*² API.

Example:

This example checks several random compile-time options and displays if they were defined at compile time.

```
names = .array-of('DEFAULT_AUTOVACUUM', 'THREADSAFE', 'TEMP_STORE', -  
    '4_BYTE_ALIGNED_MALLOC', 'CURDIR', 'SQLITE_THREADSAFE')  
do i = 1 to names~items  
    say 'Option' names[i] ':' .ooSQLite~compileOptionUsed(names[i])  
end  
  
/* Output might be:  
  
Option DEFAULT_AUTOVACUUM : 0  
Option THREADSAFE : 1  
Option TEMP_STORE : 1  
Option 4_BYTE_ALIGNED_MALLOC : 0  
Option CURDIR : 1  
Option SQLITE_THREADSAFE : 1  
  
*/
```

3.1.6. complete

```
>>--complete(--text--)-----><
```

Determines if the specific text seems to form a complete SQL statement.

² http://www.sqlite.org/c3ref/compileoption_get.html

Arguments:

The arguments are:
 text [required]
 The text to check.

Return value:

Returns 0 if the text is incomplete and 1 if the text seems complete. Note that the database engine allocates some memory when this method is executed. If memory allocation fails, the database engine will return the NOMEM constant. This is very unlikely to happen.

Remarks:

This method is useful during command-line input to determine if the currently entered text seems to form a complete SQL statement or if additional input is needed before sending the text into ooSQLite for parsing. A statement is judged to be complete if it ends with a semicolon token and is not a prefix of a well-formed CREATE TRIGGER statement. Semicolons that are embedded within string literals or quoted identifier names or comments are not independent tokens (they are part of the token in which they are embedded) and thus do not count as a statement terminator. Whitespace and comments that follow the final semicolon are ignored.

Details

The functionality of the *complete* method is similar to that of the SQLite [sqlite3_complete](#)³ API.

Example:

This example show some possible output of using this method:

```
text = 'SELECT * from foods'
say 'Is "'text'" a complete SQL statement?' .ooSQLite~complete(text)

text = 'SELECT * from foods;'
say 'Is "'text'" a complete SQL statement?' .ooSQLite~complete(text)

say 'Value of NOMEM constant:' .ooSQLite~NOMEM

/* Output:

IS "SELECT * from foods" a complete SQL statement? 0
IS "SELECT * from foods;" a complete SQL statement? 1
Value of NOMEM constant: 7

*/
```

3.1.7. enquote

```
>>--enquote(--+-----+--)-->-
    +--values--+
```

Converts the supplied Rexx value(s) into SQL literals. This method is useful to help construct SQL statements. The specified Rexx object(s) are converted to SQL literals by adding single quotes to the beginning and end of the string value of the object, escaping single quotes within the string value of the object, and changing the **.nil** object to SQL NULL.

³ <http://www.sqlite.org/c3ref/complete.html>

Arguments:

The single argument is:

`values [optional]`

A Rexx object, or an array of Rexx objects, to be converted to SQL literals. If this argument is omitted then `NULL` is returned.

Return value:

The string value of the specified object(s) as a SQL literal, or a comma separated list, of SQL literals.

Remarks:

The `enquote` methods accepts a single argument, `values`. If `values` is an array whose size is N, then each item at index 1 through N is converted to a SQL literal. Any index 1 through N that is not assigned an item is converted to SQL `NULL`. Likewise, if an item 1 through N is the `.nil` object, that item is converted to SQL `NULL`. All other items are converted to the string value of the item, with a single quote added to the front and end of the string. If the string value contains any single quotes, those single quotes are escaped. If N is greater than 1, then each converted value is added to the returned string with a comma used as a separator.

If `values` is not an array, then it is taken to be a single value to be converted, and is converted in the same manner as a single item in an array is converted, as described above. This implies that if `values` is omitted altogether, it is converted to SQL `NULL`.

Details

The functionality of the `enquote` method is similar to that of the SQLite [sqlite3_mprintf](#)⁴ API.

Example:

This example shows how the `enquote` method can be used to create SQL INSERT statements that are not prone to SQL Injection flaws:

```
r1 = .array~new(4)
r1[1] = "Tom"
r1[2] = "Hanks"
r1[4] = "male"

r2 = .array-of("Mike", , "555-9988", .nil)

sql1 = "INSERT INTO my_table (fName, lName, phone, gender)
VALUES("ooSQLite~enquote(r1));"
sql2 = "INSERT INTO my_table (fName, lName, phone, gender)
VALUES("ooSQLite~enquote(r2));"

say sql1
say sql2

/* Output would be:

INSERT INTO my_table (fName, lName, phone, gender) VALUES('Tom', 'Hanks', NULL, 'male');
INSERT INTO my_table (fName, lName, phone, gender) VALUES('Mike', NULL, '555-9988',
NULL);
```

This example shows a conversion for a single string that has an apostrophe within it:

⁴ <http://www.sqlite.org/c3ref/mprintf.html>

```

str = "It's a happy day!"
say .ooSQLite~Enquote(str)

/* Output would be:

'It''s a happy day!

*/

```

3.1.8. errStr

```
>>--errStr(--resultCode--)-----><
```

Retrieves the English language descriptive string for a result code.

Arguments:

The arguments are:

resultCode [required]

One of the result *code* constants, or one of the ooSQLite specific result *code*, whose descriptive sting is to be retrieved.

Return value:

Returns the descriptive, English language, string for the specified *result code*.

Remarks:

This method is useful for getting the description of a result code without needing a *ooSQLiteConnection* or *ooSQLiteStmt* object. The *errMsg* method will return the descriptive string for the most recent result code associated with the database connection. The *errStr* method can be used at any time to get the descriptive string for a result code.

Details

The functionality of the *errStr* method is similar to that of the SQLite *sqlite3_errstr*⁵ API.

Example:

This example is a small snippet of code to print out the description of the first 27 result codes. Inspection of the **ooSQLite.cls** file shows us that the first 27 result code constants are in numerically consecutive order:

```

first = .ooSQLite~OK
last  = .ooSQLite~NOTADB

do i = first to last
    say .ooSQLite~errStr(i)
end
say

/* Output would be:

not an error
SQL logic error or missing database
unknown error
access permission denied

```

⁵ <http://www.sqlite.org/c3ref/errcode.html>

```

callback requested query abort
database is locked
database table is locked
out of memory
attempt to write a readonly database
interrupted
disk I/O error
database disk image is malformed
unknown operation
database or disk is full
unable to open database file
locking protocol
table contains no data
database schema has changed
string or blob too big
constraint failed
datatype mismatch
library routine called out of sequence
large file support is disabled
authorization denied
auxiliary database format error
bind or column index out of range
file is encrypted or is not a database

*/

```

3.1.9. libVersion

>>--libVersion-----><

Returns the embedded SQLite library version as a string.

Arguments:

There are no arguments to this method.

Return value:

Returns the library version number as a string.

Remarks:

This method can be used to determine exactly which version of the SQLite database engine is in use. It can also be used to verify that the *ooSQLite.cls* file matches the compiled library file, (*oosqlite.dll*, *oosqlite.so*, or *oosqlite.dylib* depending on the operating system.) The value returned by this method should be exactly equal to the the compile-time version *VERSION* constant. If it is not, the *ooSQLite* installation is incorrect.

Details

The functionality of the *libVersion* method is similar to that of the SQLite *sqlite3.libversion*⁶ API.

Example:

This example checks that **ooSQLite.cls** is loading the correct library and aborts if it is not:

```

if .ooSQLite~libVersion() \== .ooSQLiteConstants~VERSION then do
    say 'The ooSQLite class file and library file do NOT match.'
    say 'The ooSQLite extension is not installed correctly on this system.'

```

⁶ <http://www.sqlite.org/c3ref/libversion.html>

```

say 'Aborting application.'
return 99
end

```

3.1.10. libVersionNumber

```
>>--libVersionNumber-----><
```

Returns the embedded SQLite library version as a number.

Arguments:

There are no arguments for this method.

Return value:

The number of the library version in use, perhaps 3007013.

Remarks:

This method is similar to the [libVersion](#) method, returning a whole number rather than a dotted version string.

The value returned by this method should be exactly equal to the the compile-time version [VERSION_NUMBER](#) constant.

Details

The functionality of the *libVersionNumber* method is similar to that of the SQLite [sqlite3_libversion_number](#)⁷ API.

3.1.11. memoryHighWater

```
>>--memoryHighWater(---+-----+---)-----><
      +--reset--+
```

Returns the maximum value of the number of bytes of memory malloced but not freed (memory in use) of the database engine since the high-water mark was last reset.

Arguments:

The single argument is:

reset [optional]

True if the high-water mark should be reset, false if it should not be reset. The default if the argument is omitted is false.

Return value:

The highest amount of memory used by the database engine, since any previous reset, in bytes.

Remarks:

The memory high-water mark is reset to the current number of bytes of memory in use, if and only if the *reset* arg true. The value returned by this method is the high-water mark prior to the reset if a reset is done.

⁷ <http://www.sqlite.org/c3ref/libversion.html>

Details

The functionality of the *memoryHighWater* method is similar to that of the SQLite [*sqlite3_memory_highwater*](http://www.sqlite.org/c3ref/memory_highwater.html)⁸ API.

Example:

This example shows the effect of using / not using the *reset* argument:

```

say
say 'Memory high water:          ' .ooSQLite~memoryHighWater
say 'Memory high water:          ' .ooSQLite~memoryHighWater
say 'Memory high water (reset):' .ooSQLite~memoryHighWater(.true)
say 'Memory high water:          ' .ooSQLite~memoryHighWater
say

/* Output might be:

Memory high water:      93664
Memory high water:      93664
Memory high water (reset): 93664
Memory high water:      91712

-- Note that the return when the reset is done is the current high water mark, not the
-- value after the reset.
*/

```

3.1.12. memoryUsed

```
>>--memoryUsed-----><
```

Returns the number of bytes of memory currently in use by the database engine. That is, the number of bytes allocated and not yet freed.

Arguments:

There are no arguments to this method.

Return value:

The amount of memory the database is currently using, in bytes.

Details

The functionality of the *memoryUsed* method is similar to that of the SQLite [*sqlite3_memory_used*](http://www.sqlite.org/c3ref/memory_used.html)⁹ API.

3.1.13. releaseMemory

```
>>--releaseMemory(--count--)-----><
```

This method causes the database engine to attempt to free *count* bytes of heap memory by deallocating non-essential memory allocations held by the database library.

⁸ http://www.sqlite.org/c3ref/memory_highwater.html

⁹ http://www.sqlite.org/c3ref/memory_used.html

Arguments:

The single argument is:

count [required]

The number of bytes the database engine should try to release.

Return value:

Returns the number of bytes actually released, which may be more or less than requested.

Remarks:

Memory used to cache database pages to improve performance is an example of non-essential memory. This method would be a no-op returning zero if SQLite was not compiled with the `SQLITE_ENABLE_MEMORY_MANAGEMENT` compile-time option. However, ooSQLite does use that option when it is built, so this method is not a no-op.

Details

The functionality of the `releaseMemory` method is similar to that of the SQLite [`sqlite3_release_memory`¹⁰](#) API.

3.1.14. softHeapLimit64

```
>>--softHeapLimit64(--n--)-----><
```

Imposes a limit on the heap size, or queries the current size.

Arguments:

The single argument is:

TERM

A 64-bit signed whole number. If n is positive the soft heap limit is set to its value. If n is zero then the soft heap limit is disabled. If n is negative then no change is made to the current limit.

Return value:

On success, returns the size of the soft heap limit prior to the method invocation. On error a negative number is returned.

Remarks:

To query the current limit only, use a negative value for n .

The database engine strives to keep heap memory utilization below the soft heap limit by reducing the number of pages held in the page cache as heap memory usages approaches the limit. The soft heap limit is *soft* because even though the engine strives to stay below the limit, it will exceed the limit rather than generate a `NOMEM` error. In other words, the soft heap limit is advisory only.

Details

The functionality of the `softHeapLimit64` method is similar to that of the SQLite [`sqlite3_soft_heap_limit64`¹¹](#) API.

Example:

This example queries the current soft heap limit, sets the limit, and queries it again. It also displays the current memory usage.

¹⁰ http://www.sqlite.org/c3ref/release_memory.html

¹¹ http://www.sqlite.org/c3ref/soft_heap_limit64.html

```

say 'Current soft heap limit:' .ooSQLite~softHeapLimit64(-1)
say 'Setting soft heap limit:' .ooSQLite~softHeapLimit64(64000000)
say 'Current soft heap limit:' .ooSQLite~softHeapLimit64(-1)
say 'Current memory used:'     .ooSQLite~memoryUsed

/* Output might be:

Current soft heap limit: 0
Setting soft heap limit: 0
Current soft heap limit: 64000000
Current memory used:     0

*/

```

3.1.15. sourceID

>>--sourceID-----><

Returns the embedded SQLite library source ID string.

Arguments:

There are no arguments to this method.

Return value:

The source ID string of the running database engine.

Remarks:

This method is similar to the [libVersion](#) method, but rather than a dotted version string it returns an unique identifier that contains the date and time of the library source code release and a globally unique nubmer hexadecimal number.

The value returned by this method should be exactly equal to the the compile-time version [SOURCE_ID](#) constant.

Details

The functionality of the `sourceID` method is similar to that of the SQLite [sqlite3_sourceid](#)¹² API.

Example:

This example displays the source ID for the currently running database engine:

```

say 'Source ID:'           .ooSQLite~sourceID

/* Output might be:

Source ID: 2012-06-11 02:05:22 f5b5a13f7394dc143aa136f1d4fab6839eaa6dc

*/

```

3.1.16. sqlite3Version

¹² <http://www.sqlite.org/c3ref/libversion.html>

```
>>--sqlite3Version-----><
```

The *sqlite3Version* method returns the value of the SQLite C code `sqlite3_version()` string constant.

Arguments:

There are no arguments to this method.

Return value:

The SQLite version string.

Remarks:

This method is included for completeness. The string returned is exactly the same as the string returned from *libVersion*.

Details

The value returned by the *sqlite3Version* method is the value of the SQLite `sqlite3_version`¹³ constant string.

Example:

This example shows the equivalence of the two **ooSQLite** class methods:

```
say 'Return from libVersion method:' .ooSQLite~libVersion
say 'Constant string value:'           .ooSQLite~sqlite3Version

/* Output would be, depending on the exact library version of the embedded database
   engine:

Return from libVersion method: 3.7.13
Constant string value:          3.7.13

*/
```

3.1.17. status

```
>>--status(--optParam-, -result--+-----+--)><
      +-,-reset--+
```

Retrieves runtime status information about the performance of the database engine, and optionally resets various highwater marks.

Arguments:

The arguments are:

`optParam` [required]

One of the runtime `status` parameter constants. This specifies which status parameter information is required.

`result` [required IN/OUT]

A **Directory** object in which the retrieved information is returned. On a successful return the directory object will contain the following two indexes:

CURRENT:

The current value of the parameter specified by *optParam*.

¹³ <http://www.sqlite.org/c3ref/libversion.html>

HIGHWATER:

The highest recorded value of the parameter specified by *optParam*.

reset [optional]

A logical value, true or false, to specify whether the high water mark should be reset. The default if omitted is false.

Return value:

Returns one of the ooSQLite *result* code constants. OK on success, otherwise an error code. On error no indexes of the *result* object are set.

Remarks:

Some parameters do not record the highest value. Other parameters record only the highwater mark and not the current value. For parameters that do not record the highest value, the *reset* argument is ignored.

Details

The functionality of the *status* method is similar to that of the SQLite *sqlite3_status*¹⁴ API.

Example:

This example prints out all the status parameters:

```
a = .array~of(
    .ooSQLite~STATUS_MEMORY_USED      '-'
    .ooSQLite~STATUS_PAGECACHE_USED   '-'
    .ooSQLite~STATUS_PAGECACHE_OVERFLOW '-'
    .ooSQLite~STATUS_SCRATCH_USED    '-'
    .ooSQLite~STATUS_SCRATCH_OVERFLOW '-'
    .ooSQLite~STATUS_MALLOC_SIZE     '-'
    .ooSQLite~STATUS_PARSER_STACK    '-'
    .ooSQLite~STATUS_PAGECACHE_SIZE   '-'
    .ooSQLite~STATUS_SCRATCH_SIZE    '-'
    .ooSQLite~STATUS_MALLOC_COUNT    '-'
)

n = .array~of(
    'STATUS_MEMORY_USED'      '-'
    'STATUS_PAGECACHE_USED'   '-'
    'STATUS_PAGECACHE_OVERFLOW' '-'
    'STATUS_SCRATCH_USED'    '-'
    'STATUS_SCRATCH_OVERFLOW' '-'
    'STATUS_MALLOC_SIZE'     '-'
    'STATUS_PARSER_STACK'    '-'
    'STATUS_PAGECACHE_SIZE'   '-'
    'STATUS_SCRATCH_SIZE'    '-'
    'STATUS_MALLOC_COUNT'    '-'
)

values = .directory~new
do i = 1 to a~items
    .ooSQLite~status(a[i], values)
    say n[i]': current:' values~current~left(10) 'high water:' values~highwater
end

/* Output might be:
   STATUS_MEMORY_USED      : current: 88864      high water: 93664
```

¹⁴ <http://www.sqlite.org/c3ref/status.html>

```

STATUS_PAGECACHE_USED      : current: 0           high water: 0
STATUS_PAGECACHE_OVERFLOW: current: 16288        high water: 16288
STATUS_SCRATCH_USED       : current: 0           high water: 0
STATUS_SCRATCH_OVERFLOW   : current: 0           high water: 0
STATUS_MALLOC_SIZE         : current: 512          high water: 64000
STATUS_PARSER_STACK        : current: 0           high water: 0
STATUS_PAGECACHE_SIZE     : current: 1272         high water: 1272
STATUS_SCRATCH_SIZE        : current: 0           high water: 0
STATUS_MALLOC_COUNT        : current: 95          high water: 105
*/
```

3.1.18. threadSafe

```
>>--threadSafe-----><
```

Tests to see if the currently running database engine is using a library that is thread safe.

Arguments:

There are no arguments for this method.

Return value:

The return is the value of the compile-time option SQLITE_THREADSAFE, 0, 1, or 2.

Remarks:

Currently, ooSQLite is compiled with SQLITE_THREADSAFE=1 and therefore the return from this method is always 1. When SQLITE_THREADSAFE is set to 1, as it is in ooSQLite, then SQLite uses its *Serialized* threading model. In serialized mode, SQLite can be safely used by multiple threads with no restriction.

Details

The functionality of the *threadSafe* method is similar to that of the SQLite [sqlite3_threadsafe](#)¹⁵ API.

Example:

This example shows the return from *threadSafe*:

```

say 'Thread safe value:' .ooSQLite~threadSafe
/* Output will be:
Thread safe value: 1
*/
```

3.1.19. version

```
>>--version(--+---+---)-----><
    +-type--+
```

Produces a version string in the format specified by the *type* argument.

¹⁵ <http://www.sqlite.org/c3ref/threadsafe.html>

Arguments:

The single argument is:

type [optional]

If not omitted, exactly one of the following keywords, case is not significant and only the first letter is needed:

Compact
Full

OneLine
LibVersion

Number
SourceID

Compact

The compact format is a simple dotted version, similar to **1.0.0.7925**

Full

Produces a long version listing of several lines. The ooSQLite version, the ooRexx version, and the SQLite version are all shown. The example below has a sample output of this format.

Oneline

A version string similar to **ooSQLite Version 1.0.0.7925 (64 bit)**. This is the default if the *type* argument is omitted.

LibVersion

A version string similar to **3.7.13**. This is the embedded SQLite database engine version and is identical to the output from the *libVersion* method.

Number

A version number, similar to **3007013**. This is the embedded SQLite database engine version number and is identical to the output from the *libVersionNumber* method.

SourceID

A version string similar to **2012-06-11 02:05:22**

f5b5a13f7394dc143aa136f1d4faba6839eaa6dc. This is the embedded SQLite database engine source ID and is identical to the output from the *sourceID* method. text

Return value:

Returns a string formatted as specified above.

Details

This attribute is provided by ooSQLite, there is no similar feature provided by SQLite.

Example:

This example prints out the full format version string:

```
say 'ooSQLite version (full)'; say
say .ooSQLite~version('F')

/* Output might be:

ooSQLite version (full):

ooSQLite: ooSQLite Version 1.0.0.7925 (64 bit)
          Built Jun 19 2012 12:40:58
          Copyright (c) RexxLA 2012-2012.
          All Rights Reserved.

Rexx:    Open Object Rexx Version 4.2.0

SQLite:  SQLite Library Version 3.7.13
```

2012-06-11 02:05:22

*/

3.2. The ooSQLiteBackup Class

One feature of SQLite is an online backup API. The *online* part means that a database can be backed up while it is in use. The backup API copies the content of the source database into the destination database, overwriting the original contents of the destination database. It is useful either for creating backups of databases or for copying in-memory databases to or from persistent files.

The copy operation may be done incrementally, in which case the source database does not need to be locked for the duration of the copy, only for the brief periods of time when it is actually being read from. This allows other database users to continue uninterrupted while a backup of an online database is made.

The **ooSQLiteBackup** class provides a complete interface to the SQLite backup API. The authoritative [documentation](#)¹⁶ for using the online backup API is the SQLite documentation. The basic process to perform a backup using the **ooSQLiteBackup** object is as follows:

- Initialize the backup by *instantiating* a new backup object with the source and destination databases.
- Use the *step* method to copy some or all of the pages of the source database to the destination database.
- Repeat invocations of the *step* method until all pages are copied, or a fatal error occurs, or it is determined the backup should be abandoned.
- Invoke the *finish* method to release the system resources used for the backup.

By using a backup object, ooSQLite is able to optimize this process a little for the Rexx programmer. During the *step* method, when it is determined that all the pages have been copied successfully, or that a fatal error has occurred, the *finish* method is invoked automatically. This means the programmer only needs to use *finish* when it is determined that the backup should be abandoned before it is done.

The source database can be accessed while the backup is in progress. It is only locked while the backup is reading from the database, it is not locked continuously for the entire backup operation. This implies that the source database is more accessible when a smaller number of pages are copied during each *step*.

When the source database is in use while the backup is in progress, if the database is written to, the database engine may restart the backup. Whether or not the backup process is restarted as a result of writes to the source database mid-backup, the user can be sure that when the backup operation is completed the backup database contains a consistent and up-to-date snapshot of the original. However, if the source database is big and the backup gets restarted often, it is possible that the backup will never finish. This would be a case where it might be needed to abandon the backup.

3.2.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with backup objects using the **ooSQLiteBackup** class.

¹⁶ http://www.sqlite.org/c3ref/backup_finish.html#sqlite3backupinit

Table 3.3. ooSQLiteBackup Methods and Attributes

Method	Documentation
Class Methods	
new	Instantiates a new ooSQLite backup object.
Attribute Methods	
finished	Reflects the finished state of the backup, that is, if finish has been invoked.
initCode	Used to check if the backup object was initialized correctly during new .
lastErrCode	Reflects the value of the last recorded SQLite result code.
lastErrmsg	An English text message describing, to a degree, the meaning of the result code contained in the lastErrCode attribute.
pageCount	Reflects the total number of pages in the source database.
remaining	Reflects the number of pages still to be backed up.
saveDestConn	Determines if ooSQLite will close the destination database connection automatically during finish .
Instance Methods	
finish	Releases all resources associated with the backup operation.
getDestConn	Retrieves the destination database connection after the backup has finished..
step	Copies the specified number of pages from the source database to the destination database of this backup object.

3.2.2. new (Class method)

Instantiates and initializes a new backup object.

Arguments:

The arguments are:

srcDb [required]

Specifies the source database for the backup. This argument must be a database [connection](#) object that is open and not in an error state.

dstDb [required]

Specifies the destination database. This argument can either be an open database *connection* or it specifies the file name for the database. If the argument is a database connection, it must be an open connection that is not in an error state. If the argument specifies a file name, then a destination database *connection* object is instantiated using the specified name.

To specify the destination database as a file name use either a string or a **File** object. When a **File** object is used, case it is treated exactly if it was a string file name and all remarks concerning a file name versus a database connection apply. When specified as a **File** object, the `absolutePath()` method of the object is used to obtain the database file name. When specified as a string, the name is used as is, implying it could be a relative file name. I.e., `myBackup.db` would be created in the current directory.

Normally, when this argument is a file name, the connection object is closed during the [finish](#) method. This behavior can be changed either by setting the `save` argument to true, or by setting the `saveDestConn` attribute to true at any time prior to invoking [finish](#). Both the argument and the attribute default to false.

save [optional]

If `save` is true **and** `dstDB` is a file name, then the destination database *connection* will not be closed during [finish](#). Normally when the destination database is specified by a file name, the connection is closed during [finish](#).

If the `dstDb` argument is a database connection, this argument is ignored completely. The programmer is responsible for closing the connection.

srcName [optional]

The source database name. This is not the database *file* name, but rather the *main*, *temp*, or *attached as*, name. If this argument is omitted, the name is set to *main*.

dstName [optional]

The destination database name. Again, this is not the database *file* name, but rather the *main*, *temp*, or *attached as*, name. If this argument is omitted, the name is set to *main*.

If the `dstDb` argument is a file name rather than a database connection, this argument is ignored completely. In this case the only possible name is *main* and ooSQLite sets that internally when it instantiates the database connection object.

Return value:

Returns the newly instantiated backup object. If an error occurs during initialization, the [finish](#) method will have been invoked and the object can not be used to perform a backup. Check the `initCode`, `lastErrCode`, or `lastErrMsg` attributes to determine if errors have occurred.

Remarks:

The SQLite doc says: *The application must guarantee that the destination database connection is not passed to any other API (by any thread) after sqlite3_backup_init() is called and before the corresponding call to sqlite3_backup_finish(). SQLite does not currently check to see if the application incorrectly accesses the destination database connection and so no error code is reported, but the operations may malfunction nevertheless. Use of the destination database connection while a backup is in progress might also cause a mutex deadlock.*

In ooSQLite, the destination database *connection* object will raise a syntax condition if any of the methods of the object are invoked between the time the `new` method of the **ooSQLiteBackup** object is invoked and the [finish](#) method is invoked. This prevents malfunctions and deadlock.

For the backup to work effectively the source and destination database connections should have a busy [handler](#) or a busy [timeout](#) handler. This prevents a possible cause of failure of the backup. With an argument to `new` that is a database *connection*, the programmer is responsible for configuring the connection correctly, the backup object does not fiddle with the connection. When the `dstDb` argument is a file name, the database *connection* object is instantiated internally by ooSQLite. In this case, ooSQLite will add a busy timeout handler of 3 seconds.

Usually, it does not matter if the page-sizes of the source database and the destination database are different before the contents of the destination are overwritten. The page-size of the destination database is simply changed as part of the backup operation. The exception is if the destination database happens to be an in-memory database. In this case, if the page sizes are not the same at the start of the backup operation, then the operation fails with a SQLITE_READONLY error.

This second possible cause of failure can be prevented by setting the page-size of the in-memory database to the same size as that of the source database. When the *dstDB* argument is `:memory:` then ooSQLite will read the page-size of the source database, open a new in-memory database connection, and set its page size to match the source database page-size. Page-size can only be changed in a database before anything is put in it. If *dstDB* is passed in as a connection to an in-memory database, then the programmer is responsible for correctly setting the page-size.

Details

The functionality of the *new* method is similar to that of the SQLite [sqlite3_backup_init](#)¹⁷ API. Note, however that the arguments to *new* have been re-ordered so that the optional arguments come last.

Example:

This example loads a database from disk into an in-memory database and exits if there is an error in initialization:

```
srcConn = .ooSQLiteConnection~new("contacts.rdbx")

srcConn~busyTimeout(3000) -- 3 seconds.

bu = .ooSQLiteBackup~new(srcConn, ":memory:", .true)
if bu~initCode <> bu~OK then do
    say 'Error opening backup object:' bu~lastErrCode bu~lastErrMsg
    srcConn~close
    return 99
end
```

3.2.3. finished (Attribute)

```
>>--finished-----><
>>--finished--varName-----><
```

This attribute can be used to determine if *finish* has been invoked on the backup object.

finished get:

If *finished* is true the backup is finished and its resources have been released. If false the backup is still in progress.

finished set:

The programmer can not set the *finished* attribute. It is set internally by ooSQLite.

Remarks:

If *finished* is true, other methods of the backup object that access the database engine can not be invoked. Those are the *finish* and *step* methods, and the *pageCount* and *remaining* attributes.

The *finished* attribute can be accessed at any time, before or after *finish* has been invoked.

3.2.4. initCode (Attribute)

¹⁷ http://www.sqlite.org/c3ref/backup_finish.html#sqlite3backupinit

```
>>--initCode-----><
>>--initCode--varName-----><
```

The *initCode* attribute is used to check if the backup object is initialized correctly during [new](#).

initCode get:

The value of the *initCode* attribute reflects the state of the initialization of the backup object. It is set during [new](#) and is not changed afterwards.

initCode set:

The programmer can not set the *initCode* attribute. It is set internally by ooSQLite.

Remarks:

When the *initCode* is not 0 (. **ooSQLite~OK**) an error in initialization has occurred and a backup operation is not possible. The [finish](#) method will already have been invoked, the programmer does not need to call [finish](#) to clean up resources.

In most cases the *initCode* value will be a SQLite [result](#) code and the same as the [lastErrCode](#) attribute. However it also could be one of the ooSQLite specific [result](#) codes, either **OO_UNEXPECTED_RESULT** or **OO_BACKUP_DB_ERRSTATE**.

The *initCode* attribute can be accessed at any time, before or after [finish](#) has been invoked.

Details

This attribute is provided by ooSQLite, there is no similar feature provided by SQLite.

Example:

This example instantiates an **ooSQLiteBackup** object and checks the *initCode* value to be sure it is safe to proceed:

```
buObj = .ooSQLiteBackup~new(srcConn, dstConn)
if buObj~initCode <> buObj~OK then do ...
```

3.2.5. lastErrCode (Attribute)

```
>>--lastErrCode-----><
>>--lastErrCode--varName-----><
```

The *lastErrCode* reflects the value of the last recorded SQLite [result](#) code.

lastErrCode get:

The *lastErrCode* attribute reflects the value of the last error code recorded by the SQLite database engine during the backup operation. It is set during initialization, (during [new](#),) and is also updated during an invocation of [step](#) and [finish](#).

lastErrCode set:

The programmer can not set the *lastErrCode* attribute. It is set internally by ooSQLite.

Remarks:

During a backup operation, the database engine sets error codes in the destination database connection. The *lastErrCode* attribute is the value of the error code in the destination database connection. If the programmer has a reference to the destination database connection, that

reference can be used to get the same value through the `errorCode` method. However, if the programmer initializes the backup object using the file name of the destination database she may not have a reference to that database connection.

The `lastErrorCode` attribute can be accessed at any time, before or after `finish` has been invoked.

Details

The functionality of the `lastErrorCode` attribute is similar to that of the SQLite `sqlite3_errCode`¹⁸ API when used with the destination database connection.

3.2.6. lastErrMsg (Attribute)

```
>>--lastErrMsg-----><
>>--lastErrMsg==varName-----><
```

The `lastErrMsg` attribute contains an English text message describing, to a degree, the meaning of the result code contained in the `lastErrorCode` attribute.

lastErrMsg get:

The value of this attribute is a string message specific to the value of the last error result code. For example if the value of the `lastErrorCode` is `.ooSQLite~OK`, the text message would be *no error*.

The `lastErrMsg` attribute is updated every time the value of the `lastErrorCode` is changed.

lastErrMsg set:

The programmer can not set the `lastErrMsg` attribute. It is set internally by ooSQLite.

Remarks:

During a backup operation, the database engine sets error codes and messages in the destination database connection. The `lastErrMsg` attribute is the value of the error message in the destination database connection. If the programmer has a reference to the destination database connection, that reference can be used to get the same value through the `errMsg` method. However, if the programmer initializes the backup object using the file name of the destination database he may not have a reference to that database connection.

The `lastErrMsg` attribute can be accessed at any time, before or after `finish` has been invoked.

Details

The functionality of the `lastErrMsg` attribute is similar to that of the SQLite `sqlite3ErrMsg`¹⁹ API when used with the destination database connection.

