21.21. `cookielib` — Cookie handling for HTTP clients¶

Note

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

New in version 2.4.

The `cookielib` module defines classes for automatic handling of HTTP cookies. It is useful for accessing web sites that require small pieces of data – *cookies* – to be set on the client machine by an HTTP response from a web server, and then returned to the server in later HTTP requests.

Both the regular Netscape cookie protocol and the protocol defined by **RFC 2965** are handled. RFC 2965 handling is switched off by default. **RFC 2109** cookies are parsed as Netscape cookies and subsequently treated either as Netscape or RFC 2965 cookies according to the 'policy' in effect. Note that the great majority of cookies on the Internet are Netscape cookies. `cookielib` attempts to follow the de-facto Netscape cookie protocol (which differs substantially from that set out in the original Netscape specification), including taking note of the `max-age` and `port` cookie-attributes introduced with RFC 2965.

Note

The various named parameters found in *Set-Cookie* and *Set-Cookie2* headers (eg. `domain` and `expires`) are conventionally referred to as *attributes*. To distinguish them from Python attributes, the documentation for this module uses the term *cookie-attribute* instead.

The module defines the following exception:

exception `cookielib.LoadError`¶

Instances of `FileCookieJar` raise this exception on failure to load cookies from a file.

Note

For backwards-compatibility with Python 2.4 (which raised an `IOError`), `LoadError` is a subclass of `IOError`.

The following classes are provided:

class `cookielib.CookieJar`(*policy=None*)¶

policy is an object implementing the `CookiePolicy` interface.

The `CookieJar` class stores HTTP cookies. It extracts cookies from HTTP requests, and returns them in HTTP responses. `CookieJar` instances automatically expire contained cookies when necessary. Subclasses are also responsible for storing and retrieving cookies from a file or database.

class `cookielib.FileCookieJar`(*filename*, *delayload=None*, *policy=None*)¶

policy is an object implementing the `CookiePolicy` interface. For the other arguments, see the documentation for the corresponding attributes.

A `CookieJar` which can load cookies from, and perhaps save cookies to, a file on disk. Cookies are **NOT** loaded from the named file until either the `load()` or `revert()` method is called. Subclasses of this class are documented in section *FileCookieJar subclasses and co-operation with web browsers*.

class `cookielib.CookiePolicy`¶

This class is responsible for deciding whether each cookie should be accepted from / returned to the server.

class `cookielib.DefaultCookiePolicy`(*blocked_domains=None*, *allowed_domains=None*, *netscape=True*, *rfc2965=False*, *rfc2109_as_netscape=None*, *hide_cookie2=False*, *strict_domain=False*, *strict_rfc2965_unverifiable=True*, *strict_ns_unverifiable=False*, *strict_ns_domain=DefaultCookiePolicy.DomainLiberal*, *strict_ns_set_initial_dollar=False*, *strict_ns_set_path=False*)¶

Constructor arguments should be passed as keyword arguments only. *blocked_domains* is a sequence of domain names that we never accept cookies from, nor return cookies to. *allowed_domains* if not `None`, this is a sequence of the only domains for which we accept and return cookies. For all other arguments, see the documentation for `CookiePolicy` and `DefaultCookiePolicy` objects.

`DefaultCookiePolicy` implements the standard accept / reject rules for Netscape and RFC 2965 cookies. By default, RFC 2109 cookies (ie. cookies received in a *Set-Cookie* header with a version cookie-attribute of 1) are treated according to the RFC 2965 rules. However, if RFC 2965 handling is turned off or

`rfc2109_as_netscape` is True, RFC 2109 cookies are 'downgraded' by the `CookieJar` instance to Netscape cookies, by setting the `version` attribute of the `Cookie` instance to 0. `DefaultCookiePolicy` also provides some parameters to allow some fine-tuning of policy.

class `cookielib.Cookie`¶

This class represents Netscape, RFC 2109 and RFC 2965 cookies. It is not expected that users of `cookielib` construct their own `Cookie` instances. Instead, if necessary, call `make_cookies()` on a `CookieJar` instance.

See also

Module `urllib2`

URL opening with automatic cookie handling.

Module `Cookie`

HTTP cookie classes, principally useful for server-side code. The `cookielib` and `Cookie` modules do not depend on each other.

http://wwwsearch.sf.net/ClientCookie/

Extensions to this module, including a class for reading Microsoft Internet Explorer cookies on Windows.

http://wp.netscape.com/newsref/std/cookie_spec.html

The specification of the original Netscape cookie protocol. Though this is still the dominant protocol, the 'Netscape cookie protocol' implemented by all the major browsers (and `cookielib`) only bears a passing resemblance to the one sketched out in `cookie_spec.html`.

