

---

# **extensions**

***Release 1.0***

**Mark Anacker**

**Dec 07, 2023**



# CONTENTS

- 1 extensions package** **1**
  - 1.1 Submodules . . . . . 1
    - 1.1.1 extensions.fileext module . . . . . 1
    - 1.1.2 extensions.mqtttext module . . . . . 4
    - 1.1.3 extensions.queueext module . . . . . 7
    - 1.1.4 extensions.redisext module . . . . . 11
    - 1.1.5 extensions.sqliteext module . . . . . 16
    - 1.1.6 extensions.tdictext module . . . . . 22
    - 1.1.7 extensions.utilext module . . . . . 26
  - 1.2 Module contents . . . . . 28
- Python Module Index** **31**
- Index** **33**



## EXTENSIONS PACKAGE

## 1.1 Submodules

### 1.1.1 extensions.fileext module

fileext - file handling extension.

This extension provides some simple text file handling. It limits the file locations to a pre-set root directory.

Make these functions available to a script by adding:

```
loadExtension_('fileext')
```

to it. Functions exported by this extension:

**readLines\_()**

read a text file line-by-line or all-at-once

**writeLines\_()**

write a line (or lines) to a text file, creating the file if need be. \*

**appendLines\_()**

write a line (or lines) to the end of a text file, extending it. \*

**listFiles\_()**

return a list of files in the given path \*

- may be disabled by option settings

#### Credits

- version: 1.0.0
- last update: 2023-Nov-13
- License: MIT
- Author: Mark Anacker <[closecrowd@pm.me](mailto:closecrowd@pm.me)>
- Copyright (c) 2023 by Mark Anacker

**class FileExt** (*api, options={}*)

Bases: object

This class manages commands to read and write files.

`__init__(api, options={})`

Constructs an instance of the FileExt class.

This instance will manage all named queues. There will be only one of these instances at a time.

#### Parameters

- **api** – an instance of ExtensionAPI connecting us to the engine
- **options** – a dict of option settings passed down to the extension

#### Options

**‘file\_root’** - a path prepended to all filenames, restricting access to files below this point.

**‘read\_only’** = If True, the **writeLines\_** and **appendLines\_** functions are not installed.

**‘list\_files’** - If True, allows the **list\_files\_** call

`appendLines_(filepath, data, handler=None, maxlines=0)`

Write lines of data to a text file, appending to any previous file

Append lines to a text file all-at-once, or line by line.

#### Args:

**filepath** [filename (and optional path) to read from.] Absolute (/) and parent (..) paths are not allowed. Path will be under “file\_root” if configured. Required.

**data** [A string to write to the file as a line of text,] or a list[] of strings that will be written as multiple lines. Required.

**handler** [The name of a script function that will be] called to obtain a line of text to be written to the file. If the handler returns False, file writing is stopped. Optional.

**maxlines** [The number of lines to write before stopping. If] omitted or 0, write until the data is exhausted. Optional.

**Returns:** None : If there was an error writing to the file

**count** [int The number of lines sent to the] handler, or the number of bytes sent if no handler.

`listFiles_(filepath=“”)`

return a list of files

`readLines_(filepath, handler=None, maxlines=0)`

Read data from a file in text mode.

Read a text file all-at-once, or line by line.

#### Args:

**filepath** [filename (and optional path) to read from.] Absolute (/) and parent (..) paths are not allowed. Path will be under “file\_root” if configured. Required.

**handler** [The name of a script function that will be] called with every line of text. The function will be called with a single str parameter containing the text line. If the handler returns False, file reading is stopped. Optional.

**maxlines** [The number of lines to read before stopping. If] omitted or 0, read until the end of the file. Optional.

**Returns:** None : If there was an error opening the file

**data** [str The lines from the file if no handler] was specified

count : int The number of lines sent to the handler

### **register()**

Make this extension's functions available to scripts

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

---

### **Note:**

### **Functions installed:**

- readLines\_() : read a text file line-by-line or all-at-once
  - writeLines\_() [write a line (or lines) to a text file,] creating the file if need be.
  - appendLines\_() [write a line (or lines) to the end of a text] file, extending it.
  - listFiles\_() : return a list of files in the given path
- 

### **Parameters None**

### **Returns**

**True** [Commands are installed and the extension is] ready to use.

**False** [Commands are NOT installed, and the extension] is inactive.

### **shutdown()**

Perform a graceful shutdown

### **unregister()**

Remove this extension's commands

### **writeLines\_ (filepath, data, handler=None, maxlines=0)**

Write lines of data to a text file, overwriting any previous file

Write to a text file all-at-once, or line by line.

### **Args:**

**filepath** [filename (and optional path) to read from.] Absolute (/) and parent (..) paths are not allowed. Path will be under "file\_root" if configured. Required.

**data** [A string to write to the file as a line of text,] or a list[] of strings that will be written as multiple lines. Required.

**handler** [The name of a script function that will be] called to obtain a line of text to be written to the file. If the handler returns False, file writing is stopped. Optional.

**maxlines** [The number of lines to write before stopping. If] omitted or 0, write until the data is exhausted. Optional.

**Returns:** None : If there was an error opening the file

**count** [int The number of lines sent to the] handler, or the number of bytes sent if no handler.

### **lineWriter (api, fp, data, handler=None, maxlines=0, mode='w')**

Write lines to a file in either overwrite or append mode

### **Parameters**

- **fp** – Filepath to write to
- **data** – Data to write if no handler
- **handler**
- **maxlines** – Max number of lines to write
- **mode** – “w” - overwrite. “a” - append

Returns:

**typeWrite** (*f*, *data*)

## 1.1.2 extensions.mqtttext module

mqtttext -mqtt client commands

This extension implements a client for the MQTT pub/sub protocol. It can handle multiple connections to MQTT brokers, publish messages to topics, and subscribe to topics. Incoming messages on subscribed topics may be delivered by polling, or by callbacks.

The link to a broker is represented by a connection name. Each connection is separate from the others, and each may have multiple topics subscribed to it.

Make the functions available to a script by adding:

```
loadExtension_('mqtttext')
```

to it. Functions exported by this extension:

**mqtt\_connect\_()**

Create a named connection to a broker

**mqtt\_disconnect\_()**

Close a named connection

**mqtt\_list\_()**

List all currently-active connections

**mqtt\_subscribe\_()**

Subscribe a connection to a topic

**mqtt\_unsubscribe\_()**

Remove a subscription from a connection

**mqtt\_isrunning\_()**

True is the connection is attached to a broker

**mqtt\_waiting\_()**

The number of messages waiting to be read

**mqtt\_readmsg\_()**

Return the first available message on the connection

**mqtt\_sendmsg\_()**

Send a message to a given topic

---

**Note:** Required Python modules:

paho.mqtt.client

---



## Credits

- version: 1.0.0
- last update: 2023-Nov-13
- License: MIT
- Author: Mark Anacker <[closecrowd@pm.me](mailto:closecrowd@pm.me)>
- Copyright (c) 2023 by Mark Anacker

**class MqttConnection** (*name, api*)

Bases: object

This class represents a connection to a broker. There can be several connections active simultaneously, linked to different brokers. Or all to the same broker, but that would be inefficient.