3.2.7. pageCount (Attribute)

```
>>--pageCount-----><
>>--pageCount==varName-----><
```

The `pageCount` attribute reflects the total number of pages in the source database file.

¹⁸ <http://www.sqlite.org/c3ref/errcode.html>

¹⁹ <http://www.sqlite.org/c3ref/errcode.html>

pageCount get:

The value of this attribute is the number of pages in the source database file as reported by SQLite. The database engine only updates this value during a *step* operation. If the source database is modified during a backup operation, then the value is not updated to account for the size of the source database file changing.

pageCount set:

The programmer can not set the *pageCount* attribute. It is set internally by ooSQLite.

Remarks:

The *pageCount* and *remainung* attributes can be used to determine the progress of the backup. The percentage completion of the backup process may be calculated as:

$$\text{Completion} = 100\% * (\text{buObj-pagecount} - \text{buObj-remaining}) / \text{buObj-pagecount}$$

The database engine reports the page count and remaining values stored by the previous step operation, it does not actually inspect the source database file. This means that if the source database is written to by another thread or process after the call to *step* returns but before the values returned by the *pageCount* and *remaining* attributes are used, the values may be technically incorrect. This is not usually a problem.

The *pageCount* attribute must not be accessed after *finish* has been invoked. It calls into the database engine and the resources allowing that call have been released.

Details

The functionality of the *pageCount* attribute is similar to that of the SQLite *sqlite3_backup_pagecount*²⁰ API.

3.2.8. remaining (Attribute)

```
>>--remaining-----><
>>--remaining--varName-----><
```

The *remaining* attribute reflects the number of pages still to be backed up

remaining get:

The value of this attribute is the number of pages still to be backed up as reported by SQLite. The database engine only updates this value during a *step* operation. If the source database is modified during a backup operation, then the value is not updated to account for any extra pages that need to be updated.

remaining set:

The programmer can not set the *remaining* attribute. It is set internally by ooSQLite.

Remarks:

The *remaining* and *pageCount* attributes can be used to determine the progress of the backup. The percentage completion of the backup process may be calculated as:

$$\text{Completion} = 100\% * (\text{buObj-pagecount} - \text{buObj-remaining}) / \text{buObj-pagecount}$$

²⁰ http://www.sqlite.org/c3ref/backup_finish.html#sqlite3backupfinish

The database engine reports the page count and remaining values stored by the previous step operation, it does not actually inspect the source database file. This means that if the source database is written to by another thread or process after the call to *step* returns but before the values returned by the *remaining* and *pageCount* attributes are used, the values may be technically incorrect. This is not usually a problem.

The *remaining* attribute must not be accessed after *finish* has been invoked. It calls into the database engine and the resources allowing that call have been released.

Details

The functionality of the *remaining* attribute is similar to that of the SQLite *sqlite3_remaining*²¹ API.

3.2.9. saveDestConn (Attribute)

```
>>--saveDestConn----->
>>--saveDestConn--varName----->
```

The *saveDestConn* attribute allows the programmer to change the default behavior of the **ooSQLiteBackup** object during *finish*. This behavior is to automatically close the destination connection if it was opened the connection during *new*.

saveDestConn get:

The value of the *saveDestConn* is false if the ooSQLite framework is going to close the destination database connection, **that it opened**, during *finish* and true if the framework is not going to close the connection. The ooSQLite framework **never** closes a connection it did not open. This implies that the value of the *saveDestConn* is ignored unless the destination connection was opened internally by ooSQLite.

saveDestConn set:

The programmer can set the value of the *saveDestConn* to true or false. Attempting to set the value to anything else will raise a syntax condition.

Remarks:

The *saveDestCon* attribute can be accessed at any time, before or after *finish* has been invoked. Note that setting the attribute after *finish* has been invoked has no effect.

Details

This attribute is provided by ooSQLite, there is no similar API provided by SQLite.

Example:

This example loads a database from disk into an in-memory database. It uses the *saveDestConn* attribute to prevent ooSQLite from closing the in-memory connection and uses the *getDestConn* method to retrieve the database connection:

```
bu = .ooSQLiteBackup~new(srcConn, ":memory:")
if bu~initCode <> bu~OK then do
    -- handle error ...
end

bu~saveDestConn = .true
```

²¹ http://www.sqlite.org/c3ref/backup_finish.html#sqlite3backupfinish

```

bu~step(-1)

memConn = bu~getDestConn
...

```

3.2.10. finish

```
>>--finish-----><
```

Releases all resources associated with the backup operation.

Arguments:

There are no arguments to this method

Return value:

Returns .ooSQLite~OK if no errors occurred during a *step* invocation, whether or not the backup operation completed. If an out-of-memory condition or IO error occurred during any prior invocation of *step* on this **ooSQLiteBackup** object, then *finish* returns the corresponding *error* code.

Remarks:

There should be exactly one invocation of *finish* for each successful invocation of *new*. Note that during *step*, if the backup finishes successfully, or a fatal error occurs, *finish* is invoked automatically by ooSQLite. Thus, the programmer should only invoke *finish* to abandon (halt) the backup before it is finished.

Details

The functionality of the *finish* method is similar to that of the SQLite [sqlite3_backup_finish](#)²² API.

Example:

This example shows a online backup in progress. The source database is in use in a busy application. The backup operation is expected to complete in less than 4 hours. If it does not, the operation is abandoned and the application reschedules the backup for another time:

```

count = 0
do while .true
    ret = bu~step(2)
    if ret == bu~DONE then leave

    if ret <> bu~OK, ret <> bu~BUSY, ret <> bu~LOCKED then do
        say 'Fatal error during back up:' bu~lastErrCode bu~lastErrMsg
        leave
    end

    j = SysSleep(.5)
    count += 1

    if count > (count * 2 * 60 * 60 * 4) then do
        say "Backup has not completed in 4 hours, going to abandon the operation."
        bu~finish
        leave
    end
end

```

²² http://www.sqlite.org/c3ref/backup_finish.html#sqlite3backupfinish

3.2.11. getDestConn

```
>>--getDestConn-----><
```

The `getDestConn` method can be used to obtain the destination database `connection`, under certain circumstances, after the backup has finished.

Arguments:

There are no arguments to this method.

Return value:

Returns the destination database connection if the following conditions are met, otherwise returns the `.nil` object.

- The destination database was specified as a file name in the `new` method.
- The `saveDestConn` attribute has been set to true.
- The `finish` method has been invoked on this backup object.

Details

This attribute is provided by ooSQLite, there is no similar API provided by SQLite.

Example:

This example loads a database from disk into an in-memory database. It sets the `saveDestConn` attribute to true to prevent ooSQLite from closing the in-memory connection. It then uses the `getDestConn` method to retrieve the database connection:

```
bu = .ooSQLiteBackup~new(srcConn, ":memory:")
if bu~initCode <> bu~OK then do
    -- handle error ...
end

bu~saveDestConn = .true
bu~step(-1)

memConn = bu~getDestConn
...
```

3.2.12. step

```
>>--step(--+-----+---)-----><
    +-count--+
```

Copies up to `count` pages between the source and destination databases of this backup object.

Arguments:

The single argument is:

`count` [optional]

The number of pages in the source database to copy to the destination. If this argument is negative, then all `remaining` pages are copied. If this argument is omitted, `count` defaults to 5.

Return value:

If `count` pages are successfully copied, and there are still more pages to be copied, then `OK` (`.ooSQLite~OK`) is returned. If `step` successfully finishes copying all pages from source to

destination, then DONE (.ooSQLite~DONE) is returned. Otherwise an error code is returned. Some errors are fatal and some are not. The remarks section further discusses this.

Remarks:

If the database engine can not obtain a required lock than *step* returns BUSY (.ooSQLite~BUSY). If the source database connection is being used to write to the source database when *step* is invoked, then LOCKED is returned. The return code can also be NOMEM, READONLY, or one of the IO_ERR_XXX codes. After BUSY or LOCKED, *step* can be tried again. But NOMEM, READONLY, and IO_ERR_XXX are considered fatal. There is no point in retrying if any of those codes are returned. The application must accept that the backup operation has failed and invoke *finish* to release associated resources.

Internally, when either DONE or a fatal error return is detected, ooSQLite invokes *finish*. The programmer does not need to, and should not invoke *finish* after *step* returns any of those codes.

Details

The functionality of the *step* method is similar to that of the SQLite [sqlite3_backup_step](#)²³ API.

Example:

This example backs up a small database. Since the database is small, it simple copies all the pages at one time:

```
...
buObj = .ooSQLiteBackup~new(srcConn, dstConn)

if buObj~initCode == buObj~OK then ret = buObj~step(-1)
else ret = buObj~lastErrCode

if ret <> buObj~DONE then do
  -- back up failed, handle error
  ...
end

return 0
```

3.3. The ooSQLiteConnection Class

An **ooSQLiteConnection** object represents a SQLite database, or to be more precise a connection to a SQLite database file. Each database in SQLite is contained in a single file. The files are platform independent, that is a database created and used on a Windows system can be moved to a Linux or Mac OS X system and will work unchanged.

Instantiating a connection object implicitly opens the SQLite database. SQLite supports multiple open connections to the same database. The Rexx programmer can open up multiple connections by instantiating multiple **ooSQLiteConnection** objects using the same database file name. The ooSQLite native extension builds the SQLite database engine in *serialized multi-threading* mode. In this mode the database engine can be safely used by multiple threads with no restriction. Therefore a connection object can be used in any thread in the Rexx program.

The **close** method should always be invoked when the connection object is no longer needed. This frees up the system resources used by the connection. The **close** method should be invoked even if an error occurred during instantiation. Once **close** is invoked, the object can no longer be used to work

²³ http://www.sqlite.org/c3ref/backup_finish.html#sqlite3backupfinish

with the database. It is an error to invoke database methods with a closed connection object. However, the `close` method can always be invoked. The method is a harmless nop if the connection has already been closed.

3.3.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with database connection objects using the `ooSQLiteConnection` class.

Table 3.4. `ooSQLiteConnection` Methods and Attributes

Method	Documentation
Class Methods	
<code>new</code>	Instantiates a new <code>ooSQLite</code> database connection
Attribute Methods	
<code>backupDestination</code>	Reflects whether this database connection is currently in use as the destination for an online <code>backup</code> .
<code>closed</code>	Reflects the open or closed state of this database connection.
<code>fileName</code>	Reflects the database file name used to instantiate this connection.
<code>initCode</code>	Reflects the status of the initialization of the database connection.
<code>lastErrCode</code>	Reflects the last error code set by for the <code>ooSQLiteConnection</code> object.
<code>lastErrMsg</code>	Reflects a human readable explanation, a message, of the last error code recorded by the connection object.
<code>null</code>	Reflects the default representation for the SQL NULL value that is returned by the interpreter, for this database connection, for database values that are NULL.
<code>recordFormat</code>	Sets or queries the default format for result sets returned through this database connection.
Instance Methods	
<code>busyHandler</code>	Installs a user defined busy handler.
<code>busyTimeOut</code>	Sets a busy handler that sleeps for the specified amount of time when a table is locked.
<code>changes</code>	Returns the number of database rows that were changed or inserted or deleted by the most recently completed SQL statement on this database connection.
<code>close</code>	Releases all systems resources that have been allocated for the database connection.
<code>commitHook</code>	Registers a callback method to be invoked whenever a transaction is committed.
<code>dbFileName</code>	Returns the database file name of the database specified by the <code>name</code> argument.
<code>dbMutex</code>	Retrieves an <code>ooSQLitMutex</code> object that represents the SQLite mutex that serializes access to this database connection.
<code>dbReadOnly</code>	Determines if the named database on this connection is read only.
<code>dbReleaseMemory</code>	Causes the SQLite database engine to attempt to free as much heap memory as possible from this database connection.
<code>dbStatus</code>	Retrieves runtime status information about this database connection.

Method	Documentation
<code>errCode</code>	Returns the numeric result <code>code</code> for the most recent failed SQLite API call associated with this database connection.
<code>errMsg</code>	Returns the English-language text that describes the current <code>errCode</code> value.
<code>exec</code>	Executes the specified SQL statement(s). A callback is invoked for each result row coming out of the evaluated SQL statements.
<code>extendedErrCode</code>	Returns the numeric result <code>code</code> for the most recent failed SQLite API call associated with this database connection.
<code>extendedResultCodes</code>	This is a nop in ooSQLite, extended result codes are always on.
<code>getAutocommit</code>	Determines if this database connection is, or is not, in autocommit mode.
<code>interrupt</code>	Causes any pending database operation on this database connection to abort and return at the earliest opportunity.
<code>lastInsertRowID</code>	Returns the rowid of the most recent successful INSERT into the database from this database connection.
<code>limit</code>	Queries or sets the limiting size of various constructs on this database connection.
<code>nextStmt</code>	Returns the next prepared statement <code>object</code> after the specified statement, or the first prepared statement if <code>stmt</code> is omitted or the <code>.nil</code> object.
<code>profile</code>	Registers an user callback method that can be used for profiling. The callback method is invoked as each SQL statement finishes.
<code>progressHandler</code>	Registers an user callback method that is invoked periodically during long running calls to <code>exec</code> and <code>step</code> methods for this database connection.
<code>pragma</code>	Executes a PRAGMA statement. A PRAGMA statement is a SQLite specific SQL extension, probably unknown to any other database engine.
<code>rollbackHook</code>	Registers a callback method to be invoked whenever a transaction is rolled back.
<code>setAuthorizer</code>	Registers an authorizer callback method that is invoked as SQL statements are being compiled by instantiating an <code>ooSQLiteStmt</code> object.
<code>tableColumnMetadata</code>	Retrieves metadata about a specific column of a specific table of this database connection.
<code>totalChanges</code>	Determines the number of row changes caused by INSERT, UPDATE or DELETE statements since the database connection was opened.
<code>trace</code>	Registers an user callback method that can be used for tracing. The callback method is invoked at various times when an SQL statement is being executed.
<code>updateHook</code>	Registers a callback method to be invoked whenever a row is updated, inserted, or deleted.

3.3.2. new (Class method)

Instantiates a new database connection object and opens the specified database for use.

Arguments:

The arguments are:

file [required]

The file name of the database to open. The special string `:memory:` can be used to open an in memory database.

This argument can be specified as a string, or as a **File** object. When specified as a **File** object, the `absolutePath()` method of the object is used to obtain the database file name.

When specified as a string, the name is used as is, implying it could be a relative file name. I.e., `myDatabase.db` would have to be in the current directory to be opened successfully.

If the `dbFileName` argument is a string, it can also be an URI. Refer to the SQLite documentation for details.

opts [optional]

One or more of the file `open` constants. This argument controls how the database is opened. Do not use any constant marked as *VFS only*. Use the `merge` method of the `ooSQLiteConstants` class to combine two or more of the constant values together, if needed.

The 3 common flags are `OPEN_READWRITE`, `OPEN_READONLY`, and `OPEN_CREATE`. If this argument is omitted, the `OPEN_READWRITE` combined with `OPEN_CREATE` flags are used.

defFormat [optional]

Specifies one of the `ooSQLite` Result Set Format *Constants* that define how a result set is formatted and sets the `recordFormat` attribute for this database connection. This is a convenience argument, the `recordFormat` attribute can always be set directly. If this argument is omitted, then the `recordFormat` attribute is set to the value of the `ooSQLite` class `recordFormat` attribute.

reserved [optional]

Reserved for future use. This argument is completely ignored in the current implementation.

Return value:

Returns a newly instantiated **ooSQLiteConnection** object.

Remarks:

Errors can occur when opening the underlying database, in which case the returned **ooSQLiteConnection** object can not be used to interact with the database. The `initCode` attribute is used to check for errors. The `initCode` attribute will be set to one of `ooSQLite` *result* code constants. If it is the `OK` constant there were no errors and the database can be used. Otherwise, the database can not be used and the `lastErrCode` and `lastErrMsg` attributes can be used to determine the exact nature of the failure.

When an error has occurred, the resources used by the database engine for a connection are released automatically. Otherwise, every successful open of the database through the instantiation of a **ooSQLiteConnection** object must be matched by an invocation of the `close` method to free the resources. Once the database is closed, other methods of the **ooSQLiteConnection** object can not be used. Note that this does not apply to the attribute methods of the object, which are always accessible. As noted, if an error with the database engine happens during `new`, the database engine resources are freed. This is an implicit close of the database.

Do not invoke the methods of the **ooSQLiteConnection** object when an error occurs during `new`.

Details:

The functionality of the *new* method is similar to that of the SQLite [sqlite3_open_v2](#)²⁴ API.

Example:

This example opens up a connection to the **phoneBook** database, located in the current directory of the application, and checks for error:

```
db = .ooSQLiteConnection~new('phoneBook.rdbx')

if db~initCode <> 0 then do
    -- handle error in some fashion
    ...
end
```

3.3.3. backupDestination (Attribute)

```
>>>--> backupDestination-----><
>>>--> backupDestination = varName-----><
```

The *backupDestination* attribute reflects whether this database connection is currently in use as the destination for an online backup.

backupDestination get:

When the value of this attribute is true, this database connection is currently in use as the destination database of an online backup. At all other times, the value is false.

backupDestination set:

The programmer can not set the value of this attribute. Its value is set internally by ooSQLite.

Remarks:

The destination database connection can not be used, by any thread, after the backup is initialized and before the backup is *finished*. Because SQLite does not currently check to see if the application incorrectly accesses the destination database connection, no error code is reported, but the operations may malfunction nevertheless. Use of the destination database connection while a backup is in progress might also cause a mutex deadlock.

To prevent this situation, if this database connection is currently in use as the destination for an online backup, this object is locked. Any invocation of any **method** of this object will raise an error condition. Note that all **attributes** of this object are still accessible.

Normally, the programmer would be fully aware of which database connection is currently in use as a destination database. However, in the event the programmer is not sure of the state of this database connection, the *backupDestination* attribute can be used to determine the state.

Details:

This attribute is provided by ooSQLite, there is no applicable API provided by SQLite.

3.3.4. closed (Attribute)

²⁴ <http://www.sqlite.org/c3ref/open.html>

```
>>--closed-----><
>>--closed = varName-----><
```

Reflects the open or closed state of this database connection.

closed get:

If the database connection has been closed the value of the *closed* attribute is true, otherwise it is false.

closed set:

The Rexx programmer can not set the value of this attribute. It is set internally by the ooSQLite framework.

Remarks:

It is an error to invoke most methods of the connection object once the connection is closed. The only exceptions to this are the attributes of the **ooSQLiteConnection** object and the [close](#) method. The *closed* attribute can be used to check if the connection has already been closed.

3.3.5. fileName (Attribute)

```
>>--fileName-----><
>>--fileName = varName-----><
```

Reflects the database file name used to instantiate this connection.

fileName get:

Returns the file name used to open up the database connection in the [new](#) method.

fileName set:

The programmer can not set this attribute, it is set internally by the ooSQLite framework.

Remarks:

The file name is set during initialization of the connection object. It never changed after that.

3.3.6. initCode (Attribute)

```
>>--initCode-----><
>>--initCode = varName-----><
```

Reflects the status of the initialization of the database connection. Any value other than 0, (.ooSQLite~OK,) indicates that an error occurred during initialization and that the connection is not open.

initCode get:

The value of the *initCode* is one of the result [code](#) constants and indicates the status of the attempt to open the connection to the database.

initCode set:

The Rexx programmer can not set the value of this attribute, it is set internally by the ooSQLite framework.

Remarks:

Errors can occur during instantiation of a database connection object. The *initCode* attribute can be checked to determine if an error occurred. The cautious programmer would always check the init code after instantiating a connection object to ensure that the connections was opened without error.

Example:

This example attempts to open up a connection to the **ooFoods . rdbx** database and checks that the connection was opened successfully, aborting if it was not:

```
dbName = 'ooFotods.rdbx'
dbConn = .ooSQLiteConnection~new(dbName, .ooSQLite~OPEN_READWRITE)

if dbConn~initCode <> 0 then do
    errRC = dbConn~lastErrCode
    errMsg = dbConn~lastErrMsg

    say 'ooSQLiteConnection initialization error:' dbConn~initCode
    say ' Error code:' errRC "('errMsg')'

    dbConn~close
    return errRC
end
...
```

3.3.7. lastErrCode (Attribute)

```
>>>lastErrCode-----><
>>>lastErrCode = varName-----><
```

Reflects the last error code set by for the **ooSQLiteConnection** object.

lastErrCode get:

The value of the *lastErrCode* attribute will be a SQLite result **code** or one of the ooSQLite specific result **codes**.

lastErrCode set:

The programmer can not set the value of this attribute, it is set internally by the ooSQLite framework.

Remarks:

The last error code attribute is similar to the **lastErrMsg** attribute. Its value is the last status code recorded by ooSQLite. The *lastErrCode* and the *lastErrMsg* attributes are always updated together. The error message is always the message that goes with the error code.

Each of the three major ooSQLite objects, an **ooSQLiteConnection**, an **ooSQLiteStmt**, and an **ooSQLiteBackup** object have a *lastErrMsg* and a *lastErrCode* attribute. These attributes all serve the same basic purpose, to hold the last error message and code recorded by ooSQLite.

The attributes are most useful when the invocation of *new* fails to initialize the object correctly due to an error. The *lastErrMsg* and *lastErrCode* attributes are always updated during *new*. In general, these objects do not update the attributes after every method invocation.

Example:

This example uses the *lastErrCode* attribute to produce a meaningful error message when a database connection fails to open:

```
dbName = 'ooFotods.rdbx'
dbConn = .ooSQLiteConnection~new(dbName, .oosSQLite~OPEN_READWRITE)

if dbConn~initCode <> 0 then do
    errRC = dbConn~lastErrCode
    errMsg = dbConn~lastErrMsg

    say 'ooSQLiteConnection initialization error:' dbConn~initCode
    say ' Error code:' errRC "('errMsg')'

    dbConn~close
    return errRC
end
...
```

3.3.8. lastErrMsg (Attribute)

```
>>--lastErrMsg-----><
>>--lastErrMsg = varName-----><
```

Reflects a human readable explanation, a message, of the last error code recorded by the connection object.

lastErrMsg get:

Returns a string message that corresponds to the last error code.

lastErrMsg set:

The programmer can not set this attribute, it is set internally by the ooSQLite framework.

Remarks:

The last error message attribute is similar to the *lastErrCode* attribute. Its value is the last status message recorded by ooSQLite. The *lastErrCode* and the *lastErrMsg* attributes are always updated together. The error message is always the message that goes with the error code.

Each of the three major ooSQLite objects, an *ooSQLiteConnection*, an *ooSQLiteStmt*, and an *ooSQLiteBackup* object have a *lastErrMsg* and a *lastErrCode* attribute. These attributes all serve the same basic purpose, to hold the last error message and code recorded by ooSQLite.

The attributes are most useful when the invocation of *new* fails to initialize the object correctly due to an error. The *lastErrMsg* and *lastErrCode* attributes are always updated during *new*. In general, these objects do not update the attributes after every method invocation.

Example:

This example uses the *lastErrMsg* attribute to produce a meaningful error message when a database connection fails to open:

```
dbName = 'ooFotods.rdbx'
dbConn = .ooSQLiteConnection~new(dbName, .oosSQLite~OPEN_READWRITE)

if dbConn~initCode <> 0 then do
    errRC = dbConn~lastErrCode
```

```

errMsg = dbConn~lastErrMsg

say 'ooSQLiteConnection initialization error:' dbConn~initCode
say '  Error code:' errRC "('errMsg')'

dbConn~close
return errRC
end
...

```

3.3.9. null (Attribute)

```

>>--null-----><
>>--null = nullObj-----><

```

Reflects the default representation for the SQL NULL value that is returned by the interpreter, for this database connection, for database values that are NULL.

null get:

Returns the current object the interpreter uses for this database connection for the SQL NULL value. If the programmer has not changed this attribute, its value is the value of the [null](#) attribute of the [ooSQLite](#) class. Normally this is the [.nil](#) object.

null set:

Set this attribute to either the [.nil](#) object, or some alternative *string* value.

Remarks:

By default, ooSQLite uses the [.nil](#) object to represent the SQL NULL value. Queries for values stored in the database will return the [.nil](#) object for any value that is SQL NULL. However, by changing the value of the *null* attribute, the Rexx programmer can change the value the interpreter returns for NULL for any statement executed through this database connection. Typically this would be done when the returned values are going to be displayed as text and the programmer would prefer to work with a string directly. Perhaps the value *NULL*, or *no value* would be assigned.

When a [ooSQLiteConnection](#) object is instantiated, the *null* attribute is assigned the value of the [null](#) attribute of the [ooSQLiteConnection](#).

Note that this attribute does not affect the value the programmer must use to assign a SQL NULL to the database. The programmer must use the [.nil](#) object for that.

Details

Raises syntax errors when incorrect usage is detected.

This attribute is provided by ooSQLite, there is no similar feature provided by SQLite.

Example:

This example sets the *null* attribute of the database connection to *NULL*. This allows the application to invoke the *left* method on the returned value without having to check that the return is the [.nil](#) object. Note that invoking the *left* method on the [.nil](#) object will raise a syntax condition:

```

dbConn = .ooSQLiteConnection~new(dbName, .oosSQLite~OPEN_READWRITE)
dbConn~null = 'NULL'

```

```

sql = 'SELECT * FROM foods ORDER BY name;'
stmt = .ooSQLiteStmt~new(dbConn, sql)

say stmt~columnName(1)~left(25) || stmt~columnName(2)~left(25) ||
stmt~columnName(3)~left(25)
say '='~copies(80)

do while stmt~step == stmt~ROW
    say stmt~columnText(1)~left(25) || stmt~columnText(2)~left(25) ||
stmt~columnText(3)~left(25)
end

```

3.3.10. recordFormat (Attribute)

```

>>--recordFormat----->|
>>--recordFormat = varName----->|

```

Over-rides the process-wide default record format set through the [ooSQLite class](#) for this database connection.

recordFormat get:

The value of this attribute is one of the [ooSQLite Result Set Format Constants](#) that define how a result set is formatted. This value defines the format of all result sets produced by this connection. If the programmer has not changed the value of the attribute explicitly, its value is the same as the default value set by the [ooSQLite](#) classes [recordFormat](#) attribute.

recordFormat set:

To over-ride the application-wide record format for result sets, assign one one of the [ooSQLite Result Set Format Constants](#) to this attribute. This default can still be over-ridden at [statement](#) or method level. If not over-ridden, all returned result sets from this database connection will have the format specified by this attribute.

3.3.11. busyHandler

```

>>--busyHandler(--callBackObj-----+---+---+---+---+----->|
                +-, -mthName--+  +-, -userData-+

```

Installs a user defined busy handler.

Arguments:

The arguments are:

callBackObj [required]

An instantiated object with a method that might be invoked whenever an attempt is made to open a database table that another thread or process has locked.

However, this argument can also be `.nil` to indicate that any installed busy handler is to be removed. When no busy handler is installed then `BUSY` or `IOERR_BLOCKED` is returned immediately upon encountering the lock.

mthName [optional]

The method name that will be invoked during a call back. By default, the method invoked will be *busyCallBack()*. However, the user can specify an alternative method if desired. This argument is ignored when the *callbackObj* argument is *.nil*.

userData [optional]

This can be any Rexx object the user desires. The object will be sent as the second argument to the busy callback method when it is invoked. This argument is ignored when the *callbackObj* argument is *.nil*.

Return value:

Returns a SQLite result *code*.

Remarks:

By default, there is no busy handler installed.

There can only be one busy handler installed. Setting a new busy handler automatically clears any previously installed handler. Note that invoking *busyTimeOut* can also set or clear the busy handler.

The busy handler should not take any actions which modify the database connection that invoked the busy handler. Any such actions result in undefined behavior.

A busy handler must not close the database connection or prepared statement that invoked the busy handler.

Details:

The functionality of the *busyHandler* method is similar to that of the SQLite *sqlite3_busy_handler*²⁵ API.

Example:

This example installs a busy handler with a *onTimeOut* method that is to be invoked. It passes the busy handler object itself as the *userData* argument:

```
helper = .MyHelperClass~new

db = .ooSQLiteConnection~new('phoneBook.rdbx')
if db~initCode <> 0 then return db~lastErrCode

db~busyHandler(helper, onTimeOut, helper)
...

::class 'MyHelperClass'

::method onTimeOut unguarded
use arg count, helperObj

if helperObj~query(count) == "ABANDON_TIMEOUT" then return 0
else return 1

::method query private unguarded
use strict arg count

{ code that determines what to return }
...
```

²⁵ http://www.sqlite.org/c3ref/busy_handler.html

3.3.11.1. busyCallBack

```
>>--busyCallBack(--countInvoked, --userData--)----><
```

The *busyCallBack* method is an example of a user callback method for the *busyHandler* method. Here the method name of *busyCallBack* is used, because it is the default method name if the programmer does not specify his own name in the *busyHandler* method. Any method name can be used by specifying it as the second argument to *busyHandler*.

Note: there is no *busyCallBack* method in any ooSQLite class. This method is just used to illustrate how to define a user callback method to be used as a busy handler.

Arguments:

The arguments sent to the callback method are:

countInvoked [required]

The number of times that the busy handler has been invoked for this locking event.

userData [optional]

The user data object specified by the programmer as the third argument to the *busyHandler* method. If the programmer did not specify a user data argument, this argument is omitted when the callback is invoked.

Return value:

The programmer must return a value from the callback. If the method returns 0, then no additional attempts are made to access the database by the SQLite database engine and SQLITE_BUSY or SQLITE_IOERR_BLOCKED is returned. If the callback returns non-zero, then another attempt is made to open the database for reading and the cycle repeats.

Details:

The implementation of a *busy handler* method is discussed on the SQLite [sqlite3_busy_handler²⁶](#) page.

3.3.12. busyTimeOut

```
>>--busyTimeOut(--milliseconds--)-----><
```

Sets a busy handler that sleeps for the specified amount of time when a table is locked. The handler sleeps multiple times until the specified time has been accumulated. After the specified milliseconds of sleeping, the handler returns 0 which causes *step* to return BUSY or IOERR_BLOCKED.

Arguments:

The arguments are:

milliseconds

The whole number of milliseconds the busy handler should sleep when a table is locked. If *milliseconds* is less than or equal to zero all busy handlers are turned off.

²⁶ http://www.sqlite.org/c3ref/busy_handler.html

Return value:

This method will always return **OK**.

Remarks:

After the specified milliseconds of sleeping, the handler returns 0 which causes *step* to return BUSY or IOERR_BLOCKED. There can only be a single busy handler for a particular database connection any any given moment. If another busy handler was defined, using *busyHandler*, prior to calling this routine, that other busy handler is cleared.

Details:

The functionality of the *busyTimeOut* method is similar to that of the SQLite [sqlite3_busy_timeout](#)²⁷ API.

Example:

This example sets a busy timeout value of 3 seconds:

```
db = .ooSQLiteConnection~new('phoneBook.rdbx')
if db~initCode <> 0 then return db~lastErrCode

db~busyTimeout(3000)
```

3.3.13. changes

>>--changes-----><

Returns the number of database rows that were changed or inserted or deleted by the most recently completed SQL statement on this database connection.

Arguments:

There are no arguments to this method.

Return value:

The number of changed, deleted, or inserted rows.

Remarks:

Only changes that are directly specified by the INSERT, UPDATE, or DELETE statement are counted. Auxiliary changes caused by triggers or foreign key actions are not counted. Use the *totalChanges* method to find the total number of changes including changes caused by triggers and foreign key actions.

Details:

The functionality of the *changes* method is similar to that of the SQLite [sqlite3_changes](#)²⁸ API.

3.3.14. close

>>--close-----><

²⁷ http://www.sqlite.org/c3ref/busy_timeout.html

²⁸ <http://www.sqlite.org/c3ref/changes.html>

The `close` method releases all systems resources that have been allocated for the database connection. Once the connection has been closed, it is an error to invoke any method of the `ooSQLiteConnection` object that interacts with the SQLite database. All connection objects should be closed when they are no longer needed, even connection objects that were instantiated with an error.

Arguments:

There are no arguments for this method.

Return value:

An ooSQLite *result* code. Returns OK if the connection is successfully closed and all associated resources are deallocated. Returns BUSY if the connection is associated with unfinalized prepared statements or unfinished backup objects.

Remarks:

Programs should finalize all prepared statements, close all BLOB handles, and finish all backup objects associated with the connection object prior to attempting to close the object. It is a harmless nop to invoke *close* on a connection object that has already been closed. The attributes of the object are still valid after the connection is closed, but invoking other methods of a closed connection object is an error.

