

# Developer Interface

Fork me on GitHub

This part of the documentation covers all the interfaces of Requests. For parts where Requests depends on external libraries, we document the most important right here and provide links to the canonical documentation.

## Main Interface

All of Requests' functionality can be accessed by these 7 methods. They all return an instance of the **Response** object.

`requests.request(method, url, **kwargs)` [\[source\]](#)  
Constructs and sends a **Request**.

- Parameters:**
- **method** – method for the new **Request** object: GET, OPTIONS, HEAD, POST, PUT, PATCH, or DELETE.
  - **url** – URL for the new **Request** object.
  - **params** – (optional) Dictionary, list of tuples or bytes to send in the query string for the **Request**.
  - **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the **Request**.
  - **json** – (optional) A JSON serializable Python object to send in the body of the **Request**.
  - **headers** – (optional) Dictionary of HTTP Headers to send with the **Request**.
  - **cookies** – (optional) Dict or CookieJar object to send with the **Request**.
  - **files** – (optional) Dictionary of 'name': file-like-objects (or {'name': file-tuple}) for multipart encoding upload. file-tuple can be a 2-tuple ('filename', fileobj), 3-tuple ('filename', fileobj, 'content\_type') or a 4-tuple ('filename', fileobj, 'content\_type', custom\_headers), where 'content\_type' is a string defining the content type of the given file and custom\_headers a dict-like object containing additional headers to add for the file.
  - **auth** – (optional) Auth tuple to enable Basic/Digest/Custom HTTP Auth.
  - **timeout** (*float or tuple*) – (optional) How many seconds to wait for the server to send data before giving up, as a float, or a ([connect timeout](#), [read timeout](#)) tuple.
  - **allow\_redirects** (*bool*) – (optional) Boolean. Enable/disable GET/OPTIONS/POST/PUT/PATCH/DELETE/HEAD redirection. Defaults to True.
  - **proxies** – (optional) Dictionary mapping protocol to the URL of the proxy.
  - **verify** – (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use. Defaults to True.
  - **stream** – (optional) if `False`, the response content will be immediately downloaded.
  - **cert** – (optional) if String, path to ssl client cert file (.pem). If Tuple, ('cert', 'key') pair.

**Returns:** **Response** object

**Return type:** [requests.Response](#)

Usage:

```
>>> import requests
>>> req = requests.request('GET', 'https://httpbin.org/get')
>>> req
<Response [200]>
```

`requests.head(url, **kwargs)` [\[source\]](#)  
Sends a HEAD request.

**Parameters:**

- **url** – URL for the new **Request** object.

v: latest

- **\*\*kwargs** – Optional arguments that `request` takes. If `allow_redirects` is not provided, it will be set to `False` (as opposed to the default **request** behavior).

**Returns:** **Response** object

**Return type:** [requests.Response](#)

`requests.get(url, params=None, **kwargs)` [source]

Sends a GET request.

- Parameters:**
- **url** – URL for the new **Request** object.
  - **params** – (optional) Dictionary, list of tuples or bytes to send in the query string for the **Request**.
  - **\*\*kwargs** – Optional arguments that `request` takes.

**Returns:** **Response** object

**Return type:** [requests.Response](#)

`requests.post(url, data=None, json=None, **kwargs)` [source]

Sends a POST request.

- Parameters:**
- **url** – URL for the new **Request** object.
  - **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the **Request**.
  - **json** – (optional) A JSON serializable Python object to send in the body of the **Request**.
  - **\*\*kwargs** – Optional arguments that `request` takes.

**Returns:** **Response** object

**Return type:** [requests.Response](#)

`requests.put(url, data=None, **kwargs)` [source]

Sends a PUT request.

- Parameters:**
- **url** – URL for the new **Request** object.
  - **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the **Request**.
  - **json** – (optional) A JSON serializable Python object to send in the body of the **Request**.
  - **\*\*kwargs** – Optional arguments that `request` takes.

**Returns:** **Response** object

**Return type:** [requests.Response](#)

`requests.patch(url, data=None, **kwargs)` [source]

Sends a PATCH request.

- Parameters:**
- **url** – URL for the new **Request** object.
  - **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the **Request**.
  - **json** – (optional) A JSON serializable Python object to send in the body of the **Request**.
  - **\*\*kwargs** – Optional arguments that `request` takes.

**Returns:** **Response** object

**Return type:** [requests.Response](#)

`requests.delete(url, **kwargs)` [source]

Sends a DELETE request.

- Parameters:**
- **url** – URL for the new **Request** object.
  - **\*\*kwargs** – Optional arguments that `request` takes.

**Returns:** **Response** object

**Return type:** [requests.Response](#)

## Exceptions

`exception requests.RequestException(*args, **kwargs)` [source]

There was an ambiguous exception that occurred while handling your request.

<code>exception requests.ConnectionError(*args, **kwargs)</code> A Connection error occurred.	[source]
<code>exception requests.HTTPError(*args, **kwargs)</code> An HTTP error occurred.	[source]
<code>exception requests.URLRequired(*args, **kwargs)</code> A valid URL is required to make a request.	[source]
<code>exception requests.TooManyRedirects(*args, **kwargs)</code> Too many redirects.	[source]
<code>exception requests.ConnectTimeout(*args, **kwargs)</code> The request timed out while trying to connect to the remote server.  Requests that produced this error are safe to retry.	[source]
<code>exception requests.ReadTimeout(*args, **kwargs)</code> The server did not send any data in the allotted amount of time.	[source]
<code>exception requests.Timeout(*args, **kwargs)</code> The request timed out.  Catching this error will catch both <b>ConnectTimeout</b> and <b>ReadTimeout</b> errors.	[source]
<code>exception requests.JSONDecodeError(*args, **kwargs)</code> Couldn't decode the text into json	[source]

## Request Sessions

<code>class requests.Session</code> A Requests session.	[source]
--	----------

Provides cookie persistence, connection-pooling, and configuration.

