

---

# pyTCP Documentation

*Release unknown*

**nimpsch**

**Mar 22, 2020**



---

## Contents

---

<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Usage</b>	<b>5</b>
<b>3</b>	<b>Note</b>	<b>7</b>
<b>4</b>	<b>Contents</b>	<b>9</b>
4.1	pyTCP . . . . .	9
4.2	License . . . . .	10
4.3	Contributors . . . . .	11
4.4	Changelog . . . . .	11
4.5	pyTCP . . . . .	11
<b>5</b>	<b>Indices and tables</b>	<b>17</b>
	<b>Python Module Index</b>	<b>19</b>
	<b>Index</b>	<b>21</b>



A small tcp package to send and receive tcp messages.



# CHAPTER 1

---

## Installation

---

```
pip install pyTCP
```





## CHAPTER 2

---

### Usage

---

synchronous:

```
from pyTCP import EchoServer, TcpClient

echo_server = EchoServer("127.0.0.1", 12345)
echo_server.start_server()
client = TcpClient("127.0.0.1", 12345)
client.connect()

data_to_send = b"Test message"
client.send(data_to_send)
client_received = client.receive()
server_received = echo_server.last_received
assert data_to_send == client_received
assert data_to_send == server_received

# or with a delimiter
data_to_send = b"Test\nmessage"
client.send(data_to_send)
client_received = client.receive_until(delimiter=b'\n')
assert b"Test" == client_received

echo_server.stop_server()
client.close()
```

async:

```
import asyncio

from pyTCP import AsyncTcpClient, EchoServer

async def main():
    echo_server = EchoServer("127.0.0.1", 12345)
```

(continues on next page)

(continued from previous page)

```
echo_server.start_server()
client = AsyncTcpClient("127.0.0.1", 12345)
await client.connect()

data_to_send = b"Test message"
await client.send(data_to_send)
data = await client.receive()
assert data == data_to_send

echo_server.stop_server()
client.close()

if __name__ == "__main__":
    asyncio.run(main())
```

## CHAPTER 3

---

### Note

---

This project has been set up using PyScaffold 3.2.3. For details and usage information on PyScaffold see <https://pyscaffold.org/>.



# CHAPTER 4

---

## Contents

---

### 4.1 pyTCP

A small tcp package to send and receive tcp messages.

#### 4.1.1 Installation

```
pip install pyTCP
```

#### 4.1.2 Usage

synchronous:

```
from pyTCP import EchoServer, TcpClient

echo_server = EchoServer("127.0.0.1", 12345)
echo_server.start_server()
client = TcpClient("127.0.0.1", 12345)
client.connect()

data_to_send = b"Test message"
client.send(data_to_send)
client_received = client.receive()
server_received = echo_server.last_received
```

(continues on next page)

(continued from previous page)

```
assert data_to_send == client_received
assert data_to_send == server_received

# or with a delimiter
data_to_send = b"Test\nmessage"
client.send(data_to_send)
client_received = client.receive_until(delimiter=b'\n')
assert b"Test" == client_received

echo_server.stop_server()
client.close()
```

async:

```
import asyncio

from pyTCP import AsyncTcpClient, EchoServer

async def main():
    echo_server = EchoServer("127.0.0.1", 12345)
    echo_server.start_server()
    client = AsyncTcpClient("127.0.0.1", 12345)
    await client.connect()

    data_to_send = b"Test message"
    await client.send(data_to_send)
    data = await client.receive()
    assert data == data_to_send

    echo_server.stop_server()
    client.close()

if __name__ == "__main__":
    asyncio.run(main())
```

### 4.1.3 Note

This project has been set up using PyScaffold 3.2.3. For details and usage information on PyScaffold see <https://pyscaffold.org/>.

## 4.2 License

The MIT License (MIT)

Copyright (c) 2020 nimpsch

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## 4.3 Contributors

- nimpsch <snimpsch@gmx.com>

## 4.4 Changelog

### 4.4.1 Version 0.0.1

- First version

## 4.5 pyTCP

### 4.5.1 pyTCP package

#### Submodules

#### pyTCP.async\_client module

```
class pyTCP.async_client.AsyncTcpClient(host: str = '127.0.0.1', port: int = 8080,  
                                         auto_reconnect: bool = True)
```

Bases: `object`

Asynchronous tcp client

**host**

The ip address of the tcp server.

**Type** `str`

**port**

The port of the tcp server.

**Type** `int`

**reader**

Instance of the StreamReader

**Type** `obj:`

**writer**

Instance of the StreamWriter

**Type** `obj:`

**auto\_reconnect**

If true, a reconnect will be made on connection loss.

**Type** `bool`

**logger**

An instance of the logging module.

**Type**

`obj`

**buffer**

The split part of the msg which was not returned.