Details:

The functionality of the `close` method is similar to that of the SQLite `sqlite3_close`²⁹ API.

Example:

This example opens a database to do some work with it, then closes it. Note that if an error occurs trying to open the database, the connection object is still closed:

```
dbFile = 'ooFoods.rdbx'

db = .ooSQLiteConnection~new(dbFile, .ooSQLite~OPEN_READWRITE)
if db~initCode <> 0 then do
    -- do error stuff
    ...
    db~close
    return 99
end

-- work with the database
...
db~close
```

3.3.15. commitHook

Registers a callback method to be invoked whenever a transaction is committed.

Arguments:

The arguments are:

²⁹ <http://www.sqlite.org/c3ref/close.html>

callBackObj [required]

An instantiated object with a method that will be invoked whenever a transaction is committed.

However, this argument can also be `.nil` to indicate that any installed commit hook is to be removed.

mthName [optional]

The method name that will be invoked during a call back. By default, the method invoked will be `commitHookCallBack()`. However, the user can specify an alternative method if desired. This argument is ignored when the `callbackObj` argument is `.nil`.

userData [optional]

This can be any Rexx object the user desires. The object will be sent as the first and only argument to the commit hook callback method when it is invoked. This argument is ignored when the `callbackObj` argument is `.nil`.

Return value:

The `userData` argument to a previous invocation of the `commitHook` method on the same database connection, or `.nil` if there has not been a previous invocation or the `userData` argument was not used on the previous invocation.

Details:

The functionality of the `commitHook` method is similar to that of the SQLite [sqlite3_commit_hook](#)³⁰ API.

3.3.15.1. commitHookCallBack

```
>>--commitHookCallBack(--userData--)-----><
```

The `commitHookCallBack` method is an example of a user callback method for the `commitHook` method. Here the method name of `commitHookCallBack` is used, because it is the default method name if the programmer does not specify his own name in the `commitHook` method. Any method name can be used by specifying it as the second argument to the `commitHook` method.

Note: there is no `commitHookCallBack` method in any ooSQLite class. This method is just used to illustrate how to define a user callback method to be used with the commit hook.

Arguments:

The single argument sent to the callback method is:

userData [required]

The user data object specified by the programmer as the third argument to the `commitHook` method. If the programmer did not specify a user data argument, this will be the `.nil` object.

Return value:

The programmer must return a value from the callback. If the method returns 0, then the commit is allowed to continue normally. If the callback returns non-zero, then the COMMIT is converted into a ROLLBACK. If a `rollback` hook has been installed, it is invoked just as it would be for any other rollback.

³⁰ http://www.sqlite.org/c3ref/commit_hook.html

Remarks:

The callback method must not do anything that will modify the database connection that invoked the callback. Any actions to modify the database connection must be deferred until after the completion of the *step* invocation that triggered the commit hook to begin with. Running any other SQL statements, including SELECT statements, or merely instantiating a new *statement* object, or executing another *step* method will modify the database connection.

Details:

The implementation of a *commit hook* method is discussed on the SQLite [sqlite3_commit_hook³¹](#) page.

3.3.16. createCollation

```
>>--createCollation(---+-----+--)
           +--type--+-----><
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.3.17. createFunction

```
>>--createFunction(---+-----+--)
           +--type--+-----><
```

xx

Arguments:

The arguments are:

³¹ http://www.sqlite.org/c3ref/commit_hook.html

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.3.18. dbFileName

```
>>--dbFileName(--name--)-----><
```

Returns the database file name of the database specified by *name*.

Arguments:

The arguments are:

name [required]

The database *name*, as known internally by the SQLite database engine, of the database to get the file name for.

Return value:

The file name of the database specified by *name* on success. If there is no *name* database on the database connection, or if *name* is a temporary or in-memory database, then the empty string is returned.

Remarks:

The filename returned will be an absolute pathname, even if the filename used to open the database originally was a URI or relative pathname.

Details:

The functionality of the *dbFileName* method is similar to that of the SQLite *sqlite3_db_filename*³² API.

3.3.19. dbMutex

```
>>--dbMutex-----><
```

Retrieves an *ooSQLitMutex* object that represents the SQLite mutex that serializes access to this database connection.

³² http://www.sqlite.org/c3ref/db_filename.html

Arguments:

This method takes no arguments.

Return value:

An **ooSQLiteMutex** object that gives access to the underlying SQLite mutex serializing access to this database connection. The mutex object could be a *closed null* mutex, see the remarks section.

Remarks:

If the threading *mode* for this database connection is not serialized, then the returned mutex object will be a closed, null, mutex. This is not likely.

The *dbMutex* method is provided for completeness. It is expected that Rexx programmers who do not understand mutexes well and / or do not understand how SQLite itself works with respect to the mutex that serializes access to the database connection, will not use this method.

Since the **ooSQLiteMutex** object returned by the *dbMutex* method represents a mutex in use by the SQLite database engine, invoking *free* on the object closes the Rexx object, but does not actually close the underlying SQLite mutex.

Details:

The functionality of the *dbMutex* method is similar to that of the SQLite *sqlite3_db_mutex*³³ API.

3.3.20. dbReadOnly

```
>>--dbReadOnly( --name-- )-----><
```

Determines if the named database on this connection is read only.

Arguments:

The single argument is:

name [required]

The database *name*, as known internally by the SQLite database engine, of the database to check for read only.

Return value:

Returns 1 if the database is read only, 0 if the database is read / write, and -1 if *name* is not an opened database on this connection.

Details:

The functionality of the *dbReadOnly* method is similar to that of the SQLite *sqlite3_db_READONLY*³⁴ API.

Example:

This example illustrates the *dbReadOnly* method:

```
dbName = 'ooFoods.rdbx'

dbConn = .ooSQLiteConnection~new(dbName, .oosSQLite~OPEN_READONLY)
```

³³ http://www.sqlite.org/c3ref/db_mutex.html

³⁴ http://www.sqlite.org/c3ref/db_READONLY.html

```

if dbConn~initCode <> 0 then do
    -- Do error stuff and return
end

dbConn~exec("ATTACH DATABASE 'phonebook.db' AS phone;", .true)

dbConn~exec("ATTACH DATABASE 'ooFoodsCopy.rdbx' AS dupe;", .true)

say 'Read only? main: ' dbConn~dbReadOnly('main')
say 'Read only? phone: ' dbConn~dbReadOnly('phone')
say 'Read only? dupe: ' dbConn~dbReadOnly('dupe')
say 'Read only? temp: ' dbConn~dbReadOnly('temp')

ret = dbConn~close

/* Output might be:

Read only? main: 1
Read only? phone: 1
Read only? dupe: 1
Read only? temp: -1

*/

```

3.3.21. dbReleaseMemory

>>--dbReleaseMemory-----><

Causes the SQLite database engine to attempt to free as much heap memory as possible from this database connection.

Arguments:

This method has no arguments.

Return value:

Returns an ooSQLite *result* code. Although the SQLite documentation does not explicitly state this, it seems likely that OK is always returned.

Remarks:

See also the *releaseMemory* method of the *ooSQLite* class.

Details:

The functionality of the *DbReleaseMemory* method is similar to that of the SQLite *sqlite3_db_release_memory*³⁵ API.

3.3.22. dbStatus

>>--dbStatus(--opt--, --result---+-----+-)---><
 +-,-reset--+

Retrieves runtime status information about this database connection.

³⁵ http://www.sqlite.org/c3ref/db_release_memory.html

Arguments:

The arguments are:

opt [required]

A DB status parameter *constant* that specifies what status information is requested.

result [required]

A **Directory** object whose indexes will hold the request information on return. On success the following indexes in the object will be valid:

CURRENT

This index will contain the current value for the status information queried.

HIGHWATER

This index will hold the high water mark for the status information queried.

reset [optional]

Must be true or false to specify whether the high water mark should be reset, or not. The default if this argument is omitted is false, do not reset the high water mark.

Return value:

Returns a SQLite result *code*, **OK** on success otherwise an error code on failure.

Remarks:

If the high water mark is reset, it is reset to the current value of the status information. The ooSQLite DB status constants reflect the currently available SQLite DB status options. The set of SQLite DB status options is likely to grow in future releases of SQLite. When, or if, those options grow, the ooSQLite constants will be updated to reflect the new options

Details:

The functionality of the *dbStatus* method is similar to that of the SQLite *sqlite3_db_status*³⁶ API.

Example:

This example checks the values of the page memory used by the caches for the database connection and does not reset the high water mark:

```
dbName = 'ooFoods.rdbx'

dbConn = .ooSQLiteConnection~new(dbName, .oosSQLite~OPEN_READWRITE)

if dbConn~initCode <> 0 then do
    -- do error handling and return
end

values = .directory~new

ret = dbConn~dbStatus(.oosSQLite~DBSTATUS_CACHE_USED, values, .false)
if ret == .dbConn~OK then do
    say 'Bytes of page memory used by all caches on this database connection:'
    say ' Current:' values~current ' High water:' values~highwater
    say
end
else do
    say 'Error returned from dbStatus():' ret
    say
end
```

³⁶ http://www.sqlite.org/c3ref/db_status.html

```
/* Output might be:

Bytes of page memory used by all caches on this database connection:
  Current: 16824  High water: 0

*/
```

3.3.23. errCode

>>--errCode-----><

Returns the numeric result `code` for the most recent failed SQLite API call associated with this database connection.

Arguments:

This method has no arguments

Return value:

The return is a numeric result code.

Remarks:

The `errMsg` method provides an English-language description of the current value of `errCode`. If a previous SQLite API call failed, but the most recent call succeeded, the return from `errCode` and `errMsg` is undefined. If a SQLite API fails with a result code of MISUSE, that means the API was invoked incorrectly by the program. In that case, the error code and message may or may not be set.

In SQLite, extended result codes are turned off by default, so the SQLite API provides an extended error code API that returns the extended result codes even if extended result codes are turned off. In ooSQLite however, the extended result codes are turned on during compilation. This makes the `errCode` method and the `extendedErrCode` method exactly equivalent. The `extendedErrCode` method is provided for completeness

Details:

The functionality of the `errCode` method is similar to that of the SQLite `sqlite3_errcode`³⁷ API.

3.3.24. errMsg

>>--errMsg-----><

Returns the English-language text that describes the current `errCode` value.

Arguments:

This method has no arguments.

Return value:

A string describing the current `errCode` value.

³⁷ <http://www.sqlite.org/c3ref/errcode.html>

Remarks:

The `errMsg` method provides an English-language description of the current value of `errCode`. If a previous SQLite API call failed, but the most recent call succeeded, the return from `errCode` and `errMsg` is undefined. If a SQLite API fails with a result code of MISUSE, that means the API was invoked incorrectly by the program. In that case, the error code and message may or may not be set.

In SQLite, extended result codes are turned off by default, so the SQLite API provides an extended error code API that returns the extended result codes even if extended result codes are turned off. In ooSQLite however, the extended result codes are turned on during compilation. This makes the `errCode` method and the `extendedErrCode` method exactly equivalent. The `extendedErrCode` method is provided for completeness.

Details:

The functionality of the `errMsg` method is similar to that of the SQLite `sqlite3_errmsg`³⁸ API.

3.3.25. exec

```
>>> exec(sql, useCB, format, cbObj, mthName, uData)
```

Executes the specified SQL statement(s). A callback is invoked for each result row coming out of the evaluated SQL statements. This callback can either be a callback internal to the ooSQLite framework, or a callback to a method in an object supplied by the programmer.

Arguments:

The arguments are:

sql [required]

A string containing the SQL statement, or statements, to execute. More than 1 statement can be executed by including semi-colons in the string. Each SQL statement is ended with a semi-colon.

useCB [optional]

True or false to indicate if the callback feature should be used or not.

If this argument is false, the *sql* is simply executed and a result code returned. The other arguments are ignored. The default if this argument is omitted is false.

If this argument is true, then the callback feature is used. If the *cbObj* arg is omitted then an internal callback of the ooSQLite framework is used and a result set is returned. The result set could be empty if the sql does not produce a result set. The format for the returned result set will be the format specified by the *recordFormat* attribute of this database connection. However, the default format can be overridden for this invocation of exec() through the optional *format* argument.

Otherwise, if *cbObj* is not omitted, then the call back method of that object is invoked for each result row coming out of the evaluated SQL statements. In this case the return is the result code from the database engine's execution.

³⁸ <http://www.sqlite.org/c3ref/errcode.html>

format [optional]

This argument can be used to specify the format of the result row(s) coming out of the evaluated SQL statements. If this argument is omitted the default *format* value for this connection is used. If specified, it must be one of the ooSQLite Result Set Format *Constants* that define how a result set is formatted. The format effects the result set if a result set is returned, and the format of the result row sent to a user defined callback when a user defined callback is used.

cbObj [optional]

Specifies that a user defined callback should be used rather than the ooSQLite internal callback. A callback method in this object is invoked for each row coming out of the evaluated SQL statement(s). By default the method invoked in the object will be **execCallBack**. However the optional *mthName* argument can be used to change this.

If this argument omitted, the internal ooSQLite callback is used. This argument is ignored if *useCB* is false.

mthName [optional]

Names the method to be invoked in *cbObj*. This argument is ignored if *cbObj* is omitted, or *useCB* is false. By default the method invoked in *cbObj* is **execCallBack**.

uData [optional]

User data that is passed to the user defined callback method. This can be any Rexx object the programmer wishes to use. The object is passed as the third argument to the callback method. This argument is ignored if *useCB* is false, or if *cbObj* is omitted.

Return value:

The return value is dependent on whether the internal ooSQLite callback is used or not. When the internal callback is used, a result set is returned containing all the result rows produced by the SQL statement(s). In all other cases a result *code* is returned.

Remarks:

The easiest approach to using this method is to use the internal callback of the ooSQLite framework. However, it is possible that more control might be desired in the processing of the result rows than the internal callback provides. In this case, a user defined callback can be used. The **execCallBack** method explains the details of a user defined callback method.

Details:

The functionality of the exec method is similar to that of the *sqlite3_exec*³⁹ SQLite API.

Example:

This example shows the exec method invocation to use the internal callback of the ooSQLite framework. The format of the returned result set is specified to be an array of arrays:

```
dbName = 'ooFoods.rdbx'

dbConn = .ooSQLiteConnection~new(dbName, .ooSQLite~OPEN_READWRITE)

sql = 'SELECT * FROM foods ORDER BY name;'
resultSet = dbConn~exec(sql, .true, .ooSQLite~OO_ARRAY_OF_ARRAYS)

z = printResultSet(resultSet)
```

³⁹ <http://www.sqlite.org/c3ref/exec.html>

3.3.25.1. execCallBack

```
>>--execCallBack(--row--, --rowNum--, --userData--)-----><
```

The `execCallBack` method is an example of a user callback method for the `exec` method. Here the method name of `execCallBack` is used because it is the default method name if the programmer does not specify her own name in the `exec` method. Any method name can be used by specifying it as the fifth argument to `exec`.

Note: there is no `execCallBack` method in any ooSQLite class. This method is just used to illustrate how to define a user callback method.

Arguments:

The arguments sent to the callback method are:

`row`

The current result row produced by executing the SQL statement(s) passed to the `exec` method. The exact format of this argument is dependent on the result set format in use. See the remarks for details.

`rowNum`

The current result row number.

`userData`

The user data object specified by the programmer as the sixth argument to the `exec` method. If the programmer did not specify a user data argument, this argument is omitted when invoking the callback.

Return value:

The programmer must return a value from the callback. This value can be any of the ooSQLite result `code` constants, but if it is not the `OK` constant, then the SQLite database engine aborts without invoking the callback again and without running any subsequent SQL statements. Note that returning some other result code than `OK` allows the callback to halt the processing of the result rows at an early stage.

Remarks:

The value of the `row` argument is dependent on the default result set format in use for the invocation of the `exec` method that generates the callback invocation. These are the possible formats:

`OO_ARRAY_OF_ARRAYS`:

The `row` argument will be an array with exactly 2 indexes. Index 1 will be an array of the column names for the result row. Index 2 will be an array for the corresponding values of the column.

`OO_ARRAY_OF_DIRECTORIES`:

The `row` argument will be a `Directory` object where the indexes of the directory are the column names and the value of the index is the value of the column.

`OO_STEM_OF_STEMS`:

The `row` argument will be a `Stem` object where the indexes of the stem are the column names and the value of the index is the value of the column.

Example:

This example is just used to show the principles of a user defined callback. The actual processing has no benefit over using the internal callback. A user class is defined with a callback method. This

is passed to the exec method. The default record format is OO_OO_ARRAY_OF_DIRECTORIES. In the callback method, each record is added to the user data object, which in this case is an array. On return from the exec method, if there were no errors, the *resultObj* array will contain all the result rows produced by executing the SQL statement:

```
dbConn = .ooSQLiteConnection~new(dbName, .oosSQLite~OPEN_READWRITE)

resultObj = .array~new(500)
cbObj = .UserCallBack~new

sql = 'SELECT * FROM foods ORDER BY name;'
ret = dbConn~exec(sql, .true, , cbObj, , resultObj)

''

::class 'UserCallBack' inherit ooSQLiteConstants

::method execCallBack
    use arg row, rowNum, userObj

    userObj[rowNum] = row

    return self~OK
```

3.3.26. extendedErrCode

>>--extendedErrCode-----><

Returns the numeric result *code* for the most recent failed SQLite API call associated with this database connection. The *extendedErrCode* and the *errCode* methods are functionally equivalent, see the remarks.

Arguments:

This method takes no arguments

Return value:

Returns a numeric result code.

Remarks:

The *errMsg* method provides an English-language description of the current value of *extendedErrCode*. If a previous SQLite API call failed, but the most recent call succeeded, the return from *extendedErrCode* and *errMsg* is undefined. If a SQLite API fails with a result code of MISUSE, that means the API was invoked incorrectly by the program. In that case, the error code and message may or may not be set.

In SQLite, extended result codes are turned off by default, so the SQLite API provides an extended error code API that returns the extended result codes even if extended result codes are turned off. In ooSQLite however, the extended result codes are turned on during compilation. This makes the *errCode* method and the *extendedErrCode* method exactly equivalent. The *extendedErrCode* method is provided for completeness.

Details:

The functionality of the `extendedErrCode` method is similar to that of the SQLite [`sqlite3_extended_errcode`⁴⁰](#) API.

3.3.27. extendedResultCodes

```
>>--extendedResultCodes(--onOff--)-----><
```

This is a nop in ooSQLite, extended result codes are always on.

Arguments:

The single argument is:

onOff

True or false to turn extended result codes on or off. However, this argument is currently ignored, extended result codes are always on in ooSQLite.

Return value:

Returns OK, always.

Remarks:

The method is provided for completeness. In SQLite, extended result code are off by default for historical reasons. ooSQLite, however, has no history to make this applicable, so extended result code are always on.

Details:

The functionality of the `extendedResultCode` method is similar to that of the SQLite [`sqlite3_extended_result_codes`⁴¹](#) API.

3.3.28. getAutocommit

```
>>--getAutocommit-----><
```

Determines if this database connection is, or is not, in autocommit mode.

Arguments:

This method does not take any arguments.

Return value:

Returns true if the connection is in autocommit mode, otherwise false.

Remarks:

Autocommit mode is on by default. Autocommit mode is disabled by a BEGIN statement.

Autocommit mode is re-enabled by a COMMIT or ROLLBACK. If certain kinds of errors happen on a statement within a multi-statement transaction then the transaction might be rolled back automatically. The only way to find out whether SQLite automatically rolled back the transaction after an error is to use this function.

⁴⁰ <http://www.sqlite.org/c3ref/errcode.html>

⁴¹ http://www.sqlite.org/c3ref/extended_result_codes.html

Details:

The functionality of the `getAutoCommit` method is similar to that of the SQLite [sqlite3_get_autocommit](#)⁴² API.

3.3.29. interrupt

```
>>--interrupt-----><
```

Interrupt causes any pending database operation to abort and return at the earliest opportunity.

Arguments:

There are no arguments to this method.

Return value:

XX

Remarks:

This method is safe to call from a different thread than the thread that is currently running the database operation. But it is not safe to call this routine with a database connection that might close before *interrupt* returns. If a SQL operation is very nearly finished at the time when *interrupt* is invoked, then it might not have an opportunity to be interrupted and might continue to completion. A SQL operation that is interrupted will return INTERRUPT result code. If the interrupted SQL operation is an INSERT, UPDATE, or DELETE that is inside an explicit transaction, then the entire transaction will be rolled back automatically.

The *interrupt* call is in effect until all currently running SQL statements on this database connection complete. Any new SQL statements that are started after the *interrupt* call and before the running statement count reaches zero are interrupted as if they had been running prior to the *interrupt* call. New SQL statements that are started after the running statement count reaches zero are not effected by the *interrupt*. A call to *interrupt* that occurs when there are no running SQL statements is a no-op and has no effect on SQL statements that are started after the *interrupt* call returns.

If this database connection closes while *interrupt* is running then bad things will likely happen.

Details:

The functionality of the *interrupt* method is similar to that of the SQLite [sqlite3_interrupt](#)⁴³ API.

3.3.30. lastInsertRowID

```
>>--lastInsertRowID-----><
```

Returns the rowid of the most recent successful INSERT into the database from this database connection.

Arguments:

This method has no arguments.

⁴² http://www.sqlite.org/c3ref/get_autocommit.html

⁴³ <http://www.sqlite.org/c3ref/interrupt.html>

Return value:

Returns the unique whole number *rowid* of the most recent, successful INSERT statement on this database connection

Remarks:

Every row of every SQLite table has a 64-bit signed integer key that uniquely identifies the row within its table. This integer is usually called the *rowid*. The *rowid* value can be accessed using one of the special case-independent names *rowid*, *oid*, or *_rowid_* in place of a column name. If a table contains a user defined column named using any of these 3 special names, then that name always refers the explicitly declared column and cannot be used to retrieve the integer *rowid* value.

The SQLite documentation contains very detailed and complete documentation concerning the *last*⁴⁴ *rowid* and the *rowid*⁴⁵ itself. That documentation should be consulted to fully understand the *rowid* concept.

Details:

The functionality of the *lastInsertRowID* method is similar to that of the SQLite [*sqlite3_last_insert_rowid*](#)⁴⁶ API.

3.3.31. limit

```
>>--limit(--id--, --value--)-----><
```

Queries or sets the limiting size of various constructs on this database connection.

Arguments:

The arguments are:

id [required]

One of the run time limit *constants*. This identifies the limit construct that is to be size limited.

value [required]

The new value of the limit. If this value is negative, then the limit is unchanged.

Return value:

Returns the existing limit at the time *limit* is invoked, even if the invocation does not change the limit. Because of this, the way to query the current value of a limit is to invoke the method with a negative number.

Remarks:

For each limit there is a hard upper bound set when ooSQLite is built. Attempts to increase a limit above its hard upper bound are silently truncated by the database engine to the hard upper bound. The *limit* method can be used by an application that allows an untrusted source to enter data into the database to lower the built in limits. Perhaps to prevent denial of service attacks. Programmers may also want to use the [*setAuthorizer*](#) method and / or limiting database size using the *max_page_count PRAGMA* when the application works with untrusted sources.

New run time limits may be introduced by SQLite in the future.

⁴⁴ http://www.sqlite.org/c3ref/last_insert_rowid.html

⁴⁵ http://www.sqlite.org/lang_createtable.html#rowid

Details:

The functionality of the *limit* method is similar to that of the SQLite [sqlite3_limit](#)⁴⁷ API.

3.3.32. nextStmt

```
>>--nextStmt(---+-----+---)----->|
    +--stmt--+
```

Returns the next prepared statement *object* after the specified statement, or the first prepared statement if *stmt* is omitted or the `.nil` object.

Arguments:

The single argument is:

stmt [optional]

A `ooSQLiteStmt` object specifying the starting point of the search. If this argument is omitted, or the `.nil` object then the search starts from the beginning of the prepared statement list.

Return value:

Returns the found statement, or `.nil` if no next prepared statement is found.

Remarks:

The search finds the first statement *after* the specified *stmt*. It is okay to use a statement that has been [finalized](#) as the starting point of the search. Omitting the *stmt* argument or using `.nil` essentially finds the *first* prepared statement.

Details:

The functionality of the *nextStmt* method is similar to that of the SQLite [sqlite3_next_stmt](#)⁴⁸ API.

3.3.33. profile

```
>>--profile(--callBackObj---+-----+---+-----+---)----->|
    +-,-mthName--+  +-,-userData-+
```

Registers an user callback method that can be used for profiling. The callback method is invoked is invoked as each SQL statement finishes.

Arguments:

The arguments are:

callBackObj [required]

An instantiated object with a method that will be invoked for profiling.

However, this argument can also be `.nil` to indicate that any installed profile callback is to be removed.

⁴⁷ <http://www.sqlite.org/c3ref/limit.html>

⁴⁸ http://www.sqlite.org/c3ref/next_stmt.html

mthName [optional]

The method name that will be invoked during a call back. By default, the method invoked will be *profileCallBack()*. However, the user can specify an alternative method if desired. This argument is ignored when the *callbackObj* argument is *.nil*.

userData [optional]

This can be any Rexx object the user desires. The object will be sent as the third argument to the profile callback method when it is invoked. This argument is ignored when the *callbackObj* argument is *.nil*.

Return value:

The *userData* argument to a previous invocation of the *trace* method on the same database connection, or *.nil* if there has not been a previous invocation or the *userData* argument was not used on the previous invocation.

Remarks:

By default, there is no profile callback installed. There can only be one profile callback per database connection. Setting a new profile callback automatically clears any previously installed callback.

The callback method is invoked as each SQL statement finishes. The profile callback contains the original statement text and an estimate of wall-clock time of how long that statement took to run. The example *profileCallBack* method has complete details.

Details:

The functionality of the *profile* method is similar to that of the SQLite *sqlite3_profile*⁴⁹ API.

3.3.33.1. profileCallBack

```
>>>profileCallBack(--sql--,--nanoSeconds--,--userData--)----->
```

The *profileCallBack* method is an example of a user callback method for the *profile* method. Here the method name of *profileCallBack* is used, because it is the default method name if the programmer does not specify his own name in the *profile* method. Any method name can be used by specifying it as the second argument to the *profile* method.

The profile callback is invoked as each SQL statement finishes executing.

Note: there is no *profileCallBack* method in any ooSQLite class. This method is just used to illustrate how to define a user callback method to be used with when profiling.

Arguments:

The arguments sent to the callback method are:

sql [required]

The original SQL statement text.

nanoSeconds [required]

The wall clock time it took to execute the SQL statement. See the remarks section.

⁴⁹ <http://www.sqlite.org/c3ref/profile.html>

userData [required]

The user data object specified by the programmer as the third argument to the *trace* method. If the programmer did not specify a user data argument, this will be the `.nil` object.

Return value:

The programmer must return a whole number value from the callback, the exact number does not matter.

Remarks:

Although the wall clock time is in nanoseconds, the current SQLite implementation is only capable of millisecond resolution making the six least significant digits in the time are meaningless. Future versions of SQLite might provide greater resolution on the profiler callback. The `sqlite3_profile()` function is considered experimental and is subject to change in future versions of SQLite.

Details:

The implementation of a *profile* callback method is discussed on the SQLite [sqlite3_profile](#)⁵⁰ page.

3.3.34. progressHandler

```
>>--progressHandler(--callBackObj---+-----+-----+-----+---><
+,-instrcts-+ +-,-mName--+ +-,-urData-+
```

Registers an user callback method that is invoked periodically during long running calls to `exec`, and `step` for this database connection. An example use for this interface is to keep a GUI updated during a large query.

Arguments:

The arguments are:

callBackObj [required]

An instantiated object with a method that will be invoked during long running exec or step methods.

However, this argument can also be `.nil` to indicate that any installed progress handler is to be removed.

instrcts [required]

The number of virtual machine instructions that are evaluated between successive invocations of the callback. If this argument is less than 1, it also has the effect of removing any installed callback.

mName [optional]

The method name that will be invoked during a call back. By default, the method invoked will be `progressCallBack()`. However, the user can specify an alternative method if desired. This argument is ignored when the `callbackObj` argument is `.nil`.

⁵⁰ <http://www.sqlite.org/c3ref/profile.html>

uData [optional]

This can be any Rexx object the user desires. The object will be sent as the first and only argument to the progress handler callback method when it is invoked. This argument is ignored when the callbackObj argument is `.nil`.

Return value:

Returns a SQLite result `code`.

Remarks:

By default, there is no progress handler installed. There can only be one progress handler per database connection. Setting a new progress handler automatically clears any previously installed handler.

Details:

The functionality of the `progressHandler` method is similar to that of the SQLite `sqlite3_progress_handler`⁵¹ API.

3.3.34.1. progressCallBack

```
>>>progressCallBack(--userData--)-----><
```

The `progressCallBack` method is an example of a user callback method for the `progressHandler` method. Here the method name of `progressCallBack` is used, because it is the default method name if the programmer does not specify his own name in the `progressHandler` method. Any method name can be used by specifying it as the second argument to the `progressHandler` method.

The `progressHandler` callback is invoked periodically after a number of virtual machine code instructions are evaluated. This number is specified by the second argument to the `progressHandler` method.

Note: there is no `progressCallBack` method in any `ooSQLite` class. This method is just used to illustrate how to define a user callback method to be used as a progress handler.

Arguments:

The single argument sent to the callback method is:

userData [required]

The user data object specified by the programmer as the fourth argument to the `progressHandler` method. If the programmer did not specify a user data argument, this will be the `.nil` object.

Return value:

The programmer must return a whole number value from the callback. If the progress handler returns non-zero, the SQLite operation is interrupted. If 0 is returned the operation continues.

Remarks;

The progress handler must not do anything that will modify the database connection that invoked the progress handler. Note that both instantiating a new `ooSQLiteStmt` and the `step` method modify their database connections.

⁵¹ http://www.sqlite.org/c3ref/progress_handler.html

Details:

The implementation of a *progressHandler* callback method is discussed on the SQLite [sqlite3_progress_handler⁵²](http://www.sqlite.org/c3ref/progress_handler.html) page.

3.3.35. pragma