Basic Usage:

```
>>> import requests
>>> s = requests.Session()
>>> s.get('https://httpbin.org/get')
<Response [200]>
```

Or as a context manager:

```
>>> with requests.Session() as s:
...     s.get('https://httpbin.org/get')
<Response [200]>
```

### auth

Default Authentication tuple or object to attach to **Request**.

### cert

SSL client certificate default, if String, path to ssl client cert file (.pem). If Tuple, ('cert', 'key') pair.

### close()

Closes all adapters and as such the session

[source]

### cookies

A CookieJar containing all currently outstanding cookies set on this session. By default it is a **RequestsCookieJar**, but may be any other `cookieclib.CookieJar` compatible object.

### delete(url, \*\*kwargs)

Sends a DELETE request. Returns **Response** object.

[source]

**Parameters:** • **url** – URL for the new **Request** object.

- **\*\*kwargs** – Optional arguments that `request` takes.

**Return type:** [requests.Response](#)

**get(url, \*\*kwargs)** [source]

Sends a GET request. Returns **Response** object.

- Parameters:**
- **url** – URL for the new **Request** object.
  - **\*\*kwargs** – Optional arguments that `request` takes.

**Return type:** [requests.Response](#)

**get\_adapter(url)** [source]

Returns the appropriate connection adapter for the given URL.

**Return type:** [requests.adapters.BaseAdapter](#)

**get\_redirect\_target(resp)**

Receives a Response. Returns a redirect URI or `None`

**head(url, \*\*kwargs)** [source]

Sends a HEAD request. Returns **Response** object.

- Parameters:**
- **url** – URL for the new **Request** object.
  - **\*\*kwargs** – Optional arguments that `request` takes.

**Return type:** [requests.Response](#)

## headers

A case-insensitive dictionary of headers to be sent on each **Request** sent from this **Session**.

## hooks

Event-handling hooks.

## max\_redirects

Maximum number of redirects allowed. If the request exceeds this limit, a **TooManyRedirects** exception is raised. This defaults to `requests.models.DEFAULT_REDIRECT_LIMIT`, which is 30.

**merge\_environment\_settings(url, proxies, stream, verify, cert)** [source]

Check the environment and merge it with some settings.

**Return type:** [dict](#)

**mount(prefix, adapter)** [source]

Registers a connection adapter to a prefix.

Adapters are sorted in descending order by prefix length.

**options(url, \*\*kwargs)** [source]

Sends a OPTIONS request. Returns **Response** object.

- Parameters:**
- **url** – URL for the new **Request** object.
  - **\*\*kwargs** – Optional arguments that `request` takes.

**Return type:** [requests.Response](#)

## params

Dictionary of querystring data to attach to each **Request**. The dictionary values may be lists for representing multivalued query parameters.

**patch(url, data=None, \*\*kwargs)** [source]

Sends a PATCH request. Returns **Response** object.

- Parameters:**
- **url** – URL for the new **Request** object.
  - **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the **Request**.
  - **\*\*kwargs** – Optional arguments that `request` takes.

**Return type:** [requests.Response](#)

**post**(*url*, *data=None*, *json=None*, *\*\*kwargs*) [source]

Sends a POST request. Returns **Response** object.

- Parameters:**
- **url** – URL for the new **Request** object.
  - **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the **Request**.
  - **json** – (optional) json to send in the body of the **Request**.
  - **\*\*kwargs** – Optional arguments that `request` takes.

**Return type:** [requests.Response](#)

**prepare\_request**(*request*) [source]

Constructs a **PreparedRequest** for transmission and returns it. The **PreparedRequest** has settings merged from the **Request** instance and those of the **Session**.

**Parameters:** **request** – **Request** instance to prepare with this session's settings.

**Return type:** [requests.PreparedRequest](#)

## proxies

Dictionary mapping protocol or protocol and host to the URL of the proxy (e.g. {'http': 'foo.bar:3128', 'http://host.name': 'foo.bar:4012'}) to be used on each **Request**.

**put**(*url*, *data=None*, *\*\*kwargs*) [source]

Sends a PUT request. Returns **Response** object.

- Parameters:**
- **url** – URL for the new **Request** object.
  - **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the **Request**.
  - **\*\*kwargs** – Optional arguments that `request` takes.

**Return type:** [requests.Response](#)

**rebuild\_auth**(*prepared\_request*, *response*)

When being redirected we may want to strip authentication from the request to avoid leaking credentials. This method intelligently removes and reapplies authentication where possible to avoid credential loss.

**rebuild\_method**(*prepared\_request*, *response*)

When being redirected we may want to change the method of the request based on certain specs or browser behavior.

**rebuild\_proxies**(*prepared\_request*, *proxies*)

This method re-evaluates the proxy configuration by considering the environment variables. If we are redirected to a URL covered by NO\_PROXY, we strip the proxy configuration. Otherwise, we set missing proxy keys for this URL (in case they were stripped by a previous redirect).

This method also replaces the Proxy-Authorization header where necessary.

**Return type:** [dict](#)

**request**(*method*, *url*, *params=None*, *data=None*, *headers=None*, *cookies=None*, *files=None*, *auth=None*, *timeout=None*, *allow\_redirects=True*, *proxies=None*, *hooks=None*, *stream=None*, *verify=None*, *cert=None*, *json=None*) [source]

Constructs a **Request**, prepares it and sends it. Returns **Response** object.

