

---

# uzmq Documentation

*Release*

**Author**

November 27, 2012



# CONTENTS



# UZMQ

libuv interface for ZeroMQ for your Python programs.

With uzmq you can use `zmq` sockets with the libuv event loop binding proposed by the [pyuv library](#)

## 1.1 Features

- Simple interface to zeromq with the libuv event loop
- *ZMQPoll handle*: Poll handle
- *ZMQ handle*: ZMQ handle

---

**Note:** uzmq source code is hosted on [Github](#)

---

## 1.2 Example of usage

Example of an echo server using a Poll handle:

```
import pyuv
import zmq
import uzmq

loop = pyuv.Loop.default_loop()

ctx = zmq.Context()
s = ctx.socket(zmq.REP)
s.bind('tcp://127.0.0.1:5555')

def rep_handler(handle, events, errors):
    # We don't know how many recv's we can do?
    msg = s.recv()
    # No guarantee that we can do the send. We need a way of putting the
    # send in the event loop.
    s.send(msg)

poll = uzmq.ZMQPoll(loop, s)
poll.start(pyuv.UV_READABLE, rep_handler)
```

```
loop.run()
```

The same but using a ZMQ handle:

```
import pyuv
import zmq
import uzmq

loop = pyuv.Loop.default_loop()

ctx = zmq.Context()
s = ctx.socket(zmq.REP)
s.bind('tcp://127.0.0.1:5555')

stream = uzmq.ZMQ(loop, s)

def echo(handle, msg, err):
    print " ".join(msg)
    stream.write_multipart(msg)

stream.start_read(echo)

loop.run()
```

Contents:

## 1.2.1 API

### uzmq Package

#### Classes

- `ZMQ`: *ZMQ handle* class
- `ZMQPoll`: *ZMQPoll handle* class

#### ZMQPoll handle

ZMQPoll: ZMQ Poll handle

```
class uzmq.poll.ZMQPoll(loop, socket)
    Bases: object
```

##### Parameters

- **loop** – loop object where this handle runs (accessible through `Poll.loop`).
- **socket** (*int*) – zmq socket to be monitored for readability or writability.

`ZMQPoll` `ZMQPoll` handles can be used to monitor any ZMQ sockets for readability or writability.

##### loop

*Read only*

`pyuv.Loop` object where this handle runs.

**active***Read only*

Indicates if this handle is active.

**close** (*callback=None*)**Parameters** **callback** (*callable*) – Function that will be called after the `ZMQPoll` handle is closed.Close the `ZMQPoll` handle. After a handle has been closed no other operations can be performed on it.**closed***Read only*

Indicates if this handle is closing or already closed.

**start** (*events, callback*)**Parameters**

- **events** – int Mask of events that will be detected. The possible events are `pyuv.UV_READABLE` or `pyuv.UV_WRITABLE`.
- **callback** – callable Function that will be called when the `Poll` handle receives events.

Callback signature: `callback(poll_handle, events, errno)`.Start or update the event mask of the `ZMQPoll` handle.**stop** ()Stop the `Poll` handle.**ZMQ handle****class** `uzmq.sock.ZMQ` (*loop, socket*)Bases: `object`**Parameters**

- **loop** – loop object where this handle runs (accessible through `Poll.loop`).
- **socket** (*int*) – zmq socket

The ZMQ handles provides qynchronous ZMQ sockets fonctionnality both for bound and connected sockets.

**close** ()

Close the ZMQ handle. After a handle has been closed no other operations can be performed on it.

**flush** ()

Flush pending messages.

This method safely handles all pending incoming and/or outgoing messages, bypassing the inner loop, passing them to the registered callbacks.

**start\_read** (*callback, copy=True, track=False*)**Parameters**

- **callback** – callable callback must take exactly one argument, which will be a `/iist`, as returned by `socket.recv_multipart()` if callback is `None`, `recv` callbacks are disabled.
- **copy** – bool copy is passed directly to `recv`, so if copy is `False`, callback will receive Message objects. If copy is `True`, then callback will receive bytes/str objects.

- **track** – bool Should the message be tracked for notification that ZMQ has finished with it? (ignored if copy=True)

Callback signature: `callback(zmq_handle, msg, error)`.

Start reading for incoming messages from the remote endpoint.

**stop()**

Stop the ZMQ handle

**stop\_read()**

Stop reading data from the remote endpoint.

**write(msg, flags=0, copy=True, track=False, callback=None)**

### Parameters

- **msg** – object, str, Frame The content of the message
- **flags** – int Any supported flag
- **copy** – bool Should the message be tracked for notification that ZMQ has finished with it? (ignored if copy=True)
- **track** – bool Should the message be tracked for notification that ZMQ has finished with it? (ignored if copy=True)

Callback signature: `callback(zmq_handle, msg, status)`.

Send a message. See `zmq.socket.send` for details.

**write\_multipart(msg, flags=0, copy=True, track=False, callback=None)**

### Parameters

- **msg** – object, str, Frame, the content of the message
- **flags** – int Any supported flag
- **copy** – bool Should the message be tracked for notification that ZMQ has finished with it? (ignored if copy=True)
- **track** – bool Should the message be tracked for notification that ZMQ has finished with it? (ignored if copy=True)

Callback signature: `callback(zmq_handle, msg, status)`.

Send a multipart message. See `zmq.socket.send_multipart` for details.

## 1.2.2 CHANGES

### 2012/11/07 - version 0.1.1

- fix license headers
- fix doc typos

### 2012/11/07 - version 0.1.0

Initial release



# INDICES AND TABLES

- *genindex*
- *modindex*
- *search*



# PYTHON MODULE INDEX

## U

uzmq.\_\_init\_\_, ??  
uzmq.poll, ??  
uzmq.sock, ??