```
>>--pragma(--name---+-----+---)--><
     +-,-value--+
```

Executes a PRAGMA statement. A PRAGMA statement is a SQLite specific SQL extension, probably unknown to any other database engine. The *pragma* method implements the functionality of the SQLite [PRAGMA⁵³](http://www.sqlite.org/prAGMA.html) statement. The SQLite documentation should be read to understand how PRAGMAS work

The *pragma* method is intended to be a convenience method for the Rexx programmer. For any specific PRAGMA, the *pragma* method could be replaced by using a prepared *statement*, or by using the *exec* method. When a PRAGMA is executed using a prepared statement or the *exec* method, a result set is returned. However, executing the vast majority of PRAGMAS returns either a single value or no value. As a convenience, the *pragma* method returns a single value for those cases. A result set is only returned when necessary.

In addition, some PRAGMAS require an argument, some only work with no argument, and SQLite ignores unknown PRAGMA keywords. In all these cases of mistaken usage, no error messages or error codes are generated by the SQLite engine. This means if there is mistaken usage of a PRAGMA statement SQLite does not inform the user of the fact. However, the ooSQLite framework attempts to detect any errors when using the *pragma* method and raises an exception when it does. Because of this, the Rexx programmer is informed of incorrect usage the first time he tests his program.

Pragmas can have an optional database name before the pragma name. The form is: the database name, a dot character, and the pragma name. The database name is the name of an *attached* database or it can be *main* or *temp* for the main and the TEMP databases. If the optional database name is omitted, *main* is assumed. In some pragmas, the database name is meaningless and is simply ignored. An example for the *secure_delete* pragma could be: **main.secure_delete**

Arguments:

The arguments are:

name [required]

The name of the pragma to be executed. Case is not significant for the name. Each recognized pragma is listed in the remarks section, along with details on what to expect for that pragma.

value [optional]

Most PRAGMAS accept, or use, an argument. The second *value* argument to the *pragma* method is used as the argument to a PRAGMA. When a pragma is used to set a value, the *value* argument is the value to set. For these pragmas, the existence of the argument determines if a query or a set is done. When the argument is omitted, a query is done. When the argument is used, a set is done.

⁵² http://www.sqlite.org/c3ref/progress_handler.html

⁵³ <http://www.sqlite.org/prAGMA.html>

In some cases the PRAGMA argument is a qualifier and the *value* argument is passed on to the PRAGMA as that qualifier. In some cases, a PRAGMA does not accept an argument, and in others an argument is required. See the remarks section for the details on each individual PRAGMA.

Return value:

The return is dependent on the pragma in use. Some pragmas return result sets, some return a single value, and others return a SQLite result *code*. In all cases a value is returned. See the remarks section for the type of return to expect for each individual PRAGMA.

Although an error is unlikely when using the *pragma* method, errors are always possible. In many cases, an error return is not distinguishable from a valid return. For instance, the *page_count* pragma could return 21 pages. 21 is also the value of the MISUSE result code. In all cases, the programmer should check the value of the last error *code* attribute to determine if an error occurred. The value of the attribute will always be `.ooSQLiteConstants~OK` if no error happened. The value will never be OK if there was an error.

Remarks:

PRAGMA statement can be used to set / configure database values, to trigger an action, and to query most of the configurable database values. SQLite *deprecated* and *debug only* PRAGMAs are not recognized in ooSQLite.

The following lists all PRAGMAs that are recognized, with details for each PRAGMA following. Return values are categorized as single values (the most common,) result codes, or result sets. When a result set is returned, it could be an empty result set. That is it could contain no rows. Both single value returns and empty result sets could also be an error return. See the Return value section above for a discussion of error returns. For each pragma the return category is listed for the pragma with no *value* argument and with a *value* argument.

<code>application_id</code>	<code>freelist_count</code>	<code>quick_check</code>
<code>auto_vacuum</code>	<code>fullfsync</code>	<code>read_uncommitted</code>
<code>automatic_index</code>	<code>ignore_check_constraints</code>	<code>recursive_triggers</code>
<code>busy_timeout</code>	<code>incremental_vacuum</code>	<code>reverse_unordered_selects</code>
<code>cache_size</code>	<code>index_info</code>	<code>schema_version</code>
<code>cache_spill</code>	<code>index_list</code>	<code>secure_delete</code>
<code>case_sensitive_like</code>	<code>integrity_check</code>	<code>shrink_memory</code>
<code>checkpoint_fullfsync</code>	<code>journal_mode</code>	<code>soft_heap_limit</code>
<code>collation_list</code>	<code>journal_size_limit</code>	<code>synchronous</code>
<code>compile_options</code>	<code>legacy_file_format</code>	<code>table_info</code>
<code>database_list</code>	<code>locking_mode</code>	<code>temp_store</code>
<code>defer_foreign_keys</code>	<code>max_page_count</code>	<code>user_version</code>
<code>encoding</code>	<code>mmap_size</code>	<code>wal_autocheckpoint</code>
<code>foreign_key_check</code>	<code>page_count</code>	<code>wal_checkpoint</code>
<code>foreign_key_list</code>	<code>page_size</code>	<code>writable_schema</code>
<code>foreign_keys</code>	<code>query_only</code>	

`application_id`

Queries or sets the 32-bit unsigned big-endian *Application ID* integer. Consult the SQLite [application_id](#)⁵⁴ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

⁵⁴ http://www.sqlite.org/pragma.html#pragma_application_id

value: integer

auto_vacum

Queries or sets the auto-vacuum status in the database. Consult the SQLite [auto_vacum](#)⁵⁵ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: 0 | NONE | 1 | FULL | 2 | INCREMENTA

automatic_index

Queries, sets, or clears the automatic indexing capability. Consult the SQLite [automatic_index](#)⁵⁶ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: true | false

busy_timeout

Queries or changes the setting of the *busy* timeout. This pragma is an alternative to the *busyTimeOut* method. SQLite makes the pragma available as a pragma for use with language bindings that do not provide direct access to the `sqlite3_busy_timeout()` API.

ooSQLite of course does provide direct access through both the *busyTimeOut* method and the [oosqlBusyTimeOut](#) routine. The pragma is included here for completeness. Consult the SQLite [busy_timeout](#)⁵⁷ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: single value.

value: milliseconds

cache_size

Queries or changes the suggested maximum number of database disk pages that SQLite will hold in memory at once per open database file. Consult the SQLite [cache_size](#)⁵⁸ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: pages | -kibibytes

cache_spill

Enables or disables the ability of the pager to spill dirty cache pages to the database file in the middle of a transaction. Cache_spill is enabled by default and most applications should leave it that way as cache spilling is usually advantageous. Consult the SQLite [cache_spill](#)⁵⁹ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: true or false

⁵⁵ http://www.sqlite.org/pragma.html#pragma_auto_vacuum

⁵⁶ http://www.sqlite.org/pragma.html#pragma_automatic_index

⁵⁷ http://www.sqlite.org/pragma.html#pragma_busy_timeout

⁵⁸ http://www.sqlite.org/pragma.html#pragma_cache_size

⁵⁹ http://www.sqlite.org/pragma.html#pragma_cache_spill

case_sensitive_like

Installs a new application-defined LIKE function that is either case sensitive or insensitive depending on the value of the `case_sensitive_like` pragma. By default LIKE is case-insensitive. Consult the SQLite [case_sensitive_like⁶⁰](#) documentation for complete details.

No `value` argument: invalid, condition raised.

With `value` argument: single value

`value`: true or false

checkpoint_fullfsync

Query or change the fullfsync flag for checkpoint operations. Consult the SQLite [checkpoint_fullfsync⁶¹](#) documentation for complete details.

No `value` argument returns: single value.

With `value` argument returns: result code.

`value`: true or false

collation_list

Returns a list of the collating sequences defined for the current database connection. Consult the SQLite [collation_list⁶²](#) documentation for complete details.

No `value` argument returns: result set.

With `value` argument: invalid, condition raised

`value`: invalid

compile_options

Returns the names of compile-time options used when building SQLite, one option per row. The `SQlite_` prefix is omitted from the returned option names. Consult the SQLite [compile_options⁶³](#) documentation for complete details.

No `value` argument returns: result set.

With `value` argument: invalid, condition raised

`value`: invalid

database_list

Works like a query to return one row for each database attached to the current database connection. Consult the SQLite [database_list⁶⁴](#) documentation for complete details.

No `value` argument returns: result set.

With `value` argument: invalid, condition raised

`value`: invalid

defer_foreign_keys

When the `defer_foreign_keys` PRAGMA is on, enforcement of all foreign key constraints is delayed until the outermost transaction is committed. The `defer_foreign_keys` pragma defaults to OFF so that foreign key constraints are only deferred if they are created as "DEFERRABLE INITIALLY DEFERRED". This pragma is only meaningful if foreign key constraints are enabled, of course. Consult the SQLite [database_list⁶⁵](#) documentation for complete details.

⁶⁰ http://www.sqlite.org/pragma.html#pragma_case_sensitive_like

⁶¹ http://www.sqlite.org/pragma.html#pragma_checkpoint_fullfsync

⁶² http://www.sqlite.org/pragma.html#pragma_collation_list

⁶³ http://www.sqlite.org/pragma.html#pragma_compile_options

⁶⁴ http://www.sqlite.org/pragma.html#pragma_database_list

⁶⁵ http://www.sqlite.org/pragma.html#pragma_database_list

No *value* argument returns: single value.

With *value* argument returns: result code.

value: true or false

encoding

When used as a query, if the main database has already been created, then this pragma returns the text encoding used by the main database, one of "UTF-8", "UTF-16le" (little-endian UTF-16 encoding) or "UTF-16be" (big-endian UTF-16 encoding). If the main database has not already been created, then the value returned is the text encoding that will be used to create the main database, if it is created by this session. Consult the SQLite [encoding](#)⁶⁶ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: UTF-8 | UTF-16 | UTF-16le | UTF-16be

foreign_key_check

Checks the database, or the table specified, for foreign key constraints that are violated and returns one row of output for each violation. Consult the SQLite [foreign_key_check](#)⁶⁷ documentation for complete details.

No *value* argument returns: result set.

With *value* argument returns: result set.

value: table_name, not required

foreign_key_list

Returns one row for each foreign key that references a column in the argument *table*. Consult the SQLite [foreign_key_list](#)⁶⁸ documentation for complete details.

No *value* argument: invalid, condition raised.

With *value* argument returns: result set.

value: table_name

foreign_keys

Query, set, or clear the enforcement of foreign key constraints. Consult the SQLite [foreign_keys](#)⁶⁹ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: true or false

freelist_count

Returns the number of unused pages in the database file. Consult the SQLite [freelist_count](#)⁷⁰ documentation for complete details.

No *value* argument returns: single value.

With *value* argument: invalid, condition raised.

value: invalid

⁶⁶ http://www.sqlite.org/pragma.html#pragma_encoding

⁶⁷ http://www.sqlite.org/pragma.html#pragma_foreign_key_check

⁶⁸ http://www.sqlite.org/pragma.html#pragma_foreign_key_list

⁶⁹ http://www.sqlite.org/pragma.html#pragma_foreign_keys

⁷⁰ http://www.sqlite.org/pragma.html#pragma_freelist_count

fullfsync

Query or change the fullfsync flag. Consult the SQLite [fullfsync](#)⁷¹ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: true or false

ignore_check_constraints

Enables or disables the enforcement of CHECK constraints. The default setting is off, meaning that CHECK constraints are enforced by default. Consult the SQLite [ignore_check_constraints](#)⁷² documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: true or false

incremental_vacuum

Causes up to N pages to be removed from the freelist. The database file is truncated by the same amount. If there are fewer than N pages on the freelist, N is omitted, or N is less than 1, all pages are removed. Consult the SQLite [incremental_vacuum](#)⁷³ documentation for complete details.

No *value* argument returns: result code.

With *value* argument returns: result code.

value: N, not required

index_info

Returns one row each column in the named index. The first column of the result is the rank of the column within the index. The second column of the result is the rank of the column within the table. The third column of output is the name of the column being indexed. Consult the SQLite [index_info](#)⁷⁴ documentation for complete details.

No *value* argument: invalid, condition raised.

With *value* argument returns: result set

value: index_name, required

index_list

Returns one row for each index associated with the given table. Columns of the result set include the index name and a flag to indicate whether or not the index is UNIQUE. Consult the SQLite [index_list](#)⁷⁵ documentation for complete details.

No *value* argument: invalid, condition raised.

With *value* argument returns: result set

value: table_name, required

integrity_check

Does an integrity check of the entire database. It looks for out-of-order records, missing pages, malformed records, and corrupt indices. If any problems are found, then strings are

⁷¹ http://www.sqlite.org/pragma.html#pragma_fullfsync

⁷² http://www.sqlite.org/pragma.html#pragma_ignore_check_constraints

⁷³ http://www.sqlite.org/pragma.html#pragma_incremental_vacuum

⁷⁴ http://www.sqlite.org/pragma.html#pragma_index_info

⁷⁵ http://www.sqlite.org/pragma.html#pragma_index_list

returned (as multiple rows with a single column per row) which describe the problems. At most N errors will be reported before the analysis quits. The default value for N is 100. If no errors are found, a single row with the value *ok* is returned. Consult the SQLite [integrity_check](#)⁷⁶ documentation for complete details.

No *value* argument returns: result set.

With *value* argument returns: result set.

value: N, not required

journal_mode

Queries or sets the journal mode for databases associated with the current database connection. Consult the SQLite [journal_mode](#)⁷⁷ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: single value

value: DELETE | TRUNCATE | PERSIST | MEMORY | WAL | OFF

journal_size_limit

This pragma can be used to limit the size (N bytes) of rollback-journal and WAL files left in the file-system after transactions or checkpoints. Consult the SQLite [journal_size_limit](#)⁷⁸ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: single value.

value: N or -1, no limit

legacy_file_format

Sets or queries the value of the legacy_file_format flag. When this flag is on, new SQLite databases are created in a file format that is readable and writable by all versions of SQLite going back to 3.0.0. When the flag is off, new databases are created using the latest file format which might not be readable or writable by versions of SQLite prior to 3.3.0. Consult the SQLite [legacy_file_format](#)⁷⁹ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: true or false

locking_mode

Sets or queries the database connection locking-mode. The locking-mode is either NORMAL or EXCLUSIVE. Consult the SQLite [locking_mode](#)⁸⁰ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: NORMAL | EXCLUSIVE

max_page_count

Queries or set the maximum number of pages (N) in the database file. Both forms of the pragma return the maximum page count. The set form attempts to modify the maximum page

⁷⁶ http://www.sqlite.org/pragma.html#pragma_integrity_check

⁷⁷ http://www.sqlite.org/pragma.html#pragma_journal_mode

⁷⁸ http://www.sqlite.org/pragma.html#pragma_journal_size_limit

⁷⁹ http://www.sqlite.org/pragma.html#pragma_legacy_file_format

⁸⁰ http://www.sqlite.org/pragma.html#pragma_locking_mode

count. The maximum page count cannot be reduced below the current database size. Consult the SQLite [max_page_count](#)⁸¹ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: single value.

value: N

mmap_size

Query or change the maximum number of bytes (N) that are set aside for memory-mapped I/O on a single database. Consult the SQLite [page_count](#)⁸² documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: N

page_count

Returns the total number of pages in the database file. Consult the SQLite [page_count](#)⁸³ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: invalid, condition raised.

value: invalid

page_size

Queries or sets the page size (N) of the database. The page size must be a power of two between 512 and 65536 inclusive. Consult the SQLite [page_size](#)⁸⁴ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: N

query_only

Prevents all changes to database files when enabled. Consult the SQLite [query_only](#)⁸⁵ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: true or false

quick_check

Performs an integrity check, like the `integrity_check` pragma, except that it does not verify that index content matches table content. By skipping the verification of index content, `quick_check` is able to run much faster than `integrity_check`. At most N errors will be reported before the analysis quits. The default value for N is 100. If no errors are found, a single row with the value `ok` is returned. Otherwise the two pragmas are the same. Consult the SQLite [quick_check](#)⁸⁶ documentation for complete details.

⁸¹ http://www.sqlite.org/pragma.html#pragma_max_page_count

⁸² http://www.sqlite.org/pragma.html#pragma_mmap_size

⁸³ http://www.sqlite.org/pragma.html#pragma_page_count

⁸⁴ http://www.sqlite.org/pragma.html#pragma_page_size

⁸⁵ http://www.sqlite.org/pragma.html#pragma_query_only

⁸⁶ http://www.sqlite.org/pragma.html#pragma_quick_check

No *value* argument returns: result set.

With *value* argument returns: result set.

value: N, not required

read_uncommitted

Query, set, or clear READ UNCOMMITTED isolation. Consult the SQLite [read_uncommitted](#)⁸⁷ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: true or false

recursive_triggers

Query, set, or clear the recursive trigger capability. Consult the SQLite [recursive_triggers](#)⁸⁸ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: true or false

reverse_unordered_selects

When enabled, this pragma causes SELECT statements without an ORDER BY clause to emit their results in the reverse order of what they normally would. This can help debug applications that are making invalid assumptions about the result order. Consult the SQLite [reverse_unordered_selects](#)⁸⁹ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: true or false

schema_version

Used to set or get the value of the schema-version. The pragmas schema_version and user_version are used to set or get the value of the schema-version and user-version, respectively. The schema-version and the user-version are big-endian 32-bit signed integers stored in the database header.

The schema-version is usually only manipulated internally by SQLite. Using the schema_version pragma to modify the schema-version is potentially dangerous and may lead to program crashes or database corruption. Use with caution.

The user-version is not used internally by SQLite. It may be used by applications for any purpose. Consult the SQLite [schema_version](#)⁹⁰ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: 32-bit integer

⁸⁷ http://www.sqlite.org/pragma.html#pragma_read_uncommitted

⁸⁸ http://www.sqlite.org/pragma.html#pragma_recursive_triggers

⁸⁹ http://www.sqlite.org/pragma.html#pragma_reverse_unordered_selects

⁹⁰ http://www.sqlite.org/pragma.html#pragma_schema_version

secure_delete

Queries or changes the secure-delete setting. When secure-delete on, SQLite overwrites deleted content with zeros. Consult the SQLite [secure_delete](#)⁹¹ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: single value.

value: true or false

shrink_memory

Causes the database connection on which it is invoked to free up as much memory as it can. Consult the SQLite [shrink_memory](#)⁹² documentation for complete details.

No *value* argument returns: result code.

With *value* argument: invalid, condition raised

value: invalid

soft_heap_limit

This pragma invokes the [softHeapLimit64](#) method with the argument N, if N is specified and is a non-negative integer. The soft_heap_limit pragma always returns the same integer that would be returned by the [softHeapLimit64](#) method with an argument of -1. C-language function. Consult the SQLite [shrink_memory](#)⁹³ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: single value

value: N

synchronous

Queries or changes the setting of the "synchronous" flag. Consult the SQLite [synchronous](#)⁹⁴ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: 0 | OFF | 1 | NORMAL | 2 | FULL

table_info

Returns a row for each column in the named table. Columns in the result set include the column name, data type, whether or not the column can be NULL, and the default value for the column. Consult the SQLite [table_info](#)⁹⁵ documentation for complete details.

No *value* argument: invalid, condition raised

With *value* argument returns: result set.

value: table_name, required

temp_store

Queries or changes the setting of the *temp_store* parameter. Consult the SQLite [temp_store](#)⁹⁶ documentation for complete details.

⁹¹ http://www.sqlite.org/pragma.html#pragma_secure_delete

⁹² http://www.sqlite.org/pragma.html#pragma_shrink_memory

⁹³ http://www.sqlite.org/pragma.html#pragma_soft_heap_limit

⁹⁴ http://www.sqlite.org/pragma.html#pragma_synchronous

⁹⁵ http://www.sqlite.org/pragma.html#pragma_table_info

⁹⁶ http://www.sqlite.org/pragma.html#pragma_temp_store

No *value* argument returns: single value.

With *value* argument returns: result code.

value: 0 | DEFAULT | 1 | FILE | 2 | MEMORY

user_version

Used to set or get the value of the user-version. The pragmas schema_version and user_version are used to set or get the value of the schema-version and user-version, respectively. The schema-version and the user-version are big-endian 32-bit signed integers stored in the database header.

The schema-version is usually only manipulated internally by SQLite. Using the schema_version pragma to modify the schema-version is potentially dangerous and may lead to program crashes or database corruption. Use with caution.

The user-version is not used internally by SQLite. It may be used by applications for any purpose. Consult the SQLite [schema_version](#)⁹⁷ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: result code.

value: 32-bit integer

wal_autocheckpoint

Queries or sets the write-ahead log auto-checkpoint interval. When the write-ahead log is enabled (via the journal_mode pragma) a checkpoint will be run automatically whenever the write-ahead log equals or exceeds N pages in length. Setting the auto-checkpoint size (N) to zero or a negative value turns auto-checkpointing off. Consult the SQLite [wal_autocheckpoint](#)⁹⁸ documentation for complete details.

No *value* argument returns: single value.

With *value* argument returns: single value.

value: N

wal_checkpoint

If the write-ahead log is enabled (via the journal_mode pragma), this pragma causes a checkpoint operation to run on the named *database* database, or on all attached databases if *database* is omitted. If write-ahead log mode is disabled, this pragma is a harmless no-op. Consult the SQLite [wal_checkpoint](#)⁹⁹ documentation for complete details.

No *value* argument returns: result set.

With *value* argument returns: result set.

value: PASSIVE | FULL | RESTART

writable_schema

When this pragma is on, the SQLITE_MASTER tables in the database can be changed using ordinary UPDATE, INSERT, and DELETE statements. Warning: misuse of this pragma can easily result in a corrupt database file. Consult the SQLite [writable_schema](#)¹⁰⁰ documentation for complete details.

No *value* argument returns: singel value.

With *value* argument returns: result code.

⁹⁷ http://www.sqlite.org/pragma.html#pragma_schema_version

⁹⁸ http://www.sqlite.org/pragma.html#pragma_wal_autocheckpoint

⁹⁹ http://www.sqlite.org/pragma.html#pragma_wal_checkpoint

¹⁰⁰ http://www.sqlite.org/pragma.html#pragma_writable_schema

value: true or false

Example:

This example uses the table_info pragma to get information on the food_types table and then print it to the screen:

```
dbName      = 'ooFoods.rdbx'
tablename  = 'food_types'

dbConn = .ooSQLiteConnection~new(dbName, .ooSQLite~OPEN_READWRITE)

resultSet = dbConn~pragma('table_info', tablename)

z = printResultSet(resultSet)

/* Output might be for example:

cid      name       type        notnull    dflt_value   pk
=====
0        id         integer     0          NULL        1
1        name        text       0          NULL        0
*/
```

3.3.36. rollbackHook

```
>>> rollbackHook(--callBackObj--+-----+---+-----+--><
           +-,-mthName--+  +-,-userData-+
```

Registers a callback method to be invoked whenever a transaction is rolled back.

Arguments:

The arguments are:

callBackObj [required]

An instantiated object with a method that will be invoked whenever a transaction is rolled back.

However, this argument can also be .nil to indicate that any installed rollback hook is to be removed.

mthName [optional]

The method name that will be invoked during a call back. By default, the method invoked will be *rollbackHookCallBack()*. However, the user can specify an alternative method if desired. This argument is ignored when the *callbackObj* argument is .nil.

userData [optional]

This can be any REXX object the user desires. The object will be sent as the first and only argument to the rollback hook callback method when it is invoked. This argument is ignored when the *callbackObj* argument is .nil.

Return value:

The *userData* argument to a previous invocation of the *rollbackHook* method on this database connection, or .nil if there has not been a previous invocation or the *userData* argument was not used on the previous invocation.

Details:

The functionality of the *rollbackHook* method is similar to that of the SQLite [sqlite3_rollback_hook](#)¹⁰¹ API.

3.3.36.1. rollbackHookCallBack

```
>>-- rollbackHookCallBack(--userData--)-----><
```

The *rollbackHookCallBack* method is an example of a user callback method for the *rollbackHook* method. Here the method name of *rollbackHookCallBack* is used, because it is the default method name if the programmer does not specify his own name in the *rollbackHook* method. Any method name can be used by specifying it as the second argument to the *rollbackHook* method.

Note: there is no *rollbackHookCallBack* method in any ooSQLite class. This method is just used to illustrate how to define a user callback method to be used with the rollback hook.

Arguments:

The single argument sent to the callback method is:

userData [required]

The user data object specified by the programmer as the third argument to the *rollbackHook* method. If the programmer did not specify a user data argument, this will be the `.nil` object.

Return value:

The programmer must return a whole number value from the callback. However, the actual value returned makes no difference to the SQLite database engine. Typically, the programmer would just return 0.

Remarks:

The callback method must not do anything that will modify the database connection that invoked the callback. Any actions to modify the database connection must be deferred until after the completion of the *step* invocation that triggered the rollback hook to begin with. Running any other SQL statements, including SELECT statements, or merely instantiating a new *statement* object, or executing another *step* method will modify the database connection.

For the purposes of this method, a transaction is said to have been rolled back if an explicit ROLLBACK statement is executed, or an error or constraint causes an implicit rollback to occur. However, the callback is not invoked if a transaction is automatically rolled back because the database connection is closed.

Details:

The implementation of a *rollback hook* method is discussed on the SQLite [sqlite3_rollback_hook](#)¹⁰² page.

3.3.37. setAuthorizer

```
>>-- setAuthorizer(--callBackObj---+-----+-----+---)-----><
```

¹⁰¹ http://www.sqlite.org/c3ref/commit_hook.html

¹⁰² http://www.sqlite.org/c3ref/commit_hook.html

+ -, -mthName-- +	+ -, -userData-- +
-------------------	--------------------

Registers an authorizer callback method that is invoked as SQL statements are being compiled by instantiating an [ooSQLiteStmt](#) object. The callback allows the Rexx program to disallow certain actions of the SQL, or reject the SQL statement entirely.

Arguments:

The arguments are:

callBackObj [required]

An instantiated object with a method that will be invoked during the compilation of a SQL statement

However, this argument can also be .nil to indicate that any installed update hook is to be removed.

mthName [optional]

The method name that will be invoked during a call back. By default, the method invoked will be *authorizerCallBack()*. However, the user can specify an alternative method if desired. This argument is ignored when the *callbackObj* argument is .nil.

userData [optional]

This can be any Rexx object the user desires. The object will be sent as the last argument to the authorizer callback method when it is invoked. This argument is ignored when the *callbackObj* argument is .nil.

Return value:

Returns a SQLite result [code](#). Currently, it appears that the SQLite database engine always returns OK.

Remarks:

At various points during the compilation process of a statement, as logic is being created to perform various actions, the authorizer callback is invoked to see if those actions are allowed. The authorizer callback should return OK to allow the action, IGNORE to disallow the specific action but allow the SQL statement to continue to be compiled, or DENY to cause the entire SQL statement to be rejected with an error. If the authorizer callback returns any value other than IGNORE, OK, or DENY then the instantiation of the [ooSQLiteStmt](#) object that triggered the authorizer will fail with an error [message](#).

An authorizer is used when preparing SQL statements from an untrusted source, to ensure that the SQL statements do not try to access data they are not allowed to see, or that they do not try to execute malicious statements that damage the database. For example, an application may allow a user to enter arbitrary SQL queries for evaluation by a database. But the application does not want the user to be able to make arbitrary changes to the database. An authorizer could then be put in place while the user-entered SQL is being prepared that disallows everything except SELECT statements.

Programs that need to process SQL from untrusted sources might also consider lowering resource limits using the [limit](#) method and / or limiting database size using the [max_page_count](#) [PRAGMA](#) in addition to using an authorizer.

Details:

The functionality of the `setAuthorizer` method is similar to that of the SQLite `sqlite3_set_authorizer`¹⁰³ API.

3.3.37.1. authorizerCallBack

```
>>--authorizerCallBack(--code--, --str1--, --str2--, --str3--, --str4--, --userData--)-<
```

The `authorizerCallBack` method is an example of a user callback method for the `setAuthorizer` method. Here the method name of `authorizerCallBack` is used, because it is the default method name if the programmer does not specify his own name in the `setAuthorizer` method. Any method name can be used by specifying it as the second argument to the `setAuthorizer` method.

Note: there is no `authorizerCallBack` method in any `ooSQLite` class. This method is just used to illustrate how to define a user callback method to be used with the set authorizer hook.

Arguments:

The arguments sent to the callback method are:

code [required]

One of the authorizer `constants` that specifies the particular action to be authorized.

str1 [required]

String 1 that contains additional details about the action to be authorized. The string varies depending on the action that triggered the callback. See the table in the Remarks section for possible values.

str2 [required]

String 2 that contains additional details about the action to be authorized. The string varies depending on the action that triggered the callback. See the table in the Remarks section for possible values.

str3 [required]

The `name` of the database (`main`, `temp`, etc.,) if applicable. If not applicable, the empty string.

str4 [required]

The name of the inner-most trigger or view that is responsible for the access attempt or the empty string if this access attempt is directly from top-level SQL code.

userData [required]

The user data object specified by the programmer as the third argument to the `setAuthorizer` method. If the programmer did not specify a user data argument, this will be the `.nil` object.

Return value:

The programmer must return 1 of the 3 authorizer return code `constants`. Any other return will cause the instantiation of the `ooSQLiteStmt` object to fail with an error.

Remarks:

At various points during the compilation process of a statement, as logic is being created to perform various actions, the authorizer callback is invoked to see if those actions are allowed.

¹⁰³ http://www.sqlite.org/c3ref/set_authorizer.html

In ooSQLite, the compilation process of a statement takes place during the initialization of an [ooSQLiteStmt](#) object.

The authorizer callback should return OK to allow the action, IGNORE to disallow the specific action but allow the SQL statement to continue to be compiled, or DENY to cause the entire SQL statement to be rejected with an error. If the authorizer callback returns any value other than IGNORE, OK, or DENY then the instantiation of the [ooSQLiteStmt](#) object that triggered the authorizer will fail with an error [message](#).

If the action code is READ and the callback returns IGNORE then the prepared statement statement is constructed to substitute a NULL value in place of the table column that would have been read if OK had been returned. The IGNORE return can be used to deny an untrusted user access to individual columns of a table. If the action code is DELETE and the callback returns IGNORE then the DELETE operation proceeds but the truncate optimization is disabled and all rows are deleted individually.

The callback method must not do anything that will modify the database connection that invoked the callback. Any actions to modify the database connection must be deferred until after the completion of the [step](#) invocation that triggered the update hook to begin with. Running any other SQL statements, including SELECT statements, or merely instantiating a new [statement](#) object, or executing another [step](#) method will modify the database connection.

The following table lists the values for *str1* and *str2* for each of the possible action codes, the value of the *code* argument

Table 3.5. The Authorizer Callback Arguments

Code	Str1	Str2
CREATE_INDEX	Index Name	Table Name
CREATE_TABLE	Table Name	Empty String
CREATE_TEMP_INDEX	Index Name	Table Name
CREATE_TEMP_TABLE	Table Name	Empty String
CREATE_TEMP_TRIGGER	Trigger Name	Table Name
CREATE_TEMP_VIEW	View Name	Empty String
CREATE_TRIGGER	Trigger Name	Table Name
CREATE_VIEW	View Name	Empty String
DELETE	Table Name	Empty String
DROP_INDEX	Index Name	Table Name
DROP_TABLE	Table Name	Empty String
DROP_TEMP_INDEX	Index Name	Table Name
DROP_TEMP_TABLE	Table Name	Empty String
DROP_TEMP_TRIGGER	Trigger Name	Table Name
DROP_TEMP_VIEW	View Name	Empty String
DROP_TRIGGER	Trigger Name	Table Name
DROP_VIEW	View Name	Empty String
INSERT	Table Name	Empty String
PRAGMA	Pragma Name	userData arg or Empty String
READ	Table Name	Column Name
SELECT	Empty String	Empty String

Code	Str1	Str2
TRANSACTION	Operation	Empty String
UPDATE	Table Name	Column Name
ATTACH	Filename	Empty String
DETACH	Database Name	Empty String
ALTER_TABLE	Database Name	Table Name
REINDEX	Index Name	Empty String
ANALYZE	Table Name	Empty String
CREATE_VTABLE	Table Name	Module Name
DROP_VTABLE	Table Name	Module Name
FUNCTION	Empty String	Function Name
SAVEPOINT	Operation	Savepoint Name

Details:

The implementation of an *authorizer* callback method is discussed on the SQLite [sqlite3_set_authorizer](#)¹⁰⁴ page.

3.3.38. tableColumnMetadata

```
>>>-tableColumnMetadata(--tableName--, --colName--, --results--+-----+--)--><
      +-, -dbName--+
```

Retrieves metadata about a specific column of a specific table of this database connection.

Arguments:

The arguments are:

tableName [required]

The name of the table containing the column whose metadata is being sought.

[required]

The name of the column whose metadata is being sought.

results [required]

A **Directory** object whose indexes will hold the metadata on return. On success the following indexes in the object will be valid:

DATATYPE

The data type of the column.

COLLATIONSEQUENCE

The name of the default collation sequence for the column.

NOTNULL

True if the column has a NOT NULL constraint, otherwise false.

¹⁰⁴ http://www.sqlite.org/c3ref/set_authorizer.html

PRIMARYKEY

True if the column is part of the PRIMARY KEY, otherwise false.

AUTOINCREMENT

True if the column is AUTOINCREMENT, otherwise false.

dbName[optional]

The *name* of the database (*main*, *temp*, etc..) If this argument is omitted, then all attached databases are searched for the table by the database engine, using the same algorithm as is used by SQLite to resolve unqualified table references.

Return value:

Returns a SQLite result *code*, OK on success, otherwise an error code.

Remarks:

If this methods fails, the *results* object is unchanged.

If the specified table is actually a view, an error code is returned. If the specified column is *rowid*, *oid*, or *_rowid_* and an INTEGER PRIMARY KEY column has been explicitly declared, then the output parameters are set for the explicitly declared column. If there is no explicitly declared INTEGER PRIMARY KEY column, then the output parameters are set as follows:

- **Data type:** "INTEGER"
- **Collation sequence:** "BINARY"
- **Not null:** false
- **Primary key:** true
- **Auto increment** false

Details:

The functionality of the *tableColumnMetadata* method is similar to that of the SQLite *sqlite3_table_column_metadata*¹⁰⁵ API. Note that the arguments to *tableColumnMetadata* are in a slightly different order than in the SQLite API. This is to place the optional database name at the end of the argument list.

3.3.39. totalChanges