- Parameters:**
- **method** – method for the new **Request** object.
  - **url** – URL for the new **Request** object.
  - **params** – (optional) Dictionary or bytes to be sent in the query string for the **Request**.
  - **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the **Request**.
  - **json** – (optional) json to send in the body of the **Request**.
  - **headers** – (optional) Dictionary of HTTP Headers to send with the **Request**.
  - **cookies** – (optional) Dict or CookieJar object to send with the **Request**.

- **files** – (optional) Dictionary of 'filename': file-like-objects for multi-part encoding upload.
- **auth** – (optional) Auth tuple or callable to enable Basic/Digest/Custom HTTP Auth.
- **timeout** (*float or tuple*) – (optional) How long to wait for the server to send data before giving up, as a float, or a (*connect timeout, read timeout*) tuple.
- **allow\_redirects** (*bool*) – (optional) Set to True by default.
- **proxies** – (optional) Dictionary mapping protocol or protocol and hostname to the URL of the proxy.
- **hooks** – (optional) Dictionary mapping hook name to one event or list of events, event must be callable.
- **stream** – (optional) whether to immediately download the response content. Defaults to **False**.
- **verify** – (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use. Defaults to **True**. When set to **False**, requests will accept any TLS certificate presented by the server, and will ignore hostname mismatches and/or expired certificates, which will make your application vulnerable to man-in-the-middle (MitM) attacks. Setting verify to **False** may be useful during local development or testing.
- **cert** – (optional) if String, path to ssl client cert file (.pem). If Tuple, ('cert', 'key') pair.

**Return type:** [requests.Response](#)

**resolve\_redirects**(*resp, req, stream=False, timeout=None, verify=True, cert=None, proxies=None, yield\_requests=False, \*\*adapter\_kwargs*)

Receives a Response. Returns a generator of Responses or Requests.

**send**(*request, \*\*kwargs*)

[\[source\]](#)

Send a given PreparedRequest.

**Return type:** [requests.Response](#)

**should\_strip\_auth**(*old\_url, new\_url*)

Decide whether Authorization header should be removed when redirecting

**stream**

Stream response content default.

**trust\_env**

Trust environment settings for proxy configuration, default authentication and similar.

**verify**

SSL Verification default. Defaults to *True*, requiring requests to verify the TLS certificate at the remote end. If verify is set to *False*, requests will accept any TLS certificate presented by the server, and will ignore hostname mismatches and/or expired certificates, which will make your application vulnerable to man-in-the-middle (MitM) attacks. Only set this to *False* for testing.

## Lower-Level Classes

*class* requests.**Request**(*method=None, url=None, headers=None, files=None, data=None, params=None, auth=None, cookies=None, hooks=None, json=None*)

A user-created **Request** object.

[\[source\]](#)

Used to prepare a **PreparedRequest**, which is sent to the server.

**Parameters:**

- **method** – HTTP method to use.

- **url** – URL to send.

- **headers** – dictionary of headers to send.

- **files** – dictionary of {filename: fileobject} files to multipart upload.

- **data** – the body to attach to the request. If a dictionary or list of tuples [(key, value)] is provided, form-encoding will take place.

- **json** – json for the body to attach to the request (if files or data is not specified).

- **params** – URL parameters to append to the URL. If a dictionary or list of tuples `[(key, value)]` is provided, form-encoding will take place.
- **auth** – Auth handler or (user, pass) tuple.
- **cookies** – dictionary or CookieJar of cookies to attach to this request.
- **hooks** – dictionary of callback hooks, for internal usage.

Usage:

```
>>> import requests
>>> req = requests.Request('GET', 'https://httpbin.org/get')
>>> req.prepare()
<PreparedRequest [GET]>
```

**deregister\_hook(event, hook)**

Deregister a previously registered hook. Returns True if the hook existed, False if not.

**prepare()**

[\[source\]](#)

Constructs a **PreparedRequest** for transmission and returns it.

**register\_hook(event, hook)**

Properly register a hook.

**class requests.Response**

[\[source\]](#)

The **Response** object, which contains a server's response to an HTTP request.

**property apparent\_encoding**

The apparent encoding, provided by the `charset_normalizer` or `chardet` libraries.

**close()**

[\[source\]](#)

Releases the connection back to the pool. Once this method has been called the underlying `raw` object must not be accessed again.

*Note: Should not normally need to be called explicitly.*

**property content**

Content of the response, in bytes.

**cookies**

A CookieJar of Cookies the server sent back.

**elapsed**

The amount of time elapsed between sending the request and the arrival of the response (as a `timedelta`). This property specifically measures the time taken between sending the first byte of the request and finishing parsing the headers. It is therefore unaffected by consuming the response content or the value of the `stream` keyword argument.

**encoding**

Encoding to decode with when accessing `r.text`.

**headers**

Case-insensitive Dictionary of Response Headers. For example, `headers['content-encoding']` will return the value of a `'Content-Encoding'` response header.

**history**

A list of **Response** objects from the history of the Request. Any redirect responses will end up here. The list is sorted from the oldest to the most recent request.

**property is\_permanent\_redirect**

True if this Response one of the permanent versions of redirect.

**property is\_redirect**

True if this Response is a well-formed HTTP redirect that could have been processed automatically (by `Session.resolve_redirects`).

**iter\_content(chunk\_size=1, decode\_unicode=False)**

[\[source\]](#)



Iterates over the response data. When `stream=True` is set on the request, this avoids reading the content at once into memory for large responses. The chunk size is the number of bytes it should read into memory. This is not necessarily the length of each item returned as decoding can take place.

`chunk_size` must be of type `int` or `None`. A value of `None` will function differently depending on the value of `stream`. `stream=True` will read data as it arrives in whatever size the chunks are received. If `stream=False`, data is returned as a single chunk.

