

---

**httpsuite**

**Felipe Faria**

**Apr 02, 2021**



# CONTENTS

<b>1</b>	<b>Quickstart</b>	<b>1</b>
1.1	Installing . . . . .	1
1.2	Getting Started . . . . .	1
1.3	Modify . . . . .	2
1.4	Compile . . . . .	2
1.5	More . . . . .	3
<b>2</b>	<b>http</b>	<b>5</b>
<b>3</b>	<b>interface</b>	<b>7</b>
<b>4</b>	<b>Basic Use</b>	<b>9</b>
4.1	Request . . . . .	9
4.2	Response . . . . .	10
<b>5</b>	<b>Advance Use</b>	<b>11</b>
5.1	Sockets . . . . .	11
5.2	Microservice . . . . .	12
<b>6</b>	<b>Glossary</b>	<b>15</b>
<b>7</b>	<b>Versions</b>	<b>17</b>
7.1	v1 . . . . .	17
<b>8</b>	<b>License</b>	<b>19</b>
	<b>Index</b>	<b>21</b>



## QUICKSTART

### 1.1 Installing

To get started with `httpsuite`, install the latest stable release from [PyPi](#):

```
pip install httpsuite
```

### 1.2 Getting Started

There are two principal abstractions to be aware of it in `httpsuite`, `Request` and `Response`.

```
from httpsuite import Request, Response
```

These classes represent an HTTP/1.x request and response message, and offer a high-level API to *modify*, *compile*, and *parse* them. The `Request` and `Response` can be initialized via two different methods.

#### 1.2.1 `__init__()`

Create a `Request` or `Response` object with the given parameter.

```
from httpsuite import Request, Response

request = Request(
    method="POST",
    target="/",
    protocol="HTTP/1.1",
    headers={"User-Agent": "httpsuite", "Content-Length": 12},
    body="Hello world."
)

response = Response(
    protocol="HTTP/1.1",
    status=200,
    status_msg="OK",
    headers={"User-Agent": "httpsuite", "Content-Length": 8},
    body="Hi back!"
)
```

### 1.2.2 .parse()

Or, parse from *bytes* to create a new Request or Response with the given details.

#### bytes

Most useful when using socket connections.

```
from httpsuite import Request, Response

req = Request.parse(b"GET / HTTP/1.1\r\nUser-Agent: httpsuite\r\nContent-Length: 12\r\n\r\nHello world")
resp = Response.parse(b"HTTP/1.1 200 OK\r\nUser-Agent: httpsuite\r\nContent-Length: 12\r\n\r\nHi back!")
```

## 1.3 Modify

The next probable step after initializing a Request or Response object is to *modify* and *compile*. Object modification is done as one would expect.

```
request.method = "POST"
response.status = 300
response.status_msg = b"Continue"
```

Notice that setting object properties is type-agnostic. Properties can be modified to either `int`, `str`, or `bytes` objects. Internally, `httpsuite` automatically converts every property of a request or response into an `Item`, which is a low-level interface to allow easy setting and comparisons on the fly. Similarly to setting properties, one can be assured of type-agnostic property comparisons.

```
request.status == 300      # True
request.status == "300"    # True
request.status == b"300"   # True
```

## 1.4 Compile

After modifying a message compilation allows the Request and Response objects to be compiled into less mutable yet useful types. Those types being `bytes` or `str`.

```
from httpsuite import Request, Response
import json

body = json.dumps({"hello": "world"})
request = Request(
    method="POST",
    target="/post",
    protocol="HTTP/1.1",
    headers={
        "Host": "httpbin.org",
        "Connection": "close",
        "Content-Length": len(body),
        "Accept": "*/*",
```

(continues on next page)

(continued from previous page)

```
    },  
    body=body,  
)
```

### 1.4.1 .raw

Useful to use with sockets.

```
print(request.raw)
```

```
b'POST /post HTTP/1.1\r\nHost: httpbin.org\r\nConnection: close\r\nContent-Length: 18\r\n\r\nAccept: */*\r\n\r\n{"hello": "world"}'
```

### 1.4.2 \_\_str\_\_

Pretty print of the object.

```
print(request)
```

```
→ POST /post HTTP/1.1  
→ Host: httpbin.org  
→ Connection: close  
→ Content-Length: 18  
→ Accept: */*  
→ {"hello": "world"}
```

## 1.5 More

If you finished this guide and want to continue learning more you can do so by reading the package's documentation found on the left menu.