```
>>--totalChanges-----><
```

Determines the number of row changes caused by INSERT, UPDATE or DELETE statements since the database connection was opened.

Arguments:

There are no arguments to this method.

Return value:

Returns the number of row changes caused by INSERT, UPDATE or DELETE statements since the database connection was opened.

¹⁰⁵ http://www.sqlite.org/c3ref/table_column_metadata.html

Remarks:

The count returned by *totalChanges* includes all changes from all trigger contexts and changes made by foreign key actions. But, the count does not include changes used to implement REPLACE constraints, do rollbacks or ABORT processing, or DROP TABLE processing. The count does not include rows of views that fire an INSTEAD OF trigger, though if the INSTEAD OF trigger makes changes of its own, those changes are counted.

The *changes* method can be used to get the number of changes caused by the most recent completion of a single SQL statement.

Details:

The functionality of the *totalChanges* method is similar to that of the SQLite [sqlite3_total_changes](#)¹⁰⁶ API.

3.3.40. trace

```
>>--trace(--callBackObj--+-----+-----+-----+-----+-----><
      +-,-mthName--+  +-,-userData--+
```

Registers an user callback method that can be used for tracing. The callback method is invoked at various times when an SQL statement is being run by *step*.

Arguments:

The arguments are:

callBackObj [required]

An instantiated object with a method that will be invoked for tracing.

However, this argument can also be *.nil* to indicate that any installed trace hook is to be removed.

mthName [optional]

The method name that will be invoked during a call back. By default, the method invoked will be *traceCallBack()*. However, the user can specify an alternative method if desired. This argument is ignored when the *callbackObj* argument is *.nil*.

userData [optional]

This can be any Rexx object the user desires. The object will be sent as the second argument to the trace callback method when it is invoked. This argument is ignored when the *callbackObj* argument is *.nil*.

Return value:

The *userData* argument to a previous invocation of the *trace* method on the same database connection, or *.nil* if there has not been a previous invocation or the *userData* argument was not used on the previous invocation.

Remarks:

By default, there is no trace callback installed. There can only be one trace callback per database connection. Setting a new trace callback automatically clears any previously installed callback.

¹⁰⁶ http://www.sqlite.org/c3ref/total_changes.html

The callback method is invoked at various times when an SQL statement is being executed by either *step* or *exec*. The trace callback is invoked with the SQL statement text as the statement first begins executing. Additional trace callbacks might occur as each triggered subprogram is entered. The callbacks for triggers contain a SQL comment that identifies the trigger.

Details:

The functionality of the *trace* method is similar to that of the SQLite [sqlite3_trace](#)¹⁰⁷ API.

3.3.40.1. traceCallBack

```
>>--commitHookCallBack(--sql--, --userData--)-----><
```

The *traceCallBack* method is an example of a user callback method for the *trace* method. Here the method name of *traceCallBack* is used, because it is the default method name if the programmer does not specify his own name in the *trace* method. Any method name can be used by specifying it as the second argument to the *trace* method.

Note: there is no *traceCallBack* method in any ooSQLite class. This method is just used to illustrate how to define a user callback method to be used with when tracing

Arguments:

The arguments sent to the callback method are:

sql [required]

When the statement first begins executing, *sql* will be the SQL statement text. Additional trace callbacks could occur if a triggered subprogram is entered. In these cases, *sql* will be a comment that identifies the trigger.

userData [required]

The user data object specified by the programmer as the third argument to the *trace* method. If the programmer did not specify a user data argument, this will be the `.nil` object.

Return value:

The programmer must return a whole number value from the callback, the exact number does not matter.

Details:

The implementation of a *trace* callback method is discussed on the SQLite [sqlite3_trace](#)¹⁰⁸ page.

3.3.41. updateHook

```
>>--updateHook(--callBackObj---+-----+-----+-----+-----+-----><
      +-,-mthName--+  +-,-userData-+
```

Registers a callback method to be invoked whenever a row is updated, inserted, or deleted.

¹⁰⁷ <http://www.sqlite.org/c3ref/profile.html>

¹⁰⁸ <http://www.sqlite.org/c3ref/profile.html>

Arguments:

The arguments are:

callBackObj [required]

An instantiated object with a method that will be invoked whenever a row is updated, inserted, or deleted.

However, this argument can also be `.nil` to indicate that any installed update hook is to be removed.

mthName [optional]

The method name that will be invoked during a call back. By default, the method invoked will be `updateHookCallBack()`. However, the user can specify an alternative method if desired.

This argument is ignored when the `callbackObj` argument is `.nil`.

userData [optional]

This can be any REXX object the user desires. The object will be sent as the last argument to the update hook callback method when it is invoked. This argument is ignored when the `callbackObj` argument is `.nil`.

Return value:

The `userData` argument to a previous invocation of the `updateHook` method on this database connection, or `.nil` if there has not been a previous invocation or the `userData` argument was not used on the previous invocation.

Details:

The functionality of the `updateHook` method is similar to that of the SQLite [sqlite3_update_hook](#)¹⁰⁹ API.

3.3.41.1. updateHookCallBack

```
>>--updateHookCallBack(--op--, --dbName--, --tableName--, --rowID--, --userData--)-><
```

The `updateHookCallBack` method is an example of a user callback method for the `updateHook` method. Here the method name of `updateHookCallBack` is used, because it is the default method name if the programmer does not specify his own name in the `updateHook` method. Any method name can be used by specifying it as the second argument to the `updateHook` method.

Note: there is no `updateHookCallBack` method in any ooSQLite class. This method is just used to illustrate how to define a user callback method to be used with the update hook.

Arguments:

The arguments sent to the callback method are:

op [required]

One of 3 authorizer *constants* that indicate what the update operation was. Either INSERT, DELETE, or UPDATE.

dbName [required]

The database *name* of the database containing the affected row.

¹⁰⁹ http://www.sqlite.org/c3ref/commit_hook.html

`tableName` [required]

The table name of the table containing the affected row.

`rowID` [required]

The row ID of the affected row. Every row of every SQLite table has a whole number key that uniquely identifies the row within its table. In the case of an update, this is the `rowid` after the update takes place.

`userData` [required]

The user data object specified by the programmer as the third argument to the `updateHook` method. If the programmer did not specify a user data argument, this will be the `.nil` object.

Return value:

The programmer must return a whole number value from the callback. However, the actual value returned makes no difference to the SQLite database engine. Typically, the programmer would just return 0.

Remarks:

The callback method must not do anything that will modify the database connection that invoked the callback. Any actions to modify the database connection must be deferred until after the completion of the `step` invocation that triggered the update hook to begin with. Running any other SQL statements, including SELECT statements, or merely instantiating a new `statement` object, or executing another `step` method will modify the database connection.

The update hook is not invoked when internal SQLite system tables are modified (i.e. `sqlite_master` and `sqlite_sequence`). In the current SQLite database engine implementation, the update hook is not invoked when duplicate rows are deleted because of an ON CONFLICT REPLACE clause. Nor is the update hook invoked when rows are deleted using the truncate optimization. The exceptions defined in this paragraph might change in a future release of SQLite.

Details:

The implementation of a `update hook` method is discussed on the SQLite [sqlite3_update_hook](#)¹¹⁰ page.

3.4. The ooSQLiteMutex Class

text

text

3.4.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with mutex objects using the ooSQLiteMutex class.

Table 3.6. ooSQLiteMutex Methods and Attributes

Method	Documentation
Class Methods	
new	Instantiates a new ooSQLite mutex
Attribute Methods	

¹¹⁰ http://www.sqlite.org/c3ref/update_hook.html

Method	Documentation
Instance Methods	

3.4.2. new (Class method)

```
>>--new( -+-----+ )-----><
      +--type--+
```

XX

Arguments:

The single argument is:

type [optional]

One of the SQLite *>muxtex* type constants. However this can only be: MUTEX_RECURSIVE or MUTEX_FAST. This argument defaults to MUTEX_FAST

Return value:

XX

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example ...

3.4.3. closed (Attribute)

```
>>--closed-----><
>>--closed--varName-----><
```

XX

closed get:

details about get

closed set:

details about set

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.4.4. isNull (Attribute)

```
>>--isNull-----><
>>--isNull--varName-----><
```

XX

isNull get:

details about get

isNull set:

details about set

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.4.5. enter

```
>>--enter(--+-----+--)--><
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.4.6. free

```
>>--free(---+-----+--)
      +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>leave(---+-----+--)
      +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.4.8. try

```
>>--try(--+-----+--)
      +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5. The ooSQLiteStmt Class

The **ooSQLiteStmt** class represents a prepared statement in SQLite. An instance of this object represents a single SQL statement. This object is usually referred to as a *prepared statement*, a *compiled SQL statement* or simply as a *statement*.

The life time of a statement object generally goes like this:

- Instantiate a *new* statement.
- Bind values to host parameters using one of the *bind* methods such as *bindText*, *bindInt*, or *bindNull*.
- Execute the SQL by *step* one or more times.
- Possibly, reset the statement using *reset*, then go back to step 2. Do this zero or more times.
- Release the system resources used by the statement by invoking *finalize*.

3.5.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with prepared statement objects using the ooSQLiteStmt class.

Table 3.7. ooSQLiteStmt Methods and Attributes

Method	Documentation
Class Methods	
<i>new</i>	Instantiates a new ooSQLite prepared statement.
Attribute Methods	

Method	Documentation
<i>finalized</i>	Reflects the finalized state of this statement.
<i>null</i>	Reflects the representation for the SQL NULL value that is returned by the interpreter, for this statement, for database values that are NULL.
Instance Methods	

3.5.2. new (Class method)

```
>>--new(--dbConn--, --sql---+-----+--)-->-
    +-,-format-+
```

Instantiates a new **ooSQLiteStmt** object and prepares the statement SQL to be executed.

Arguments:

The arguments are:

dbConn [required]

The open *connection* to the database that the statement will be executed on.

sql

The SQL statement to be prepared for execution

format [optional]

This argument can be used to specify the default format of the result row(s) coming out of the execution of this statement. This argument sets the *recordFormat* attribute. If this argument is omitted the default *format* value of the *dbConn* argument is used to set the attribute.

If specified, it must be one of the ooSQLite Result Set Format *Constants* that define how a result set is formatted.

Return value:

Returns a newly instantiated **ooSQLiteConnection**

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example ...

3.5.3. initCode (Attribute)

```
>>--initCode----->-
>>--initCode--varName----->-
```

initCode get:

details about get

initCode set:

details about set

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.4. finalized (Attribute)

```
>>--finalized-----><
>>--finalized---varName-----><
```

Reflects the finalized state of this statement.

finalized get:

Returns true if this statement has been finalized, otherwise false.

finalized set:

details about set

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.5. lastErrCode (Attribute)

```
>>--lastErrCode-----><
>>--lastErrCode---varName-----><
```

The *lastErrCode* attribute reflects the last error code recorded by ooSQLite.

lastErrCode get:

The value of this attribute is the last error code recorded by ooSQLite for this prepared statement.

lastErrCode set:

The Rexx programmer can not set the value of the last error code, it is set internally by ooSQLite.

Remarks:

The last error code attribute is similar to the *lastErrMsg* attribute. Its value is the last status code recorded by ooSQLite. The *lastErrCode* and the *lastErrMsg* attributes are always updated together. The error message is always the message that goes with the error code.

Each of the three major ooSQLite objects, an *ooSQLiteConnection*, an *ooSQLiteStmt*, and an *ooSQLiteBackup* object have a *lastErrMsg* and a *lastErrCode* attribute. These attributes all serve the same basic purpose, to hold the last error message and code recorded by ooSQLite.

The attributes are most useful when the invocation of *new* fails to initialize the object correctly due to an error. The *lastErrMsg* and *lastErrCode* attributes are always updated during *new*. In general, these objects do not update the attributes after every method invocation.

Details

Anything?

Example:

This example ...

```
>>--lastErrMsg-----><
>>--lastErrMsg--varName-----><
```

xx

errMsg get:

This attribute holds the value of the last error message recorded by ooSQLite.

errMsg set:

The programmer can not set the value of this attribute. It is set internally by ooSQLite.

Remarks:

The last error message attribute is similar to the *lastErrCode* attribute. Its value is the last status message recorded by ooSQLite. The *lastErrCode* and the *lastErrMsg* attributes are always updated together. The error message is always the message that goes with the error code.

Each of the three major ooSQLite objects, an *ooSQLiteConnection*, an *ooSQLiteStmt*, and an *ooSQLiteBackup* object have a *lastErrMsg* and a *lastErrCode* attribute. These attributes all serve the same basic purpose, to hold the last error message and code recorded by ooSQLite.

The attributes are most useful when the invocation of *new* fails to initialize the object correctly due to an error. The *lastErrMsg* and *lastErrCode* attributes are always updated during *new*. In general, these objects do not update the attributes after every method invocation.

Details

In some circumstances, the error message from the SQLite `sqlite3_errmsg`¹¹¹ API is copied into the `lastErrMsg`. At other times the error message is specific to ooSQLite.

Example:

This example initializes a new `ooSQLiteStmt` object. When the `initCode` attribute is checked, it is seen that an error occurred, and the last error message is printed to the screen:

```
stmt = .ooSQLiteStmt~new(dbConn, 'SELECT * FROM fooods ORDER BY name;')
if stmt~initCode <> 0 then do
    say 'ooSQLiteStmt initialization error:' stmt~initCode
    say '  Error code:' stmt~initCode "('stmt~lastErrMsg')"

/* output would be similar to:

ooSQLiteStmt initialization error: 1
  Error code: 1 (no such table: fooods)

*/
```

3.5.7. null (Attribute)

```
>>--null-----><
>>--null = nullObj-----><
```

Reflects the representation for the SQL NULL value that is returned by the interpreter, for this statement, for database values that are NULL.

null get:

Returns the current object the interpreter uses for this statement for the SQL NULL value. If the programmer has not changed this attribute, its value is the value of the `null` attribute of the database connection this statement is assigned to. Normally this is the `.nil` object.

null set:

Set this attribute to either the `.nil` object, or some alternative *string* value.

Remarks:

By default, ooSQLite uses the `.nil` object to represent the SQL NULL value. Queries for values stored in the database will return the `.nil` object for any value that is SQL NULL. However, by changing the value of the `null` attribute, the Rexx programmer can change the value the interpreter returns for NULL when this statement is executed. Typically this would be done when the returned values are going to be displayed as text and the programmer would prefer to work with a string directly. Perhaps *NULL*, or *no value*.

When a `ooSQLiteStmt` object is instantiated, the `null` attribute is assigned the value of the `null` object that is used to instantiate the statement.

Note that this attribute does not affect the value the programmer must use to assign a SQL NULL to the database. The programmer must use the `.nil` object for that.

¹¹¹ <http://www.sqlite.org/c3ref/errcode.html>

Details

Raises syntax errors when incorrect usage is detected.

This attribute is provided by ooSQLite, there is no similar feature provided by SQLite.

Example:

This example sets the *null* attribute of the statement to *no value*. This allows the application to invoke the *left* method on the returned value without having to check that the return is the **.nil** object. Note that invoking the *left* method on the **.nil** object will raise a syntax condition:

```
dbConn = .ooSQLiteConnection~new(dbName, .oosQLite~OPEN_READWRITE)

sql = 'SELECT * FROM foods ORDER BY name;'
stmt = .ooSQLiteStmt~new(dbConn, sql)
stmt~null = 'no value'

say stmt~columnName(1)~left(25) || stmt~columnName(2)~left(25) ||
stmt~columnName(3)~left(25)
say '='~copies(80)

do while stmt~step == stmt~ROW
    say stmt~columnText(1)~left(25) || stmt~columnText(2)~left(25) ||
stmt~columnText(3)~left(25)
end
```

3.5.8. recordFormat (Attribute)

```
>>--recordFormat----->
>>--recordFormat--varName----->
```

XX

recordFormat get:

details about get

recordFormat set:

details about set

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.9. bindBlob

```
>>--bindBlob(-----)----->
```

```
+--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--bindDouble(--+-----+--)--><  
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--bindInt(--+-----+--)--><
```

```
+--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.12. bindInt64

```
>>--bindInt64(--+-----+--)--><  
      +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.13. bindNull

```
>>--bindNull(--+-----+--)--><
```

```
+--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--bindParameterCount(--+-----+--)--><  
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--bindParameterIndex(--+-----+--)--><
```

```
+--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--bindParameterName(--+-----+--)--><  
      +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.17. bindText

```
>>--bindText(--+-----+--)--><
```

```
+--type--+<
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--bindValue(--+-----+--)--><  
      +--type--+<
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--bindZeroBlob(--+-----+--)--><
```

```
+--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--clearBindings(--+-----+--)--><  
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.21. columnBlob

```
>>--columnBlob(--+-----+--)--><
```

```
+--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--columnBytes(--+-----+--)--><
```

3.5.22. columnBytes

```
>>--columnBytes(--+-----+--)--><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--columnBytes(--+-----+--)--><
```

3.5.23. columnCount

```
>>--columnCount(--+-----+--)--><
```

```
+--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--columnDataBaseName( --+-----+-- )-----><  
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--columnDeclType( --+-----+-- )-----><
```

```
+--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--columnDouble(--+-----+--)-->--><  
      +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.27. columnIndex

```
>>--columnIndex(--+-----+--)-->--><
```

```
+--type--+<
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--columnInt(--+-----+--)--><----->  
      +--type--+<
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--columnInt64(--+-----+--)--><----->  
      +--type--+<
```

```
+--type--+<
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--columnName(--+-----+--)--><  
      +--type--+<
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.31. columnOriginName

```
>>--columnOriginName(--+-----+--)--><
```

```
+--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--columnTableName(--+-----+--)--><  
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--columnText(--+-----+--)--><
```

```
+--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--columnType(--+-----+--)
               +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--columnValue(--+-----+--)
```

```
+--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--dataCount(--+-----+--)--><
```

```
+--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

```
>>--dbHandle(--+-----+--)--><
```

3.5.37. dbHandle

```
>>--dbHandle(--+-----+--)--><
```

```
+--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.38. finalize

```
>>--finalize(--+-----+--)--><  
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.39. reset

```
>>--reset(--+-----+--)--><
```

```
+--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.40. step

```
>>--step(--+-----+--)--><  
      +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.41. stmtBusy

```
>>--stmtBusy(--+-----+--)--><
```

```
+--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.42. stmt_READONLY

```
>>--stmt_READONLY(--+-----+--)--><  
      +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.43. stmt_STATUS

```
>>--stmt_STATUS(--+-----+--)--><
```

```
+--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

3.5.44. value

```
>>--value(--+-----+--)--><
      +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

User Defined Extension Classes

SQLite has the ability to load extensions (including new application-defined SQL functions, collating sequences, virtual tables, and VFSes) at run-time. This feature allows the code for extensions to be developed and tested separately from the application and then loaded on an as-needed basis. An SQLite extension is a shared library or DLL and typically coded in C / C++.

ooSQLite provides complete support for loading and using these external extensions. In addition, the object orientated interface includes enhancements to using external extensions, such as automatic registration when any database connection is opened. The classes related to using and working with external extensions are documented in this chapter. Most of the work to use the SQLite database engine is done through the primary ooSQLite classes which are documented in their own [chapter](#).

Support for user defined extensions written in REXX are also part of ooSQLite. This support is done through the [createCollation](#) and [createFunction](#). In the future support for user defined virtual tables is intended to be added to ooSQLite through a [createModule](#) method.

The following table lists the classes used to work with user defined external extensions and the classes needed to implement user defined extensions in REXX code:

Table 4.1. ooSQLite External Extensions Class Listing

Class	Description
The ooSQLCollation Class	Some text.
The ooSQLCollationNeeded Class	Some text.
The ooSQLExtensions Class	Some text.
The ooSQLFunction Class	Some text.
The ooSQLLibrary Class	Some text.
The ooSQLPackage Class	Some text.
The ooSQLResult Class	Some text. This class is used in the implementation of user defined extensions in REXX
The ooSQLValue Class	Some text. This class is used in the implementation of user defined extensions in REXX

4.1. The ooSQLCollation Class

text

text

4.1.1. new (Class method)

```
>>--new(---+-----+--)-->-
    +-type--+
```

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example ...

4.2. The ooSQLCollationNeeded Class

text

text

4.2.1. new (Class method)

```
>>--new(---+-----+---)----->-
    +--type--+
```

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example ...

4.3. The ooSQLExtensions Class

The **ooSQLExtensions** class provides utilities to work with user defined SQLite extensions written in native code, typically C / C++. In general these extensions are packaged in external shared libraries, although there are a few extensions that are statically linked in to the ooSQLite library. The class can be thought of as a manager of these extensions.

text

4.3.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with ???

Table 4.2. ooSQLExtensions Methods and Attributes

Method	Documentation
Class Attribute Methods	
<i>lastErrCode</i>	.
<i>lastErrMsg</i>	.
Class Methods	
<i>autoBuiltin</i>	.
<i>autoCollationNeeded</i>	.
<i>autoCollation</i>	.
<i>autoFunction</i>	.
<i>autoPackage</i>	.
<i>CancelAutoBuiltin</i>	.
<i>getLibrary</i>	.
<i>getPackage</i>	.
<i>listBuiltins</i>	.
<i>loadLibrary</i>	.
<i>loadPackage</i>	.
<i>resetAutoBuiltin</i>	.
<i>registerBuiltin</i>	.

4.3.2. lastErrCode (Attribute)

```
>>--lastErrCode----->
>>--lastErrCode = varName----->
```

Reflects the last error code set by for the **ooSQLPackage** object.

lastErrCode get:

The value of the *lastErrCode* attribute will be a SQLite result [code](#) or one of the ooSQLite specific result [codes](#).

lastErrCode set:

The programmer can not set the value of this attribute, it is set internally by the ooSQLite framework.

Remarks:

The last error code attribute is similar to the *lastErrMsg* attribute. Its value is the last status code recorded by ooSQLite. The *lastErrCode* and the *lastErrMsg* attributes are always updated together. The error message is always the message that goes with the error code.

Each of the three major ooSQLite objects, an [ooSQLPackage](#), an [ooSQLiteStmt](#), and an [ooSQLiteBackup](#) object have a *lastErrMsg* and a *lastErrCode* attribute. These attributes all serve the same basic purpose, to hold the last error message and code recorded by ooSQLite.

The attributes are most useful when the invocation of *new* fails to initialize the object correctly due to an error. The *lastErrMsg* and *lastErrCode* attributes are always updated during *new*. In general, these objects do not update the attributes after every method invocation.

Example:

This example uses the *lastErrCode* attribute to produce a meaningful error message when a database connection fails to open:

```
dbName = 'ooFoods.rdbx'
dbConn = .ooSQLiteConnection~new(dbName, .oosQLite~OPEN_READWRITE)

-- Load the package and automatically register everything in the package for
-- our open database connection.
success = .ooSQLExtensions~loadPackage('myPackage', dbConn)
if \ success then do
    say 'Failed to load package'
    say '  Error code:  ' .ooSQLExtensions~lastErrCode
    say '  Error message:' .ooSQLExtensions~lastErrMsg

    return .ooSQLExtensions~lastErrCode
end
```

4.3.3. lastErrMsg (Attribute)

```
>>--lastErrMsg-----><
>>--lastErrMsg = varName-----><
```

XX

lastErrMsg get:

details about get

lastErrMsg set:

details about set

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

4.3.4. autoBuiltin (Class method)

```
>>--autoBuiltin(-----+---)-----><
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

[REDACTED]

4.3.5. autoCollationNeeded (Class method)

```
>>--autoCollationNeeded(--+-----+--)--><  
      +--type--+  
[REDACTED]
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

[REDACTED]

4.3.6. autoCollation

```
>>--autoCollation(--+-----+--)--><  
      +--type--+  
[REDACTED]
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

4.3.7. autoFunction

```
>>--autoFunction(--+-----+--)------><  
      +-type--+  
-----
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

4.3.8. autoPackage

```
>>--autoPackage(--+-----+--)------><  
      +-type--+  
-----
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

4.3.9. cancelAutoBuiltIn

```
>>--cancelAutoBuiltIn(--+-----+--)--><
      +-type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

4.3.10. getLibrary

```
>>--getLibrary(--+-----+--)--><
      +-type--+
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

4.3.11. getPackage

```
>>--getPackage(--+-----+--)--><  
      +-type--+  
-----
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

4.3.12. listBuiltins

```
>>--listBuiltins(--+-----+--)--><  
      +-type--+  
-----
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

4.3.13. loadLibrary

```
>>>loadLibrary(-----><
           +--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

4.3.14. loadPackage

```
>>>loadPackage(-----><
           +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

4.3.15. resetAutoBuiltin

```
>>--resetAutoBuiltin(--+-----+--)--><
      +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

4.3.16. registerBuiltin

```
>>--registerBuiltin(--+-----+--)--><
```

```
+--type--+
```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

xx

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...

4.4. The ooSQLFunction Class

An **ooSQLFunction** object represents a user defined function that has been loaded from an external shared library and is usable in the ooSQLite program that loaded it. The Rexx programmer can not instantiate the object, it is instantiated internally by ooSQLite through the *ooSQLExtensions* class.

text

4.4.1. new (Class method)

```
>>--new(---+-----+--)------><
    +-type--+
```

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example ...

4.5. The ooSQLLibrary Class

text

text

4.5.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with mutex objects using the **ooSQLLibrary** class.

Table 4.3. **ooSQLLibrary** Methods and Attributes

Method	Documentation
Class Methods	
new	Instantiates a new ooSQLite mutex
Attribute Methods	
Instance Methods	

4.5.2. new (Class method)

```
>>--new(---+-----+--)--><
      +-type--+
```

XX

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example ...

4.5.3. lastErrCode (Attribute)

```
>>--lastErrCode-----><
>>--lastErrCode = varName-----><
```

Reflects the last error code set by for the **ooSQLLibrary** object.

lastErrCode get:

The value of the *lastErrCode* attribute will be a SQLite result [code](#) or one of the ooSQLite specific result [codes](#).

lastErrCode set:

The programmer can not set the value of this attribute, it is set internally by the ooSQLite framework.

Remarks:

The last error code attribute is similar to the *lastErrMsg* attribute. Its value is the last status code recorded by ooSQLite. The *lastErrCode* and the *lastErrMsg* attributes are always updated together. The error message is always the message that goes with the error code.

Each of the three major ooSQLite objects, an *ooSQLLibrary*, an *ooSQLiteStmt*, and an *ooSQLiteBackup* object have a *lastErrMsg* and a *lastErrCode* attribute. These attributes all serve the same basic purpose, to hold the last error message and code recorded by ooSQLite.

The attributes are most useful when the invocation of *new* fails to initialize the object correctly due to an error. The *lastErrMsg* and *lastErrCode* attributes are always updated during *new*. In general, these objects do not update the attributes after every method invocation.

Example:

This example uses the *lastErrCode* attribute to produce a meaningful error message when a database connection fails to open:

```
dbName = 'ooFotods.rdbx'
dbConn = .ooSQLLibrary~new(dbName, .ooSQLite~OPEN_READWRITE)

if dbConn~initCode <> 0 then do
    errRC = dbConn~lastErrCode
    errMsg = dbConn~lastErrMsg

    say 'oosQLLibrary initialization error:' dbConn~initCode
    say ' Error code:' errRC "('errMsg')'

    dbConn~close
    return errRC
end
...
```

4.5.4. lastErrMsg (Attribute)

```
>>--lastErrMsg-----><
>>--lastErrMsg = varName-----><
```

Reflects a human readable explanation, a message, of the last error code recorded by the connection object.

lastErrMsg get:

Returns a string message that corresponds to the last error code.

lastErrMsg set:

The programmer can not set this attribute, it is set internally by the ooSQLite framework.

Remarks:

The last error message attribute is similar to the *lastErrCode* attribute. Its value is the last status message recorded by ooSQLite. The *lastErrCode* and the *lastErrMsg* attributes are always updated together. The error message is always the message that goes with the error code.

Each of the three major ooSQLite objects, an *ooSQLLibrary*, an *ooSQLiteStmt*, and an *ooSQLiteBackup* object have a *lastErrMsg* and a *lastErrCode* attribute. These attributes all serve the same basic purpose, to hold the last error message and code recorded by ooSQLite.

The attributes are most useful when the invocation of *new* fails to initialize the object correctly due to an error. The *lastErrMsg* and *lastErrCode* attributes are always updated during *new*. In general, these objects do not update the attributes after every method invocation.

Example:

This example ...

4.6. The ooSQLPackage Class

text

text

4.6.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with mutex objects using the ooSQLPackage class.

Table 4.4. ooSQLPackage Methods and Attributes

Method	Documentation
Class Methods	
<i>new</i>	Instantiates a new ooSQLite mutex
Attribute Methods	
<i>lastErrCode</i>	.
<i>lastErrMsg</i>	.
Instance Methods	
<i>getCollation</i>	.
<i>getCollationNeeded</i>	.
<i>getFunction</i>	.
<i>register</i>	.

4.6.2. new (Class method)

```
>>--new-----><
```

An **ooSQLPackage** object can not be instantiated from REXX code. Rather, the programmer uses the *getPackage* method of the **ooSQLExtensions** class to get a package object.

Remarks:

In general an external ooSQLite package is loaded through the *loadPackage* method of the **ooSQLExtensions** class. For many use cases, this may be sufficient. Some use cases, such as registering a single function from the package to a specific database connection, require the package object. The loaded package object can be obtained through the *getPackage* method.

Details

Raises an error condition if invoked from REXX code.

Example:

This example is from a program where only a single function in an ooSQLite package is needed to be registered with a database connection:

```
-- Load the package
success = .ooSQLExtensions~loadPackage(packageFile)
if \ success then do
    say 'Failed to load package'
    say '  Error code:  ' .ooSQLExtensions~lastErrCode
    say '  Error message:' .ooSQLExtensions~lastErrMsg

    return .ooSQLExtensions~lastErrCode
end

...
dbConn = .ooSQLiteConnection~new(dbName, .ooSQLite~OPEN_READWRITE)

-- Get the package and register a single function
package = .ooSQLExtensions~getPackage('examplePackage')

function = package~getFunction('half')
if function == .nil then do
    say 'Failed to get function: half'
    say '  Error code:  ' package~lastErrCode
    say '  Error message:' package~lastErrMsg

    return package~lastErrCode
end

dbConn~createFunction('half', function)
```

4.6.3. lastErrCode (Attribute)

```
>>--lastErrCode-----><
>>--lastErrCode = varName-----><
```

Reflects the last error code set by for the **ooSQLPackage** object.

lastErrCode get:

The value of the *lastErrCode* attribute will be a SQLite result [code](#) or one of the ooSQLite specific result [codes](#).

lastErrCode set:

The programmer can not set the value of this attribute, it is set internally by the ooSQLite framework.

Remarks:

The last error code attribute is similar to the [lastErrMsg](#) attribute. Its value is the last status code recorded by ooSQLite. The *lastErrCode* and the *lastErrMsg* attributes are always updated together. The error message is always the message that goes with the error code.

Example:

This example uses the *lastErrCode* attribute as part of producing a meaningful error message when an **ooSQLFunction** object can not be retrieved from a package:

```
function = package~getFunction('half')
```

```

if function == .nil then do
    say 'Failed to get function: half'
    say '  Error code: ' package~lastErrCode
    say '  Error message:' package~lastErrMsg

    return package~lastErrCode
end

```

4.6.4. lastErrMsg (Attribute)

```

>>--lastErrMsg-----><
>>--lastErrMsg = varName-----><

```

Reflects a human readable explanation, a message, of the last error code recorded by the connection object.

lastErrMsg get:

Returns a string message that corresponds to the last error code.

lastErrMsg set:

The programmer can not set this attribute, it is set internally by the ooSQLite framework.

Remarks:

The last error message attribute is similar to the *lastErrCode* attribute. Its value is the last status message recorded by ooSQLite. The *lastErrCode* and the *lastErrMsg* attributes are always updated together. The error message is always the message that goes with the error code.

Example:

This example uses the *lastErrMsg* attribute to produce a meaningful error message when an **ooSQLFunction** object can not be retrieved from a package:

```

function = package~getFunction('half')
if function == .nil then do
    say 'Failed to get function: half'
    say '  Error code: ' package~lastErrCode
    say '  Error message:' package~lastErrMsg

    return package~lastErrCode
end

```

4.6.5. getCollation

```

>>--getCollation(-----+)
      +--type--+

```

xx

Arguments:

The arguments are:

TERM

xx

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...



4.6.6. getCollationNeeded

```
>>>getCollationNeeded(--+-----+--)
                  +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...