If `decode_unicode` is `True`, content will be decoded using the best available encoding based on the response.

**`iter_lines(chunk_size=512, decode_unicode=False, delimiter=None)`** [\[source\]](#)

Iterates over the response data, one line at a time. When `stream=True` is set on the request, this avoids reading the content at once into memory for large responses.

### Note:

This method is not reentrant safe.

**`json(**kwargs)`** [\[source\]](#)

Returns the json-encoded content of a response, if any.

**Parameters:** `**kwargs` – Optional arguments that `json.loads` takes.

**Raises:** [requests.exceptions.JSONDecodeError](#) – If the response body does not contain valid json.

**`property links`**

Returns the parsed header links of the response, if any.

**`property next`**

Returns a `PreparedRequest` for the next request in a redirect chain, if there is one.

**`property ok`**

Returns `True` if `status_code` is less than 400, `False` if not.

This attribute checks if the status code of the response is between 400 and 600 to see if there was a client error or a server error. If the status code is between 200 and 400, this will return `True`. This is **not** a check to see if the response code is `200 OK`.

**`raise_for_status()`** [\[source\]](#)

Raises `HTTPError`, if one occurred.

**`raw`**

File-like object representation of response (for advanced usage). Use of `raw` requires that `stream=True` be set on the request. This requirement does not apply for use internally to Requests.

**`reason`**

Textual reason of responded HTTP Status, e.g. “Not Found” or “OK”.

**`request`**

The `PreparedRequest` object to which this is a response.

**`status_code`**

Integer Code of responded HTTP Status, e.g. 404 or 200.

**`property text`**

Content of the response, in unicode.

If `Response.encoding` is `None`, encoding will be guessed using `charset_normalizer` or `chardet`.

The encoding of the response content is determined based solely on HTTP headers, following RFC 2616 to the letter. If you can take advantage of non-HTTP knowledge to make a better guess at the



encoding, you should set `r.encoding` appropriately before accessing this property.

## **url**

Final URL location of Response.

# Lower-Lower-Level Classes

## `class requests.PreparedRequest`

[\[source\]](#)

The fully mutable **PreparedRequest** object, containing the exact bytes that will be sent to the server.

Instances are generated from a **Request** object, and should not be instantiated manually; doing so may produce undesirable effects.

Usage:

```
>>> import requests
>>> req = requests.Request('GET', 'https://httpbin.org/get')
>>> r = req.prepare()
>>> r
<PreparedRequest [GET]>

>>> s = requests.Session()
>>> s.send(r)
<Response [200]>
```

## **body**

request body to send to the server.

## **deregister\_hook(event, hook)**

Deregister a previously registered hook. Returns True if the hook existed, False if not.

## **headers**

dictionary of HTTP headers.

## **hooks**

dictionary of callback hooks, for internal usage.

## **method**

HTTP verb to send to the server.

## *property* **path\_url**

Build the path URL to use.

**prepare(method=None, url=None, headers=None, files=None, data=None, params=None, auth=None, cookies=None, hooks=None, json=None)** [\[source\]](#)

Prepares the entire request with the given parameters.

**prepare\_auth(auth, url='')** [\[source\]](#)

Prepares the given HTTP auth data.

**prepare\_body(data, files, json=None)** [\[source\]](#)

Prepares the given HTTP body data.

**prepare\_content\_length(body)** [\[source\]](#)

Prepare Content-Length header based on request method and body

**prepare\_cookies(cookies)** [\[source\]](#)

Prepares the given HTTP cookie data.

This function eventually generates a `Cookie` header from the given cookies using `cookiecrlib`. Due to `cookiecrlib`'s design, the header will not be regenerated if it already exists, meaning this function can only be called once for the life of the **PreparedRequest** object. Any subsequent calls to `prepare_cookies` will have no actual effect, unless the “Cookie” header is removed beforehand.

**prepare\_headers(headers)** [\[source\]](#)

Prepares the given HTTP headers.

**prepare\_hooks**(*hooks*) [\[source\]](#)

Prepares the given hooks.

**prepare\_method**(*method*) [\[source\]](#)

Prepares the given HTTP method.

**prepare\_url**(*url*, *params*) [\[source\]](#)

Prepares the given HTTP URL.

**register\_hook**(*event*, *hook*)

Properly register a hook.

**url**

HTTP URL to send the request to.

**class** requests.adapters.**BaseAdapter** [\[source\]](#)

The Base Transport Adapter

**close**() [\[source\]](#)

Cleans up adapter specific items.

**send**(*request*, *stream=False*, *timeout=None*, *verify=True*, *cert=None*, *proxies=None*) [\[source\]](#)

Sends PreparedRequest object. Returns Response object.

- Parameters:**
- **request** – The **PreparedRequest** being sent.
  - **stream** – (optional) Whether to stream the request content.
  - **timeout** (*float or tuple*) – (optional) How long to wait for the server to send data before giving up, as a float, or a ([connect timeout](#), [read timeout](#)) tuple.
  - **verify** – (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use
  - **cert** – (optional) Any user-provided SSL certificate to be trusted.
  - **proxies** – (optional) The proxies dictionary to apply to the request.

**class** requests.adapters.**HTTPAdapter**(*pool\_connections=10*, *pool\_maxsize=10*, *max\_retries=0*, *pool\_block=False*) [\[source\]](#)

The built-in HTTP Adapter for urllib3.

Provides a general-case interface for Requests sessions to contact HTTP and HTTPS urls by implementing the Transport Adapter interface. This class will usually be created by the **Session** class under the covers.