**Type** `bool`, default=True

**close()**

Closes the socket connection if it open

**connect** (*timeout: float = 10.0*)

Tries to connect to the given host. Waits 0.5 seconds until another try will be made.

**Parameters** **timeout** (*float, default 10.0*) – The maximum time this function will try to connect until a ClientTimeoutError is raised.

**Raises** ClientTimeoutError – If no connection could be established in the given time a ClientTimeoutError is raised.

**is\_connected**

Returns True if connected.

**Type** `bool`

**receive** (*bytes\_to\_receive: int = 4096*) → bytes

Receives messages from the socket. If an socket.error is raised and auto\_connect is enabled, a reconnect will be executed, otherwise an empty byte string will be returned.

**Parameters** **bytes\_to\_receive** (*int, default 4096*) – Reads the number bytes from the socket. Returns fewer bytes than bytes\_to\_receive if fewer are available.

**Returns** The received data from the socket. Or an empty byte string if socket.error is raised.

**Return type** `bytes`

**receive\_until** (*bytes\_to\_receive: int = 4096, delimiter: bytes = '\n', timeout: float = 1.0*) → bytes

Receives messages from the socket until the given delimiter is recognized.

The data will be split at the delimiter. The delimiter will be removed from the message and returned. If the received message contains a message after the delimiter, it will be stored in a buffer and prepended to the next message. If an socket.error is raised and auto\_connect is enabled, a reconnect will be executed, otherwise an empty byte string will be returned.

**Parameters**

- **bytes\_to\_receive** (*int, default 4096*) – Reads the number bytes from the socket. Returns fewer bytes than bytes\_to\_receive if fewer are available.
- **delimiter** (*bytes, default '\n'*) – Splits the read data at the delimiter
- **timeout** (*float, default 1.0*) – The maximum time this function will wait until a ClientTimeoutError is raised.

**Returns** The received data from the socket. Or an empty byte string if socket.error is raised.

**Return type** `bytes`



**Raises** `ClientTimeoutError` – Raises if no data was read or no delimiter was found withing the given time.

**send** (*data: bytes*)

Sends a message to the socket. If an `socket.error` is raised and `auto_connect` is enabled, a reconnect will be executed.

**Parameters** `data` (*bytes*) – Sends the given bytes to the socket.

## pyTCP.client module

**class** `pyTCP.client.TcpClient` (*host: str = '127.0.0.1', port: int = 8080, auto\_reconnect: bool = True*)

Bases: `object`

A tcp client

**host**

The ip address of the tcp server.

**Type** `str`

**port**

The port of the tcp server.

**Type** `int`

**auto\_reconnect**

If true, a reconnect will be made on connection loss.

**Type** `bool`

**logger**

An instance of the logging module.

**Type**

`obj`

**buffer**

The split part of the msg which was not returned.

**Type** `bool`, default=True

**close** ()

Closes the socket connection if it open

**connect** (*timeout: float = 10.0*)

Tries to connect to the given host. Waits 0.5 seconds until another try will be made.

**Parameters** `timeout` (*float, default 10.0*) – The maximum time this function will try to connect until a `ClientTimeoutError` is raised.

**Raises** `ClientTimeoutError` – If no connection could be established in the given time a `ClientTimeoutError` is raised.

**is\_connected**

Returns True if connected.

**Type** `bool`

**receive** (*bytes\_to\_receive: int = 4096*) → bytes

Receives messages from the socket. If an `socket.error` is raised and `auto_connect` is enabled, a reconnect will be executed, otherwise an empty byte string will be returned.

**Parameters** `bytes_to_receive` (*int*, default 4096) – Reads the number bytes from the socket. Returns fewer bytes than `bytes_to_receive` if fewer are available.

**Returns** The received data from the socket. Or an empty byte string if `socket.error` is raised.

**Return type** `bytes`

**receive\_until** (*bytes\_to\_receive: int = 4096, delimiter: bytes = '\n', timeout: float = 1.0*) → `bytes`  
Receives messages from the socket until the given delimiter is recognized.

The data will be split at the delimiter. The delimiter will be removed from the message and returned. If the received message contains a message after the delimiter, it will be stored in a buffer and prepended to the next message. If an `socket.error` is raised and `auto_connect` is enabled, a reconnect will be executed, otherwise an empty byte string will be returned.

**bytes\_to\_receive** [*int*, default 4096] Reads the number bytes from the socket. Returns fewer bytes than `bytes_to_receive` if fewer are available.

**delimiter** [*bytes*, default '\n'] Splits the read data at the delimiter

**timeout** [*float*, default 1.0] The maximum time this function will wait until a `ClientTimeoutError` is raised.

**bytes** The received data from the socket. Or an empty byte string if `socket.error` is raised.

**ClientTimeoutError** Raises if no data was read or no delimiter was found withing the given time.

**send** (*data: bytes*)

Sends a message to the socket. If an `socket.error` is raised and `auto_connect` is enabled, a reconnect will be executed.