RFC 2109 - HTTP State Management Mechanism

Obsoleted by RFC 2965. Uses *Set-Cookie* with version=1.

RFC 2965 - HTTP State Management Mechanism

The Netscape protocol with the bugs fixed. Uses *Set-Cookie2* in place of *Set-Cookie*. Not widely used.

http://kristol.org/cookie/errata.html

Unfinished errata to RFC 2965.

RFC 2964 - Use of HTTP State Management

21.21.1. CookieJar and FileCookieJar Objects¶

`CookieJar` objects support the *iterator* protocol for iterating over contained `Cookie` objects.

`CookieJar` has the following methods:

`CookieJar.add_cookie_header`(*request*)¶

Add correct *Cookie* header to *request*.

If policy allows (ie. the `rfc2965` and `hide_cookie2` attributes of the `CookieJar`'s `CookiePolicy` instance are true and false respectively), the *Cookie2* header is also added when appropriate.

The *request* object (usually a `urllib2.Request` instance) must support the methods `get_full_url()`, `get_host()`, `get_type()`, `unverifiable()`, `get_origin_req_host()`, `has_header()`, `get_header()`, `header_items()`, and `add_unredirected_header()`,as documented by `urllib2`.

`CookieJar.extract_cookies`(*response*, *request*)¶

Extract cookies from HTTP *response* and store them in the `CookieJar`, where allowed by policy.

The `CookieJar` will look for allowable *Set-Cookie* and *Set-Cookie2* headers in the *response* argument, and store cookies as appropriate (subject to the `CookiePolicy.set_ok()` method's approval).

The *response* object (usually the result of a call to `urllib2.urlopen()`, or similar) should support an `info()` method, which returns an object with a `getallmatchingheaders()` method (usually a `mimetools.Message` instance).

The *request* object (usually a `urllib2.Request` instance) must support the methods `get_full_url()`, `get_host()`, `unverifiable()`, and `get_origin_req_host()`, as documented by `urllib2`. The request is used to set default values for cookie-attributes as well as for checking that the cookie is allowed to be set.

`CookieJar.set_policy`(*policy*)¶

Set the `CookiePolicy` instance to be used.

`CookieJar.make_cookies`(*response*, *request*)¶

Return sequence of `Cookie` objects extracted from *response* object.

See the documentation for `extract_cookies()` for the interfaces required of the *response* and *request* arguments.

`CookieJar.set_cookie_if_ok`(*cookie*, *request*)¶

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

`CookieJar.set_cookie`(*cookie*)¶

Set a `Cookie`, without checking with policy to see whether or not it should be set.

`CookieJar.clear`([*domain*[, *path*[, *name*]]])¶

Clear some cookies.

If invoked without arguments, 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 *domain* and URL *path* are removed. If given three arguments, then the cookie with the specified *domain*, *path* and *name* is removed.

Raises `KeyError` if no matching cookie exists.

`CookieJar.clear_session_cookies`()¶

Discard all session cookies.

Discards all contained cookies that have a true `discard` attribute (usually because they had either no `max-age` or `expires` cookie-attribute, or an explicit `discard` cookie-attribute). For interactive browsers, the end of a session usually corresponds to closing the browser window.

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

`FileCookieJar` implements the following additional methods:

`FileCookieJar.save`(*filename=None*, *ignore_discard=False*, *ignore_expires=False*)¶

Save cookies to a file.

This base class raises `NotImplementedError`. Subclasses may leave this method unimplemented.

filename is the name of file in which to save cookies. If *filename* is not specified, `self.filename` is used (whose default is the value passed to the constructor, if any); if `self.filename` is `None`, `ValueError` is raised.

ignore_discard: save even cookies set to be discarded. *ignore_expires*: save even cookies that have expired

The file is overwritten if it already exists, thus wiping all the cookies it contains. Saved cookies can be restored later using the `load()` or `revert()` methods.

`FileCookieJar.load`(*filename=None*, *ignore_discard=False*, *ignore_expires=False*)¶

Load cookies from a file.

Old cookies are kept unless overwritten by newly loaded ones.

Arguments are as for `save()`.

The named file must be in the format understood by the class, or `LoadError` will be raised. Also, `IOError` may be raised, for example if the file does not exist.

Note

For backwards-compatibility with Python 2.4 (which raised an `IOError`), `LoadError` is a subclass of `IOError`.