- Parameters:**
- **pool\_connections** – The number of urllib3 connection pools to cache.
  - **pool\_maxsize** – The maximum number of connections to save in the pool.
  - **max\_retries** – The maximum number of retries each connection should attempt. Note, this applies only to failed DNS lookups, socket connections and connection timeouts, never to requests where data has made it to the server. By default, Requests does not retry failed connections. If you need granular control over the conditions under which we retry a request, import urllib3's **Retry** class and pass that instead.
  - **pool\_block** – Whether the connection pool should block for connections.

Usage:

```
>>> import requests
>>> s = requests.Session()
>>> a = requests.adapters.HTTPAdapter(max_retries=3)
>>> s.mount('http://', a)
```

**add\_headers**(*request*, *\*\*kwargs*) [\[source\]](#)

Add any headers needed by the connection. As of v2.0 this does nothing by default, but is left for overriding by users that subclass the **HTTPAdapter**.

This should not be called from user code, and is only exposed for use when subclassing the **HTTPAdapter**.

- Parameters:**
- **request** – The **PreparedRequest** to add headers to.
  - **kwargs** – The keyword arguments from the call to `send()`.

**build\_connection\_pool\_key\_attributes**(*request, verify, cert=None*)

Build the PoolKey attributes used by urllib3 to return a connection. [\[source\]](#)

This looks at the PreparedRequest, the user-specified verify value, and the value of the cert parameter to determine what PoolKey values to use to select a connection from a given urllib3 Connection Pool.

The SSL related pool key arguments are not consistently set. As of this writing, use the following to determine what keys may be in that dictionary:

- If `verify` is `True`, `"ssl_context"` will be set and will be the default Requests SSL Context
- If `verify` is `False`, `"ssl_context"` will not be set but `"cert_reqs"` will be set
- If `verify` is a string, (i.e., it is a user-specified trust bundle) `"ca_certs"` will be set if the string is not a directory recognized by `os.path.isdir`, otherwise `"ca_certs_dir"` will be set.
- If `"cert"` is specified, `"cert_file"` will always be set. If `"cert"` is a tuple with a second item, `"key_file"` will also be present

To override these settings, one may subclass this class, call this method and use the above logic to change parameters as desired. For example, if one wishes to use a custom `ssl.SSLContext` one must both set `"ssl_context"` and based on what else they require, alter the other keys to ensure the desired behaviour.

- Parameters:**
- **request** (**PreparedRequest**) – The PreparedRequest being sent over the connection.
  - **verify** – Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use.
  - **cert** – (optional) Any user-provided SSL certificate for client authentication (a.k.a., mTLS). This may be a string (i.e., just the path to a file which holds both certificate and key) or a tuple of length 2 with the certificate file path and key file path.

**Returns:** A tuple of two dictionaries. The first is the "host parameters" portion of the Pool Key including scheme, hostname, and port. The second is a dictionary of SSLContext related parameters.

**build\_response**(*req, resp*) [\[source\]](#)

Builds a **Response** object from a urllib3 response. This should not be called from user code, and is only exposed for use when subclassing the **HTTPAdapter**


- Parameters:**
- **req** – The **PreparedRequest** used to generate the response.
  - **resp** – The urllib3 response object.

**Return type:** [requests.Response](#)

**cert\_verify**(*conn, url, verify, cert*) [\[source\]](#)

Verify a SSL certificate. This method should not be called from user code, and is only exposed for use when subclassing the **HTTPAdapter**.

- Parameters:**
- **conn** – The urllib3 connection object associated with the cert.
  - **url** – The requested URL.
  - **verify** – Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use
  - **cert** – The SSL certificate to verify.

 v: latest ▾

**close**() [\[source\]](#)

Disposes of any internal state.

Currently, this closes the PoolManager and any active ProxyManager, which closes any pooled connections.

**get\_connection**(*url*, *proxies=None*) [\[source\]](#)

DEPRECATED: Users should move to *get\_connection\_with\_tls\_context* for all subclasses of HTTPAdapter using Requests>=2.3.2.2.

Returns a urllib3 connection for the given URL. This should not be called from user code, and is only exposed for use when subclassing the **HTTPAdapter**.

**Parameters:**

- **url** – The URL to connect to.
- **proxies** – (optional) A Requests-style dictionary of proxies used on this request.

**Return type:** urllib3.ConnectionPool

**get\_connection\_with\_tls\_context**(*request*, *verify*, *proxies=None*, *cert=None*) [\[source\]](#)

Returns a urllib3 connection for the given request and TLS settings. This should not be called from user code, and is only exposed for use when subclassing the **HTTPAdapter**.

**Parameters:**

- **request** – The **PreparedRequest** object to be sent over the connection.
- **verify** – Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use.
- **proxies** – (optional) The proxies dictionary to apply to the request.
- **cert** – (optional) Any user-provided SSL certificate to be used for client authentication (a.k.a., mTLS).

**Return type:** urllib3.ConnectionPool

**init\_poolmanager**(*connections*, *maxsize*, *block=False*, *\*\*pool\_kwargs*) [\[source\]](#)

Initializes a urllib3 PoolManager.

This method should not be called from user code, and is only exposed for use when subclassing the **HTTPAdapter**.

**Parameters:**

- **connections** – The number of urllib3 connection pools to cache.
- **maxsize** – The maximum number of connections to save in the pool.
- **block** – Block when no free connections are available.
- **pool\_kwargs** – Extra keyword arguments used to initialize the Pool Manager.

**proxy\_headers**(*proxy*) [\[source\]](#)

Returns a dictionary of the headers to add to any request sent through a proxy. This works with urllib3 magic to ensure that they are correctly sent to the proxy, rather than in a tunnelled request if CONNECT is being used.

This should not be called from user code, and is only exposed for use when subclassing the **HTTPAdapter**.