**connect\_** (*\*\*kwargs*)

**connectclient** ()

**disconnect\_** ()

**isrunning\_** ()

**on\_connect** (*client, userdata, flags, rc*)

**on\_disconnect** (*client, userdata, rc*)

**on\_message** (*client, userdata, message*)

**on\_topic\_message** (*client, userdata, message*)

**on\_wildcard\_message** (*client, userdata, message*)

**readmsg\_** (*blocking=True, timeout=1*)

**resuball** ()

**sendmsg\_** (*dest, data, qos=0, retain=False*)

**subtopic\_** (*topic, handler=None, qos=0*)

**unsubtopic\_** (*topic*)

**waiting\_** ()

**class MqttExt** (*api, options={}*)

Bases: object

This class manages commands to send/receive mqtt messages

**\_\_init\_\_** (*api, options={}*)

Constructs an instance of the MqttExt class.

This instance will manage all connections to mqtt brokers. There will be only once of these instances at a time.

### Parameters

- **api** – an instance of ExtensionAPI connecting us to the engine.
- **options** – a dict of option settings passed down to the extension.

**Returns** None

**\_\_api**

An instance of ExtensionAPI passed by the host, used to call back into the engine. Copied from api.

**\_\_options**

A dict of options from the host that may or may not apply to this extension. Copied from options.

**\_\_cmddict**

Dispatch table of our script command names and their functions.

**\_\_conns**

The table of active connections, indexed by name.

**\_\_locktimeout**

Timeout in seconds to wait for a mutex.

**\_\_lock**

Thread-locking mutex.

**connect\_** (*cname*, *\*\*kwargs*)

Handles the mqtt\_connect\_() function.

Create a connection object and establish a connection to a broker.

Args:

*cname* : The name of the connection

*\*\*kwargs* : A dict with all of the required parameters

Returns:

    The return value. True for success, False otherwise.

**disconnect\_** (*cname*)

Handles the mqtt\_disconnect\_() function.

Closes an open connection to a broker, and removes the connection from the table.

Args:

*cname*: The name of the connection to remove

Returns:

    The return value. True for success, False otherwise.

**isrunning\_** (*cname*)

**listConns\_** ()

Return a list[] of open connections.

**Args:** None

**Returns:** A list[] of active connections. The list may be empty if there are no connections.

    None if there was an error.

**readmsg\_** (*cname*, *blocking=True*, *timeout=1*)

**register** ()

Make this extension's functions available to scripts.

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

---

**Note:**

**Functions installed:**

- `mqtt_connect_()` : Create a named connection to a broker
- `mqtt_disconnect_()` : Close a named connection
- `mqtt_list_()` : List all currently-active connections
- `mqtt_subscribe_()` : Subscribe a connection to a topic
- `mqtt_unsubscribe_()` : Remove a subscription from a connection
- `mqtt_isrunning_()` : True is the connection is attached to a broker
- `mqtt_waiting_()` : The number of messages waiting to be read
- `mqtt_readmsg_()` : Return the first available message on the connection
- `mqtt_sendmsg_()` : Send a message to a given topic

**Args:** None

Returns:

True : Commands are installed and the extension is ready to use.

False : Commands are NOT installed, and the extension is inactive.

**sendmsg\_** (*cname, dest, data, qos=0, retain=False*)

**shutdown** ()

Perform a graceful shutdown.

Close all of the active MQTT connections. This gets called by the extension manager just before the extension is unloaded.

**subtopic\_** (*cname, topic, handler=None, qos=0*)

**unregister** ()

Remove this extension's functions from the engine.

**unsubtopic\_** (*cname, topic*)

**waiting\_** (*cname*)

**debug** (\*args)

**topicmatch** (*subsc, topic*)

### 1.1.3 extensions.queueext module

queueext - Named multi-thread queues.

This extension provides thread-safe queues for communicating between threads (and callback handlers). It has all the queue management functions anyone should need.

Queues are referenced by name (set in the `queue_open_()` function), and may be either a classic First-In-First-Out queue, or a Last-In-First-Out stack. These queues are an excellent way to pass data from an event-driven handler function to a main processing loop.

Make the functions available to a script by adding:

```
loadExtension_('queueext')
```

to it. Functions exported by this extension:

**queue\_open\_** ()

Create a named queue

**queue\_close\_()**

Flush and destroy an existing queue

**queue\_put\_()**

Put data into a queue

**queue\_get\_()**

Get the next item from a queue

**queue\_clear\_()**

Clear all items from a queue

**queue\_len\_()**

Return the number of items in a queue

**queue\_isempty\_()**

Return True if the queue is empty

**queue\_list\_()**

Return a list[] of the current queue names

### Credits

- version: 1.0.0
- last update: 2023-Nov-13
- License: MIT
- Author: Mark Anacker <[closecrowd@pm.me](mailto:closecrowd@pm.me)>
- Copyright (c) 2023 by Mark Anacker

**class QueueExt** (*api, options={}*)

Bases: object

This class provides thread-safe queues.

**\_\_init\_\_** (*api, options={}*)

Constructs an instance of the QueueExt class.

This instance will manage all named queues. There will be only one of these instances at a time.

#### Parameters

- **api** – an instance of ExtensionAPI connecting us to the engine
- **options** – a dict of option settings passed down to the extension

**queue\_clear\_** (*name*)

Handles the queue\_clear\_() function.

Removes all items in a queue.

**Args:** name : The name of the queue to get from.

**Returns:** The return value. True for success, False otherwise.

**queue\_close\_** (*name*)

Handles the queue\_close\_() function.

Close an open queue and clear any data currently in it.

**Args:** name : The name of the queue to close.

**Returns:** The return value. True for success, False otherwise.

**queue\_get\_**(*name*, *\*\*kwargs*)

Handles the queue\_get\_() function.

Gets an item from the queue. A 'fifo' queue returns the item at the HEAD of the queue. A 'lifo' type will return the item at the END of the queue.

The get operation may be either blocking or non-blocking. If the block option in kwargs is False, the operation will return immediately. If the queue was empty, the return will be None.

If block is True (default) and the queue is empty, the get operation will be retried at 1-second intervals until either: an item is added to the queue, the timeout count is reached, or the queue is closed. A timeout of 0 will retry until either success or close.

**Args:** name : The name of the queue to get from.

**\*\*kwargs :** Options to pass down to the Queue.

**Options:** block : If True, block until timeout if the queue is empty.

timeout : Seconds to wait when blocking. 0 means wait forever.

**Returns:** The next value from the queue, or None.

**queue\_isempty\_**(*name*)

Handles the queue\_isempty\_() function.

Returns True if the queue is empty.

**Args:** name : The name of the queue to check.

**Returns:** True if the named queue is empty, or there was an error.

**queue\_len\_**(*name*)

Handles the queue\_len\_() function.

Returns the number of entries in a queue.

**Args:** name : The name of the queue to check.

**Returns:** The number of items in the named queue. Any error returns 0.

**queue\_list\_**()

Handles the queue\_list\_() function.

Returns a list with the names of all open queues.

**Args:** None

**Returns:** A list[] of open queues. The list may be empty if there are no queues open.

None if there was an error.

**queue\_open\_**(*name*, *\*\*kwargs*)

Handles the queue\_open\_() function.

Creates a new queue and add it's name to the table.

**Args:** cname : The queue name to use. Must not be in use.

**\*\*kwargs :** Options to pass down to the Queue.

**Returns:** True if the queue was created.

False if an error occurred.