4.6.7. getFunction

```
>>>getFunction(--+-----+--)
                  +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...



4.6.8. register

```
>>>register(---+-----+--><
           +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Anything?

Example:

This example ...



4.7. The ooSQLResult Class

text

text

4.7.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with ???

Table 4.5. ooSQLResult Methods and Attributes

Method	Documentation
Class Methods	
blob	

4.7.2. blob (Class method)

```
>>--blob( ---+-----+---)-----><
      +--type--+
```

XX

Arguments:

The single argument is:
type [optional]

Return value:

XX

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example ...

4.8. The ooSQLValue Class

text

text

4.8.1. Method Table

The following table provides links to the documentation for the primary methods and attributes used in working with mutex objects using the ooSQLValue class.

Table 4.6. ooSQLValue Methods and Attributes

Method	Documentation
Class Methods	
blob	

4.8.2. blob (Class method)

```
>>--blob( ---+-----+---)-----><
```

```
+-- type --+
```

xx

Arguments:

The single argument is:
type [optional]

Return value:

xx

Remarks:

Additional comments.

Details

Raises syntax errors when incorrect arguments are detected.

Example:

This example ...

The ooSQLite Constants

ooSQLite provides a number of constant values that are used to interact with the SQLite database engine. These constants are needed in both the object orientated and classical Rexx interfaces, so they have been documented separately in this chapter. Although the constants are provided through a class object, the syntax for using an ooRexx constant is extremely simple and should present no problem for the ooSQLite programmer that is unfamiliar with classes and objects. The classic Rexx programmer need simply prefix each constant value with: **.ooSQLite~**.

This rule is absolute for every constant listed in this chapter. If the classic Rexx programmer wants to use the constant **CORRUPT** in her Rexx code, the prefix **.ooSQLite~** is added to **CORRUPT** like so

```
dbConn = ''
ret = oosqlOpen('ooFoods.rdbx', 'dbConn')

if ret == .ooSQLite~CORRUPT then do
    -- handle error in some fashion
    say 'Error return:' .ooSQLite~CORRUPT
    say 'The "ooFoods.rdbx" database file has been corrupted.'
    ...
end

/* Output would be, if the file was indeed detected to be corrupt:

Error return: 11
The "ooFoods.rdbx" database file has been corrupted.

*/
```

The **ooSQLiteConstants** class a mixin class that provides *constant* values defined using the **::constant** directive. Each constant maps to a SQLite constant and all SQLite constants have a corresponding ooSQLite constant.

Each ooSQLite constant is named the same as the SQLite constant, minus the **SQLITE_** part of the name. For instance, the SQLite constant, **SQLITE_OK** is the **OK** constant in the **ooSQLiteConstant** class.

All of the ooSQLite objects supplied by ooSQLite inherit the **ooSQLiteConstant** class.

5.1. All Constants Table

The following table lists *all* of the constant values provided by the ooSQLiteConstants class. Additional tables list subsets of the constant values grouped by purpose.

Table 5.1. The ooSQLite Constants

Constant	Meaning
Compile-Time Version Constants	
VERSION	"3.8.1" The value of this constant will vary depending on the exact version of the SQLite database engine embedded in ooSQLite. Typically this documentation is updated when ooSQLite is upgraded to use a new version of the SQLite library so that the value shown here is current.
VERSION_NUMBER	3008001 The value of this constant will vary depending on the exact version of the SQLite database engine embedded in ooSQLite. Typically this documentation

Constant	Meaning
	is updated when ooSQLite is upgraded to use a new version of the SQLite library so that the value shown here is current.
SOURCE_ID	"2013-10-17 12:57:35 c78be6d786c19073b3a6730dfe3fb1be54f5657a" The value of this constant will vary depending on the exact version of the SQLite database engine embedded in ooSQLite. Typically this documentation is updated when ooSQLite is upgraded to use a new version of the SQLite library so that the value shown here is current.
ooSQLite Specific Result Code Constants	
OO_INTERNAL_ERR	The ooSQLite framework encountered an error internally that it is unprepared to handle. It was not considered possible for the error to happen.
OO_WRONG_ARG_TYPE	An argument to a method or function is not the correct type. For instance the argument requires a number, but the supplied argument is not a number.
OO_UNEXPECTED_RESULT	The SQLite database engine returned a result that was considered to be impossible to produce.
OO_BACKUP_IN_PROGRESS	A method was invoked on this database connection object when the database is the destination of a backup in progress.
OO_BACKUP_DB_ERRSTATE	The specified source or destination database for a backup operation is in an error state.
ooSQLite Specific Result Set Format Constants	
OO_ARRAY_OF_ARRAYS	Result sets are returned as an array where each item in the array is an array. The array at item 1 is an array of the column names. The following items are the records of the result set.
OO_ARRAY_OF_DIRECTORIES	Result sets are returned as an array where each item in the array is a Directory object. Each directory in the array contains the values of the columns in a row, where the indexes are the names of the columns.
OO_STEM_OF_STEMS	Result sets are returned as a stem with the tails 0 through the number of rows. Tail 0 contains the number rows and tails 1 through n contain a stem for each row. The tails of stem of a row are the column names for the rows.
OO_CLASSIC_STEM	Result sets are returned as a stem with the tails 0 through the number of rows. Tail 0 contains the number rows and tails 1 through n contain compound tails for each row. The compound tails consists of the row number combined with each column name of the row.
Result Code Constants	
OK	Successful result
ERROR	SQL error or missing database
INTERNAL	Internal logic error in SQLite

Constant	Meaning
PERM	Access permission denied
ABORT	Callback routine requested an abort
BUSY	The database file is locked
LOCKED	A table in the database is locked
NOMEM	A malloc() failed
READONLY	Attempt to write a readonly database
INTERRUPT	Operation terminated by sqlite3_interrupt()
IOERR	Some kind of disk I/O error occurred
CORRUPT	The database disk image is malformed
NOTFOUND	Unknown opcode in sqlite3_file_control()
FULL	Insertion failed because database is full
CANTOPEN	Unable to open the database file
PROTOCOL	Database lock protocol error
EMPTY	Database is empty
SCHEMA	The database schema changed
TOOBIG	String or BLOB exceeds size limit
CONSTRAINT	Abort due to constraint violation
MISMATCH	Data type mismatch
MISUSE	Library used incorrectly
NOLFS	Uses OS features not supported on host
AUTH	Authorization denied
FORMAT	Auxiliary database format error
RANGE	2nd parameter to sqlite3_bind out of range
NOTADB	File opened that is not a database file
ROW	sqlite3_step() has another row ready
DONE	sqlite3_step() has finished executing
IOERR_READ	
IOERR_SHORT_READ	
IOERR_WRITE	
IOERR_FSYNC	
IOERR_DIR_FSYNC	
IOERR_TRUNCATE	
IOERR_FSTAT	
IOERR_UNLOCK	
IOERR_RDLOCK	
IOERR_DELETE	
IOERR_BLOCKED	
IOERR_NOMEM	

Constant	Meaning
IOERR_ACCESS	
IOERR_CHECKRESERVEDLOCK	
IOERR_LOCK	
IOERR_CLOSE	
IOERR_DIR_CLOSE	
IOERR_SHMOPEN	
IOERR_SHMSIZE	
IOERR_SHMLOCK	
IOERR_SHMMAP	
IOERR_SEEK	
IOERR_DELETE_NOENT	
IOERR_MMAP	
IOERR_GETTEMPPATH	
IOERR_CONVPATH	
LOCKED_SHAREDCACHE	
BUSY_RECOVERY	
BUSY_SNAPSHOT	
CANTOPEN_NOTEMPDIR	
CANTOPEN_ISDIR	
CANTOPEN_FULLPATH	
CANTOPEN_CONVPATH	
CORRUPT_VTAB	
READONLY_RECOVERY	
READONLY_CANTLOCK	
READONLY_ROLLBACK	
READONLY_DBMOVED	
ABORT_ROLLBACK	
CONSTRAINT_CHECK	
CONSTRAINT_COMMITHOOK	
CONSTRAINT_FOREIGNKEY	
CONSTRAINT_FUNCTION	
CONSTRAINT_NOTNULL	
CONSTRAINT_PRIMARYKEY	
CONSTRAINT_TRIGGER	
CONSTRAINT_UNIQUE	
CONSTRAINT_VTAB	
NOTICE_RECOVER_WAL	
NOTICE_RECOVER_ROLLBACK	

Constant	Meaning	
WARNING_AUTOINDEX		
	File Open Constants	
OPEN_READONLY	Ok for sqlite3_open_v2(). The database is opened in read-only mode. If the database does not already exist, an error is returned.	
OPEN_READWRITE	Ok for sqlite3_open_v2(). The database is opened for reading and writing if possible, or reading only if the file is write protected by the operating system. In either case the database must already exist, otherwise an error is returned	
OPEN_CREATE	Ok for sqlite3_open_v2(). When merged with OPEN_READWRITE, the database is opened for reading and writing, and is created if it does not already exist.	
OPEN_DELETEONCLOSE	VFS only	
OPEN_EXCLUSIVE	VFS only	
OPEN_AUTOPROXY	VFS only	
OPEN_URI	Ok for sqlite3_open_v2()	
OPEN_MAIN_DB	VFS only	
OPEN_TEMP_DB	VFS only	
OPEN_TRANSIENT_DB	VFS only	
OPEN_MAIN_JOURNAL	VFS only	
OPEN_TEMP_JOURNAL	VFS only	
OPEN_SUBJOURNAL	VFS only	
OPEN_MASTER_JOURNAL	VFS only	
OPEN_NOMUTEX	Ok for sqlite3_open_v2()	
OPEN_FULLMUTEX	Ok for sqlite3_open_v2()	
OPEN_SHAREDCACHE	Ok for sqlite3_open_v2()	
OPEN_PRIVATECACHE	Ok for sqlite3_open_v2()	
OPEN_WAL	VFS only	
	Authorizer Action Constants	
CREATE_INDEX		
CREATE_TABLE		
CREATE_TEMP_INDEX		
CREATE_TEMP_TABLE		
CREATE_TEMP_TRIGGER		
CREATE_TEMP_VIEW		
CREATE_TRIGGER		
CREATE_VIEW		
DELETE		
DROP_INDEX		
DROP_TABLE		

Constant	Meaning
DROP_TEMP_INDEX	
DROP_TEMP_TABLE	
DROP_TEMP_TRIGGER	
DROP_TEMP_VIEW	
DROP_TRIGGER	
DROP_VIEW	
INSERT	
PRAGMA	
READ	
SELECT	
TRANSACTION	
UPDATE	
ATTACH	
DETACH	
ALTER_TABLE	
REINDEX	
ANALYZE	
CREATE_VTABLE	
DROP_VTABLE	
FUNCTION	
SAVEPOINT	
RECURSIVE	
	Authorizer Return Code Constants
DENY	Abort the SQL statement with an error
IGNORE	Don't allow access, but don't generate an error
	xAccess VFS Method Constants
ACCESS_EXISTS	
ACCESS_READWRITE	Used by PRAGMA temp_store_directory
ACCESS_READ	Unused
	Checkpoint Operation Parameter Constants
CHECKPOINT_PASSIVE	
CHECKPOINT_FULL	
CHECKPOINT_RESTART	
	Configuration Option Constants
CONFIG_SINGLETHREAD	nil
CONFIG_MULTITHREAD	nil
CONFIG_SERIALIZED	nil
CONFIG_MALLOC	sqlite3_mem_methods*

Constant	Meaning
CONFIG_GETMALLOC	sqlite3_mem_methods*
CONFIG_SCRATCH	void*, int sz, int N
CONFIG_PAGECACHE	void*, int sz, int N
CONFIG_HEAP	void*, int nByte, int min
CONFIG_MEMSTATUS	boolean
CONFIG_MUTEX	sqlite3_mutex_methods*
CONFIG_GETMUTEX	sqlite3_mutex_methods*
CONFIG_LOOKASIDE	int int
CONFIG_PCACHE	no-op
CONFIG_GETPCACHE	no-op
CONFIG_LOG	xFunc, void*
CONFIG_URI	int
CONFIG_PCACHE2	sqlite3_pcache_methods2*
CONFIG_GETPCACHE2	sqlite3_pcache_methods2*
CONFIG_COVERING_INDEX_SCAN	int
CONFIG_SQLLOG	xSqllog, void*
CONFIG_MMAP_SIZE	sqlite3_int64, sqlite3_int64
CONFIG_WIN32_HEAPSIZE	int nByte
	DB Connection Configuration Constants
DBCONFIG_LOOKASIDE	void* int int
DBCONFIG_ENABLE_FKEY	int int*
DBCONFIG_ENABLE_TRIGGER	int int*
	DB Status Parameter Constants
DBSTATUS_LOOKASIDE_USED	
DBSTATUS_CACHE_USED	
DBSTATUS_SCHEMA_USED	
DBSTATUS_STMT_USED	
DBSTATUS_LOOKASIDE_HIT	
DBSTATUS_LOOKASIDE_MISS_SIZE	
DBSTATUS_LOOKASIDE_MISS_FULL	
DBSTATUS_CACHE_HIT	
DBSTATUS_CACHE_MISS	
DBSTATUS_CACHE_WRITE	
DBSTATUS_MAX	Largest defined DBSTATUS
	File Control Opcode Constants
FCNTL_LOCKSTATE	
GET_LOCKPROXYFILE	
SET_LOCKPROXYFILE	

Constant	Meaning
LAST_ERRNO	
FCNTL_SIZE_HINT	
FCNTL_CHUNK_SIZE	
FCNTL_FILE_POINTER	
FCNTL_SYNC OMITTED	
FCNTL_WIN32_AV_RETRY	
FCNTL_PERSIST_WAL	
FCNTL_OVERWRITE	
FCNTL_VFSNAME	
FCNTL_POWERSAFE_OVERWRITE	
FCNTL_PRAGMA	
FCNTL_BUSYHANDLER	
FCNTL_TEMPFILENAME	
FCNTL_MMAP_SIZE	
FCNTL_TRACE	
FCNTL_HAS_MOVED	
FCNTL_SYNC	
FCNTL_COMMIT_PHASETWO	
	Fundamental Datatype Constants
INTEGER	
FLOAT	
BLOB	
SQLNULL	
TEXT	
	Device Characteristic Constants
IOCAP_ATOMIC	
IOCAP_ATOMIC512	
IOCAP_ATOMIC1K	
IOCAP_ATOMIC2K	
IOCAP_ATOMIC4K	
IOCAP_ATOMIC8K	
IOCAP_ATOMIC16K	
IOCAP_ATOMIC32K	
IOCAP_ATOMIC64K	
IOCAP_SAFE_APPEND	
IOCAP_SEQUENTIAL	
IOCAP_UNDELETABLE_WHEN_OPEN	
IOCAP_POWERSAFE_OVERWRITE	

Constant	Meaning
Run-Time Limit Constants	
LIMIT_LENGTH	
LIMIT_SQL_LENGTH	
LIMIT_COLUMN	
LIMIT_EXPR_DEPTH	
LIMIT_COMPOUND_SELECT	
LIMIT_VDBE_OP	
LIMIT_FUNCTION_ARG	
LIMIT_ATTACHED	
LIMIT_LIKE_PATTERN_LENGTH	
LIMIT_VARIABLE_NUMBER	
LIMIT_TRIGGER_DEPTH	
File Locking Constants	
LOCK_NONE	
LOCK_SHARED	
LOCK_RESERVED	
LOCK_PENDING	
LOCK_EXCLUSIVE	
Mutex Type Constants	
MUTEX_FAST	
MUTEX_RECURSIVE	
MUTEX_STATIC_MASTER	
MUTEX_STATIC_MEM	sqlite3_malloc()
MUTEX_STATIC_OPEN	sqlite3BtreeOpen()
MUTEX_STATIC_PRNG	sqlite3_random()
MUTEX_STATIC_LRU	lru page list
MUTEX_STATIC_PMEM	sqlite3PageMalloc()
xShmLock VFS Constants	
SHM_UNLOCK	
SHM_LOCK	
SHM_SHARED	
SHM_EXCLUSIVE	
Destructor Behavior Constants	
STATIC	
TRANSIENT	
Status Parameter Constants	
STATUS_MEMORY_USED	
STATUS_PAGECACHE_USED	

Constant	Meaning
STATUS_PAGECACHE_OVERFLOW	
STATUS_SCRATCH_USED	
STATUS_SCRATCH_OVERFLOW	
STATUS_MALLOC_SIZE	
STATUS_PARSER_STACK	
STATUS_PAGECACHE_SIZE	
STATUS_SCRATCH_SIZE	
STATUS_MALLOC_COUNT	
	Status Parameter Constants (Stmt)
STMTSTATUS_FULLSCAN_STEP	This is the number of times that SQLite has stepped forward in a table as part of a full table scan. Large numbers for this counter may indicate opportunities for performance improvement through careful use of indices.
STMTSTATUS_SORT	This is the number of sort operations that have occurred. A non-zero value in this counter may indicate an opportunity to improvement performance through careful use of indices.
STMTSTATUS_AUTOINDEX	This is the number of rows inserted into transient indices that were created automatically in order to help joins run faster. A non-zero value in this counter may indicate an opportunity to improvement performance by adding permanent indices that do not need to be reinitialized each time the statement is run.
STMTSTATUS_VM_STEP	This is the number of virtual machine operations executed by the prepared statement if that number is less than or equal to 2147483647. The number of virtual machine operations can be used as a proxy for the total work done by the prepared statement. If the number of virtual machine operations exceeds 2147483647 then the value returned by this statement status code is undefined.
	Synchronization Constants
SYNC_NORMAL	
SYNC_FULL	
SYNC_DATAONLY	
	Text Encoding Constants
UTF8	
UTF16LE	
UTF16BE	
UTF16	Use native byte order
ANY	Deprecated
UTF16_ALIGNED	sqlite3_create_collation only
DETERMINISTIC	
	Virtual Table Config Option Constants

Constant	Meaning
VTAB_CONSTRAINT_SUPPORT	

5.2. Compile Time Version Constants

The following table lists the compile time version constants:

Table 5.2. The Compile Time Version Constants

Constant	Meaning
VERSION	"3.8.3" The value of this constant will vary depending on the exact version of the SQLite database engine embedded in ooSQLite. Typically this documentation is updated when ooSQLite is upgraded to use a new version of the SQLite library so that the value shown here is current.
VERSION_NUMBER	3008003 The value of this constant will vary depending on the exact version of the SQLite database engine embedded in ooSQLite. Typically this documentation is updated when ooSQLite is upgraded to use a new version of the SQLite library so that the value shown here is current.
SOURCE_ID	"2014-02-03 14:04:11 6c643e45c274e755dc5a1a65673df79261c774be" The value of this constant will vary depending on the exact version of the SQLite database engine embedded in ooSQLite. Typically this documentation is updated when ooSQLite is upgraded to use a new version of the SQLite library so that the value shown here is current.

5.3. ooSQLite Specific Constants

The following table lists constants specific to ooSQLite. These constants have no counterpart in SQLite:

Table 5.3. The ooSQLite Specific Constants

Constant	Meaning
ooSQLite Result Set Format Constants	
OO_ARRAY_OF_ARRAYS	Result sets are returned as an array where each item in the array is an array. The array at item 1 is an array of the column names. The following items are the records of the result set.
OO_ARRAY_OF_DIRECTORIES	Result sets are returned as an array where each item in the array is a Directory object. Each directory in the array contains the values of the columns in a row, where the indexes are the names of the columns.
OO_STEM_OF_STEMS	Result sets are returned as a stem with tails 0 through the number of rows. Tail 0 contains the number rows and tails 1 through n contain a stem for each row. The tails of the stem of a row are the column names for the rows.
OO_CLASSIC_STEM	Result sets are returned as a stem with tails 0 through the number of rows. Tail 0 contains the number rows and tails 1 through n contain a compound index, n.columnName for each row. The value of each stm.n.columnName variable is the value of the columnName column for the nth row.

5.4. ooSQLite Specific Result Code Constants

The following table lists result code constants specific to ooSQLite. These constants have no counterpart in SQLite:

Table 5.4. The ooSQLite Specific Result Code Constants

Constant	Meaning
ooSQLite Result Code Constants	
OO_INTERNAL_ERR	The ooSQLite framework encountered an error internally that it is unprepared to handle. It was not considered possible for the error to happen.
OO_WRONG_ARG_TYPE	An argument to a method or function is not the correct type. For instance the argument requires a number, but the supplied argument is not a number.
OO_UNEXPECTED_RESULT	The SQLite database engine returned a result that was considered to be impossible to produce.
OO_BACKUP_IN_PROGRESS	A method was invoked on this database connection object when the database is the destination of a backup in progress.
OO_BACKUP_DB_ERRSTATE	The specified source or destination database for a backup operation is in an error state.

5.5. Result Code Constants

The following table lists the result code constants. Note that these result code constants also contain what the SQLite documentation refers to as the *extended* result codes. ooSQLite has the extended result codes enabled at compile time, so there is no distinction between the result codes and the extended result codes:

Table 5.5. The Result Code Constants

Constant	Meaning
OK	Successful result
ERROR	SQL error or missing database
INTERNAL	Internal logic error in SQLite
PERM	Access permission denied
ABORT	Callback routine requested an abort
BUSY	The database file is locked
LOCKED	A table in the database is locked
NOMEM	A malloc() failed
READONLY	Attempt to write a readonly database
INTERRUPT	Operation terminated by sqlite3_interrupt()
IOERR	Some kind of disk I/O error occurred
CORRUPT	The database disk image is malformed
NOTFOUND	Unknown opcode in sqlite3_file_control()
FULL	Insertion failed because database is full
CANTOPEN	Unable to open the database file
PROTOCOL	Database lock protocol error

Constant	Meaning
EMPTY	Database is empty
SCHEMA	The database schema changed
TOOBIG	String or BLOB exceeds size limit
CONSTRAINT	Abort due to constraint violation
MISMATCH	Data type mismatch
MISUSE	Library used incorrectly
NOLFS	Uses OS features not supported on host
AUTH	Authorization denied
FORMAT	Auxiliary database format error
RANGE	2nd parameter to sqlite3_bind out of range
NOTADB	File opened that is not a database file
NOTICE	Notifications from sqlite3_log()
WARNING	Warnings from sqlite3_log()
ROW	sqlite3_step() has another row ready
DONE	sqlite3_step() has finished executing
IOERR_READ	
IOERR_SHORT_READ	
IOERR_WRITE	
IOERR_FSYNC	
IOERR_DIR_FSYNC	
IOERR_TRUNCATE	
IOERR_FSTAT	
IOERR_UNLOCK	
IOERR_RDLCK	
IOERR_DELETE	
IOERR_BLOCKED	
IOERR_NOMEM	
IOERR_ACCESS	
IOERR_CHECKRESERVEDLOCK	
IOERR_LOCK	
IOERR_CLOSE	
IOERR_DIR_CLOSE	
IOERR_SHMOPEN	
IOERR_SHMSIZE	
IOERR_SHMLOCK	
IOERR_SHMMAP	
IOERR_SEEK	
IOERR_DELETE_NOENT	

Constant	Meaning
IOERR_MMAP	
IOERR_GETTEMPPATH	
IOERR_CONVPATH	
LOCKED_SHAREDCACHE	
BUSY_RECOVERY	
BUSY_SNAPSHOT	
CANTOPEN_NOTETEMPDIR	
CANTOPEN_ISDIR	
CANTOPEN_FULLPATH	
CANTOPEN_CONVPATH	
CORRUPT_VTAB	
READONLY_RECOVERY	
READONLY_CANTLOCK	
READONLY_ROLLBACK	
READONLY_DBMOVED	
ABORT_ROLLBACK	
CONSTRAINT_CHECK	
CONSTRAINT_COMMITHOOK	
CONSTRAINT_FOREIGNKEY	
CONSTRAINT_FUNCTION	
CONSTRAINT_NOTNULL	
CONSTRAINT_PRIMARYKEY	
CONSTRAINT_TRIGGER	
CONSTRAINT_UNIQUE	
CONSTRAINT_VTAB	
NOTICE_RECOVER_WAL	
NOTICE_RECOVER_ROLLBACK	
WARNING_AUTOINDEX	

5.6. File Open Constants

The following table lists the file open constants:

Table 5.6. The File Open Constants

Constant	Meaning
OPEN_READONLY	Ok for sqlite3_open_v2(). The database is opened in read-only mode. If the database does not already exist, an error is returned.
OPEN_READWRITE	Ok for sqlite3_open_v2(). The database is opened for reading and writing if possible, or reading only if the file is write protected by the operating system. In either case the database must already exist, otherwise an error is returned

Constant	Meaning
OPEN_CREATE	Ok for sqlite3_open_v2(). When merged with OPEN_READWRITE, the database is opened for reading and writing, and is created if it does not already exist.
OPEN_DELETEONCLOSE	VFS only
OPEN_EXCLUSIVE	VFS only
OPEN_AUTOPROXY	VFS only
OPEN_URI	Ok for sqlite3_open_v2()
OPEN_MAIN_DB	VFS only
OPEN_TEMP_DB	VFS only
OPEN_TRANSIENT_DB	VFS only
OPEN_MAIN_JOURNAL	VFS only
OPEN_TEMP_JOURNAL	VFS only
OPEN_SUBJOURNAL	VFS only
OPEN_MASTER_JOURNAL	VFS only
OPEN_NOMUTEX	Ok for sqlite3_open_v2()
OPEN_FULLMUTEX	Ok for sqlite3_open_v2()
OPEN_SHAREDCACHE	Ok for sqlite3_open_v2()
OPEN_PRIVATECACHE	Ok for sqlite3_open_v2()
OPEN_WAL	VFS only

5.7. Authorizer Action Constants

The following table lists the authorizer action constants:

Table 5.7. The Authorizer Action Constants

Constant	Meaning
CREATE_INDEX	
CREATE_TABLE	
CREATE_TEMP_INDEX	
CREATE_TEMP_TABLE	
CREATE_TEMP_TRIGGER	
CREATE_TEMP_VIEW	
CREATE_TRIGGER	
CREATE_VIEW	
DELETE	
DROP_INDEX	
DROP_TABLE	
DROP_TEMP_INDEX	
DROP_TEMP_TABLE	
DROP_TEMP_TRIGGER	
DROP_TEMP_VIEW	

Constant	Meaning
DROP_TRIGGER	
DROP_VIEW	
INSERT	
PRAGMA	
READ	
SELECT	
TRANSACTION	
UPDATE	
ATTACH	
DETACH	
ALTER_TABLE	
REINDEX	
ANALYZE	
CREATE_VTABLE	
DROP_VTABLE	
FUNCTION	
SAVEPOINT	
RECURSIVE	

5.8. Authorizer Return Code Constants

The following table lists the authorizer return code constants:

Table 5.8. The Authorizer Return Code Constants

Constant	Meaning
DENY	Abort the SQL statement with an error
IGNORE	Don't allow access, but don't generate an error
OK	The operation requested is ok

5.9. xAccess VFS Method Constants

The following table lists the authorizer return code constants:

Table 5.9. The xAccess VFS Method Constants

Constant	Meaning
ACCESS_EXISTS	
ACCESS_READWRITE	Used by PRAGMA temp_store_directory
ACCESS_READ	Unused

5.10. Checkpoint Operation Parameter Constants

The following table lists the checkpoint operation parameter constants:

Table 5.10. The Checkpoint Operation Parameter Constants

Constant	Meaning
CHECKPOINT_PASSIVE	
CHECKPOINT_FULL	
CHECKPOINT_RESTART	

5.11. Configuration Option Constants

The following table lists the configuration options constants:

Table 5.11. The Configuration Options Constants

Constant	Meaning
CONFIG_SINGLETHREAD	nil
CONFIG_MULTITHREAD	nil
CONFIG_SERIALIZED	nil
CONFIG_MALLOC	sqlite3_mem_methods*
CONFIG_GETMALLOC	sqlite3_mem_methods*
CONFIG_SCRATCH	void*, int sz, int N
CONFIG_PAGECACHE	void*, int sz, int N
CONFIG_HEAP	void*, int nByte, int min
CONFIG_MEMSTATUS	boolean
CONFIG_MUTEX	sqlite3_mutex_methods*
CONFIG_GETMUTEX	sqlite3_mutex_methods*
CONFIG_LOOKASIDE	int int
CONFIG_PCACHE	no-op
CONFIG_GETPCACHE	no-op
CONFIG_LOG	xFunc, void*
CONFIG_URI	int
CONFIG_PCACHE2	sqlite3_pcache_methods2*
CONFIG_GETPCACHE2	sqlite3_pcache_methods2*
CONFIG_COVERING_INDEX_SCAN	
CONFIG_SQLLOG	xSqllog, void*
CONFIG_MMAP_SIZE	sqlite3_int64, sqlite3_int64
CONFIG_WIN32_HEAPSIZE	int nByte

5.12. DB Connection Configuration Constants

The following table lists the DB connection configuration constants:

Table 5.12. The DB Connection Configuration Constants

Constant	Meaning
DBCONFIG_LOOKASIDE	void* int int
DBCONFIG_ENABLE_FKEY	int int*

Constant	Meaning
DBCONFIG_ENABLE_TRIGGER	int int*

5.13. DB Status Parameter Constants

The following table lists the DB status parameter constants:

Table 5.13. The DB Status Parameter Constants

Constant	Meaning
DBSTATUS_LOOKASIDE_USED	The number of lookaside memory slots currently checked out.
DBSTATUS_LOOKASIDE_HIT	The number of malloc attempts that were satisfied using lookaside memory. Only the high-water value is meaningful; the current value is always zero.
DBSTATUS_LOOKASIDE_MISS_SIZE	The number malloc attempts that might have been satisfied using lookaside memory but failed due to the amount of memory requested being larger than the lookaside slot size. Only the high-water value is meaningful, the current value is always zero.
DBSTATUS_LOOKASIDE_MISS_FULL	The number malloc attempts that might have been satisfied using lookaside memory but failed due to all lookaside memory already being in use. Only the high-water value is meaningful, the current value is always zero.
DBSTATUS_CACHE_USED	The approximate number of bytes of heap memory used by all pager caches associated with the database connection. The highwater mark associated with DBSTATUS_CACHE_USED is always 0.
DBSTATUS_SCHEMA_USED	The approximate number of bytes of heap memory used to store the schema for all databases associated with the connection, main, temp, and any ATTACHED databases. The full amount of memory used by the schemas is reported, even if the schema memory is shared with other database connections due to shared cache mode being enabled. The highwater mark associated with DBSTATUS_SCHEMA_USED is always 0.
DBSTATUS_STMT_USED	the approximate number of bytes of heap and lookaside memory used by all prepared statements associated with the database connection. The highwater mark associated with DBSTATUS_STMT_USED is always 0.
DBSTATUS_CACHE_HIT	The number of pager cache hits that have occurred. The highwater mark associated with DBSTATUS_CACHE_HIT is always 0.
DBSTATUS_CACHE_MISS	The number of pager cache misses that have occurred. The highwater mark associated with DBSTATUS_CACHE_MISS is always 0.

