

---

# **S3M Documentation**

*Release 1.1.0*

**Ivan Konovalov**

**Jun 13, 2018**



---

## Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	What else is different from <i>sqlite3</i> ? . . . . .	1
1.2	Example . . . . .	1
<b>2</b>	<b>Installation</b>	<b>3</b>
<b>3</b>	<b>Documentation</b>	<b>5</b>
3.1	Using <code>with</code> statement . . . . .	8
	<b>Python Module Index</b>	<b>9</b>



# CHAPTER 1

---

## Introduction

---

One of the problems of the built-in *sqlite3* module is that doesn't work very well with multithreading. S3M - is a wrapper of *sqlite3* that allows you to easily do multithreading:

- It locks parallel database operations so that only one can run at a time.
- It can also lock transactions (enabled by default) so that only one transaction can be active at a time.
- You won't get an *OperationalError* saying that the database is locked. All the database operations will just run in a queue.

Keep in mind that this library can only help you with **threads**, not **processes**.

### 1.1 What else is different from *sqlite3*?

- You can freely share connections between threads (not that you have to), given `check_same_thread=False`.
- You can use the `with` statement with the connection object to acquire the locks.

### 1.2 Example

```
import random
import threading

# Try replacing 's3m' with 'sqlite3' and see what happens
import s3m

# Open the database file,
# isolation_level=None is needed to prevent sqlite3 from starting transactions on its_
↳ own
conn = s3m.connect("s3m_example.db", isolation_level=None)
```

(continues on next page)

```
# Create table if it doesn't already exist
conn.execute("CREATE TABLE IF NOT EXISTS numbers(number INTEGER)")

def thread_func():
    conn = s3m.connect("s3m_example.db", isolation_level=None)
    conn.execute("BEGIN IMMEDIATE")
    conn.execute("INSERT INTO numbers VALUES(?)", (random.randint(1, 100),))

    # Imagine there's some intense database work going on
    time.sleep(1)

    conn.commit()

# Make 10 threads
threads = [threading.Thread(target=thread_func) for i in range(10)]

# Start them
for thread in threads:
    thread.start()

for thread in threads:
    thread.join()

# Now let's look at what we got
result = conn.execute("SELECT * FROM numbers").fetchall()
print(result)
```

As you can see from this example, the usage is pretty much the same as with built-in *sqlite3*.

## CHAPTER 2

---

### Installation

---

Run

```
pip install s3m
```

or

```
python setup.py install
```





S3M - sqlite3 wrapper for multithreaded applications

`s3m.connect` (*path*, *lock\_transactions=True*, *lock\_timeout=-1*, *single\_cursor\_mode=False*, *factory=<class 's3m.Connection'>*, \**args*, \*\**kwargs*)  
Analogous to `sqlite3.connect()`

#### Parameters

- **path** – Path to the database
- **lock\_transactions** – If *True*, parallel transactions will be blocked
- **lock\_timeout** – Maximum amount of time the connection is allowed to wait for a lock. If the timeout is exceeded, *LockTimeoutError* will be thrown. *-1* disables the timeout.
- **single\_cursor\_mode** – Use only one cursor (default: *False*)
- **factory** – Connection class (default: *Connection*)

**class** `s3m.Connection` (*path*, *lock\_transactions=True*, *lock\_timeout=-1*, *single\_cursor\_mode=False*, \**args*, \*\**kwargs*)

The connection class. It won't let multiple database operations execute in parallel. It can also block parallel transactions (with *lock\_transactions=True*).

*with* statement is also supported, it acquires the locks, thus blocking all the competing threads. This can be useful to ensure that database queries will complete in the specified order.

#### Parameters

- **path** – Path to the database
- **lock\_transactions** – If *True*, parallel transactions will be blocked
- **lock\_timeout** – Maximum amount of time the connection is allowed to wait for a lock. If the timeout is exceeded, *LockTimeoutError* will be thrown. *-1* disables the timeout.
- **single\_cursor\_mode** – Use only one cursor (default: *False*)

**acquire** (*lock\_transactions=None*)

Acquire the connection locks.

Parameters `lock_transactions` – *bool*, acquire the transaction lock  
(*self.lock\_transactions* is the default value)

**arraysize**

Analogous to `sqlite3.Cursor.arraysize`

Works only in single cursor mode.

**close ()**

Close the connection

**commit ()**

Analogous to `sqlite3.Connection.commit`

**create\_aggregate (\*args, \*\*kwargs)**

Analogous to `sqlite3.Connection.create_aggregate`

**create\_collation (\*args, \*\*kwargs)**

Analogous to `sqlite3.Connection.create_collation`

**create\_function (\*args, \*\*kwargs)**

Analogous to `sqlite3.Connection.create_function`

**cursor ()**

Analogous to `sqlite3.Connection.cursor`

**description**

Analogous to `sqlite3.Cursor.description`

Works only in single cursor mode.

**enable\_load\_extension (\*args, \*\*kwargs)**

Analogous to `sqlite3.Connection.enable_load_extension`

**execute (\*args, \*\*kwargs)**

Analogous to `sqlite3.Cursor.execute`

**executemany (\*args, \*\*kwargs)**

Analogous to `sqlite3.Cursor.executemany`

**executescript (\*args, \*\*kwargs)**

Analogous to `sqlite3.Cursor.executescript`

**fetchall ()**

Analogous to `sqlite3.Cursor.fetchall`.

Works only in single cursor mode.

**fetchmany (\*args, \*\*kwargs)**

Analogous to `sqlite3.Cursor.fetchmany`.

Works only in single cursor mode.

**fetchone ()**

Analogous to `sqlite3.Cursor.fetchone`.