**Options:** type : 'fifo' or 'lifo' - queue or stack

**queue\_put\_** (*name, value, \*\*kwargs*)

Handles the queue\_put\_() function.

Adds an item to the queue. Items are always added to the back of the queue.

The put operation may be either blocking or non-blocking. If the block option in kwargs is False, the item is simply sent to the queue. If the queue is full, the item will be silently discarded. If block is True (default), the put operation will be retried at 1-second intervals until either: the item is added to the queue, the timeout count is reached, or the queue is closed. A timeout of 0 will retry until either success or close.

**Args:** name : The name of the queue to add to.

value : The data item to add to the queue

**\*\*kwargs** : Options to pass down to the Queue.

**Options:** block : If True, block until timeout if the queue is full.

timeout : Seconds to wait when blocking. 0 means wait forever.

**Returns:** The return value. True for success, False otherwise.

**register** ()

Make this extension's functions available to scripts

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

---

**Note:**

**Functions installed:**

- queue\_open\_() : Create a named queue
- queue\_close\_() : Flush and destroy an existing queue
- queue\_put\_() : Put data on a queue
- queue\_get\_() : Get the next item from a queue
- queue\_clear\_() : Clear all items from a queue
- queue\_len\_() : Return the number of items in a queue
- queue\_isempty\_() : Return True if the queue is empty
- queue\_list\_() : Return a list[] of the current queue names

Args:

None

Returns:

True : Commands are installed and the extension is ready to use.

False : Commands are NOT installed, and the extension is inactive.

---

**shutdown** ()

Perform a graceful shutdown.

This gets called by the extension manager just before the extension is unloaded.

```
unregister ()
```

Remove this extension's functions from the engine.

```
class ThreadQueue_ (name, api, qtype)
```

Bases: `object`

This class implements the actual thread-safe queue.

There is one of these for every named queue in the table.

```
tqueue_clear_ ()
```

```
tqueue_close_ ()
```

```
tqueue_get_ (**kwargs)
```

```
tqueue_isempty_ ()
```

```
tqueue_len_ ()
```

```
tqueue_put_ (value, **kwargs)
```

```
clearq__ (q)
```

## 1.1.4 extensions.redisext module

redisext - basic interface to the Redis cache.

This extension provides a client connection to a redis in-memory cache. Not all redis functions are available (there are a *lot* of them), but a lot of the useful ones are included.

Familiarity with redis is helpful. See: <https://redis.io/commands/>

Make the functions available to a script by adding:

```
loadExtension_('redisext')
```

to it. Functions exported by this extension:

```
redis_connect_ ()
```

Connect to a redis server

```
redis_disconnect_ ()
```

Disconnect from the server

```
redis_list_ ()
```

Return a list[] of active connections

```
redis_cmd_ ()
```

Pass a command string to redis \*

these map directly to redis\_cli commands:

```
redis_set_ ()
```

Set a value in the cache

```
redis_get_ ()
```

Get a value from the cache

```
redis_del_ ()
```

Remove a keyed entry from the cache

```
redis_incr_ ()
```

Increments the number stored at key by one

**redis\_decr\_()**

Decrements the number stored at key by one

**redis\_hset\_()**

Sets the value associated in a hash stored at key.

**redis\_hmset\_()**

Sets multiple values in a hash

**redis\_hget\_()**

Returns the value associated in a hash stored at key.

**redis\_hdel\_()**

Removes the specified fields from the hash stored at key

**redis\_hkeys\_()**

Returns all field names in the hash stored at key.

**redis\_hvals\_()**

Returns all values in the hash stored at key.

- may be disabled by option settings

### Options

**allow\_redis\_cmds** If True, installs the “**redis\_cmd\_**” function to pass complete redis command strings to the server. This is a potential security risk.

---

**Note:** Required Python modules:

redis

---

### Credits

- version: 1.0.0
- last update: 2023-Nov-20
- License: MIT
- Author: Mark Anacker <[closecrowd@pm.me](mailto:closecrowd@pm.me)>
- Copyright (c) 2023 by Mark Anacker

**class RedisConnection** (*name, api*)

Bases: object

This class represents a connection to a Redis server. There can be several connections active simultaneously, open to different servers (or possibly the same server).

**cmds\_** (\*args)

Send a command directly to the server.

**connect\_** (*host='127.0.0.1', port=6379, \*\*kwargs*)

Connect to the Redis server.

**disconnect\_** ()

Disconnect from the Redis server.



**hmset\_** (\*args)

Load a whole dict into a redis hashmap.

**class RedisExt** (api, options={})

Bases: object

This class manages connections to a Redis cache server.

**\_\_init\_\_** (api, options={})

Constructs an instance of the RedisExt class.

This instance will manage all connections to one or more redis servers. There will be only once of these instances at a time.

Args:

api : an instance of ExtensionAPI connecting us to the engine.

options : a dict of option settings passed down to the extension.

Returns:

None

**Attributes:**

**\_\_api** [An instance of ExtensionAPI passed by the host, used] to call back into the engine.  
Copied from api.

**\_\_options** [A dict of options from the host that may or may not] apply to this extension.  
Copied from options.

**\_\_cmddict** [Dispatch table of our script command names and their] functions.

**\_\_conns** : The table of active connections, indexed by name.

**\_\_cmdsflag** [If True, install the "redis\_cmd\_()" function. Set] in the options dict passed in.

**\_\_locktimeout** : Timeout in seconds to wait for a mutex.

**\_\_lock** : Thread-locking mutex.

**cmd\_** (cname, \*args)

Handles the redis\_cmd\_() function.

Sends a raw command directly to the redis server. This is potentially dangerous, and is only available if specifically enabled by the host application via the options.

Args:

cname: The name of the connection to use.

\*args: The arguments to pass to the execute\_command function.

Returns:

The results of the command.

None if there was an error

**connect\_** (cname, host='127.0.0.1', port=6379, \*\*kwargs)

Handles the redis\_connect\_() function.

This function establishes a named connection to a redis server. Successful completion is required before any other functions may be used (except **redis\_list\_**).

Args:

cname: The name of the connection  
host: The hostname or ip address of the server  
port: The port to connect to the server on  
**\*\*kwargs**: Optional arguments

Returns:

The return value. True for success, False otherwise.

**decr\_** (*cname*, *key*, *val=1*)

Handles the redis\_decr\_() function.

**del\_** (*cname*, *key*)

Handles the redis\_del\_() function.

**disconnect\_** (*cname*)

Handles the redis\_disconnect\_() function.

Closes an open connection to a redis server and removes the connection from the table.

Args:

cname: The name of the connection to remove

Returns:

The return value. True for success, False otherwise.

**get\_** (*cname*, *key*)

Handles the redis\_get\_() function.

**hdel\_** (*cname*, *hashname*, *key*)

Handles the redis\_hdel\_() function.

**hget\_** (*cname*, *hashname*, *key*)

Handles the redis\_hget\_() function.

**hkeys\_** (*cname*, *hashname*)

Handles the redis\_hkeys\_() function.

**hmsset\_** (*cname*, *hashname*, *map*)

Handles the redis\_hmsset\_() function.

**hset\_** (*cname*, *hashname*, *key*, *value=""*)

Handles the redis\_hset\_() function.

**hvals\_** (*cname*, *hashname*)

Handles the redis\_hvals\_() function.

**incr\_** (*cname*, *key*, *val=1*)

Handles the redis\_incr\_() function.