Constant	Meaning
DBSTATUS_CACHE_WRITE	The number of dirty cache entries that have been written to disk. Specifically, the number of pages written to the wal file in wal mode databases, or the number of pages written to the database file in rollback mode databases. Any pages written as part of transaction rollback or database recovery operations are not included. If an IO or other error occurs while writing a page to disk, the effect on subsequent DBSTATUS_CACHE_WRITE requests is undefined. The highwater mark associated with DBSTATUS_CACHE_WRITE is always 0.
DBSTATUS_MAX	Largest defined DBSTATUS

5.14. File Control Opcode Constants

The following table lists the file control opcode constants:

Table 5.14. The File Control Opcode Constants

Constant	Meaning
FCNTL_LOCKSTATE	
GET_LOCKPROXYFILE	
SET_LOCKPROXYFILE	
LAST_ERRNO	
FCNTL_SIZE_HINT	
FCNTL_CHUNK_SIZE	
FCNTL_FILE_POINTER	
FCNTL_SYNC OMITTED	
FCNTL_WIN32_AV_RETRY	
FCNTL_PERSIST_WAL	
FCNTL_OVERWRITE	
FCNTL_VFSNAME	
FCNTL_POWERSAFE_OVERWRITE	
FCNTL_PRAGMA	
FCNTL_BUSYHANDLER	
FCNTL_TEMPFILENAME	
FCNTL_MMAP_SIZE	
FCNTL_TRACE	
FCNTL_HAS_MOVED	
FCNTL_SYNC	
FCNTL_COMMIT_PHASETWO	

5.15. Fundamental Datatype Constants

The following table lists the fundamental datatype constants:

Table 5.15. The Fundamental Datatype Constants

Constant	Meaning
INTEGER	
FLOAT	
BLOB	
NULL	
TEXT	

5.16. Device Characteristic Constants

The following table lists the device characteristic constants:

Table 5.16. The Device Characteristic Constants

Constant	Meaning
IOCAP_ATOMIC	
IOCAP_ATOMIC512	
IOCAP_ATOMIC1K	
IOCAP_ATOMIC2K	
IOCAP_ATOMIC4K	
IOCAP_ATOMIC8K	
IOCAP_ATOMIC16K	
IOCAP_ATOMIC32K	
IOCAP_ATOMIC64K	
IOCAP_SAFE_APPEND	
IOCAP_SEQUENTIAL	
IOCAP_UNDELETABLE_WHEN_OPEN	
IOCAP_POWERSAFE_OVERWRITE	

5.17. Run-Time Limit Constants

The following table lists the run-time limit constants:

Table 5.17. The Run-Time Limit Constants

Constant	Meaning
LIMIT_LENGTH	The maximum size of any string or BLOB or table row, in bytes.
LIMIT_SQL_LENGTH	The maximum length of an SQL statement, in bytes.
LIMIT_COLUMN	The maximum number of columns in a table definition or in the result set of a SELECT or the maximum number of columns in an index or in an ORDER BY or GROUP BY clause.
LIMIT_EXPR_DEPTH	The maximum depth of the parse tree on any expression.
LIMIT_COMPOUND_SELECT	The maximum number of terms in a compound SELECT statement.

Constant	Meaning
LIMIT_VDBE_OP	The maximum number of instructions in a virtual machine program used to implement an SQL statement. This limit is not currently enforced, though that might be added in some future release of SQLite.
LIMIT_FUNCTION_ARG	The maximum number of arguments on a function.
LIMIT_ATTACHED	The maximum number of attached databases.
LIMIT_LIKE_PATTERN_LENGTH	The maximum length of the pattern argument to the LIKE or GLOB operators.
LIMIT_VARIABLE_NUMBER	The maximum index number of any parameter in an SQL statement.
LIMIT_TRIGGER_DEPTH	The maximum depth of recursion for triggers.

5.18. File Locking Constants

The following table lists the file locking constants:

Table 5.18. The File Locking Constants

Constant	Meaning
LOCK_NONE	
LOCK_SHARED	
LOCK_RESERVED	
LOCK_PENDING	
LOCK_EXCLUSIVE	

5.19. Mutex Type Constants

The following table lists the mutex type constants:

Table 5.19. The Mutex Type Constants

Constant	Meaning
MUTEX_FAST	
MUTEX_RECURSIVE	
MUTEX_STATIC_MASTER	
MUTEX_STATIC_MEM	sqlite3_malloc()
MUTEX_STATIC_OPEN	sqlite3BtreeOpen()
MUTEX_STATIC_PRNG	sqlite3_random()
MUTEX_STATIC_LRU	lru page list
MUTEX_STATIC_PMEM	sqlite3PageMalloc()

5.20. xShmLock VFS Constants

The following table lists the xShmLock VFS constants:

Table 5.20. The xShmLock VFS Constants

Constant	Meaning
SHM_UNLOCK	
SHM_LOCK	
SHM_SHARED	
SHM_EXCLUSIVE	

5.21. Destructor Behavior Constants

The following table lists the destructor behavior constants:

Table 5.21. The Destructor Behavior Open Constants

Constant	Meaning
STATIC	
TRANSIENT	

5.22. Status Parameter Constants

The following table lists the status parameter constants:

Table 5.22. The Status Parameter Constants

Constant	Meaning
STATUS_MEMORY_USED	This parameter is the current amount of memory checked out using the SQLite malloc routine, either directly or indirectly. The figure includes calls made to the routine by the application and internal memory usage by the SQLite library. Scratch memory controlled by CONFIG_SCRATCH and auxiliary page-cache memory controlled by CONFIG_PAGECACHE is not included in this parameter. The amount returned is the sum of the allocation sizes as reported by the xSize method in sqlite3_mem_methods.
STATUS_PAGECACHE_USED	
STATUS_PAGECACHE_OVERFLOW	
STATUS_SCRATCH_USED	
STATUS_SCRATCH_OVERFLOW	
STATUS_MALLOC_SIZE	
STATUS_PARSER_STACK	
STATUS_PAGECACHE_SIZE	
STATUS_SCRATCH_SIZE	
STATUS_MALLOC_COUNT	

5.23. Status Parameter (stmt) Constants

The following table lists the status parameter (stmt) constants:

Table 5.23. The Status Parameter (stmt) Constants

Constant	Meaning
STMTSTATUS_FULLSCAN_STEP	This is the number of times that SQLite has stepped forward in a table as part of a full table scan. Large numbers for this counter may indicate opportunities for performance improvement through careful use of indices.
STMTSTATUS_SORT	This is the number of sort operations that have occurred. A non-zero value in this counter may indicate an opportunity to improvement performance through careful use of indices.
STMTSTATUS_AUTOINDEX	This is the number of rows inserted into transient indices that were created automatically in order to help joins run faster. A non-zero value in this counter may indicate an opportunity to improvement performance by adding permanent indices that do not need to be reinitialized each time the statement is run.
STMTSTATUS_VM_STEP	This is the number of virtual machine operations executed by the prepared statement if that number is less than or equal to 2147483647. The number of virtual machine operations can be used as a proxy for the total work done by the prepared statement. If the number of virtual machine operations exceeds 2147483647 then the value returned by this statement status code is undefined.

5.24. Synchronization Constants

The following table lists the synchronization constants:

Table 5.24. The Synchronization Constants

Constant	Meaning
SYNC_NORMAL	
SYNC_FULL	
SYNC_DATAONLY	

5.25. Text Encoding Constants

The following table lists the text encoding constants:

Table 5.25. The Text Encoding Constants

Constant	Meaning
UTF8	
UTF16LE	
UTF16BE	
UTF16	Use native byte order
ANY	Deprecated
UTF16_ALIGNED	sqlite3_create_collation only
DETERMINISTIC	

5.26. Virtual Table Config Option Constants

The following table lists the virtual table config option constants:

Table 5.26. The Virtual Table Config Option Constants

Constant	Meaning
VTAB_CONSTRAINT_SUPPORT	

5.27. merge (Class method)

```
+--, --+
V   |
>>--merge(--value--+--)--><
```

Performs a bit-wise *or* operation on the arguments and returns the result.

Arguments:

The arguments are:

value

One or more whole numbers that will be *merged* together. The *value* argument can repeat any number of times, but the series of values can not omit an argument in the middle of the series. Although this method will work with any numbers, it is intended to be used with values that are **ooSQLiteConstant** values.

Return value:

Returns the result of performing a bit-wise *or* operation on the supplied numbers.

Remarks:

In some cases when the SQLite constants are used as arguments to a method or function, the constants are actually bit flags that are meant to be *or'd* together. The *opts* argument in the *new* method of the **ooSqliteConnection** class is an example of this. This is a common practice in C / C++ programming, not so common in Rexx. The *merge* method is provided as a convenience to the Rexx programmer.

Example:

This example

```
dbName = 'ooFoods.rdbx'
openOpts = .ooSQLite~merge(.ooSQLite~OPEN_READWRTITE, .ooSQLite~OPEN_CREATE)

dbConn = .ooSQLiteConnection~new(dbName, openOpts)
```

The Classic Rexx Interface to SQLite

The classic Rexx interface to SQLite provides a complete functional interface to SQLite. This allows the Rexx programmer who prefers to not program with objects the same access to the SQLite database [engine](#) as the object-orientated Rexx programmer.

The intent is for the classic Rexx interface to allow access to the complete functionality and feature set of SQLite. The first release of ooSQLite will not meet, and is not intended to meet, that goal. Lesser used functionality will be added over time.

The [object-orientated](#) and classic Rexx interfaces are developed in tandem. As each new feature or functionality of SQLite is added to ooSQLite, access to the feature is added to both interfaces at the same time. There is no SQLite functionality in the classic Rexx interface that can not be accessed through the object-orientated interface. And, vice versa.

The following table lists all of the routines used in the functional (classic Rexx) interface of the ooSQLite package:

Table 6.1. ooSQLite Routine Listing

Routine	Description
ooSQLiteEnquote()	Converts the supplied Rexx value(s) into SQL literals.
ooSQLiteMerge()	description
ooSQLiteRegisterBuiltin()	description
ooSQLiteVersion()	description
oosqlAutoExtension()	.
oosqlBackupFinish()	.
oosqlBackupInit()	.
oosqlBackupPageCount()	.
oosqlBackupRemaining()	.
oosqlBackupStep()	.
oosqlBindBlob()	.
oosqlBindDouble()	.
oosqlBindInt()	.
oosqlBindInt64()	.
oosqlBindNull()	.
oosqlBindParameterCount()	.
oosqlBindParameterIndex()	.
oosqlBindParameterName()	.
oosqlBindText()	.
oosqlBindValue()	.
oosqlBindZeroBlob()	.
oosqlBusyHandler()	.
oosqlBusyTimeOut()	.
oosqlChanges()	.
oosqlClearBindings()	.

Routine	Description
oosqlClose()	.
oosqlColumnBlob()	.
oosqlColumnBytes()	.
oosqlColumnCount()	.
oosqlColumnDatabaseName()	.
oosqlColumnDeclType()	.
oosqlColumnDouble()	.
oosqlColumnIndex()	.
oosqlColumnInt()	.
oosqlColumnInt64()	.
oosqlColumnName()	.
oosqlColumnOriginName()	.
oosqlColumnTableName()	.
oosqlColumnText()	.
oosqlColumnType()	.
oosqlColumnValue()	.
oosqlCollationNeeded()	.
oosqlCommitHook()	.
oosqlCompileOptionGet()	.
oosqlCompileOptionUsed()	.
oosqlComplete()	.
oosqlCreateCollation()	.
oosqlCreateFunction()	.
oosqlDataCount()	.
oosqlDbFileName()	.
oosqlDbHandle()	.
oosqlDbMutex()	.
oosqlDbReadOnly()	.
oosqlDbReleaseMemory()	.
oosqlDbStatus()	.
oosqlEnableLoadExtension()	.
oosqlErrCode()	.
oosqlErrMsg()	.
oosqlErrStr()	Retrieves the English language descriptive string for a result code.
oosqlExec()	.
oosqlExtendedErrCode()	.
oosqlExtendedResultCodes()	.
oosqlFinalize()	.

6.1. Online Backup Feature

SQLite provides an online backup feature and the classic Rexx interface of ooSQLite provides the functions necessary to make complete use of this feature. To effectively use the online backup feature with the classic Rexx interface, the Rexx programmer should read the SQLite [documentation](#)¹ for the *Online Backup API*.. SQLite also provides an [article](#)² with two examples and commentary showing how to use the online backup API. Readers may also find useful the information in the object orientated interface's [support](#) for this feature.

These functions in ooSQLite provide the access to the online backup feature of SQLite:

- [*oosqlBackupFinish*](#).
- [*oosqlBackupInit*](#).
- [*oosqlPageCount*](#).
- [*oosqlBackupRemaining*](#).
- [*oosqlBackupStep*](#).

¹ http://www.sqlite.org/c3ref/backup_finish.html

² <http://www.sqlite.org/backup.html>

ooSQLite Specific Functions

Almost all of the functions in the Classic Rexx interface of ooSQLite have a one-to-one mapping with the SQLite API functions. A few functions are unique to ooSQLite. These functions are used to provide SQLite functionality that does not directly translate into Rexx. All functions of this type use a naming convention of ooSQLiteXXXX(), while all functions that map directly to SQLite functions use a naming convention of oosqlXXXX(). The ooSQLite unique functions are documented in this chapter.

7.1. ooSQLiteEnquote

```
>>--oosQLiteEnquote(-----+)
      +--values--+
```

Converts the supplied Rexx value(s) into SQL literals. This function is useful to help construct SQL statements. The specified Rexx object(s) are converted to SQL literals by adding single quotes to the beginning and end of the string value of the object, escaping single quotes within the string value of the object, and changing the `.nil` object to SQL NULL.

Arguments:

The single argument is:

`values` [optional]

A Rexx object, or an array-like stem of Rexx objects, to be converted to SQL literals. If this argument is omitted then `NULL` is returned.

Return value:

The string value of the specified object(s) as a SQL literal, or a comma separated list of SQL literals.

Remarks:

The `ooSQLiteEnquote` function accepts a single argument `values`. If `values` is a stem, then it must be a stem containing tails that are positive whole number indexes. The stem can contain tails 1 through N where N is the count of values to convert, and *must* contain the tail 0 whose value is N. Any tail 1 through N that is not assigned a value is converted to SQL NULL. Any tail 1 through N whose assigned value is the `.nil` object is also converted to SQL NULL. For all other tails 1 through N, the value assigned to the tail is converted to a string enclosed in single quotes. If the string contains single quotes, those single quotes are escaped. If the stem contains any other tails, other than 0 through N, those tails are ignored. If N is greater than 1, then each converted value is added to the string with a comma used as a separator.

If `values` is not a stem, then it is taken to be a single value to be converted, and is converted in the same manner as a single tail of a stem is converted, as described above. This implies that if `values` is omitted altogether, it is converted to SQL NULL.

Details

The functionality of the `ooSQLiteEnQuote` function is similar to that of the `sqlite3_mprintf`¹ SQLite API

¹ <http://www.sqlite.org/c3ref/mprintf.html>

Example:

This example shows how the enquote function can be used to create SQL INSERT statements that are not prone to SQL Injection flaws:

```
r1.0 = 4
r1.1 = "Tom"
r1.2 = "Hanks"
r1.4 = "male"

r2.0 = 4
r2.1 = "Mike"
r2.3 = "555-9988"
r2.4 = .nil

sql1 = "INSERT INTO my_table (fName, lName, phone, gender)
VALUES("ooSQLiteEnquote(r1.))";
sql2 = "INSERT INTO my_table (fName, lName, phone, gender)
VALUES("ooSQLiteEnquote(r2.))";

say sql1
say sql2

/* Output would be:

INSERT INTO my_table (fName, lName, phone, gender) VALUES('Tom', 'Hanks', NULL, 'male');
INSERT INTO my_table (fName, lName, phone, gender) VALUES('Mike', NULL, '555-9988',
NULL);

*/
```

This example shows a conversion for a single string that has an apostrophe within it:

```
str = "It's a happy day!"
say ooSQLiteEnquote(str)

/* Output would be:

'It''s a happy day!'

*/
```

7.2. ooSQLiteMerge

```
>>--ooSQLiteMerge(--+-----+--)
               +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



7.3. ooSQLiteRegisterBuiltin

```
>>--ooSQLiteRegisterBuiltin(---+-----+---+-----><
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



7.4. ooSQLiteVersion

```
>>--ooSQLiteVersion(---+-----+---+-----><
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



ooSQLite Functions A - F

8.1. oosqlAutoExtension

```
>>--oosqlAutoExtension(--+-----+--)
    +-type--+
```

XX

Arguments:

The arguments are:

TERM
XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.2. oosqlBackupFinish

```
>>--oosqlBackupFinish(--buHandle--)><
```

Called to release all resources associated with the backup operation. This function is part of the *support* for the online backup feature of SQLite.

Arguments:

The arguments are:

buHandle [required]

The non-null handle to the backup returned from *[oosqlBackupInit](#)*.

Return value:

Returns .ooSQLite~OK if no errors occurred during a call to *[oosqlBackupStep](#)*, whether or not the backup operation completed. If an out-of-memory condition or IO error occurred during any prior call of *[oosqlBackupStep](#)* using the *buHandle* argument, then *[oosqlBackupFinish](#)* returns the corresponding *[error](#)* code.

Remarks:

There should be exactly one call to *[oosqlBackupFinish](#)* for each successful call of *[oosqlBackupInit](#)*. Once *[oosqlBackupFinish](#)* has been called, *buHandle* is no longer valid and must not be used in any other call to a function.

Details

The functionality of the `oosqlBackupFinish` routine is similar to that of the [sqlite3_backup_finish](#)¹ SQLite API

Example:

This example shows part of the code that performs an online backup of a very large database which is concurrently in heavy use. If the backup does not finish in four hours, the application abandons the backup and reschedules it to a different time:

```
-- This function will return DONE if completed and BUSY if abandoned. Any other
-- return would be a fatal error.
::routine backupWithTimeLimit
  use strict arg buHandle, limit

  count = 0
  do while .true
    ret = oosqlBackupStep(buHandle, 2)
    if ret == .ooSQLite~DONE then do
      say 'Backup finished with no error.'
      outcome = .ooSQLite~DONE
      leave
    end

    if ret <> .ooSQLite~OK, ret <> .ooSQLite~BUSY, ret <> .ooSQLite~LOCKED then do
      say 'Fatal error during back up.'
      outcome = ret
      leave
    end

    if count * 2 > limit then do
      say
      say "Backup has not completed within the time limit, going to abandon the
operation."
      say
      outcome = .ooSQLite~BUSY
      leave
    end

    j = SysSleep(.5)
    count += 1

  end

  ret = oosqlBackupFinish(buHandle)

  return outcome
```

8.3. oosqlBackupInit

```
>>--oosqlBackupInit(--dstConn-, -srcConn--+-----+-----+-----+-----+-----><
          +-,-dstName--+  +-,srcName--+
```

Called once to initialize an online backup. This function is part of the [support](#) for the online backup feature of SQLite.

¹

Arguments:

The arguments are:

dstConn [required]

An open database *connection*. This database is used as the destination of the backup.

srcConn [required]

An open database *connection*. This database is used as the source of the backup.

dstName [optional]

The name of the destination backup. This is not the file name but rather the *main*, *temp*, or attached as name. If omitted, *main* is used.

srcName [optional]

The name of the source backup. Again, this is not the file name but rather the *main*, *temp*, or attached as name. If omitted, *main* is used.

Return value:

Returns a *handle* that is used in the calls to other functions that are part of the online backup feature. This handle may be null if an error occurred. See the remarks for a discussion on this.

Remarks:

If an error occurs within the call to *oosqlBackupInit*, then the handle returned will be null. Use the *oosqlIsHandleNull* routine to check for this. On error, an error code and error message are stored in the destination database connection. This error code and message can be retrieved using the *oosqlErrCode* and *oosqlErrMsg* functions. A successful call to *oosqlBackupInit* returns a non-null handle. The handle may be used with the *oosqlBackupStep* and *oosqlBackupFinish*, functions to perform the specified backup operation. The non-null handle is also used in the *oosqlBackupPageCount* and *oosqlBackupRemaining* functions.

Details

The functionality of the *oosqlBackupInit* routine is similar to that of the *sqlite3_backup_init*² SQLite API

Example:

This example initializes a backup and checks for error:

```

buHandle = oosqlBackupInit(destDB, srcDB)
if oosqlIsHandleNull(buHandle) then do
    say 'Error initializing backup. Error code:' oosqlErrCode(destDB)
    oosqlErrMsg(destDB)
    r = oosqlClose(srcDB)
    r = oosqlClose(destDB)
    return 99.
end

```

8.4. oosqlBackupPageCount

```
>>--oosqlBackupPageCount( --buHandle-- )-----><
```

² http://www.sqlite.org/c3ref/backup_finish.html#sqlite3backupinit

Returns the total number of pages in the source database file. This function is part of the [support](#) for the online backup feature of SQLite.

Arguments:

The single argument is:

buHandle [required]

The non-null handle to the backup returned from [oosqlBackupInit](#).

Return value:

The total number of pages in the source database file.

Details

The functionality of the [oosqlBackupPageCount](#) routine is similar to that of the [sqlite3_backup_pagecount](#)³ SQLite API.

8.5. oosqlBackupRemaining

```
>>--oosqlBackupRemaining(--buHandle--)-----><
```

Returns the number of pages still to be backed up in the source database file. This function is part of the [support](#) for the online backup feature of SQLite.

Arguments:

The single argument is:

buHandle [required]

The non-null handle to the backup returned from [oosqlBackupInit](#).

Return value:

The number of pages in the source database file that still need to be backed up.

Details

The functionality of the [oosqlBackupRemaining](#) routine is similar to that of the [sqlite3_backup_remaining](#)⁴ SQLite API .

Example:

This example shows how to calculate the percentage complete of a backup. The code snippet would be executed after a call to [oosqlBackupStep\(\)](#). Maybe every call, or every 5th call, ...

```
remain = oosqlBackupRemaining(buHandle)
pages  = oosqlBackupPageCount(buHandle)

percentComplete = 100 * (pages - remain) / pages

say "Backup" percentComplete "percent complete..."

/* Output might be:

Backup 8 percent complete...
Backup 16 percent complete...
Backup 24 percent complete...
Backup 32 percent complete...
```

³ http://www.sqlite.org/c3ref/backup_finish.html#sqlite3backupfinish

⁴ http://www.sqlite.org/c3ref/backup_finish.html#sqlite3backupfinish

```

Backup 40 percent complete...
Backup 48 percent complete...

*/

```

8.6. oosqlBackupStep

```
>>--oosqlBackupStep(--buHandle--, --count--)-----><
```

Copies up to *count* pages between the source and destination databases specified by the *buHandle* argument. This function is part of the [support](#) for the online backup feature of SQLite.

Arguments:

The arguments are:

buHandle [required]

The non-null handle to the backup returned from [oosqlBackupInit](#).

count

The whole number count of database pages to copy. If *count* is negative, all of the remaining pages are copied. If *count* is omitted it defaults to 5.

Return value:

If *count* pages are successfully copied, and there are still more pages to be copied, then OK (.ooSQLite~OK) is returned. If *oosqlBackupStep* successfully finishes copying all pages from source to destination, then DONE (.ooSQLite~DONE) is returned. Otherwise an error code is returned. Some errors are fatal and some are not. The remarks section further discusses this.

Remarks:

If the database engine can not obtain a required lock then *oosqlBackupStep* returns BUSY (.ooSQLite~BUSY.) If the source database connection is being used to write to the source database when *oosqlBackupStep* is called, then LOCKED is returned. The return code can also be NOMEM, READONLY, or one of the IO_ERR_XXX codes. After BUSY or LOCKED, *oosqlBackupStep* can be tried again. But NOMEM, READONLY, and IO_ERR_XXX are considered fatal. There is no point in retrying if any of those codes are returned. The application must accept that the backup operation has failed and call [finish](#) to release associated resources.

Details

The functionality of the *oosqlStep* routine is similar to that of the [sqlite3_backup_step](#)⁵ SQLite API.

Example:

This example initializes a backup, copies everything in one step, checks for error, and cleans up:

```

...
buObj = oosqlBackupInit(srcConn, dstConn)

if oosqlIsHandleNull(buObj) then do
    -- handle error
    ...
end

```

⁵ http://www.sqlite.org/c3ref/backup_finish.html#sqlite3backupfinish

```

ret = oosqlBackupStep(buObj, -1)

if ret \== .ooSQLite~DONE then do
    -- back up failed, handle error
    ret = oosqlBackupFinish(buObj)
    ...
end

-- Backup okay, we are done with the connections
-- and the backup handle, close everything ...
ret = oosqlBackupFinish(buObj)
ret = oosqlClose(dstConn)
ret = oosqlClose(srcConn)

return 0

```

8.7. oosqlBindBlob

>>--oosqlBindBlob(--+-----+--)------><
 +--type--+

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.8. oosqlBindDouble

>>--oosqlBindDouble(--+-----+--)------><
 +--type--+

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.9. oosqlBindInt

```
>>>-oosqlBindInt(---+-----+---)-><
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.10. oosqlBindInt64

```
>>>-oosqlBindInt64(---+-----+---)-><
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.11. oosqlBindNull

```
>>--oosqlBindNull(---+-----+---)-----><
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.12. oosqlBindParameterCount

```
>>--oosqlBindParameterCount(---+-----+---)-----><
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.13. oosqlBindParameterIndex

```
>>--oosqlBindParameterIndex(---+-----+--)
    +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.14. oosqlBindParameterName

```
>>--oosqlBindParameterName(---+-----+--)
    +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.15. oosqlBindText

```
>>--oosqlBindText(---+-----+---)-----><
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.16. oosqlBindValue

```
>>--oosqlBindValue(---+-----+---)-----><
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.17. oosqlBindZeroBlob

```
>>--oosqlBindZeroBlob(--+-----+--)
                         +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.18. oosqlBusyHandler

```
>>--oosqlBusyHandler(--+-----+--)
                         +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.19. oosqlBusyTimeOut

```
>>--oosqlBusyTimeOut(---+-----+--)<br/>
     +--type--+<br/>
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.20. oosqlCancelAutoBuiltin

```
>>--oosqlCancelAutoBuiltin(---+-----+--)<br/>
     +--type--+<br/>
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.21. oosqlChanges

```
>>>-oosqlChanges(---+-----+---)-><
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.22. oosqlClearBindings

```
>>>-oosqlClearBindings(---+-----+---)-><
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.23. oosqlClose

```
>>--oosqlClose(--+-----+--)
               +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.24. oosqlColumnBlob

```
>>--oosqlColumnBlob(--+-----+--)
               +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



8.25. oosqlColumnBytes

```
>>>-oosqlColumnBytes(--+-----+--)<----->
      +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

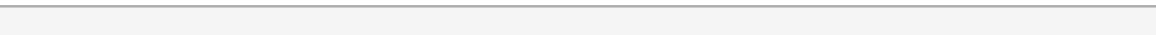
Additional comments.

Details

Additional details

Example:

This example ...



8.26. oosqlColumnCount

```
>>>-oosqlColumnCount(--+-----+--)<----->
      +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.27. oosqlColumnName

```
>>--oosqlColumnName(--+-----+--)
      +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.28. oosqlColumnDeclType

```
>>--oosqlColumnDeclType(--+-----+--)
      +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.29. oosqlColumnDouble

```
>>--oosqlColumnDouble(--+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.30. oosqlColumnIndex

```
>>--oosqlColumnIndex(--stmt--, --colName--)-----><
```

Returns the index of the column with the specified column name in the result set of a SELECT statement.

Arguments:

The arguments are:

stmt [required]

The *handle* to the *prepared* statement to be queried. The handle must not be *null* and the statement must not have been *finalized*.

colName [required]

The name of the column whose index is desired. The name is case-insensitive because SQLite does not allow column names that differ only by case.

Return value:

Returns the one-based index of the column that matches the specified name, or 0 if there is no match.

Details

This function does not access any of the SQLite APIs. It is specific to ooSQLite.

Example:

This example uses the `oosqlColumnIndex` function to get the index of the `name` column in the `foods` table:

```
dbConn = ''
ret = oosqlOpen('ooFoods.rdbx', 'dbConn')

stmt = ''
ret = oosqlPrepare(dbConn, "SELECT * FROM foods", 'stmt')
index = oosqlColumnIndex(stmt, 'name')

do while oosqlStep(stmt) == .ooSQLite~ROW
    say oosqlColumnText(stmt, index)
end

dbConn~close
stmt~finalize
```

8.31. oosqlColumnInt

```
>>--oosqlColumnInt(--+-----+--)-->--<
      +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.32. oosqlColumnInt64

```
>>--oosqlColumnInt64(---+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.33. oosqlColumnName

```
>>--oosqlColumnName(---+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.34. oosqlColumnOriginName

```
>>--oosqlColumnOriginName(---+-----+---)
    +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.35. oosqlColumnName

```
>>--oosqlColumnName(---+-----+---)
    +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.36. oosqlColumnText

```
>>--oosqlColumnText(---+-----+---)-----><
    +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.37. oosqlColumnType

```
>>--oosqlColumnType(---+-----+---)-----><
    +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.38. oosqlColumnValue

```
>>--oosqlColumnValue(--+-----+--)--><
    +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.39. oosqlCollationNeeded

```
>>--oosqlCollationNeeded(--+-----+--)--><
    +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.40. oosqlCommitHook

```
>>--oosqlCommitHook(---+-----+---)-----><
    +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.41. oosqlCompileOptionGet

```
>>--oosqlCompileOptionGet(---+-----+---)-----><
    +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.42. oosqlCompileOptionUsed

```
>>--oosqlCompileOptionUsed(---+-----+---)
    +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.43. oosqlComplete

```
>>--oosqlComplete(---+-----+---)
    +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.44. oosqlCreateCollation

```
>>--oosqlCreateCollation(---+-----+---)
    +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.45. oosqlCreateFunction

```
>>--oosqlCreateFunction(---+-----+---)
    +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.46. oosqlDataCount

```
>>--oosqlDataCount(---+-----+---)-----><
      +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.47. oosqlDbFileName

```
>>--oosqlDbFileName(---+-----+---)-----><
      +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.48. oosqlDbHandle

```
>>--oosqlDbHandle(---+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.49. oosqlDbMutex

```
>>--oosqlDbMutex(---+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.50. oosqlDbReadOnly

```
>>--oosqlDbReadOnly(--+-----+--)--><
    +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.51. oosqlDbReleaseMemory

```
>>--oosqlDbReleaseMemory(--+-----+--)--><
    +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.52. oosqlDbStatus

```
>>--oosqlDbStatus(---+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.53. oosqlEnableLoadExtension

```
>>--oosqlEnableLoadExtension(---+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.54. oosqlErrorCode

```
>>--oosqlErrorCode(--+-----+--)
      +--type--+--><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.55. oosqlErrMsg

```
>>--oosqlErrMsg(--+-----+--)
      +--type--+--><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.56. oosqlErrStr

```
>>--oosqlErrStr(--resultCode--)-----><
```

Retrieves the English language descriptive string for a result code.

Arguments:

The arguments are:

`resultCode` [required]

One of the result `code` constants, or one of the ooSQLite specific result `code`, whose descriptive sting is to be retrieved.

Return value:

Returns the descriptive, English language, string for the specified *result code*.

Remarks:

This routine is useful for getting the description of a result code without needing a *open* database connection. The `oosqlErrMsg` routine will return the descriptive string for the most recent result code associated with the database connection, but the routine requires a handle to an open database connection. The `oosqlErrStr` routine can be used at any time to get the descriptive string for a result code.

Details

The functionality of the `oosqlErrStr` function is similar to that of the `sqlite3_errstr`⁶ SQLite API

Example:

This example is a small snippet of code to print out the description of the first 27 result codes. Inspection of the `ooSQLite.cls` file shows us that the first 27 result code constants are in numerically consecutive order:

```
first = .ooSQLite~OK
last  = .ooSQLite~NOTADB

do i = first to last
    say oosqlErrStr(i)
end
say

/* Output would be:

not an error
SQL logic error or missing database
unknown error
access permission denied
callback requested query abort
database is locked
database table is locked
out of memory
attempt to write a readonly database
interrupted
disk I/O error
database disk image is malformed
unknown operation
```

⁶ <http://www.sqlite.org/c3ref/errcode.html>

```

database or disk is full
unable to open database file
locking protocol
table contains no data
database schema has changed
string or blob too big
constraint failed
datatype mismatch
library routine called out of sequence
large file support is disabled
authorization denied
auxiliary database format error
bind or column index out of range
file is encrypted or is not a database

*/

```

8.57. oosqlExec

```
>>--oosqlExec(--+-----+--><
      +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.58. oosqlExtendedErrCode

```
>>--oosqlExtendedErrCode(--+-----+--><
      +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.59. oosqlExtendedResultCodes

```
>>--oosqlExtendedResultCodes(---+-----+---)-----><
      +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

8.60. oosqlFinalize

```
>>--oosqlFinalize(---+-----+---)-----><
      +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

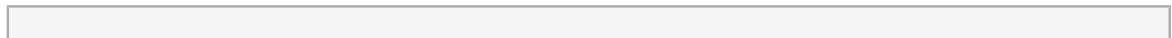
Additional comments.

Details

Additional details

Example:

This example ...



ooSQLite Functions G - R

9.1. oosqlGetAutocommit

```
>>--oosqlGetAutocommit(--+-----+--)
    +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

9.2. oosqlInterrupt

```
>>--oosqlInterrupt(--+-----+--)
    +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

9.3. oosqlIsHandleNull

```
>>>--oosqlIsHandleNull(--handle--)-----><
```

This routine provides a way for the programmer to check if a [handle](#) is null. The functions in ooSQLite that return handles will return a null handle on error. A null handle should never be used as an argument to any function.

Arguments:

The single argument is:

handle

The handle to check.

Return value:

Returns true if the handle is null, false if it is not null.

Remarks:

The [oosqlBackupInit\(\)](#), [oosqlOpen\(\)](#), [oosqlPrepare\(\)](#) and several other functions return handles. These handles will be null on error.

If the programmer is not completely adverse to using object methods, the **isNull()** method can be invoked on a handle to test for null, rather than use the [oosqlIsHandleNull\(\)](#) function. E.g

```
-- The following code snippet:  
if handle~isNull then ...  
  
