Werkzeug and Pythons http and socketserver Library

General Information & Licensing

Code Repository	https://github.com/pallets/werkzeug
License Type	BSD 3-Clause "New" or "Revised" License PSF License Agreement. (for any Python library)
License Description	• A permissive license similar to the BSD 2-Clause License, but with a 3rd clause that prohibits others from using the name of the copyright holder or its contributors to promote derived products without written consent. • Essentially, this license just means you're allowed to use the current repository but you aren't allowed to warrant it off as your own and promoting the creator and any contributors to individuals without getting consent from these individuals. Python's License Description • This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 3.11.0 software in source or binary form and its associated documentation. • 2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works,distribute, and otherwise use Python 3.11.0 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001-2022 Python Software Foundation; All Rights Reserved" are retained in Python 3.11.0 alone or

	in any derivative version prepared by Licensee. ● 3. In the event Licensee prepares a derivative work that is based on or incorporates Python 3.11.0 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 3.11.0. ● 4. PSF is making Python 3.11.0 available to Licensee on an "AS IS" basis.PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 3.11.0 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS. ● 5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.11.0 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.11.0, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. ● 6. This License Agreement will automatically terminate upon a material breach of its terms and conditions. ● 7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party. ● 8. By copying, installing or otherwise using Python 3.11.0,

	Licensee agrees to be bound by the terms and conditions of this License Agreement.
License Restrictions	● Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. ● Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. ● Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

Once our socket and TCP connection is made, our server is now up and running and ready to accept incoming requests. As we made the server, this is where we also defined how we will be handling our incoming requests. When we call *run_simple*, we go through until we get to line 1071 where we call *make_server.* If we look at the parameters of *make_server*, we'll see that the request_handler is of type *WSGIRequestHandler.*

Since that is the case, when we get an incoming request, we move into the *WSGIRequestHandler* class where our constructor is a *BaseHTTPRequestHandler* where we have a method called *parse_request*.

This method will essentially parse the request it was given AND within this method, on line 337, we have it say "*self.headers = http.client.parse_headers(...)*. From this point, we have the code that then strips and parses the headers but it gets quite a bit difficult to follow since we start making objects of type Email and start making Parser objects in order to successfully parse these headers but this is where the headers are found. The headers then get put into our class of *WSGIRequestHandler* since that gets passed around!

Links To Code In Files (including images for some of them since I couldn't get links):

- https://github.com/pallets/werkzeug/blob/3115aa6a6276939f5fd6efa46282e0256ff21f1a/src/werkzeug/serving.py#L148
- https://github.com/pallets/werkzeug/blob/3115aa6a6276939f5fd6efa46282e0256ff21f1a/src/werkzeug/serving.py#L358

```
class BaseRequestHandler:

    """Base class for request handler classes.

    This class is instantiated for each request to be handled.  The
    constructor sets the instance variables request, client_address
    and server, and then calls the handle() method.  To implement a
    specific service, all you need to do is to derive a class which
    defines a handle() method.

    The handle() method can find the request as self.request, the
    client address as self.client_address, and the server (in case it
    needs access to per-server information) as self.server.  Since a
    separate instance is created for each request, the handle() method
    can define other arbitrary instance variables.

    """

    def __init__(self, request, client_address, server):   self: <werkzeug.
        self.request = request
        self.client_address = client_address
        self.server = server
        self.setup()
        try:
            self.handle()
        finally:
            self.finish()

    def setup(self):
        pass

    def handle(self):
        pass

    def finish(self):
        pass
```

```
class BaseHTTPRequestHandler(socketserver.StreamRequestHandler):
```

```
269 *       def parse_request(self):
```

```
336         try:
337             self.headers = http.client.parse_headers(self.rfile,
338                                                      _class=self.MessageClass)
```

```
def parse_headers(fp, _class=HTTPMessage):
    """Parses only RFC2822 headers from a file pointer.

    email Parser wants to see strings rather than bytes.
    But a TextIOWrapper around self.rfile would buffer too many bytes
    from the stream, bytes which we later need to read as bytes.
    So we read the correct bytes here, as bytes, for email Parser
    to parse.

    """
    headers = []
    while True:
        line = fp.readline(_MAXLINE + 1)
        if len(line) > _MAXLINE:
            raise LineTooLong("header line")
        headers.append(line)
        if len(headers) > _MAXHEADERS:
            raise HTTPException("got more than %d headers" % _MAXHEADERS)
        if line in (b'\r\n', b'\n', b''):
            break
    hstring = b''.join(headers).decode('iso-8859-1')
    return email.parser.Parser(_class=_class).parsestr(hstring)
```