**Parameters:** **proxy** – The url of the proxy being used for this request.

**Return type:** [dict](#)

**proxy\_manager\_for**(*proxy*, *\*\*proxy\_kwargs*) [\[source\]](#)

Return urllib3 ProxyManager for the given proxy.

This method should not be called from user code, and is only exposed for use when subclassing the **HTTPAdapter**.

**Parameters:**

- **proxy** – The proxy to return a urllib3 ProxyManager for.
- **proxy\_kwargs** – Extra keyword arguments used to configure the Proxy Manager.

**Returns:** ProxyManager

**Return type:** [urllib3.ProxyManager](#)

**request\_url**(*request*, *proxies*) [\[source\]](#)

Obtain the url to use when making the final request.

If the message is being sent through a HTTP proxy, the full URL has to be used. Otherwise, we should only use the path portion of the URL.

This should not be called from user code, and is only exposed for use when subclassing the **HTTPAdapter**.

**Parameters:**

- **request** – The **PreparedRequest** being sent.
- **proxies** – A dictionary of schemes or schemes and hosts to proxy URLs.

**Return type:** [str](#)

**send**(*request*, *stream=False*, *timeout=None*, *verify=True*, *cert=None*, *proxies=None*) [\[source\]](#)

Sends PreparedRequest object. Returns Response object.

**Parameters:**

- **request** – The **PreparedRequest** being sent.
- **stream** – (optional) Whether to stream the request content.
- **timeout** ([float](#) or [tuple](#) or [urllib3 Timeout object](#)) – (optional) How long to wait for the server to send data before giving up, as a float, or a ([connect timeout](#), [read timeout](#)) tuple.
- **verify** – (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use
- **cert** – (optional) Any user-provided SSL certificate to be trusted.
- **proxies** – (optional) The proxies dictionary to apply to the request.

**Return type:** [requests.Response](#)

## Authentication

**class** requests.auth.**AuthBase** [\[source\]](#)

Base class that all auth implementations derive from

**class** requests.auth.**HTTPBasicAuth**(*username*, *password*) [\[source\]](#)

Attaches HTTP Basic Authentication to the given Request object.

**class** requests.auth.**HTTPProxyAuth**(*username*, *password*) [\[source\]](#)

Attaches HTTP Proxy Authentication to a given Request object.

**class** requests.auth.**HTTPDigestAuth**(*username*, *password*) [\[source\]](#)

Attaches HTTP Digest Authentication to the given Request object.

## Encodings

requests.utils.**get\_encodings\_from\_content**(*content*) [\[source\]](#)

Returns encodings from given content string.

**Parameters:** **content** – bytestring to extract encodings from.

requests.utils.**get\_encoding\_from\_headers**(*headers*) [\[source\]](#)

Returns encodings from given HTTP Header Dict.

**Parameters:** **headers** – dictionary to extract encoding from.

**Return type:** [str](#)

requests.utils.**get\_unicode\_from\_response**(*r*) [\[source\]](#)

Returns the requested content back in unicode.

**Parameters:** **r** – Response object to get unicode content from.

Tried:

1. charset from content-type

2. fall back and replace all unicode characters

**Return type:** [str](#)

## Cookies

`requests.utils.dict_from_cookiejar(cj)`

[\[source\]](#)

Returns a key/value dictionary from a CookieJar.

**Parameters:** `cj` – CookieJar object to extract cookies from.

**Return type:** [dict](#)

`requests.utils.add_dict_to_cookiejar(cj, cookie_dict)`

[\[source\]](#)

Returns a CookieJar from a key/value dictionary.

**Parameters:** • `cj` – CookieJar to insert cookies into.

• `cookie_dict` – Dict of key/values to insert into CookieJar.

**Return type:** CookieJar

`requests.cookies.cookiejar_from_dict(cookie_dict, cookiejar=None, overwrite=True)`

[\[source\]](#)

Returns a CookieJar from a key/value dictionary.

**Parameters:** • `cookie_dict` – Dict of key/values to insert into CookieJar.

• `cookiejar` – (optional) A cookiejar to add the cookies to.

• `overwrite` – (optional) If False, will not replace cookies already in the jar with new ones.

**Return type:** CookieJar

`class requests.cookies.RequestsCookieJar(policy=None)`

[\[source\]](#)

Compatibility class; is a `http.cookiejar.CookieJar`, but exposes a dict interface.

This is the CookieJar we create by default for requests and sessions that don't specify one, since some clients may expect `response.cookies` and `session.cookies` to support dict operations.

Requests does not use the dict interface internally; it's just for compatibility with external client code. All requests code should work out of the box with externally provided instances of `CookieJar`, e.g. `LWPCookieJar` and `FileCookieJar`.

Unlike a regular `CookieJar`, this class is pickleable.

### Warning:

dictionary operations that are normally O(1) may be O(n).

`add_cookie_header(request)`

Add correct Cookie: header to request (`urllib.request.Request` object).

The Cookie2 header is also added unless `policy.hide_cookie2` is true.

`clear(domain=None, path=None, name=None)`

Clear some cookies.

Invoking this method without arguments will clear all cookies. If given a single argument, only cookies belonging to that domain will be removed. If given two arguments, cookies belonging to the specified path within that domain are removed. If given three arguments, then the cookie with the specified name, path and domain is removed.

Raises `KeyError` if no matching cookie exists.

`clear_expired_cookies()`

Discard all expired cookies.

You probably don't need to call this method: expired cookies are never sent back to the server (provided you're using `DefaultCookiePolicy`), this method is called by `CookieJar` itself every so of-

ten, and the `.save()` method won't save expired cookies anyway (unless you ask otherwise by passing a true `ignore_expires` argument).

