This module defines classes which implement the client side of the HTTP and HTTPS protocols. It is normally not used directly — the module `urllib` uses it to handle URLs that use HTTP and HTTPS.

Note: HTTPS support is only available if the `socket` module was compiled with SSL support.

Note: The public interface for this module changed substantially in Python 2.0. The `HTTP` class is retained only for backward compatibility with 1.5.2. It should not be used in new code. Refer to the online docstrings for usage.

The module provides the following classes:

```python
class httplib.HTTPConnection(host[, port[, strict[, timeout[, source_address]]]])
```
An `HTTPConnection` instance represents one transaction with an HTTP server. It should be instantiated passing it a host and optional port number. If no port number is passed, the port is extracted from the host string if it has the form `host:port`, else the default HTTP port (80) is used. When `True`, the optional parameter `strict` (which defaults to a false value) causes `BadStatusLine` to be raised if the status line can’t be parsed as a valid HTTP/1.0 or 1.1 status line. If the optional `timeout` parameter is given, blocking operations (like connection attempts) will timeout after that many seconds (if it is not given, the global default timeout setting is used). The optional `source_address` parameter may be a tuple of a (host, port) to use as the source address the HTTP connection is made from.

For example, the following calls all create instances that connect to the server at the same host and port:

```python
>>> h1 = httplib.HTTPConnection('www.cwi.nl')
>>> h2 = httplib.HTTPConnection('www.cwi.nl:80')
>>> h3 = httplib.HTTPConnection('www.cwi.nl', 80)
>>> h3 = httplib.HTTPConnection('www.cwi.nl', 80, timeout=10)
```

New in version 2.0.

Changed in version 2.6: `timeout` was added.
Changed in version 2.7: source_address was added.

```python
class httplib.HTTPSConnection(host[, port[, key_file[, cert_file[, strict[, timeout[, source_address]]]]]])
```

A subclass of HTTPConnection that uses SSL for communication with secure servers. Default port is 443. key_file is the name of a PEM formatted file that contains your private key. cert_file is a PEM formatted certificate chain file.

**Warning:** This does not do any verification of the server’s certificate.

New in version 2.0.

Changed in version 2.6: timeout was added.

Changed in version 2.7: source_address was added.

```python
class httplib.HTTPResponse(sock[, debuglevel=0][, strict=0])
```

Class whose instances are returned upon successful connection. Not instantiated directly by user.

New in version 2.0.

```python
class httplib.HTTPMessage
```

An HTTPMessage instance is used to hold the headers from an HTTP response. It is implemented using the mimetools.Message class and provides utility functions to deal with HTTP Headers. It is not directly instantiated by the users.

The following exceptions are raised as appropriate:

```python
exception httplib.HTTPException
```

The base class of the other exceptions in this module. It is a subclass of Exception.

New in version 2.0.

```python
exception httplib.NotConnected
```

A subclass of HTTPException.

New in version 2.0.

```python
exception httplib.InvalidURL
```

A subclass of HTTPException, raised if a port is given and is either non-numeric or empty.

New in version 2.3.

```python
exception httplib.UnknownProtocol
```
A subclass of `HTTPException`.

*New in version 2.0.*

```python
exception `httplib.UnknownTransferEncoding`
A subclass of `HTTPException`.

*New in version 2.0.*
```

```python
exception `httplib.UnimplementedFileMode`
A subclass of `HTTPException`.

*New in version 2.0.*
```

```python
exception `httplib.IncompleteRead`
A subclass of `HTTPException`.

*New in version 2.0.*
```

```python
exception `httplib.ImproperConnectionState`
A subclass of `HTTPException`.

*New in version 2.0.*
```

```python
exception `httplib.CannotSendRequest`
A subclass of `ImproperConnectionState`.

*New in version 2.0.*
```

```python
exception `httplib.CannotSendHeader`
A subclass of `ImproperConnectionState`.

*New in version 2.0.*
```

```python
exception `httplib.ResponseNotReady`
A subclass of `ImproperConnectionState`.

*New in version 2.0.*
```

```python
exception `httplib.BadStatusLine`
A subclass of `HTTPException`. Raised if a server responds with a HTTP status code that we don’t understand.

*New in version 2.0.*
```

