21.7. `httplib` — HTTP protocol client¶

Note

The `httplib` module has been renamed to `http.client` in Python 3.0. The *2to3* tool will automatically adapt imports when converting your sources to 3.0.

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:

class `httplib.HTTPConnection`(*host*[, *port*[, *strict*[, *timeout*]]])¶

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).

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

```
>>> 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.

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

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.

Note

This does not do any certificate verification.

New in version 2.0.

Changed in version 2.6: *timeout* was added.

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.

The following exceptions are raised as appropriate:

exception `httplib.HTTPException`¶

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

New in version 2.0.

exception `httplib.NotConnected`¶

A subclass of `HTTPException`.

New in version 2.0.

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.

exception `httplib.UnknownProtocol`¶

A subclass of `HTTPException`.

New in version 2.0.

exception `httplib.UnknownTransferEncoding`¶

A subclass of `HTTPException`.

New in version 2.0.

exception `httplib.UnimplementedFileMode`¶

A subclass of `HTTPException`.

New in version 2.0.

exception `httplib.IncompleteRead`¶

A subclass of `HTTPException`.

New in version 2.0.

exception `httplib.ImproperConnectionState`¶

A subclass of `HTTPException`.

New in version 2.0.

exception `httplib.CannotSendRequest`¶

A subclass of `ImproperConnectionState`.

New in version 2.0.

exception `httplib.CannotSendHeader`¶

A subclass of `ImproperConnectionState`.

New in version 2.0.

exception `httplib.ResponseNotReady`¶

A subclass of `ImproperConnectionState`.

New in version 2.0.

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:

`httplib.HTTP_PORT`¶
The default port for the HTTP protocol (always `80`).

`httplib.HTTPS_PORT`¶

The default port for the HTTPS protocol (always 443).

and also the following constants for integer status codes:

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

`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.

21.7.1. 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.

Note

Note that you must have read the whole response before you can send a new request to the server.

`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.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`()¶
Send a blank line to the server, signalling the end of the headers.

`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.

21.7.2. 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.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.

21.7.3. 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()
>>> conn.request("GET", "/parrot.spam")
>>> r2 = conn.getresponse()
>>> print r2.status, r2.reason
404 Not Found
>>> data2 = r2.read()
>>> conn.close()
```

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

```
>>> import httplib, urllib
>>> params = urllib.urlencode({'spam': 1, 'eggs': 2, 'bacon': 0})
>>> headers = {"Content-type": "application/x-www-form-urlencoded",
...            "Accept": "text/plain"}
>>> conn = httplib.HTTPConnection("musi-cal.mojam.com:80")
>>> conn.request("POST", "/cgi-bin/query", params, headers)
>>> response = conn.getresponse()
>>> print response.status, response.reason
200 OK
>>> data = response.read()
>>> conn.close()
```


Previous topic

21.6. `urllib2` — extensible library for opening URLs

Next topic

21.8. `ftplib` — FTP protocol client

This Page

- Show Source

Navigation

- index
- modules |
- next |
- previous |
- Python v2.6.4 documentation »
- The Python Standard Library »
- 21. Internet Protocols and Support »