**list\_** ()

Handles the redis\_list\_() function.

This method returns a list[] of active Redis connections.

Args:

None

Returns:

A list[] of active connections. The list may be empty if there are no connections.

None if there was an error.

**register()**

Make this extension's functions available to scripts.

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

---

**Note:****Functions installed:**

- redis\_connect\_() : Connect to a redis server
- redis\_disconnect\_() : Disconnect from the server
- redis\_list\_() : Return a list[] of active connections
- redis\_cmd\_() : Pass a command string to redis [OPT]  
these map directly to redis\_cli commands:
- redis\_set\_() : Set a value in the cache
- redis\_get\_() : Get a value from the cache
- redis\_del\_() : Remove a keyed entry from the cache
- redis\_incr\_() : Increments the number stored at key by one
- redis\_decr\_() : Decrements the number stored at key by one
- redis\_hset\_() : Sets the value associated in a hash stored at key.
- redis\_hmset\_() : Sets multiple values in a hash
- redis\_hget\_() : Returns the value associated in a hash stored at key.
- redis\_hdel\_() : Removes the specified fields from the hash stored at key
- redis\_hkeys\_() : Returns all field names in the hash stored at key.
- redis\_hvals\_() : Returns all values in the hash stored at key.

[OPT] may be disabled by option settings

Args:

None

Returns:

True : Commands are installed and the extension is ready to use.

False : Commands are NOT installed, and the extension is inactive.

---

**set\_(cname, key, value)**

Handles the redis\_set\_() function.

**shutdown()**

Perform a graceful shutdown.

Close all of the active Redis connections. This gets called by the extension manager just before the extension is unloaded.

**unregister()**

Remove this extension's functions from the engine.

### 1.1.5 extensions.sqliteext module

sqliteext - SQLite3 database functions.

This extension allows a script to store and retrieve data from a sqlite3 database. The database files are stored under a directory passed in via the "sql\_root" option.

Some basic knowledge of SQLite3 and SQL is helpful when using this extension.

All functions are mutex-protected, and can be used from multiple threads.

Make the functions available to a script by adding:

```
loadExtension_('sqliteext')
```

to it. Functions exported by this extension:

**sql\_open()**

Open a sqlite3 database, creating it if it doesn't exist.

**sql\_close()**

Close an open connection.

**sql\_execute()**

Execute a SQL command in the open database.

**sql\_commit()**

Commit any pending transactions (in autocommit == False)

**sql\_rollback()**

Roll back any pending transactions

**sql\_changes()**

Return the number of rows recently altered

**sql\_cursor\_fetch()**

Return rows from a cursor after an execute\_()

**sql\_list()**

Return a list[] of the active connection names

---

**Note:** Required Python modules:

sqlite3

---

## Credits

- version: 1.0
- last update: 2023-Nov-20
- License: MIT
- Author: Mark Anacker <[closecrowd@pm.me](mailto:closecrowd@pm.me)>
- Copyright (c) 2023 by Mark Anacker

**class SQLiteExt** (*api, options={}*)

Bases: object

This class manages commands to use a sqlite3 database.

**\_\_init\_\_** (*api, options={}*)

Constructs an instance of the SQLiteExt class.

This instance will manage all connections to sqlite3 databases. There will be only one of these instances at a time.

Args:

api : an instance of ExtensionAPI connecting us to the engine.

options : a dict of option settings passed down to the extension.

Returns:

Nothing.

**\_\_api**

An instance of ExtensionAPI passed by the host, used to call back into the engine. Copied from api.

**\_\_options**

A dict of options from the host that may or may not apply to this extension. Copied from options.

**\_\_cmddict**

Dispatch table of our script command names and their functions.

**\_\_conns**

The table of active connections, indexed by name.

**\_\_cmdsflag**

If True, install the “redis\_cmd\_()” function. Set in the options dict passed in.

**sqlroot**

The path to prepend to database file names.

**sqlext**

The file extension to append to database file names.

**\_\_locktimeout**

Timeout in seconds to wait for a mutex.

**\_\_lock**

Thread-locking mutex.

## Options

**'sql\_root'** [a path prepended to all database names,] restricting access to db files below this point.

**'sql\_ext'** [filename extension to use for database files.] Defaults to **'db'**

### **register()**

Make this extension's functions available to scripts.

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

---

### **Note:**

#### **Functions installed:**

- **sql\_open()** : Open a sqlite3 database, creating it if it doesn't exist.
- **sql\_close()** : Close an open connection.
- **sql\_execute()** : Execute a SQL command in the open database.
- **sql\_commit()** : Commit any pending transactions (if autocommit is False)
- **sql\_rollback()** : Roll back any pending transactions
- **sql\_changes()** : Return the number of rows recently altered
- **sql\_cursor\_fetch()** : Return rows from a cursor after an **execute()**
- **sql\_list()** : Return a list[] of the active connection names

---

### **Parameters None**

### **Returns**

**True** [Commands are installed and the extension is] ready to use.

**False** [Commands are NOT installed, and the extension] is inactive.

### **shutdown()**

Perform a graceful shutdown.

Close all of the active database connections. This gets called by the extension manager just before the extension is unloaded.

### **sql\_changes(cname)**

Handles the **sql\_cursor\_fetch()** function.

Returns the number of changes to the database since it was last opened.

**Args:** **cname** : The name of the connection to use.

**Returns:** The number of updates in this session.

-1 if there was an error.

### **sql\_close(cname)**

Handles the **sql\_close()** function.

Close an open db connection and remove the connection from the table.

**Args:** *cname* : The name of the connection to remove.

**Returns:** The return value. True for success, False otherwise.

**sql\_commit\_**(*cname*)

Handles the sql\_commit\_() function.

Commit pending updates to the database. If autocommit is True, this function has no effect.

**Args:** *cname* : The name of the connection to commit.

**Returns:** True for success, False otherwise.

**sql\_cursor\_fetch\_**(*cname, cursor, count=0*)

Handles the sql\_cursor\_fetch\_() function.

Return one or many rows from the supplied cursor (returned by sql\_execute\_()).

**Args:** *cname* : The name of the connection to use.

*cursor* : The cursor reference returned by sql\_execute\_().

*count* : The number of rows to return.

- -1 = get the next row as a tuple.
- 0 = get all the rows as a list of tuples.
- >0 = gets all rows <n> at a time as a list of tuples.

**Returns:** The requested number of rows, None if an error occurred.

**sql\_execute\_**(*cname, sql, \*args, \*\*kwargs*)

Handles the sql\_execute\_() function.

Execute a SQL statement in an open db.

**Args:** *cname* : The connection name to use.

*sql* : The SQL statement to execute.

*\*args* : Values to substitute in the above SQL (if any).

*\*\*kwargs* : Options to pass down to sqlite3.

**Returns:** A cursor reference if successful.

None if there was an error.

**sql\_list\_**()

Handles the sql\_list\_() function.

Returns a list with the names of all current connections.

**Args:** None

**Returns:** A list[] of active connections. The list may be empty if there are no connections.

None if there was an error.

**sql\_open\_**(*cname, dbname, \*\*kwargs*)

Handles the sql\_open\_() function.

Open a connection to a sqlite3 database file, creating the files if it doesn't exist. The connection name and the database filename do not have to be the same.