`FileCookieJar.revert`(*filename=None*, *ignore_discard=False*, *ignore_expires=False*)¶

Clear all cookies and reload cookies from a saved file.

`revert()` can raise the same exceptions as `load()`. If there is a failure, the object's state will not be altered.

`FileCookieJar` instances have the following public attributes:

`FileCookieJar.filename`¶
Filename of default file in which to keep cookies. This attribute may be assigned to.
`FileCookieJar.delayload`¶
If true, load cookies lazily from disk. This attribute should not be assigned to. This is only a hint, since this only affects performance, not behaviour (unless the cookies on disk are changing). A `CookieJar` object may ignore it. None of the `FileCookieJar` classes included in the standard library lazily loads cookies.

21.21.2. FileCookieJar subclasses and co-operation with web browsers¶

The following `CookieJar` subclasses are provided for reading and writing . Further `CookieJar` subclasses, including one that reads Microsoft Internet Explorer cookies, are available at http://wwwsearch.sf.net/ClientCookie/.

class `cookielib.MozillaCookieJar`(*filename*, *delayload=None*, *policy=None*)¶

A `FileCookieJar` that can load from and save cookies to disk in the Mozilla `cookies.txt` file format (which is also used by the Lynx and Netscape browsers).

Note

Version 3 of the Firefox web browser no longer writes cookies in the `cookies.txt` file format.

Note

This loses information about RFC 2965 cookies, and also about newer or non-standard cookie-attributes such as `port`.

Warning

Back up your cookies before saving if you have cookies whose loss / corruption would be inconvenient (there are some subtleties which may lead to slight changes in the file over a load / save round-trip).

Also note that cookies saved while Mozilla is running will get clobbered by Mozilla.

class `cookielib.LWPCookieJar`(*filename*, *delayload=None*, *policy=None*)¶
A `FileCookieJar` that can load from and save cookies to disk in format compatible with the libwww-perl library's `Set-Cookie3` file format. This is convenient if you want to store cookies in a human-readable file.

21.21.3. CookiePolicy Objects¶

Objects implementing the `CookiePolicy` interface have the following methods:

`CookiePolicy.set_ok`(*cookie*, *request*)¶

Return boolean value indicating whether cookie should be accepted from server.

cookie is a `cookielib.Cookie` instance. *request* is an object implementing the interface defined by the documentation for `CookieJar.extract_cookies()`.

`CookiePolicy.return_ok`(*cookie*, *request*)¶

Return boolean value indicating whether cookie should be returned to server.

cookie is a `cookielib.Cookie` instance. *request* is an object implementing the interface defined by the documentation for `CookieJar.add_cookie_header()`.

`CookiePolicy.domain_return_ok`(*domain*, *request*)¶

Return false if cookies should not be returned, given cookie domain.

This method is an optimization. It removes the need for checking every cookie with a particular domain (which might involve reading many files). Returning true from `domain_return_ok()` and `path_return_ok()` leaves all the work to `return_ok()`.

If `domain_return_ok()` returns true for the cookie domain, `path_return_ok()` is called for the cookie path. Otherwise, `path_return_ok()` and `return_ok()` are never called for that cookie domain. If `path_return_ok()` returns true, `return_ok()` is called with the `Cookie` object itself for a full check. Otherwise, `return_ok()` is never called for that cookie path.

Note that `domain_return_ok()` is called for every *cookie* domain, not just for the *request* domain. For example, the function might be called with both `".example.com"` and `"www.example.com"` if the request domain is `"www.example.com"`. The same goes for `path_return_ok()`.

The *request* argument is as documented for `return_ok()`.

`CookiePolicy.path_return_ok`(*path*, *request*)¶

Return false if cookies should not be returned, given cookie path.

See the documentation for `domain_return_ok()`.

In addition to implementing the methods above, implementations of the `CookiePolicy` interface must also supply the following attributes, indicating which protocols should be used, and how. All of these attributes may be assigned to.

`CookiePolicy.netscape`¶
Implement Netscape protocol.
`CookiePolicy.rfc2965`¶
Implement RFC 2965 protocol.
`CookiePolicy.hide_cookie2`¶
Don't add *Cookie2* header to requests (the presence of this header indicates to the server that we understand RFC 2965 cookies).

The most useful way to define a `CookiePolicy` class is by subclassing from `DefaultCookiePolicy` and overriding some or all of the methods above. `CookiePolicy` itself may be used as a 'null policy' to allow setting and receiving any and all cookies (this is unlikely to be useful).

21.21.4. DefaultCookiePolicy Objects¶

Implements the standard rules for accepting and returning cookies.