---

## CHAPTER TWO

---

## HTTP

```
from http suite import Message, Request, Response
```

---

---

---



---

## CHAPTER THREE

---

### INTERFACE

```
from httpsuite.interface import Item, Headers, TwoWayFrozenDict, FrozenSet
```

---

---

---



## BASIC USE

## 4.1 Request

```
1 import json
2
3 from httpsuite import Request, ENCODE
4
5 # 1. Creates the body of the request.
6 body = json.dumps({"hello": "world"})
7
8 # 2. Creates an HTTP request.
9 request = Request(
10     method="GET",
11     target="/",
12     protocol="HTTP/1.1",
13     headers={
14         "Host": "www.google.com",
15         "Connection": "keep-alive",
16         "Content-Length": len(body),
17     },
18     body=body,
19 )
20
21 # 3. Parses the equivalent request as the above.
22 request_parsed = request = Request.parse(
23     (
24         b"GET / HTTP/1.1\r\n"
25         b"Host: www.google.com\r\n"
26         b"Connection: keep-alive\r\n"
27         b"Content-Length: %i\r\n"
28         b"\r\n"
29         b"%b"
30     )
31     % (len(body), body.encode(ENCODE))
32 )
```

## 4.2 Response

```
1 import json
2
3 from httpsuite import Response, ENCODE
4
5 # 1. Creates the body of the request.
6 body = json.dumps({"hello": "world"})
7
8 # 2. Creates an HTTP response.
9 response = Response(
10     protocol="HTTP/1.1",
11     status=200,
12     status_msg="OK",
13     headers={
14         "Host": "www.google.com",
15         "Connection": "keep-alive",
16         "Content-Length": len(body),
17     },
18     body=body,
19 )
20
21 # 3. Parses the equivalent response as the above.
22 response_parsed = Response.parse(
23     (
24         b"HTTP/1.1 200 OK\r\n"
25         b"Host: www.google.com\r\n"
26         b"Connection: keep-alive\r\n"
27         b"Content-Length: %i\r\n"
28         b"\r\n"
29         b"%b"
30     )
31     % (len(body), body.encode(ENCODE))
32 )
```

## ADVANCE USE

Advance examples of httpsuite being used.

### 5.1 Sockets

```
1  """ Example of using httpsuite to communicate with an external web server using
   ↳ sockets.
2
3  1. Open a new socket.
4  2. Creates the request to be sent via Request object.
5  3. Connect to httpbin.org via socket.
6  4. Send generated request to server.
7  5. Receive raw response from server.
8  6. Parse the server's response with Response object.
9  7. Loads the response's body via JSON.
10 8. Close socket.
11 """
12
13 from httpsuite import Request, Response
14 import socket
15 import json
16
17 # 1. Open a new socket.
18 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
19
20 # 2. Creates the request to be sent via Request object.
21 body = json.dumps({"hello": "world"})
22 request = Request(
23     method="POST",
24     target="/post",
25     protocol="HTTP/1.1",
26     headers={
27         "Host": "httpbin.org",
28         "Connection": "close",
29         "Content-Length": len(body),
30         "Accept": "*/*",
31     },
32     body=body,
33 )
34
35 # Prints the raw request.
36 print("==== Raw Request =====", "\n")
```

(continues on next page)

(continued from previous page)

```

37 print(request.raw, "\n")
38
39 # 3. Connect to httpbin.org via socket.
40 s.connect(("httpbin.org", 80))
41
42 # 4. Send generated request to server.
43 s.sendall(request.raw)
44
45 # 5. Receive raw response from server.
46 response_raw = s.recv(4096)
47
48 # Prints the raw response.
49 print("==== Raw Response ====", "\n")
50 print(response_raw, "\n")
51
52 # 6. Parse the server's response with Response object.
53 response = Response.parse(response_raw)
54
55 # Prints the request and the response (pretty-print).
56 print("==== Request and Response ====", "\n")
57 print(request, "\n")
58 print(response, "\n")
59
60 # 7. Loads the response's body via JSON.
61 body = json.loads(response.body.string)
62
63 # Prints the loaded json ('dumps' for pretty-print).
64 print("==== Json ====", "\n")
65 print(json.dumps(body, indent=4))
66
67 # 8. Close socket.
68 s.close()

```

## 5.2 Microservice

```

1  """ Very primitive example of a socket microservice architecture using httpsuite.
2
3  Server and Client functions are ran through the 'multiprocessing' module, so
4  to act as two separate entities. For the sake of clarity, only the server function
5  prints anything to console. Entities are documented seperately.
6  """
7
8  from httpsuite import Request, Response
9  from multiprocessing import Process
10 import socket
11 import time
12
13
14 def server():
15     """Simple socket server that uses httpsuite to interpret and reply.
16
17     1. Opens a new socket.
18     2. Binds to 127.0.0.1:8080 and waits until new connection.
19     3. Accepts connection from external source.