A check is made to make sure the database filename isn't already in use by another connection. It's probably not a good idea to have the same file accessed by multiple connections simultaneously. That's not the same as calling this extension from multiple threads (which is supported).

**Args:** `cname` : The connection name to use. Must not be in use.

`dbname` : The Sqlite3 database file to use.

**\*\*kwargs** : Options to pass down to sqlite3.

**Returns:** True if the database was opened.

False if an error occurred.

**Options:** `autocommit` : If True, write the changes to the file after every change. Default=True.

`check_same_thread` : If True, restrict callers to a single thread. Default=False.

**sql\_rollback\_** (*cname*)

Handles the `sql_rollback_()` function.

Rollback pending changes.

If `autocommit` is False, updates to the database are held in a buffer until `sql_commit_()` is called. This function clears that buffer, effectively reverting changes back to the last `sql_commit_()`. If `autocommit` is True, this function has no effect.

**Args:** `cname` : The name of the connection to use.

**Returns:** The return value. True for success, False otherwise.

**unregister** ()

Remove this extension's functions from the engine.

**class SqlLiteConnection** (*name, dbname, api*)

Bases: `object`

This class represents a connection to a sqlite3 database. There can be several connections active simultaneously, open to different database files.

**\_\_init\_\_** (*name, dbname, api*)

Create a `SqlLiteConnection` object.

Sets up the info for a single connection to a db file.

**Args:** `name` : The name of this connection.

`dbname` : The base name of the database file.

`api` : A reference back to the engine API. Used for error messages.

**Returns:** None

**sql\_changes\_** ()

Return the number of recent updates.

Returns the number of changes to the database since it was last opened.

**Args:** None

**Returns:** The number of updates in this session. -1 if there was an error.

**sql\_close\_** ()

Close the connection to database and clean-up.

**sql\_commit\_** ()

Handles the `sql_commit_()` function.

Commit pending updates to the database. If `autocommit` is True, this function has no effect.

**Args:** None



**Returns:** The return value. True for success, False otherwise.

**sql\_cursor\_fetch\_** (*cursor*, *count=0*)

Get rows from a cursor (returned by sql\_execute\_()).

Return one or many rows from the supplied cursor.

**Args:** *cursor* : The cursor reference returned by sql\_execute\_(). *count* : The number of rows to return.

- -1 = get the next row as a tuple.
- 0 = get all the rows as a list of tuples.
- >0 = gets all rows <n> at a time as a list of tuples.

**Returns:** The requested number of rows. None if an error occurred.

**sql\_execute\_** (*sql*, *\*args*, *\*\*kwargs*)

Execute a sql statement.

Submit a SQL statement to the database.

**Args:** *sql* : The SQL statement to execute.

*\*args* : Values to substitute in the above SQL (if any).

*\*\*kwargs* : Options to pass down to sqlite3.

**Returns:** A cursor reference if successful.

None if there was an error.

**Options:** *args* - values to replace ? in the sql statement.

**autocommit=True - Commits changes to the database after** the execute\_() call. Default=False.

**sql\_get\_dbname\_** ()

Return this connection's db file name .

**sql\_open\_** (*dbpath*, *\*\*kwargs*)

Open a database for use.

This method tries to connect to a Sqlite3 database file.

**Args:** *dbpath* : The full pathname of the Sqlite3 db file.

*\*\*kwargs* : Options to pass down to sqlite3.

**Returns:** True if the database was opened.

False if there was an error.

**Options:**

- *autocommit=True* : Commits changes to the database after sql\_execute\_() call. Default=False
- *check\_same\_thread=False* : Allow updates from multiple threads.

**sql\_rollback\_** ()

Rollback pending changes.

If autocommit is False, updates to the database are held in a buffer until sql\_commit\_() is called. This function clears that buffer, effectively reverting changes back to the last sql\_commit\_(). If autocommit is True, this function has no effect.

**Args:** None

**Returns:** The return value. True for success, False otherwise.

## 1.1.6 extensions.tdicttext module

tdicttext - Thread-safe Dict

This extension provides thread-safe, named dictionaries (key,value). Python's built-in dict objects are *mostly* atomic (and therefore thread-safe), but not *always*. And not every operation is guaranteed to be safe. Scripts in apyshell may make extensive use of threads, so an assured safe dict is needed.

Each named tdict has it's own lock, so they are relatively independant once created.

See also the companion tlist extension.

Make these functions available to a script by adding:

```
loadExtension_('tdicttext')
```

to it. Functions exported by this extension:

**tdict\_open\_()**

create a named dict

**tdict\_close\_()**

delete an existing dict

**tdict\_put\_()**

add an item to the named dict

**tdict\_update\_()**

add a dict to the named dict

**tdict\_get\_()**

get an item by key

**tdict\_pop\_()**

get an item by key, then remove it

**tdict\_del\_()**

remove an item from the dict by key

**tdict\_clear\_()**

remove all items from the dict

**tdict\_keys\_()**

return a list[] of keys

**tdict\_items\_()**

return a list[] of items

**tdict\_len\_()**

return the number of items in the dict

**tdict\_copy\_()**

return a shallow copy of the dict

**tdict\_list\_()**

return a list[] of tdicts

## Credits

- version: 1.0.0
- last update: 2023-Dec-05
- License: MIT
- Author: Mark Anacker <[closecrowd@pm.me](mailto:closecrowd@pm.me)>
- Copyright (c) 2023 by Mark Anacker

**class** **TDictExt** (*api*, *options*={})

Bases: `object`

This class provides a thread-safe key-value store.

**\_\_init\_\_** (*api*, *options*={})

Constructs an instance of the TDictExt class.

This instance will manage all thread-safe dictionaries. There will be only one of these instances at a time.

Args:

*api* : an instance of ExtensionAPI connecting us to the engine.

*options* : a dict of option settings passed down to the extension.

Returns:

    Nothing.

**register** ()

Make this extension's functions available to scripts.

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

---

**Note:** Functions installed:

- `tdict_open_()` : create a named dict
  - `tdict_close_()` : delete an existing dict
  - `tdict_put_()` : add an item to the named dict
  - `tdict_update_()` : add a dict to the named dict
  - `tdict_get_()` : get an item by key
  - `tdict_pop_()` : get an item by key, then remove it
  - `tdict_del_()` : remove an item from the dict by key
  - `tdict_clear_()` : remove all items from the dict
  - `tdict_keys_()` : return a list[] of keys
  - `tdict_items_()` : return a list[] of items
  - `tdict_len_()` : return the number of items in the dict
  - `tdict_copy_()` : return a shallow copy of the dict
  - `tdict_list_()` : return a list[] of tdicts
-

**Parameters** None

**Returns**

**True** [Commands are installed and the extension is] ready to use.

**False** [Commands are NOT installed, and the extension] is inactive.

**shutdown** ()

Perform a graceful shutdown.

Clear then close all of the active dictionaries. This gets called by the extension manager just before the extension is unloaded.

**tdict\_clear\_** (*cname*)

Handles the tdict\_clear\_() function.

Removes all items from the named dict.

**Args:** *cname* : The name of the dict to use.

**Returns:** True for success, False otherwise.

**tdict\_close\_** (*cname*)

Handles the tdict\_close\_() function.

Clears out the data from an existing tdict, then removes it.

**Args:** *cname* : The name of the dict to remove.

**Returns:** True for success, False otherwise.

**tdict\_cmd** (*cmd, cname, key=None, value=None*)

Internal command dispatcher.