Both RFC 2965 and Netscape cookies are covered. RFC 2965 handling is switched off by default.

The easiest way to provide your own policy is to override this class and call its methods in your overridden implementations before adding your own additional checks:

```
import cookielib
class MyCookiePolicy(cookielib.DefaultCookiePolicy):
   def set_ok(self, cookie, request):
       if not cookielib.DefaultCookiePolicy.set_ok(self, cookie, request):
           return False
       if i_dont_want_to_store_this_cookie(cookie):
           return False
       return True
```

In addition to the features required to implement the `CookiePolicy` interface, this class allows you to block and allow domains from setting and receiving cookies. There are also some strictness switches that allow you to tighten up the rather loose Netscape protocol rules a little bit (at the cost of blocking some benign cookies).

A domain blacklist and whitelist is provided (both off by default). Only domains not in the blacklist and present in the whitelist (if the whitelist is active) participate in cookie setting and returning. Use the *blocked_domains* constructor argument, and `blocked_domains()` and `set_blocked_domains()` methods (and the corresponding argument and methods for *allowed_domains*). If you set a whitelist, you can turn it off again by setting it to `None`.

Domains in block or allow lists that do not start with a dot must equal the cookie domain to be matched. For example, `"example.com"` matches a blacklist entry of `"example.com"`, but `"www.example.com"` does not. Domains that do start with a dot are matched by more specific domains too. For example, both `"www.example.com"` and `"www.coyote.example.com"` match `".example.com"` (but `"example.com"` itself does not). IP addresses are an exception, and must match exactly. For example, if blocked_domains contains `"192.168.1.2"` and `".168.1.2"`, 192.168.1.2 is blocked, but 193.168.1.2 is not.

`DefaultCookiePolicy` implements the following additional methods:

`DefaultCookiePolicy.blocked_domains`()¶
Return the sequence of blocked domains (as a tuple).

`DefaultCookiePolicy.set_blocked_domains`(*blocked_domains*)¶
Set the sequence of blocked domains.

`DefaultCookiePolicy.is_blocked`(*domain*)¶
Return whether *domain* is on the blacklist for setting or receiving cookies.

`DefaultCookiePolicy.allowed_domains`()¶
Return `None`, or the sequence of allowed domains (as a tuple).

`DefaultCookiePolicy.set_allowed_domains`(*allowed_domains*)¶
Set the sequence of allowed domains, or `None`.

`DefaultCookiePolicy.is_not_allowed`(*domain*)¶
Return whether *domain* is not on the whitelist for setting or receiving cookies.

`DefaultCookiePolicy` instances have the following attributes, which are all initialised from the constructor arguments of the same name, and which may all be assigned to.

`DefaultCookiePolicy.rfc2109_as_netscape`¶

If true, request that the `CookieJar` instance downgrade RFC 2109 cookies (ie. cookies received in a *Set-Cookie* header with a version cookie-attribute of 1) to Netscape cookies by setting the version attribute of the `Cookie` instance to 0. The default value is `None`, in which case RFC 2109 cookies are downgraded if and only if RFC 2965 handling is turned off. Therefore, RFC 2109 cookies are downgraded by default.

New in version 2.5.

General strictness switches:

`DefaultCookiePolicy.strict_domain`¶
Don't allow sites to set two-component domains with country-code top-level domains like `.co.uk`, `.gov.uk`, `.co.nz`.etc. This is far from perfect and isn't guaranteed to work!

RFC 2965 protocol strictness switches:

`DefaultCookiePolicy.strict_rfc2965_unverifiable`¶
Follow RFC 2965 rules on unverifiable transactions (usually, an unverifiable transaction is one resulting from a redirect or a request for an image hosted on another site). If this is false, cookies are *never* blocked on the basis of verifiability

Netscape protocol strictness switches:

`DefaultCookiePolicy.strict_ns_unverifiable`¶

apply RFC 2965 rules on unverifiable transactions even to Netscape cookies

`DefaultCookiePolicy.strict_ns_domain`¶

Flags indicating how strict to be with domain-matching rules for Netscape cookies. See below for acceptable values.

`DefaultCookiePolicy.strict_ns_set_initial_dollar`¶

Ignore cookies in Set-Cookie: headers that have names starting with `'$'`.

`DefaultCookiePolicy.strict_ns_set_path`¶

Don't allow setting cookies whose path doesn't path-match request URI.

`strict_ns_domain` is a collection of flags. Its value is constructed by or-ing together (for example, `DomainStrictNoDots|DomainStrictNonDomain` means both flags are set).