```

(continues on next page)



(continued from previous page)

```

20 4. Receive the data from the client.
21 5. Parse the clients request.
22 6. Interpret the request.
23 7. Reply to the client.
24 8. Close the connection with the client.
25 """
26
27 # 1. Opens a new socket.
28 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
29
30 # 2. Binds to 127.0.0.1:8080 and waits until new connection.
31 s.bind(("127.0.0.1", 8080))
32 s.listen(1)
33
34 # 3. Accepts connection from external source.
35 conn, address = s.accept()
36
37 print("==== Connecting With New Client =====", "\n")
38 print(address, "\n")
39
40 # 4. Receive the data from the client.
41 data = conn.recv(1024)
42
43 # 5. Parse the clients request.
44 request = Request.parse(data)
45
46 print("==== Received Data From Client =====", "\n")
47 print(request, "\n")
48
49 # 6. Interpret the request.
50 response = Response(protocol="HTTP/1.1", status=200, status_msg="OK")
51 if request.target == "/":
52     response.body = "Homepage of the microservice."
53 elif request.target == "/data":
54     response.body = "You are accessing the /data directory of this microservice."
55 else:
56     response.status = 404
57     response.status_msg = "Not Found"
58
59 print("==== Replying to Client =====", "\n")
60 print(response, "\n")
61
62 # 7. Reply to the client.
63 conn.sendall(response.raw)
64
65 print("==== Closing Connection to Client =====", "\n")
66
67 # 8. Close the connection with the client, and the server.
68 conn.close()
69 s.close()
70
71
72 def client():
73     """Simple socket client that uses httpmodule to request server resource.
74
75     1. Opens a new socket.
76     2. Connects the server.

```

(continues on next page)

```
77     3. Creates a valid request to send to the server.
78     4. Sends the request.
79     5. Receives reply from the server.
80     6. Parses the reply from the server.
81     7. Closes connection with the server.
82     """
83
84     # Note: Sleeps so that the socket server can boot-up before.
85     time.sleep(1)
86
87     # 1. Opens a new socket.
88     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
89
90     # 2. Connects the server.
91     s.connect(("127.0.0.1", 8080))
92
93     # 3. Creates a valid request to send to the server.
94     request = Request(method="GET", target="/", protocol="HTTP/1.1")
95
96     # 4. Sends the request.
97     s.sendall(request.raw)
98
99     # 5. Receives reply from the server.
100    data = s.recv(1024)
101
102    # 6. Parses the reply from the server.
103    response = Response.parse(data)
104
105    # 7. Closes connection with the server.
106    s.close()
107
108
109    if __name__ == "__main__":
110        p1 = Process(target=server)
111        p2 = Process(target=client)
112
113        p1.daemon = True
114        p2.daemon = True
115
116        p1.start()
117        p2.start()
118
119        time.sleep(3)
120        raise SystemExit
```

## GLOSSARY

**compile** In the context of `httpsuite`, compiling means converting a `Response` or `Request` object into another more useable type, typically either `str` or `bytes`.

**message** The parent object of both an HTTP request and response.

**modify** In the context of `httpsuite`, modifying means changing attributes in either the `Request` or `Response` objects. When modifications are applied to these objects usually compilation follows.

**parse** In the context of `httpsuite`, parsing means interpreting an external type and creating a representative `Response` or `Request` object.



## VERSIONS

### 7.1 v1

#### 7.1.1 1.1.0

- Revamped and cleaned the codebase.
- Update documentation.

#### 7.1.2 1.0.6

- Updated core `Message` to include `protocol` by default.
- Removed `info.py` and place content inside `__init__.py`.
- Formatted some tests with Black.

#### 7.1.3 1.0.5

- Fixed documentation bug relating to `sphinx_rtd_theme`.

#### 7.1.4 1.0.4

- Fixed documentation bug relating to `m2r`.

#### 7.1.5 1.0.3

- Fixed PyLint errors.
- Added `__bool__` to `Item` for accurate comparissons (i.e. `if request.body` will return `False` when no body exists).
- Modified `Message._compile` input param format to `frmt`.

### 7.1.6 1.0.2

- Cleaned up imports and codebase.

### 7.1.7 1.0.1

- Cleaned up imports and codebase.

### 7.1.8 1.0.0

- Initial commit.

## LICENSE

MIT License

Copyright (c) 2020 Felipe Faria

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.





## INDEX

### C

compile, [15](#)

### M

message, [15](#)

modify, [15](#)

### P

parse, [15](#)