Performs the operation specified by the “cmd” argument on the dict object associated with “cname”, and returns the result.

**Args:** *cmd* : The operation to perform

*cname* : The name of the dict to use.

*key* : A key string (if needed)

*value* : A value argument (if needed)

**Returns:** Depends on the operation. See the functions below for the specific returns.

**tdict\_copy\_** (*cname*)

**tdict\_del\_** (*cname, key*)

Handles the tdict\_del\_() function.

Removes an item indexed by the key from the dict.

**Args:** *cname* : The name of the dict to use.

*key* : A key string to remove.

**Returns:** True for success, False otherwise.

**tdict\_get\_** (*cname, key, value=None*)

Handles the tdict\_get\_() function.

Retrieves an item indexed by the key from the dict, or returns a default value if the key isn't found.

**Args:** *cname* : The name of the dict to use.  
           *key* : A key string to look for.  
           *value* : A default to return if key isn't in the dict.

**Returns:** The retrieved item, or the supplied value.

**tdict\_items\_** (*cname*)

**tdict\_keys\_** (*cname*)

Handles the `tdict_keys_()` function.

Returns a list with the keys in the named dict.

**Args:** None

**Returns:** A list[] of keys in the dict. The list may be empty if there are none.

None if there was an error.

**tdict\_len\_** (*cname*)

**tdict\_list\_** ()

Handles the `tdict_list_()` function.

Returns a list with the names of all current dictionaries.

**Args:** None

**Returns:** A list[] of active dictionaries. The list may be empty if there are none.

None if there was an error.

**tdict\_open\_** (*cname*)

Handles the `tdict_open_()` function.

Creates a new `tdict` object attached to the given name.

**Args:** *cname* : The name of the dict to create.

**Returns:** True for success, False otherwise.

**tdict\_pop\_** (*cname*, *key*, *value*)

Handles the `tdict_pop_()` function.

Retrieves an item indexed by the key from the dict, or returns a default value if the key isn't found.

Removes the item from the dict if the key was found.

**Args:** *cname* : The name of the dict to use.

*key* : A key string to look for.

*value* : A default to return if key isn't in the dict.

**Returns:** The retrieved item, or the supplied value.

**tdict\_put\_** (*cname*, *key*, *value*)

Handles the `tdict_put_()` function.

Adds a new value indexed by the key to the dict, or replaces an existing value with the new one.

**Args:** *cname* : The name of the dict to use.

*key* : A key string.

*value* : A value argument to add or replace.

**Returns:** True for success, False otherwise.

**tdict\_update\_**(*cname, value*)

Handles the tdict\_update\_() function.

Merges the entire contents of an existing dict object into the named tdict.

**Args:** *cname* : The name of the dict to use.

*value* : The dict object to merge in.

**Returns:** True for success, False otherwise.

**unregister**()

Remove this extension's functions from the engine.

**class ThreadDict\_**(*name, api, timeout=20*)

Bases: `object`

**close\_**()

**cmd\_**(*cmd, key=None, value=None*)

### 1.1.7 extensions.utilext module

utilext - utility functions.

This extension adds a few useful functions callable by scripts. Some of these functions may be disabled by options passed in from apyshell.

Make these functions available to a script by adding:

`loadExtension_('utilext')`

to it. Functions exported by this extension:

**input\_**()

read characters from the terminal, return as a string.

**system\_**()

execute a string in the user's default shell. \*

**getenv\_**()

return the value of an environment variable. \*

- may be disabled by option settings

### Credits

- version: 1.0.0
- last update: 2023-Nov-17
- License: MIT
- Author: Mark Anacker <[closecrowd@pm.me](mailto:closecrowd@pm.me)>
- Copyright (c) 2023 by Mark Anacker

**class UtilExt**(*api, options={}*)

Bases: `object`

This class provides utility commands.

---

```
__init__ (api, options={})
```

Constructs an instance of the UtilExt class.

This instance supplies some useful functions.

#### Parameters

- **api** – an instance of ExtensionAPI connecting us to the engine.
- **options** – a dict of option settings passed down to the extension.

**Returns** None

### Options

**‘allow\_system’** [install the `system_()` command allowing] the script to run commands in the system shell.

**‘allow\_getenv’** [install the `getenv_()` command to allow] the script to read environment variables.

```
getenv_ (name)
```

Return an environment variable.

This function returns the value of an item in the host application’s environment. Since this might be unsafe, it has to be specifically enabled by the host application.

Args:

name : The environment variable to return

Returns:

The contents of the env variable, or None if it isn’t valid.

```
input_ (prompt=None, default=None, **kwargs)
```

Get console input.

This function will (optionally) print a prompt on the console, then wait for (and return) whatever the user types in. A default value may be set to be returned if nothing was entered. A timeout can also be set, either returning a default value or raising an exception if the timeout expires without a console entry.

The timeout default value may be different from the input default.

Args:

prompt : The prompt to display on the console

default : A value to return if there is no input

**\*\*kwargs** : Various options:

#### Options:

- timeout : input timeout in seconds.
- todef : default value to return if the timeout expires.
- toraise : If True, raise an Exception when the timeout expires.

**Returns:** A string with the input (minus the trailing newline), or a sepcified default value.

**Raises:** Exception(‘Timed out’)

**register()**

Make this extension's functions available to scripts.

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

---

**Note:**

**Functions installed:**

- `input_()` : read characters from the terminal, return as a string.
  - `system_()` : execute a string in the user's default shell.
  - `getenv_()` : return the value of an environment variable.
- 

**Parameters None**

**Returns** True : Commands are installed and the extension is ready to use.

False : Commands are NOT installed, and the extension is inactive.

**shutdown()**

Perform a graceful shutdown.

This gets called by the extension manager just before the extension is unloaded.

**system\_()**

Run a shell command - possibly dangerous. (definitely dangerous!).

This function simply runs whatever string is passed to it in the host environment. Useful for debugging or ad-hoc scripting, it has to be deliberately enabled by the host application.

Args:

cmd : A command to run.

Returns:

The output of the command.

**unregister()**

Remove this extension's functions from the engine.

## 1.2 Module contents

Extensions - Add additional functionality to scripts.

This package contains modules that add special functions to apyshell. Scripts can use the `loadExtension_()` command to add any of these modules that are in the `extensionsdir` directory, and make their commands available to the scripts. Extensions may be unloaded as well.<sup>33</sup>



## Credits

- version: 1.0.0
- last update: 2023-Nov-17
- License: MIT
- Author: Mark Anacker <[closecrowd@pm.me](mailto:closecrowd@pm.me)>
- Copyright (c) 2023 by Mark Anacker



## PYTHON MODULE INDEX

### e

- `extensions`, [28](#)
- `extensions.fileext`, [1](#)
- `extensions.mqtttext`, [4](#)
- `extensions.queueext`, [7](#)
- `extensions.redisext`, [11](#)
- `extensions.sqliteext`, [16](#)
- `extensions.tdicttext`, [22](#)
- `extensions.utilext`, [26](#)