Works only in single cursor mode.

**in\_transaction**

Analogous to `sqlite3.Connection.in_transaction`

**interrupt ()**

Analogous to `sqlite3.Connection.interrupt`

**isolation\_level**

Analogous to `sqlite3.Connection.isolation_level`

**iterdump** (\*args, \*\*kwargs)

Analogous to `sqlite3.Connection.iterdump`

**lastrowid**

Analogous to `sqlite3.Cursor.lastrowid`.

Works only in single cursor mode.

**load\_extension** (\*args, \*\*kwargs)

Analogous to `sqlite3.Connection.load_extension`

**release** (lock\_transactions=None)

Release the connection locks.

**Parameters** `lock_transactions` – *bool*, release the transaction lock  
(*self.lock\_transactions* is the default value)

**rollback** ()

Analogous to `sqlite3.Connection.rollback`

**row\_factory**

Analogous to `sqlite3.Connection.row_factory`

**rowcount**

Analogous to `sqlite3.Cursor.rowcount`.

Works only in single cursor mode.

**set\_authorizer** (\*args, \*\*kwargs)

Analogous to `sqlite3.Connection.set_authorizer`

**set\_progress\_handler** (\*args, \*\*kwargs)

Analogous to `sqlite3.Connection.set_progress_handler`

**set\_trace\_callback** (\*args, \*\*kwargs)

Analogous to `sqlite3.Connection.set_trace_callback`

**text\_factory**

Analogous to `sqlite3.Connection.text_factory`

**total\_changes**

Analogous to `sqlite3.Connection.total_changes`

**class** `s3m.Cursor` (*connection*)

The cursor class, analogous to `sqlite3.Cursor`.

**arraysize**

Analogous to `sqlite3.Cursor.arraysize`

**close** ()

Close the cursor

**connection**

Connection used by the cursor

**description**

Analogous to `sqlite3.Cursor.description`

**execute** (\*args, \*\*kwargs)

Analogous to `sqlite3.Cursor.execute`

**Returns** `self`

**executemany** (\*args, \*\*kwargs)

Analogous to `sqlite3.Cursor.executemany`

**Returns** self

**executescript** (\*args, \*\*kwargs)

Analogous to `sqlite3.Cursor.executescript`

**Returns** self

**fetchall** ()

Analogous to `sqlite3.Cursor.fetchall`

**fetchmany** (\*args, \*\*kwargs)

Analogous to `sqlite3.Cursor.fetchmany`

**fetchone** ()

Analogous to `sqlite3.Cursor.fetchone`

**lastrowid**

Analogous to `sqlite3.Cursor.lastrowid`

**rowcount**

Analogous to `sqlite3.Cursor.rowcount`

**exception** `s3m.S3MError`

The base class of all the other exceptions in this module

**exception** `s3m.LockTimeoutError` (*conn, msg=None*)

Thrown when `Lock.acquire()` took too long

## 3.1 Using with statement

The *Connection* (as well as *Cursor*) object supports the `with` statement. It acquires the locks which will result either in the current thread waiting for other threads or making other threads wait until the current thread exits the `with` block.

```
conn = s3m.connect("database.db", ...)
...
with conn: # This blocks other threads
    conn.execute(<something>)
    conn.execute(<something else>)

# The other threads are no longer blocked
```

**S**

s3m, 5



## A

acquire() (s3m.Connection method), 5  
arraysize (s3m.Connection attribute), 6  
arraysize (s3m.Cursor attribute), 7

## C

close() (s3m.Connection method), 6  
close() (s3m.Cursor method), 7  
commit() (s3m.Connection method), 6  
connect() (in module s3m), 5  
Connection (class in s3m), 5  
connection (s3m.Cursor attribute), 7  
create\_aggregate() (s3m.Connection method), 6  
create\_collation() (s3m.Connection method), 6  
create\_function() (s3m.Connection method), 6  
Cursor (class in s3m), 7  
cursor() (s3m.Connection method), 6

## D

description (s3m.Connection attribute), 6  
description (s3m.Cursor attribute), 7

## E

enable\_load\_extension() (s3m.Connection method), 6  
execute() (s3m.Connection method), 6  
execute() (s3m.Cursor method), 7  
executemany() (s3m.Connection method), 6  
executemany() (s3m.Cursor method), 7  
executescript() (s3m.Connection method), 6  
executescript() (s3m.Cursor method), 8

## F

fetchall() (s3m.Connection method), 6  
fetchall() (s3m.Cursor method), 8  
fetchmany() (s3m.Connection method), 6  
fetchmany() (s3m.Cursor method), 8  
fetchone() (s3m.Connection method), 6  
fetchone() (s3m.Cursor method), 8

## I

in\_transaction (s3m.Connection attribute), 6  
interrupt() (s3m.Connection method), 6  
isolation\_level (s3m.Connection attribute), 6  
iterdump() (s3m.Connection method), 7

## L

lastrowid (s3m.Connection attribute), 7  
lastrowid (s3m.Cursor attribute), 8  
load\_extension() (s3m.Connection method), 7  
LockTimeoutError, 8

## R

release() (s3m.Connection method), 7  
rollback() (s3m.Connection method), 7  
row\_factory (s3m.Connection attribute), 7  
rowcount (s3m.Connection attribute), 7  
rowcount (s3m.Cursor attribute), 8

## S

s3m (module), 5  
S3MError, 8  
set\_authorizer() (s3m.Connection method), 7  
set\_progress\_handler() (s3m.Connection method), 7  
set\_trace\_callback() (s3m.Connection method), 7

## T

text\_factory (s3m.Connection attribute), 7  
total\_changes (s3m.Connection attribute), 7