The constants defined in this module are:

```python
`httplib.HTTP_PORT`
The default port for the HTTP protocol (always 80).
```
The default port for the HTTPS protocol (always 443).

and also the following constants for integer status codes:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTINUE</td>
<td>100</td>
<td>HTTP/1.1, RFC 2616, Section 10.1.1</td>
</tr>
<tr>
<td>SWITCHING_PROTOCOLS</td>
<td>101</td>
<td>HTTP/1.1, RFC 2616, Section 10.1.2</td>
</tr>
<tr>
<td>PROCESSING</td>
<td>102</td>
<td>WEBDAV, RFC 2518, Section 10.1</td>
</tr>
<tr>
<td>OK</td>
<td>200</td>
<td>HTTP/1.1, RFC 2616, Section 10.2.1</td>
</tr>
<tr>
<td>CREATED</td>
<td>201</td>
<td>HTTP/1.1, RFC 2616, Section 10.2.2</td>
</tr>
<tr>
<td>ACCEPTED</td>
<td>202</td>
<td>HTTP/1.1, RFC 2616, Section 10.2.3</td>
</tr>
<tr>
<td>NON_AUTHORITATIVE_INFORMATION</td>
<td>203</td>
<td>HTTP/1.1, RFC 2616, Section 10.2.4</td>
</tr>
<tr>
<td>NO_CONTENT</td>
<td>204</td>
<td>HTTP/1.1, RFC 2616, Section 10.2.5</td>
</tr>
<tr>
<td>RESET_CONTENT</td>
<td>205</td>
<td>HTTP/1.1, RFC 2616, Section 10.2.6</td>
</tr>
<tr>
<td>PARTIAL_CONTENT</td>
<td>206</td>
<td>HTTP/1.1, RFC 2616, Section 10.2.7</td>
</tr>
<tr>
<td>MULTI_STATUS</td>
<td>207</td>
<td>WEBDAV RFC 2518, Section 10.2</td>
</tr>
<tr>
<td>IM_USED</td>
<td>226</td>
<td>Delta encoding in HTTP, RFC 3229, Section 10.4.1</td>
</tr>
<tr>
<td>MULTIPLE_CHOICES</td>
<td>300</td>
<td>HTTP/1.1, RFC 2616, Section 10.3.1</td>
</tr>
<tr>
<td>MOVED_PERMANENTLY</td>
<td>301</td>
<td>HTTP/1.1, RFC 2616, Section 10.3.2</td>
</tr>
<tr>
<td>FOUND</td>
<td>302</td>
<td>HTTP/1.1, RFC 2616, Section 10.3.3</td>
</tr>
<tr>
<td>SEE_OTHER</td>
<td>303</td>
<td>HTTP/1.1, RFC 2616, Section 10.3.4</td>
</tr>
<tr>
<td>NOT_MODIFIED</td>
<td>304</td>
<td>HTTP/1.1, RFC 2616, Section 10.3.5</td>
</tr>
<tr>
<td>USE_PROXY</td>
<td>305</td>
<td>HTTP/1.1, RFC 2616, Section 10.3.6</td>
</tr>
<tr>
<td>TEMPORARY_REDIRECT</td>
<td>307</td>
<td>HTTP/1.1, RFC 2616, Section 10.3.8</td>
</tr>
<tr>
<td>BAD_REQUEST</td>
<td>400</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.1</td>
</tr>
<tr>
<td>UNAUTHORIZED</td>
<td>401</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.2</td>
</tr>
<tr>
<td>PAYMENT_REQUIRED</td>
<td>402</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.3</td>
</tr>
<tr>
<td>FORBIDDEN</td>
<td>403</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.4</td>
</tr>
<tr>
<td>NOT_FOUND</td>
<td>404</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.5</td>
</tr>
<tr>
<td>METHOD_NOT_ALLOWED</td>
<td>405</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.6</td>