## Symbols

[\\_\\_api \(MqttExt attribute\), 5](#)  
[\\_\\_api \(SqliteExt attribute\), 17](#)  
[\\_\\_cmddict \(MqttExt attribute\), 6](#)  
[\\_\\_cmddict \(SqliteExt attribute\), 17](#)  
[\\_\\_cmdsflag \(SqliteExt attribute\), 17](#)  
[\\_\\_conns \(MqttExt attribute\), 6](#)  
[\\_\\_conns \(SqliteExt attribute\), 17](#)  
[\\_\\_init\\_\\_ \(\) \(FileExt method\), 1](#)  
[\\_\\_init\\_\\_ \(\) \(MqttExt method\), 5](#)  
[\\_\\_init\\_\\_ \(\) \(QueueExt method\), 8](#)  
[\\_\\_init\\_\\_ \(\) \(RedisExt method\), 13](#)  
[\\_\\_init\\_\\_ \(\) \(SqliteExt method\), 17](#)  
[\\_\\_init\\_\\_ \(\) \(SqliteConnection method\), 20](#)  
[\\_\\_init\\_\\_ \(\) \(TDictExt method\), 23](#)  
[\\_\\_init\\_\\_ \(\) \(UtilExt method\), 26](#)  
[\\_\\_lock \(MqttExt attribute\), 6](#)  
[\\_\\_lock \(SqliteExt attribute\), 17](#)  
[\\_\\_locktimeout \(MqttExt attribute\), 6](#)  
[\\_\\_locktimeout \(SqliteExt attribute\), 17](#)  
[\\_\\_options \(MqttExt attribute\), 5](#)  
[\\_\\_options \(SqliteExt attribute\), 17](#)

## A

[appendLines\\_ \(\) \(FileExt method\), 2](#)  
[appendLines\\_ \(\) \(in module extensions.fileext\), 1](#)

## C

[clearq\\_ \(\) \(in module extensions.queueext\), 11](#)  
[close\\_ \(\) \(ThreadDict\\_ method\), 26](#)  
[cmd\\_ \(\) \(RedisExt method\), 13](#)  
[cmd\\_ \(\) \(ThreadDict\\_ method\), 26](#)  
[cmds\\_ \(\) \(RedisConnection method\), 12](#)  
[connect\\_ \(\) \(MqttConnection method\), 5](#)  
[connect\\_ \(\) \(MqttExt method\), 6](#)  
[connect\\_ \(\) \(RedisConnection method\), 12](#)  
[connect\\_ \(\) \(RedisExt method\), 13](#)  
[connectclient \(\) \(MqttConnection method\), 5](#)

## D

[debug \(\) \(in module extensions.mqttext\), 7](#)  
[decr\\_ \(\) \(RedisExt method\), 14](#)

[del\\_ \(\) \(RedisExt method\), 14](#)  
[disconnect\\_ \(\) \(MqttConnection method\), 5](#)  
[disconnect\\_ \(\) \(MqttExt method\), 6](#)  
[disconnect\\_ \(\) \(RedisConnection method\), 12](#)  
[disconnect\\_ \(\) \(RedisExt method\), 14](#)

## E

[extensions](#)  
     [module, 28](#)  
[extensions.fileext](#)  
     [module, 1](#)  
[extensions.mqttext](#)  
     [module, 4](#)  
[extensions.queueext](#)  
     [module, 7](#)  
[extensions.redisext](#)  
     [module, 11](#)  
[extensions.sqliteext](#)  
     [module, 16](#)  
[extensions.tdictext](#)  
     [module, 22](#)  
[extensions.utilext](#)  
     [module, 26](#)

## F

[FileExt \(class in extensions.fileext\), 1](#)

## G

[get\\_ \(\) \(RedisExt method\), 14](#)  
[getenv\\_ \(\) \(in module extensions.utilext\), 26](#)  
[getenv\\_ \(\) \(UtilExt method\), 27](#)

## H

[hdel\\_ \(\) \(RedisExt method\), 14](#)  
[hget\\_ \(\) \(RedisExt method\), 14](#)  
[hkeys\\_ \(\) \(RedisExt method\), 14](#)  
[hmset\\_ \(\) \(RedisConnection method\), 12](#)  
[hmset\\_ \(\) \(RedisExt method\), 14](#)  
[hset\\_ \(\) \(RedisExt method\), 14](#)  
[hvals\\_ \(\) \(RedisExt method\), 14](#)

## I

incr\_() (*RedisExt* method), 14  
input\_() (in module *extensions.utilext*), 26  
input\_() (*UtilExt* method), 27  
isrunning\_() (*MqttConnection* method), 5  
isrunning\_() (*MqttExt* method), 6

## L

lineWriter() (in module *extensions.fileext*), 3  
list\_() (*RedisExt* method), 14  
listConns\_() (*MqttExt* method), 6  
listFiles\_() (*FileExt* method), 2  
listFiles\_() (in module *extensions.fileext*), 1

## M

module  
    *extensions*, 28  
    *extensions.fileext*, 1  
    *extensions.mqtttext*, 4  
    *extensions.queueext*, 7  
    *extensions.redisext*, 11  
    *extensions.sqlliteext*, 16  
    *extensions.tdictext*, 22  
    *extensions.utilext*, 26  
mqtt\_connect\_() (in module *extensions.mqtttext*), 4  
mqtt\_disconnect\_() (in module *extensions.mqtttext*), 4  
mqtt\_isrunning\_() (in module *extensions.mqtttext*), 4  
mqtt\_list\_() (in module *extensions.mqtttext*), 4  
mqtt\_readmsg\_() (in module *extensions.mqtttext*), 4  
mqtt\_sendmsg\_() (in module *extensions.mqtttext*), 4  
mqtt\_subscribe\_() (in module *extensions.mqtttext*), 4  
mqtt\_unsubscribe\_() (in module *extensions.mqtttext*), 4  
mqtt\_waiting\_() (in module *extensions.mqtttext*), 4  
MqttConnection (class in *extensions.mqtttext*), 5  
MqttExt (class in *extensions.mqtttext*), 5

## O

on\_connect\_() (*MqttConnection* method), 5  
on\_disconnect\_() (*MqttConnection* method), 5  
on\_message\_() (*MqttConnection* method), 5  
on\_topic\_message\_() (*MqttConnection* method), 5  
on\_wildcard\_message\_() (*MqttConnection* method), 5

## Q

queue\_clear\_() (in module *extensions.queueext*), 8  
queue\_clear\_() (*QueueExt* method), 8  
queue\_close\_() (in module *extensions.queueext*), 8  
queue\_close\_() (*QueueExt* method), 8

queue\_get\_() (in module *extensions.queueext*), 8  
queue\_get\_() (*QueueExt* method), 8  
queue\_isempty\_() (in module *extensions.queueext*), 8  
queue\_isempty\_() (*QueueExt* method), 9  
queue\_len\_() (in module *extensions.queueext*), 8  
queue\_len\_() (*QueueExt* method), 9  
queue\_list\_() (in module *extensions.queueext*), 8  
queue\_list\_() (*QueueExt* method), 9  
queue\_open\_() (in module *extensions.queueext*), 7  
queue\_open\_() (*QueueExt* method), 9  
queue\_put\_() (in module *extensions.queueext*), 8  
queue\_put\_() (*QueueExt* method), 9  
QueueExt (class in *extensions.queueext*), 8