### **`clear_session_cookies()`**

Discard all session cookies.

Note that the `.save()` method won't save session cookies anyway, unless you ask otherwise by passing a true `ignore_discard` argument.

### **`copy()`**

[\[source\]](#)

Return a copy of this `RequestsCookieJar`.

### **`extract_cookies(response, request)`**

Extract cookies from response, where allowable given the request.

### **`get(name, default=None, domain=None, path=None)`**

[\[source\]](#)

Dict-like `get()` that also supports optional domain and path args in order to resolve naming collisions from using one cookie jar over multiple domains.

#### **Warning:**

operation is  $O(n)$ , not  $O(1)$ .

### **`get_dict(domain=None, path=None)`**

[\[source\]](#)

Takes as an argument an optional domain and path and returns a plain old Python dict of name-value pairs of cookies that meet the requirements.

**Return type:** [dict](#)

### **`get_policy()`**

[\[source\]](#)

Return the `CookiePolicy` instance used.

### **`items()`**

[\[source\]](#)

Dict-like `items()` that returns a list of name-value tuples from the jar. Allows client-code to call `dict(RequestsCookieJar)` and get a vanilla python dict of key value pairs.

#### **See also:**

`keys()` and `values()`.

### **`iteritems()`**

[\[source\]](#)

Dict-like `iteritems()` that returns an iterator of name-value tuples from the jar.

#### **See also:**

`iterkeys()` and `itervalues()`.

### **`iterkeys()`**

[\[source\]](#)

Dict-like `iterkeys()` that returns an iterator of names of cookies from the jar.

#### **See also:**

`itervalues()` and `iteritems()`.

### **`itervalues()`**

[\[source\]](#)

Dict-like `itervalues()` that returns an iterator of values of cookies from the jar.

#### **See also:**

`iterkeys()` and `iteritems()`.



**keys()** [\[source\]](#)

Dict-like keys() that returns a list of names of cookies from the jar.

**See also:**

values() and items().

**list\_domains()** [\[source\]](#)

Utility method to list all the domains in the jar.

**list\_paths()** [\[source\]](#)

Utility method to list all the paths in the jar.

**make\_cookies(response, request)**

Return sequence of Cookie objects extracted from response object.

**multiple\_domains()** [\[source\]](#)

Returns True if there are multiple domains in the jar. Returns False otherwise.

**Return type:** [bool](#)

**pop(k[, d])** → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised.

**popitem()** → (k, v), remove and return some (key, value) pair as a 2-tuple; but raise KeyError if D is empty.

**set(name, value, \*\*kwargs)** [\[source\]](#)

Dict-like set() that also supports optional domain and path args in order to resolve naming collisions from using one cookie jar over multiple domains.

**set\_cookie(cookie, \*args, \*\*kwargs)** [\[source\]](#)

Set a cookie, without checking whether or not it should be set.

**set\_cookie\_if\_ok(cookie, request)**

Set a cookie if policy says it's OK to do so.

**setdefault(k[, d])** → D.get(k,d), also set D[k]=d if k not in D

**update(other)** [\[source\]](#)

Updates this jar with cookies from another CookieJar or dict-like

**values()** [\[source\]](#)

Dict-like values() that returns a list of values of cookies from the jar.

**See also:**

keys() and items().

**class requests.cookies.CookieConflictError** [\[source\]](#)

There are two cookies that meet the criteria specified in the cookie jar. Use .get and .set and include domain and path args in order to be more specific.

**add\_note()**

Exception.add\_note(note) – add a note to the exception

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

## Status Code Lookup

requests.codes

alias of {}

The `codes` object defines a mapping from common names for HTTP statuses to their numerical codes, accessible either as attributes or as dictionary items.

Example:

```
>>> import requests
>>> requests.codes['temporary_redirect']
307
>>> requests.codes.teapot
418
>>> requests.codes['\o/']
200
```

Some codes have multiple names, and both upper- and lower-case versions of the names are allowed. For example, `codes.ok`, `codes.OK`, and `codes.okay` all correspond to the HTTP status code 200.

- 100: `continue`
- 101: `switching_protocols`
- 102: `processing`, `early_hints`
- 103: `checkpoint`
- 122: `uri_too_long`, `request_uri_too_long`
- 200: `ok`, `okay`, `all_ok`, `all_okay`, `all_good`, `\o/`, `✓`
- 201: `created`
- 202: `accepted`
- 203: `non_authoritative_info`, `non_authoritative_information`
- 204: `no_content`
- 205: `reset_content`, `reset`
- 206: `partial_content`, `partial`
- 207: `multi_status`, `multiple_status`, `multi_stati`, `multiple_stati`
- 208: `already_reported`
- 226: `im_used`
- 300: `multiple_choices`
- 301: `moved_permanently`, `moved`, `\o-`
- 302: `found`
- 303: `see_other`, `other`
- 304: `not_modified`
- 305: `use_proxy`
- 306: `switch_proxy`
- 307: `temporary_redirect`, `temporary_moved`, `temporary`
- 308: `permanent_redirect`, `resume_incomplete`, `resume`
- 400: `bad_request`, `bad`
- 401: `unauthorized`
- 402: `payment_required`, `payment`
- 403: `forbidden`
- 404: `not_found`, `-o-`
- 405: `method_not_allowed`, `not_allowed`
- 406: `not_acceptable`
- 407: `proxy_authentication_required`, `proxy_auth`, `proxy_authentication`
- 408: `request_timeout`, `timeout`
- 409: `conflict`
- 410: `gone`
- 411: `length_required`
- 412: `precondition_failed`, `precondition`
- 413: `request_entity_too_large`, `content_too_large`
- 414: `request_uri_too_large`, `uri_too_long`
- 415: `unsupported_media_type`, `unsupported_media`, `media_type`
- 416: `requested_range_not_satisfiable`, `requested_range`, `range_not_satisfiable`
- 417: `expectation_failed`
- 418: `im_a_teapot`, `teapot`, `i_am_a_teapot`