</tr>
<tr>
<td>NOT_ACCEPTABLE</td>
<td>406</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.7</td>
</tr>
<tr>
<td>PROXY_AUTHENTICATION_REQUIRED</td>
<td>407</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.8</td>
</tr>
<tr>
<td>REQUEST_TIMEOUT</td>
<td>408</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.9</td>
</tr>
<tr>
<td>CONFLICT</td>
<td>409</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.10</td>
</tr>
<tr>
<td>GONE</td>
<td>410</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.11</td>
</tr>
<tr>
<td>LENGTH_REQUIRED</td>
<td>411</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.12</td>
</tr>
<tr>
<td>PRECONDITION_FAILED</td>
<td>412</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.13</td>
</tr>
<tr>
<td>REQUEST_ENTITY_TOO_LARGE</td>
<td>413</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.14</td>
</tr>
<tr>
<td>REQUEST_URI_TOO_LONG</td>
<td>414</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.15</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>--------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UNSUPPORTED_MEDIA_TYPE</td>
<td>415</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.16</td>
</tr>
<tr>
<td>REQUESTED_RANGE_NOT_SATISFIABLE</td>
<td>416</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.17</td>
</tr>
<tr>
<td>EXPECTATION_FAILED</td>
<td>417</td>
<td>HTTP/1.1, RFC 2616, Section 10.4.18</td>
</tr>
<tr>
<td>UNPROCESSABLE_ENTITY</td>
<td>422</td>
<td>WEBDAV, RFC 2518, Section 10.3</td>
</tr>
<tr>
<td>LOCKED</td>
<td>423</td>
<td>WEBDAV RFC 2518, Section 10.4</td>
</tr>
<tr>
<td>FAILED_DEPENDENCY</td>
<td>424</td>
<td>WEBDAV, RFC 2518, Section 10.5</td>
</tr>
<tr>
<td>UPGRADE_REQUIRED</td>
<td>426</td>
<td>HTTP Upgrade to TLS, RFC 2817, Section 6</td>
</tr>
<tr>
<td>INTERNAL_SERVER_ERROR</td>
<td>500</td>
<td>HTTP/1.1, RFC 2616, Section 10.5.1</td>
</tr>
<tr>
<td>NOT_IMPLEMENTED</td>
<td>501</td>
<td>HTTP/1.1, RFC 2616, Section 10.5.2</td>
</tr>
<tr>
<td>BAD_GATEWAY</td>
<td>502</td>
<td>HTTP/1.1 RFC 2616, Section 10.5.3</td>
</tr>
<tr>
<td>SERVICE_UNAVAILABLE</td>
<td>503</td>
<td>HTTP/1.1, RFC 2616, Section 10.5.4</td>
</tr>
<tr>
<td>GATEWAY_TIMEOUT</td>
<td>504</td>
<td>HTTP/1.1 RFC 2616, Section 10.5.5</td>
</tr>
<tr>
<td>HTTP_VERSION_NOT_SUPPORTED</td>
<td>505</td>
<td>HTTP/1.1, RFC 2616, Section 10.5.6</td>
</tr>
<tr>
<td>INSUFFICIENT_STORAGE</td>
<td>507</td>
<td>WEBDAV, RFC 2518, Section 10.6</td>
</tr>
<tr>
<td>NOT_EXTENDED</td>
<td>510</td>
<td>An HTTP Extension Framework, RFC 2774, Section 7</td>
</tr>
</tbody>
</table>