## R

readLines\_() (*FileExt* method), 2  
readLines\_() (in module *extensions.fileext*), 1  
readmsg\_() (*MqttConnection* method), 5  
readmsg\_() (*MqttExt* method), 6  
redis\_cmd\_() (in module *extensions.redisext*), 11  
redis\_connect\_() (in module *extensions.redisext*), 11  
redis\_decr\_() (in module *extensions.redisext*), 11  
redis\_del\_() (in module *extensions.redisext*), 11  
redis\_disconnect\_() (in module *extensions.redisext*), 11  
redis\_get\_() (in module *extensions.redisext*), 11  
redis\_hdel\_() (in module *extensions.redisext*), 12  
redis\_hget\_() (in module *extensions.redisext*), 12  
redis\_hkeys\_() (in module *extensions.redisext*), 12  
redis\_hmset\_() (in module *extensions.redisext*), 12  
redis\_hset\_() (in module *extensions.redisext*), 12  
redis\_hvals\_() (in module *extensions.redisext*), 12  
redis\_incr\_() (in module *extensions.redisext*), 11  
redis\_list\_() (in module *extensions.redisext*), 11  
redis\_set\_() (in module *extensions.redisext*), 11  
RedisConnection (class in *extensions.redisext*), 12  
RedisExt (class in *extensions.redisext*), 13  
register\_() (*FileExt* method), 3  
register\_() (*MqttExt* method), 6  
register\_() (*QueueExt* method), 10  
register\_() (*RedisExt* method), 15  
register\_() (*SqLiteExt* method), 18  
register\_() (*TDictExt* method), 23  
register\_() (*UtilExt* method), 27  
resuball\_() (*MqttConnection* method), 5

## S

sendmsg\_() (*MqttConnection* method), 5  
sendmsg\_() (*MqttExt* method), 7  
set\_() (*RedisExt* method), 15  
shutdown\_() (*FileExt* method), 3  
shutdown\_() (*MqttExt* method), 7

shutdown() (*QueueExt method*), 10  
 shutdown() (*RedisExt method*), 15  
 shutdown() (*SQLiteExt method*), 18  
 shutdown() (*TDictExt method*), 24  
 shutdown() (*UtilExt method*), 28  
 sql\_changes\_() (in module *extensions.sqliteext*), 16  
 sql\_changes\_() (*SQLiteExt method*), 18  
 sql\_changes\_() (*SQLiteConnection method*), 20  
 sql\_close\_() (in module *extensions.sqliteext*), 16  
 sql\_close\_() (*SQLiteExt method*), 18  
 sql\_close\_() (*SQLiteConnection method*), 20  
 sql\_commit\_() (in module *extensions.sqliteext*), 16  
 sql\_commit\_() (*SQLiteExt method*), 19  
 sql\_commit\_() (*SQLiteConnection method*), 20  
 sql\_cursor\_fetch\_() (in module *extensions.sqliteext*), 16  
 sql\_cursor\_fetch\_() (*SQLiteExt method*), 19  
 sql\_cursor\_fetch\_() (*SQLiteConnection method*), 21  
 sql\_execute\_() (in module *extensions.sqliteext*), 16  
 sql\_execute\_() (*SQLiteExt method*), 19  
 sql\_execute\_() (*SQLiteConnection method*), 21  
 sql\_get\_dbname\_() (*SQLiteConnection method*), 21  
 sql\_list\_() (in module *extensions.sqliteext*), 16  
 sql\_list\_() (*SQLiteExt method*), 19  
 sql\_open\_() (in module *extensions.sqliteext*), 16  
 sql\_open\_() (*SQLiteExt method*), 19  
 sql\_open\_() (*SQLiteConnection method*), 21  
 sql\_rollback\_() (in module *extensions.sqliteext*), 16  
 sql\_rollback\_() (*SQLiteExt method*), 20  
 sql\_rollback\_() (*SQLiteConnection method*), 21  
 sqlext (*SQLiteExt attribute*), 17  
 SQLiteExt (class in *extensions.sqliteext*), 17  
 SQLiteConnection (class in *extensions.sqliteext*), 20  
 sqlroot (*SQLiteExt attribute*), 17  
 subtopic\_() (*MqttConnection method*), 5  
 subtopic\_() (*MqttExt method*), 7  
 system\_() (in module *extensions.utilext*), 26  
 system\_() (*UtilExt method*), 28

## T

tdict\_clear\_() (in module *extensions.tdictext*), 22  
 tdict\_clear\_() (*TDictExt method*), 24  
 tdict\_close\_() (in module *extensions.tdictext*), 22  
 tdict\_close\_() (*TDictExt method*), 24  
 tdict\_cmd\_() (*TDictExt method*), 24  
 tdict\_copy\_() (in module *extensions.tdictext*), 22  
 tdict\_copy\_() (*TDictExt method*), 24  
 tdict\_del\_() (in module *extensions.tdictext*), 22  
 tdict\_del\_() (*TDictExt method*), 24  
 tdict\_get\_() (in module *extensions.tdictext*), 22

tdict\_get\_() (*TDictExt method*), 24  
 tdict\_items\_() (in module *extensions.tdictext*), 22  
 tdict\_items\_() (*TDictExt method*), 25  
 tdict\_keys\_() (in module *extensions.tdictext*), 22  
 tdict\_keys\_() (*TDictExt method*), 25  
 tdict\_len\_() (in module *extensions.tdictext*), 22  
 tdict\_len\_() (*TDictExt method*), 25  
 tdict\_list\_() (in module *extensions.tdictext*), 22  
 tdict\_list\_() (*TDictExt method*), 25  
 tdict\_open\_() (in module *extensions.tdictext*), 22  
 tdict\_open\_() (*TDictExt method*), 25  
 tdict\_pop\_() (in module *extensions.tdictext*), 22  
 tdict\_pop\_() (*TDictExt method*), 25  
 tdict\_put\_() (in module *extensions.tdictext*), 22  
 tdict\_put\_() (*TDictExt method*), 25  
 tdict\_update\_() (in module *extensions.tdictext*), 22  
 tdict\_update\_() (*TDictExt method*), 25  
 TDictExt (class in *extensions.tdictext*), 23  
 ThreadDict\_ (class in *extensions.tdictext*), 26  
 ThreadQueue\_ (class in *extensions.queueext*), 11  
 topicmatch\_() (in module *extensions.mqtttext*), 7  
 tqueue\_clear\_() (*ThreadQueue\_ method*), 11  
 tqueue\_close\_() (*ThreadQueue\_ method*), 11  
 tqueue\_get\_() (*ThreadQueue\_ method*), 11  
 tqueue\_isempty\_() (*ThreadQueue\_ method*), 11  
 tqueue\_len\_() (*ThreadQueue\_ method*), 11  
 tqueue\_put\_() (*ThreadQueue\_ method*), 11  
 typeWrite\_() (in module *extensions.fileext*), 4

## U

unregister\_() (*FileExt method*), 3  
 unregister\_() (*MqttExt method*), 7  
 unregister\_() (*QueueExt method*), 10  
 unregister\_() (*RedisExt method*), 16  
 unregister\_() (*SQLiteExt method*), 20  
 unregister\_() (*TDictExt method*), 26  
 unregister\_() (*UtilExt method*), 28  
 unsubtopic\_() (*MqttConnection method*), 5  
 unsubtopic\_() (*MqttExt method*), 7  
 UtilExt (class in *extensions.utilext*), 26

## W

waiting\_() (*MqttConnection method*), 5  
 waiting\_() (*MqttExt method*), 7  
 writeLines\_() (*FileExt method*), 3  
 writeLines\_() (in module *extensions.fileext*), 1