- 421: `misdirected_request`
- 422: `unprocessable_entity`, `unprocessable`, `unprocessable_content`
- 423: `locked`
- 424: `failed_dependency`, `dependency`
- 425: `unordered_collection`, `unordered`, `too_early`
- 426: `upgrade_required`, `upgrade`
- 428: `precondition_required`, `precondition`
- 429: `too_many_requests`, `too_many`
- 431: `header_fields_too_large`, `fields_too_large`
- 444: `no_response`, `none`
- 449: `retry_with`, `retry`
- 450: `blocked_by_windows_parental_controls`, `parental_controls`
- 451: `unavailable_for_legal_reasons`, `legal_reasons`
- 499: `client_closed_request`
- 500: `internal_server_error`, `server_error`, `/o\`, `x`
- 501: `not_implemented`
- 502: `bad_gateway`
- 503: `service_unavailable`, `unavailable`
- 504: `gateway_timeout`
- 505: `http_version_not_supported`, `http_version`
- 506: `variant_also_negotiates`
- 507: `insufficient_storage`
- 509: `bandwidth_limit_exceeded`, `bandwidth`
- 510: `not_extended`
- 511: `network_authentication_required`, `network_auth`, `network_authentication`

## Migrating to 1.x

This section details the main differences between 0.x and 1.x and is meant to ease the pain of upgrading.

### API Changes

- `Response.json` is now a callable and not a property of a response.

```
import requests
r = requests.get('https://api.github.com/events')
r.json() # This *call* raises an exception if JSON decoding fails
```

- The `Session` API has changed. `Sessions` objects no longer take parameters. `Session` is also now capitalized, but it can still be instantiated with a lowercase `session` for backwards compatibility.

```
s = requests.Session() # formerly, session took parameters
s.auth = auth
s.headers.update(headers)
r = s.get('https://httpbin.org/headers')
```

- All request hooks have been removed except ‘response’.
- Authentication helpers have been broken out into separate modules. See [requests-oauthlib](#) and [requests-kerberos](#).
- The parameter for streaming requests was changed from `prefetch` to `stream` and the logic was inverted. In addition, `stream` is now required for raw response reading.

```
# in 0.x, passing prefetch=False would accomplish the same thing
r = requests.get('https://api.github.com/events', stream=True)
for chunk in r.iter_content(8192):
    ...
```

- The `config` parameter to the `requests` method has been removed. Some of these options are now configured on a `Session` such as keep-alive and maximum number of redirects. The verbosity option

should be handled by configuring logging.

```
import requests
import logging

# Enabling debugging at http.client level (requests->urllib3->http.client)
# you will see the REQUEST, including HEADERS and DATA, and RESPONSE with HEADERS
# the only thing missing will be the response.body which is not logged.
try: # for Python 3
    from http.client import HTTPConnection
except ImportError:
    from httplib import HTTPConnection
HTTPConnection.debuglevel = 1

logging.basicConfig() # you need to initialize logging, otherwise you will not see
logging.getLogger().setLevel(logging.DEBUG)
requests_log = logging.getLogger("urllib3")
requests_log.setLevel(logging.DEBUG)
requests_log.propagate = True

requests.get('https://httpbin.org/headers')
```

## Licensing

One key difference that has nothing to do with the API is a change in the license from the [ISC](#) license to the [Apache 2.0](#) license. The Apache 2.0 license ensures that contributions to Requests are also covered by the Apache 2.0 license.

## Migrating to 2.x

Compared with the 1.0 release, there were relatively few backwards incompatible changes, but there are still a few issues to be aware of with this major release.

For more details on the changes in this release including new APIs, links to the relevant GitHub issues and some of the bug fixes, read Cory's [blog](#) on the subject.

## API Changes

- There were a couple changes to how Requests handles exceptions. `RequestException` is now a subclass of `IOError` rather than `RuntimeError` as that more accurately categorizes the type of error. In addition, an invalid URL escape sequence now raises a subclass of `RequestException` rather than a `ValueError`.

```
requests.get('http://%zz/') # raises requests.exceptions.InvalidURL
```

Lastly, `httplib.IncompleteRead` exceptions caused by incorrect chunked encoding will now raise a `Requests ChunkedEncodingError` instead.

- The proxy API has changed slightly. The scheme for a proxy URL is now required.

```
proxies = {
    "http": "10.10.1.10:3128", # use http://10.10.1.10:3128 instead
}

# In requests 1.x, this was legal, in requests 2.x,
# this raises requests.exceptions.MissingSchema
requests.get("http://example.org", proxies=proxies)
```

## Behavioural Changes

- Keys in the `headers` dictionary are now native strings on all Python versions, i.e. bytestrings on Python 2 and unicode on Python 3. If the keys are not native strings (unicode on Python 2 or bytestrings on Python 3) they will be converted to the native string type assuming UTF-8 encoding.

- Values in the `headers` dictionary should always be strings. This has been the project's position since before 1.0 but a recent change (since version 2.11.0) enforces this more strictly. It's advised to avoid passing header values as unicode when possible.

```
# .readthedocs.yaml
build.tools:
  python: "3.11"
sphinx:
  configuration: conf.py
python.install:
  - requirements: pip.in
```

**Docs as Code hosting** is easy on Read the Docs. Free for open source, paid for business.  
**Sign up today.**

Ad by EthicalAds · 