-- is equivalent to this snippet using the oosqlIsHandleNull() function:  
if oosqlIsHandleNull(handle) then ...
```

Details

This function does not access any of the SQLite APIs. It is specific to ooSQLite.

Example:

This example opens a database connection and checks to be sure the returned handle is not null:

```
dbConn = oosqlOpen('contacts.rdbx')  
if oosqlIsHandleNull(dbConn) then do  
    -- handle the error ...  
end
```

9.4. oosqlLastInsertRowID

```
>>>--oosqlLastInsertRowID(---+---+---+---+-----><  
      +-type--+
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

9.5. oosqlLibVersion

```
>>--oosqlLibVersion(--+-----+--)
                         +--type--+<
```

xx

Arguments:

The arguments are:
TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

9.6. oosqlLibVersionNumber

```
>>--oosqlLibVersionNumber(--+-----+--)
                           +--type--+<
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

9.7. oosqlLimit

```
>>--oosqlLimit(--+-----+--><
           +--type--+
```

xx

Arguments:

The arguments are:
TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

9.8. oosqlLoadExtension

```
>>--oosqlLoadExtension(--+-----+--><
           +--type--+
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



9.9. oosqlMemoryHighWater

```
>>--oosqlMemoryHighWater(---+-----+--)
      +--type--+-----><
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

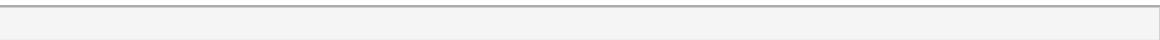
Additional comments.

Details

Additional details

Example:

This example ...



9.10. oosqlMemoryUsed

```
>>--oosqlMemoryUsed(---+-----+--)
      +--type--+-----><
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

9.11. oosqlMutexAlloc

```
>>--oosqlMutexAlloc(--+-----+--)
           +--type--+
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

9.12. oosqlMutexEnter

```
>>--oosqlMutexEnter(--+-----+--)
           +--type--+
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



9.13. oosqlMutexFree

```
>>--oosqlMutexFree(--+-----+--)
      +--type--+-----><
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

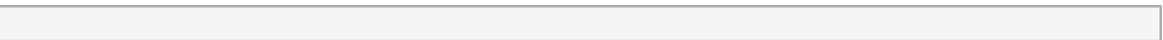
Additional comments.

Details

Additional details

Example:

This example ...



9.14. oosqlMutexLeave

```
>>--oosqlMutexLeave(--+-----+--)
      +--type--+-----><
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

9.15. oosqlMutexTry

```
>>--oosqlMutexTry(---+-----+---)-----><
      +--type--+
```

xx

Arguments:

The arguments are:
TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

9.16. oosqlINextStmt

```
>>--oosqlINextStmt(---+-----+---)-----><
      +--type--+
```

xx

Arguments:

The arguments are:

TERM
xx

Return value:

xx

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

9.17. oosqlOpen

```
>>--oosqlOpen(--dbFileName--, --'dbConn'-----+-----+---)--><
      +-,-openFlags--+  +-,-reserved--+
```

Opens a database connection.

Arguments:

The arguments are:

dbFileName [required]

The file name of the database to open. The special string `:memory:` can be used to open an in memory database. The `dbFileName` argument can also be an URI. Refer to the SQLite documentation for details.

dbConn [required]

The string `name` of a variable in the REXX program that will be set to the `handle` to the database connection. Note that this is the string name of the variable, not the variable itself. The variable may, but does not need to, already exist in the program.

openFlags [optional]

One or more of the file `open` constants. This flag controls how the database is opened. Do not use any constant marked as *VFS only*. Use `ooSQLiteMerge()` to merge two or more of the constant values together, if needed.

The 3 common flags are `OPEN_READWRITE`, `OPEN_READONLY`, and `OPEN_CREATE`. If this argument is omitted, the `OPEN_READWRITE` merged with `OPEN_CREATE` flags are used.

reserved [optional]

This argument is reserved for future enhancement. It is completely ignored in the current implementation.

Return value:

Returns one of the SQLite `result` codes. OK on success, otherwise an error code.

When the function returns, the variable named by the `dbConn` argument will always be set to a handle to the database connection. This is true even if the return code indicates an error.

Remarks:

On success, the database connection handle can be used as an argument in any other function that requires a database connection handle. To prevent resource leaks, [oosqlClose](#) must be called on each database connection returned by [oosqlOpen](#). This is true even if [oosqlOpen](#) returns an error. Never use the database connection after [oosqlClose](#) has been called. This may crash the SQLite database engine.

On error, the handle can be used in the [oosqlErrMsg](#) function to obtain a description of the error. It can also be used in the [oosqlErrCode](#) function, although at this point the return from [oosqlErrCode](#) will be the same as the return code from [oosqlOpen](#). There is one exception to this. If the error return is NOMEM the handle will be null. Do not use a null handle in any other function.

The [oosqlIsHandleNull\(\)](#) function can be used to test for a null handle. However testing for a return of NOMEM is sufficient. The handle will always be null when the return is NOMEM and not null for any other return.

The *openFlags* argument is the binary or value of the individual open constants. If the programmer is comfortable with the binary or operation, that can be used instead of the [ooSQLiteMerge\(\)](#) function.

Details

The functionality of the [oosqlOpen\(\)](#) routine is similar to that of the [sqlite3_open_v2](#)¹ SQLite API

Example:

This example opens the **ooFoods.rdbx** database and checks for any errors:

```

ret = oosqlOpen('ooFoods.rdbx', 'db')
if ret == .ooSQLite~NOMEM then do
    say 'Unrecoverable error, quitting.'
    return 99
end

if oosqlErrCode(db) <> .ooSQLite~OK then do
    -- handle the error ..
    oosqlClose(db)
    ...
end

-- We have a good connection, use the database.
...

```

9.18. oosqlPrepare

```
>>>-oosqlPrepare(--db--, --sql--, --'_stmt'---+-----+--)------><
           +-,-'_tail'---+
```

Prepares a SQL statement to be executed by the database engine. To execute a SQL statement, SQLite first compiles the statement into a byte-code program. This can be thought of as preparing, or initializing the statement.

Arguments:

The arguments are:

¹ <http://www.sqlite.org/c3ref/open.html>

db [required]

The *handle* to an *open* database connection. The handle can not be null and the connection can not have been *closed*.

sql [required]

The SQL statement to prepare.

_stmt [required]

The string *name* of a variable in the REXX program that will be set to the *handle* to the prepared statement. Note that this is the string name of the variable, not the variable itself. The variable may, but does not need to, already exist in the program.

_tail [optional]

The string *name* of a variable in the REXX program that will be set to the value of the *tail* to the prepared statement. Note that this is the string name of the variable, not the variable itself. The variable may, but does not need to, already exist in the program.

SQLite only compiles the first SQL statement in *sql*. That is up to the first semi-colon in *sql*. If *_tail* is not omitted, the the variable named by *_tail* to the substring that follows the first semicolon. This may of course be the empty string.

Return value:

Returns one of the SQLite *result* codes. OK on success, otherwise an error code.

Remarks:

The variable named by the *_stmt* argument will always be set on return from this function call. On success this will be a handle to a prepared statement which can be used in any function requiring a prepared statement. On an error return this handle will always be null. Never use a null handle in any function.

Each successful call to *oosqlPrepare* must be matched with a call to *finalize* to prevent resource leaks. *oosqlFinalize* can be called at any time after *oosqlPrepare*. Normally it would be called when the program is done with the prepared statement. Note that *oosqlFinalize* is one exception to never use a null handle in a function call. It is a harmless no-op to call *oosqlFinalize* with a null handle.

Never use the prepared statement after it has been finalized. It is a grievous error for the application to try to use a prepared statement after it has been finalized. Any use of a prepared statement after it has been finalized can result in undefined and undesirable behavior such as segfaults and heap corruption.

Note that if the *sql* argument is the empty string or contains only a comment the handle to the prepared statement will be null. Experimentation has shown the return code in this case is OK.

Details

The functionality of the *oosqlPrepare()* routine is similar to that of the *sqlite3_prepare_v2*² SQLite API.

Example:

This example creates a handle to a prepared statement and, if there is no error, executes it using the *oosqlStep* function:

```
...
```

² <http://www.sqlite.org/c3ref/prepare.html>

```

ret = oosqlPrepare(dbConn, "SELECT * FROM foods", 'stmt')
if ret == .ooSQLite~OK & \ oosqlIsHandleNull(stmt) then do
    index = oosqlColumnIndex(stmt, 'name')

    do while oosqlStep(stmt) == .ooSQLite~ROW
        say oosqlColumnText(stmt, index)
    end
end

```

9.19. oosqlProfile

```
>>--oosqlProfile(---+-----+--)
      +--type--+<
```

XX

Arguments:

The arguments are:

TERM
XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

9.20. oosqlProgressHandler

```
>>--oosqlProgressHandler(---+-----+--)
      +--type--+<
```

XX

Arguments:

The arguments are:

TERM
XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



9.21. oosqlReleaseMemory

```
>>--oosqlReleaseMemory(--+-----+--)--><
      +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



9.22. oosqlReset

```
>>--oosqlReset(--+-----+--)--><
      +-type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

9.23. oosqlResetAutoExtension

```
>>--oosqlResetAutoExtension(--+-----+--><
     +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

9.24. oosqlResultBlob

```
>>--oosqlResultBlob(--+-----+--><
     +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



9.25. oosqlResultDouble

```
>>--oosqlResultDouble(--+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



9.26. oosqlResultError

```
>>--oosqlResultError(--+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



9.27. oosqlResultErrorCode

```
>>--oosqlResultErrorCode(--+-----+--)
    +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



9.28. oosqlResultErrorNoMem

```
>>--oosqlResultErrorNoMem(--+-----+--)
    +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



9.29. oosqlResultErrorTooBig

```
>>--oosqlResultErrorTooBig(--+-----+--)
    +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



9.30. oosqlResultInt

```
>>--oosqlResultInt(--+-----+--)
    +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



9.31. oosqlResultInt64

```
>>--oosqlResultInt64(--+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



9.32. oosqlResultNull

```
>>--oosqlResultNull(--+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

```
[REDACTED]
```

9.33. oosqlResultText

```
>>--oosqlResultText(--+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

```
[REDACTED]
```

9.34. oosqlResultValue

```
>>--oosqlResultValue(--+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



9.35. oosqlResultZeroBlob

```
>>--oosqlResultZeroBlob(---+-----+---)--><
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



9.36. oosqlRollbackHook

```
>>--oosqlRollbackHook(---+-----+---)--><
      +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...



ooSQLite Functions S - Z

10.1. oosqlSetAuthorizer

```
>>--oosqlSetAuthorizer(--+-----+--)
    +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.2. oosqlSoftHeapLimit64

```
>>--oosqlSoftHeapLimit64(--+-----+--)
    +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.3. oosqlSourceID

```
>>--oosqlSourceID(--+-----+--)
    +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.4. oosqlSql

```
>>--oosqlSql(--+-----+--)
    +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.5. oosqlStatus

```
>>--oosqlStatus(--+-----+--)
           +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.6. oosqlStep

```
>>--oosqlStep(--+-----+--)
           +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.7. oosqlStmtBusy

```
>>--oosqlStmtBusy(--+-----+--)
    +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.8. oosqlStmtReadonly

```
>>--oosqlStmtReadonly(--+-----+--)
    +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.9. oosqlStmtStatus

```
>>--oosqlStmtStatus(--+-----+--)
    +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.10. oosqlStrGlob

```
>>--oosqlStrGlob(--+-----+--)
    +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.11. oosqlTableColumnMetadata

```
>>--oosqlTableColumnMetadata(--+-----+--)
    +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.12. oosqlThreadSafe

```
>>--oosqlThreadSafe(--+-----+--)
    +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.13. oosqlTotalChanges

```
>>--oosqlTotalChanges(---+-----+-->
                         +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.14. oosqlTrace

```
>>--oosqlTrace(---+-----+-->
                         +--type--+
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.15. oosqlUpdateHook

```
>>--oosqlUpdateHook(--+-----+--)
      +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.16. oosqlValueBlob

```
>>--oosqlValueBlob(--+-----+--)
      +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.17. oosqlValueBytes

```
>>--oosqlValueBytes(--+-----+--)
    +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.18. oosqlValueDouble

```
>>--oosqlValueDouble(--+-----+--)
    +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.19. oosqlValueInt

```
>>>-oosqlValueInt(---+-----+--)
           +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.20. oosqlValueNumericType

```
>>>-oosqlValueNumericType(---+-----+--)
           +--type--+<
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.21. oosqlValueText

```
>>--oosqlValueText(---+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.22. oosqlValueType

```
>>--oosqlValueType(---+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

10.23. oosqlVersion

```
>>--oosqlVersion(--+-----+--)
      +--type--+-----><
```

XX

Arguments:

The arguments are:

TERM

XX

Return value:

XX

Remarks:

Additional comments.

Details

Additional details

Example:

This example ...

Appendix A. Notices

Any reference to a non-open source product, program, or service is not intended to state or imply that only non-open source product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Rexx Language Association (RexxLA) intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-open source product, program, or service.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-open source products was obtained from the suppliers of those products, their published announcements or other publicly available sources. RexxLA has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-RexxLA packages. Questions on the capabilities of non-RexxLA packages should be addressed to the suppliers of those products.

All statements regarding RexxLA's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

A.1. Trademarks

Open Object Rexx™ and ooRexx™ are trademarks of the Rexx Language Association.

The following terms are trademarks of the IBM Corporation in the United States, other countries, or both:

1-2-3
AIX
IBM
Lotus
OS/2
S/390
VisualAge

AMD is a trademark of Advance Micro Devices, Inc.

Intel, Intel Inside (logos), MMX and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

A.2. Source Code For This Document

The source code for this document is available under the terms of the Common Public License v1.0 which accompanies this distribution and is available in the appendix [Appendix B, Common Public License Version 1.0](#). The source code is available at <https://sourceforge.net/p/oorexx/code-0/HEAD/tree/docs/>.

The source code for this document is maintained in DocBook SGML/XML format.



The railroad diagrams were generated with the help of "Railroad Diagram Generator" located at <http://bottlecaps.de/rr/ui>. Special thanks to Gunther Rademacher for creating and maintaining this tool.



Appendix B. Common Public License

Version 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

B.1. Definitions

"Contribution" means:

1. in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
2. in the case of each subsequent Contributor:
 - a. changes to the Program, and
 - b. additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

B.2. Grant of Rights

1. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.
2. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.
3. Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement

of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

4. Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

B.3. Requirements

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

1. it complies with the terms and conditions of this Agreement; and
2. its license agreement:
 - a. effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
 - b. effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
 - c. states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
 - d. states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

1. it must be made available under this Agreement; and
2. a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

B.4. Commercial Distribution

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified

Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

B.5. No Warranty

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

B.6. Disclaimer of Liability

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

B.7. General

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable.

However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. IBM is the initial Agreement Steward. IBM may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

Appendix C. Revision History

Revision 0-0 Tue Aug 7 2012

Initial creation of book by publican

David Ashley

Index

A

authorizerCallBack, 83
autoBuiltin, 121
autoCollation, 122
autoCollationNeeded, 122
autoFunction, 123
autoPackage, 123

B

backupDestination, 41
bindBlob, 99
bindDouble, 100
bindInt, 100
bindInt64, 101
bindNull, 101
bindParameterCount, 102
bindParameterIndex, 102
bindParameterName, 103
bindText, 103
bindValue, 104
bindZeroBlob, 104
blob
 ooSQLResult class, 136
 ooSQLValue class, 136
busyCallBack, 48
busyHandler, 46
busyTimeOut, 48

C

Callback Methods
 authorizerCallBack, 83
 busyCallBack, 48
 commitHookCallBack, 51
 execCallBack, 60
 profileCallBack, 66
 progressCallBack, 68
 rollbackHookCallBack, 81
 traceCallBack, 88
 updateHookCallBack, 89
cancelAutoBuiltin, 124
changes, 49
Classic Rexx Interface, 162
 online backup, 164
 oosqlAutoExtension, 169
 oosqlBackupFinish, 169
 oosqlBackupInit, 170
 oosqlBackupPageCount, 171
 oosqlBackupRemaining, 172
 oosqlBackupStep, 173
 oosqlBindBlob, 174
 oosqlBindDouble, 174

oosqlBindInt, 175
oosqlBindInt64, 175
oosqlBindNull, 176
oosqlBindParameterCount, 176
oosqlBindParameterIndex, 177
oosqlBindParameterName, 177
oosqlBindText, 178
oosqlBindValue, 178
oosqlBindZeroBlob, 179
oosqlBusyHandler, 179
oosqlBusyTimeOut, 180
oosqlCancelAutoBuiltin, 180
oosqlChanges, 181
oosqlClearBindings, 181
oosqlClose, 182
oosqlCollationNeeded, 190
oosqlColumnBlob, 182
oosqlColumnBytes, 183
oosqlColumnCount, 183
oosqlColumnNameDatabaseName, 184
oosqlColumnDeclType, 184
oosqlColumnDouble, 185
oosqlColumnIndex, 185
oosqlColumnInt, 186
oosqlColumnInt64, 187
oosqlColumnName, 187
oosqlColumnOriginName, 188
oosqlColumnNameTableName, 188
oosqlColumnText, 189
oosqlColumnType, 189
oosqlColumnValue, 190
oosqlCommitHook, 191
oosqlCompileOptionGet, 191
oosqlCompileOptionUsed, 192
oosqlComplete, 192
oosqlCreateCollation, 193
oosqlCreateFunction, 193
oosqlDataCount, 194
oosqlDbFileName, 194
oosqlDbHandle, 195
oosqlDbMutex, 195
oosqlDbReadOnly, 196
oosqlDbReleaseMemory, 196
oosqlDbStatus, 197
oosqlEnableLoadExtension, 197
oosqlErrCode, 198
oosqlErrMsg, 198
oosqlErrStr, 199
oosqlExec, 200
oosqlExtendedErrCode, 200
oosqlExtendedResultCodes, 201
oosqlFinalize, 201
oosqlGetAutocommit, 203
oosqlInterrupt, 203

oosqlIsHandleNull, 204
ooSQLiteEnquote, 165
ooSQLiteMerge, 166
ooSQLiteRegisterBuiltin, 167
ooSQLiteVersion, 167
oosqlLastInsertRowID, 204
oosqlLibVersion, 205
oosqlLibVersionNumber, 205
oosqlLimit, 206
oosqlLoadExtension, 206
oosqlMemoryHighWater, 207
oosqlMemoryUsed, 207
oosqlMutexAlloc, 208
oosqlMutexEnter, 208
oosqlMutexFree, 209
oosqlMutexLeave, 209
oosqlMutexTry, 210
oosqlNextStmt, 210
oosqlOpen, 211
oosqlPrepare, 212
oosqlProfile, 214
oosqlProgressHandler, 214
oosqlReleaseMemory, 215
oosqlReset, 215
oosqlResetAutoExtension, 216
oosqlResultBlob, 216
oosqlResultDouble, 217
oosqlResultError, 217
oosqlResultErrorCode, 218
oosqlResultErrorNoMem, 218
oosqlResultErrorTooBig, 219
oosqlResultInt, 219
oosqlResultInt64, 220
oosqlResultNull, 220
oosqlResultText, 221
oosqlResultValue, 221
oosqlResultZeroBlob, 222
oosqlRollbackHook, 222
oosqlSetAuthorizer, 224
oosqlSoftHeapLimit64, 224
oosqlSourceID, 225
oosqlSql, 225
oosqlStatus, 226
oosqlStep, 226
oosqlStmtBusy, 227
oosqlStmtReadonly, 227
oosqlStmtStatus, 228
oosqlStrGlob, 228
oosqlTableColumnMetadata, 229
oosqlThreadSafe, 229
oosqlTotalChanges, 230
oosqlTrace, 230
oosqlUpdateHook, 231
oosqlValueBlob, 231
oosqlValueBytes, 232
oosqlValueDouble, 232
oosqlValueInt, 233
oosqlValueNumericType, 233
oosqlValueText, 234
oosqlValueType, 234
oosqlVersion, 235
clearBindings, 105
close, 49
closed, 41, 91
columnBlob, 105
columnBytes, 106
columnCount, 106
columnDataBaseName, 107
columnDeclType, 107
columnDouble, 108
columnIndex, 108
columnInt, 109
columnInt64, 109
columnName, 110
columnOriginName, 110
columnTableName, 111
columnText, 111
columnType, 112
columnValue, 112
command line shell, 1
commitHook, 50
commitHookCallBack, 51
Common Public License, 238
compileOptionGet, 13
compileOptionUsed, 14
complete, 14
CPL, 238
createCollation, 52
createFunction, 52

D

dataCount, 113
dbFileName, 53
dbHandle, 113
dbMutex, 53
dbReadOnly, 54
dbReleaseMemory, 55
dbStatus, 55

E

enquote, 15
enter, 92
errCode, 57
errMsg, 57
errStr, 17
exec, 58
execCallBack, 60

extendedErrCode, 61
extendedResultCodes, 62

F

fileName, 42
finalize, 114
finalized, 96
finish, 35
finished, 30
free, 93

G

getAutocommit, 62
getCollation, 133
getCollationNeeded, 134
getDestConn, 36
getFunction, 134
getLibrary, 124
getPackage, 125

I

initCode, 30, 42, 95
interrupt, 63
isNull, 92

L

lastErrCode
 ooSQLExtensions class, 120
 ooSQLiteBackup class, 31
 ooSQLiteConnection class, 43
 ooSQLiteStmt class, 96
 ooSQLLibrary class, 129
 ooSQLPackage class, 132
lastErrMsg
 ooSQLExtensions class, 121
 ooSQLiteBackup class, 32
 ooSQLiteConnection class, 44
 ooSQLiteStmt class, 97
 ooSQLLibrary class, 130
 ooSQLPackage class, 133
lastInsertRowID, 63
leave, 93
libVersion, 18
libVersionNumber, 19
License, Common Public, 238
License, Open Object Rexx, 238
limit, 64
listBuiltins, 125
loadLibrary, 126
loadPackage, 126

M

memoryHighWater, 19

memoryUsed, 20
merge, 161

N

new, 39, 95
 ooSQLCollation class, 118
 ooSQLCollationNeeded class, 119
 ooSQLFunction class, 128
 ooSQLiteBackup class, 28
 ooSQLiteMutex class, 91
 ooSQLLibrary class, 129
 ooSQLPackage class, 131
next Stmt, 65
Notices, 236
null, 10, 45, 98

O

Object Orientated Interface, 8
 Primary Classes, 9
 User Defined Extensions, 118
Object-orientated Interface
 online backup, 27
online backup
 Classic Rexx Interface, 164
 Object-oriented Interface, 27
ooRexx License, 238
oosqlAutoExtension, 169
oosqlBackupFinish, 169
oosqlBackupInit, 170
oosqlBackupPageCount, 171
oosqlBackupRemaining, 172
oosqlBackupStep, 173
oosqlBindBlob, 174
oosqlBindDouble, 174
oosqlBindInt, 175
oosqlBindInt64, 175
oosqlBindNull, 176
oosqlBindParameterCount, 176
oosqlBindParameterIndex, 177
oosqlBindParameterName, 177
oosqlBindText, 178
oosqlBindValue, 178
oosqlBindZeroBlob, 179
oosqlBusyHandler, 179
oosqlBusyTimeOut, 180
oosqlCancelAutoBuiltin, 180
oosqlChanges, 181
oosqlClearBindings, 181
oosqlClose, 182
ooSQLCollation class, 118
 new, 118
oosqlCollationNeeded, 190
ooSQLCollationNeeded class, 119

new, 119
oosqlColumnBlob, 182
oosqlColumnBytes, 183
oosqlColumnCount, 183
oosqlColumnDatabaseName, 184
oosqlColumnDeclType, 184
oosqlColumnDouble, 185
oosqlColumnIndex, 185
oosqlColumnInt, 186
oosqlColumnInt64, 187
oosqlColumnName, 187
oosqlColumnOriginName, 188
oosqlColumnTableName, 188
oosqlColumnText, 189
oosqlColumnType, 189
oosqlColumnValue, 190
oosqlCommitHook, 191
oosqlCompileOptionGet, 191
oosqlCompileOptionUsed, 192
oosqlComplete, 192
oosqlCreateCollation, 193
oosqlCreateFunction, 193
oosqlDataCount, 194
oosqlDbFileName, 194
oosqlDbHandle, 195
oosqlDbMutex, 195
oosqlDbReadOnly, 196
oosqlDbReleaseMemory, 196
oosqlDbStatus, 197
oosqlEnableLoadExtension, 197
oosqlErrCode, 198
oosqlErrMsg, 198
oosqlErrStr, 199
oosqlExec, 200
oosqlExtendedErrCode, 200
oosqlExtendedResultCodes, 201
ooSQLExtensions class, 119
 autoBuiltin, 121
 autoCollation (Class method), 122
 autoCollationNeeded, 122
 autoFunction (Class method), 123
 autoPackage (Class method), 123
 cancelAutoBuiltin, 124
 getLibrary (Class method), 124
 getPackage (Class method), 125
 lastErrCode, 120
 lastErrMsg, 121
 listBuiltins, 125
 loadLibrary (Class method), 126
 loadPackage (Class method), 126
 registerBuiltin (Class method), 127
 resetAutoBuiltin (Class method), 127
oosqlFinalize, 201
ooSQLFunction class, 128
new, 128
oosqlGetAutocommit, 203
oosqlInterrupt, 203
oosqlIsHandleNull, 204
ooSQLite class, 9
 compileOptionGet, 13
 compileOptionUsed, 14
 complete, 14
 enquote, 15
 errStr, 17
 libVersion, 18
 libVersionNumber, 19
 memoryHighWater, 19
 memoryUsed, 20
 null, 10
 recordFormat, 11
 releaseMemory, 20
 softHeapLimit64, 21
 sourceID, 22
 sqlite3Version, 22
 status, 23
 threadSafe, 25
 version, 25
ooSQLite Constants, 138
ooSQLite package
 command line shell, 1
ooSQLiteBackup class, 27
 finish, 35
 finished, 30
 getDestConn, 36
 initCode, 30
 lastErrCode, 31
 lastErrMsg, 32
 new, 28
 pageCount, 32
 remaining, 33
 saveDestConn, 34
 step, 36
ooSQLiteConnection class, 37
 backupDestination, 41
 busyHandler, 46
 busyTimeOut, 48
 changes, 49
 close, 49
 closed, 41
 commitHook, 50
 createCollation, 52
 createFunction, 52
 dbFileName, 53
 dbMutex, 53
 dbReadOnly, 54
 dbReleaseMemory, 55
 dbStatus, 55
 errCode, 57

errMsg, 57
exec, 58
extendedErrCode, 61
extendedResultCodes, 62
fileName, 42
getAutocommit, 62
initCode, 42
interrupt, 63
lastErrCode, 43
lastErrMsg, 44
lastInsertRowID, 63
limit, 64
new, 39
nextStmt, 65
null, 45
pragma, 69
profile, 65
progressHandler, 67
recordFormat, 46
rollbackHook, 80
setAuthorizer, 81
tableColumnMetadata, 85
totalChanges, 86
trace, 87
updateHook, 88
ooSQLiteConstant class
 merge, 161
ooSQLiteConstants class, 138
ooSQLiteEnquote, 165
ooSQLiteMerge, 166
ooSQLiteMutex class, 90
 closed, 91
 enter, 92
 free, 93
 isNull, 92
 leave, 93
 new, 91
 try, 94
ooSQLiteRegisterBuiltin, 167
ooSQLiteStmt class, 94
 bindBlob, 99
 bindDouble, 100
 bindInt, 100
 bindInt64, 101
 bindNull, 101
 bindParameterCount, 102
 bindParameterIndex, 102
 bindParameterName, 103
 bindText, 103
 bindValue, 104
 bindZeroBlob, 104
 clearBindings, 105
 columnBlob, 105
 columnBytes, 106
columnCount, 106
columnDataBaseName, 107
columnDeclType, 107
columnDouble, 108
columnIndex, 108
columnInt, 109
columnInt64, 109
columnName, 110
columnOriginName, 110
columnTableName, 111
columnText, 111
columnType, 112
columnValue, 112
dataCount, 113
dbHandle, 113
finalize, 114
finalized, 96
initCode, 95
lastErrCode, 96
lastErrMsg, 97
new, 95
null, 98
recordFormat, 99
reset, 114
step, 115
stmtBusy, 115
stmtReadonly, 116
stmtStatus, 116
value, 117
ooSQLiteVersion, 167
oosqlLastInsertRowID, 204
ooSQLLibrary class, 128
 lastErrCode, 129
 lastErrMsg, 130
 new, 129
oosqlLibVersion, 205
oosqlLibVersionNumber, 205
oosqlLimit, 206
oosqlLoadExtension, 206
oosqlMemoryHighWater, 207
oosqlMemoryUsed, 207
oosqlMutexAlloc, 208
oosqlMutexEnter, 208
oosqlMutexFree, 209
oosqlMutexLeave, 209
oosqlMutexTry, 210
oosqlNextStmt, 210
oosqlOpen, 211
ooSQLPackage class, 131
 getCollation, 133
 getCollationNeeded, 134
 getFunction, 134
 lastErrCode, 132
 lastErrMsg, 133

new, 131
register, 135
oosqlPrepare, 212
oosqlProfile, 214
oosqlProgressHandler, 214
oosqlReleaseMemory, 215
oosqlReset, 215
oosqlResetAutoExtension, 216
ooSQLResult class, 135
 blob, 136
oosqlResultBlob, 216
oosqlResultDouble, 217
oosqlResultError, 217
oosqlResultErrorCode, 218
oosqlResultErrorNoMem, 218
oosqlResultErrorTooBig, 219
oosqlResultInt, 219
oosqlResultInt64, 220
oosqlResultNull, 220
oosqlResultText, 221
oosqlResultValue, 221
oosqlResultZeroBlob, 222
oosqlRollbackHook, 222
oosqlSetAuthorizer, 224
oosqlSoftHeapLimit64, 224
oosqlSourceID, 225
oosqlSql, 225
oosqlStatus, 226
oosqlStep, 226
oosqlStmtBusy, 227
oosqlStmtReadonly, 227
oosqlStmtStatus, 228
oosqlStrGlob, 228
oosqlTableColumnMetadata, 229
oosqlThreadSafe, 229
oosqlTotalChanges, 230
oosqlTrace, 230
oosqlUpdateHook, 231
ooSQLValue class, 136
 blob, 136
oosqlValueBlob, 231
oosqlValueBytes, 232
oosqlValueDouble, 232
oosqlValueInt, 233
oosqlValueNumericType, 233
oosqlValueType, 234
oosqlVersion, 235
Open Object REXX License, 238
overview, 1

P

pageCount, 32
pragma, 69

profile, 65
profileCallBack, 66
progressCallBack, 68
progressHandler, 67

R

recordFormat, 11, 46, 99
register, 135
registerBuiltin, 127
releaseMemory, 20
remaining, 33
reset, 114
resetAutoBuiltin, 127
rollbackHook, 80
rollbackHookCallBack, 81

S

saveDestConn, 34
setAuthorizer, 81
softHeapLimit64, 21
sourceID, 22
sqlite3Version, 22
status, 23
step, 36, 115
stmtBusy, 115
stmtReadonly, 116
stmtStatus, 116

T

tableColumnMetadata, 85
threadSafe, 25
totalChanges, 86
trace, 87
traceCallBack, 88
try, 94

U

updateHook, 88
updateHookCallBack, 89

V

value, 117
version, 25