**httplib. responses**

This dictionary maps the HTTP 1.1 status codes to the W3C names.

Example: `httplib.responses[httplib.NOT_FOUND]` is 'Not Found'.

*New in version 2.5.*

### 20.7.1. HTTPConnection Objects

HTTPConnection instances have the following methods:

**HTTPConnection.request**(method, url[, body[, headers]])

This will send a request to the server using the HTTP request method `method` and the selector `url`. If the `body` argument is present, it should be a string of data to send after the headers are finished. Alternatively, it may be an open file object, in which case the contents of the file is sent; this file object should support `fileno()` and `read()` methods. The header Content-Length is automatically set to the correct value. The `headers` argument should be a mapping of extra HTTP headers to send with the request.

*Changed in version 2.6: body can be a file object.*

**HTTPConnection.getresponse()**

Should be called after a request is sent to get the response from the server. Returns an HTTPResponse instance.
HTTPConnection.set_debuglevel(level)
Set the debugging level (the amount of debugging output printed). The default debug level is 0, meaning no debugging output is printed.

HTTPConnection.set_tunnel(host, port=None, headers=None)
Set the host and the port for HTTP Connect Tunnelling. Normally used when it is required to do HTTPS Connection through a proxy server.

The headers argument should be a mapping of extra HTTP headers to send with the CONNECT request.

*New in version 2.7.*

HTTPConnection.connect()
Connect to the server specified when the object was created.

HTTPConnection.close()
Close the connection to the server.

As an alternative to using the request() method described above, you can also send your request step by step, by using the four functions below.

HTTPConnection.putrequest(request, selector[, skip_host[, skip_accept_encoding]])
This should be the first call after the connection to the server has been made. It sends a line to the server consisting of the request string, the selector string, and the HTTP version (HTTP/1.1). To disable automatic sending of Host: or Accept-Encoding: headers (for example to accept additional content encodings), specify skip_host or skip_accept_encoding with non-False values.

*Changed in version 2.4:* skip_accept_encoding argument added.

HTTPConnection.putheader(header, argument[, ...])
Send an RFC 822-style header to the server. It sends a line to the server consisting of the header, a colon and a space, and the first argument. If more arguments are given, continuation lines are sent, each consisting of a tab and an argument.

HTTPConnection.endheaders(message_body=None)
Send a blank line to the server, signalling the end of the headers. The optional message_body argument can be used to pass a message body associated with the request. The message body will be sent in the same packet as the message headers if it is string, otherwise it is sent in a separate packet.
Changed in version 2.7: message_body was added.

```
HTTPConnection.send(data)
```

Send data to the server. This should be used directly only after the `endheaders()` method has been called and before `getresponse()` is called.

### 20.7.2. HTTPResponse Objects

HTTPResponse instances have the following methods and attributes:

```
HTTPResponse.read([amt])
```

Reads and returns the response body, or up to the next `amt` bytes.

```
HTTPResponse.getheader(name[, default])
```

Get the contents of the header `name`, or `default` if there is no matching header.

```
HTTPResponse.getheaders()
```

Return a list of (header, value) tuples.

*New in version 2.4.*

```
HTTPResponse.fileno()
```

Returns the `fileno` of the underlying socket.

```
HTTPResponse.msg
```

A `mimetools.Message` instance containing the response headers.

```
HTTPResponse.version
```

HTTP protocol version used by server. 10 for HTTP/1.0, 11 for HTTP/1.1.

```
HTTPResponse.status
```

Status code returned by server.

```
HTTPResponse.reason
```

Reason phrase returned by server.

### 20.7.3. Examples

Here is an example session that uses the `GET` method:

```
>>> import httplib
>>> conn = httplib.HTTPConnection("www.python.org")
>>> conn.request("GET", "/index.html")
>>> r1 = conn.getresponse()
>>> print r1.status, r1.reason
200 OK
>>> data1 = r1.read()
```
Here is an example session that uses the **HEAD** method. Note that the **HEAD** method never returns any data.

```python
>>> import httplib
>>> conn = httplib.HTTPConnection("www.python.org")
>>> conn.request("HEAD","/index.html")
>>> res = conn.getresponse()
>>> print res.status, res.reason
200 OK
>>> data = res.read()
>>> print len(data)
0
>>> data == ''
True
```

Here is an example session that shows how to **POST** requests:

```python
>>> import httplib, urllib
>>> params = urllib.urlencode({'@number': 12524, '@type': 'issue', '@action': 'show'})
>>> headers = {'Content-type': "application/x-www-form-urlencoded",
...            'Accept': "text/plain"}
>>> conn = httplib.HTTPConnection("bugs.python.org")
>>> conn.request("POST", "/", params, headers)
>>> response = conn.getresponse()
>>> print response.status, response.reason
302 Found
>>> data = response.read()
>>> data
'Redirecting to <a href="http://bugs.python.org/issue12524">http://bugs.python.org/issue12524</a>
>>> conn.close()
```