**Parameters** `data` (*bytes*) – Sends the given bytes to the socket.

## pyTCP.client\_errors module

**exception** `pyTCP.client_errors.ClientError`  
Bases: `Exception`

**exception** `pyTCP.client_errors.ClientProtocolError`  
Bases: `pyTCP.client_errors.ClientError`

**exception** `pyTCP.client_errors.ClientSocketError`  
Bases: `pyTCP.client_errors.ClientError`

**exception** `pyTCP.client_errors.ClientTimeoutError`  
Bases: `pyTCP.client_errors.ClientError`

## pyTCP.server module

**class** `pyTCP.server.EchoServer` (*ip, port, receive\_bytes=4096*)  
Bases: `object`

**last\_received**  
Returns the last received message.

**Type** `bytes`

**start\_server** ()  
Starts the tcp server.

**stop\_server()**

Stops the tcp server.

**class** pyTCP.server.**ThreadedTCPRequestHandler**(*request, client\_address, server*)

Bases: `socketserver.BaseRequestHandler`

A threaded tcp request handler

**handle()**

The handle function.

Reads data from the client as long as the server is not requested to close. Sends the read data back to the client and stores it in a queue.

**class** pyTCP.server.**ThreadedTCPServer**(*server\_address, RequestHandlerClass, bind\_and\_activate=True*)

Bases: `socketserver.ThreadingMixIn, socketserver.TCPServer`

## Module contents



## CHAPTER 5

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### p

- `pyTCP`, [15](#)
- `pyTCP.async_client`, [11](#)
- `pyTCP.client`, [13](#)
- `pyTCP.client_errors`, [14](#)
- `pyTCP.server`, [14](#)





## A

AsyncTcpClient (*class in pyTCP.async\_client*), 11  
auto\_reconnect (*pyTCP.async\_client.AsyncTcpClient attribute*), 11  
auto\_reconnect (*pyTCP.client.TcpClient attribute*), 13

## B

buffer (*pyTCP.async\_client.AsyncTcpClient attribute*), 12  
buffer (*pyTCP.client.TcpClient attribute*), 13

## C

ClientError, 14  
ClientProtocolError, 14  
ClientSocketError, 14  
ClientTimeoutError, 14  
close() (*pyTCP.async\_client.AsyncTcpClient method*), 12  
close() (*pyTCP.client.TcpClient method*), 13  
connect() (*pyTCP.async\_client.AsyncTcpClient method*), 12  
connect() (*pyTCP.client.TcpClient method*), 13

## E

EchoServer (*class in pyTCP.server*), 14

## H

handle() (*pyTCP.server.ThreadedTCPRequestHandler method*), 15  
host (*pyTCP.async\_client.AsyncTcpClient attribute*), 11  
host (*pyTCP.client.TcpClient attribute*), 13

## I

is\_connected (*pyTCP.async\_client.AsyncTcpClient attribute*), 12  
is\_connected (*pyTCP.client.TcpClient attribute*), 13

## L

last\_received (*pyTCP.server.EchoServer attribute*), 14  
logger (*pyTCP.async\_client.AsyncTcpClient attribute*), 12  
logger (*pyTCP.client.TcpClient attribute*), 13

## P

port (*pyTCP.async\_client.AsyncTcpClient attribute*), 11  
port (*pyTCP.client.TcpClient attribute*), 13  
pyTCP (*module*), 15  
pyTCP.async\_client (*module*), 11  
pyTCP.client (*module*), 13  
pyTCP.client\_errors (*module*), 14  
pyTCP.server (*module*), 14

## R

reader (*pyTCP.async\_client.AsyncTcpClient attribute*), 11  
receive() (*pyTCP.async\_client.AsyncTcpClient method*), 12  
receive() (*pyTCP.client.TcpClient method*), 13  
receive\_until() (*pyTCP.async\_client.AsyncTcpClient method*), 12  
receive\_until() (*pyTCP.client.TcpClient method*), 14

## S

send() (*pyTCP.async\_client.AsyncTcpClient method*), 13  
send() (*pyTCP.client.TcpClient method*), 14  
start\_server() (*pyTCP.server.EchoServer method*), 14  
stop\_server() (*pyTCP.server.EchoServer method*), 15

## T

TcpClient (*class in pyTCP.client*), 13

ThreadedTCPRequestHandler (class in *pyTCP.server*), [15](#)

ThreadedTCPServer (class in *pyTCP.server*), [15](#)

## W

writer (*pyTCP.async\_client.AsyncTcpClient* attribute), [11](#)