`DefaultCookiePolicy.DomainStrictNoDots`¶

When setting cookies, the 'host prefix' must not contain a dot (eg. `www.foo.bar.com` can't set a cookie for `.bar.com`, because `www.foo` contains a dot).

`DefaultCookiePolicy.DomainStrictNonDomain`¶

Cookies that did not explicitly specify a `domain` cookie-attribute can only be returned to a domain equal to the domain that set the cookie (eg. `spam.example.com` won't be returned cookies from `example.com` that had no `domain` cookie-attribute).

`DefaultCookiePolicy.DomainRFC2965Match`¶

When setting cookies, require a full RFC 2965 domain-match.

The following attributes are provided for convenience, and are the most useful combinations of the above flags:

`DefaultCookiePolicy.DomainLiberal`¶

Equivalent to 0 (ie. all of the above Netscape domain strictness flags switched off).

`DefaultCookiePolicy.DomainStrict`¶

Equivalent to `DomainStrictNoDots|DomainStrictNonDomain`.

21.21.5. Cookie Objects¶

`Cookie` instances have Python attributes roughly corresponding to the standard cookie-attributes specified in the various cookie standards. The correspondence is not one-to-one, because there are complicated rules for assigning default values, because the `max-age` and `expires` cookie-attributes contain equivalent information, and because RFC 2109 cookies may be 'downgraded' by `cookielib` from version 1 to version 0 (Netscape) cookies.

Assignment to these attributes should not be necessary other than in rare circumstances in a `CookiePolicy` method. The class does not enforce internal consistency, so you should know what you're doing if you do that.

`Cookie.version`¶

Integer or `None`. Netscape cookies have `version` 0. RFC 2965 and RFC 2109 cookies have a `version` cookie-attribute of 1. However, note that `cookielib` may 'downgrade' RFC 2109 cookies to Netscape cookies, in which case `version` is 0.

`Cookie.name`¶

Cookie name (a string).

`Cookie.value`¶

Cookie value (a string), or `None`.

`Cookie.port`¶

String representing a port or a set of ports (eg. '80', or '80,8080'), or `None`.

`Cookie.path`¶

Cookie path (a string, eg. `'/acme/rocket_launchers'`).

`Cookie.secure`¶

True if cookie should only be returned over a secure connection.

`Cookie.expires`¶

Integer expiry date in seconds since epoch, or `None`. See also the `is_expired()` method.

`Cookie.discard`¶

True if this is a session cookie.

`Cookie.comment`¶

String comment from the server explaining the function of this cookie, or `None`.

`Cookie.comment_url`¶

URL linking to a comment from the server explaining the function of this cookie, or `None`.

`Cookie.rfc2109`¶

True if this cookie was received as an RFC 2109 cookie (ie. the cookie arrived in a *Set-Cookie* header, and the value of the Version cookie-attribute in that header was 1). This attribute is provided because `cookielib` may 'downgrade' RFC 2109 cookies to Netscape cookies, in which case `version` is 0.

New in version 2.5.

`Cookie.port_specified`¶

True if a port or set of ports was explicitly specified by the server (in the *Set-Cookie* / *Set-Cookie2* header).

`Cookie.domain_specified`¶

True if a domain was explicitly specified by the server.

`Cookie.domain_initial_dot`¶

True if the domain explicitly specified by the server began with a dot (`'.'`).

Cookies may have additional non-standard cookie-attributes. These may be accessed using the following methods:

`Cookie.has_nonstandard_attr`(*name*)¶

Return true if cookie has the named cookie-attribute.

`Cookie.get_nonstandard_attr`(*name*, *default=None*)¶

If cookie has the named cookie-attribute, return its value. Otherwise, return *default*.

`Cookie.set_nonstandard_attr`(*name*, *value*)¶

Set the value of the named cookie-attribute.

The `Cookie` class also defines the following method:

`Cookie.is_expired`([*now=None*])¶

True if cookie has passed the time at which the server requested it should expire. If *now* is given (in seconds since the epoch), return whether the cookie has expired at the specified time.

21.21.6. Examples¶

The first example shows the most common usage of `cookielib`:

```
import cookielib, urllib2
cj = cookielib.CookieJar()
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))
r = opener.open("http://example.com/")
```

This example illustrates how to open a URL using your Netscape, Mozilla, or Lynx cookies (assumes Unix/Netscape convention for location of the cookies file):

```
import os, cookielib, urllib2
cj = cookielib.MozillaCookieJar()
cj.load(os.path.join(os.environ["HOME"], ".netscape/cookies.txt"))
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))
r = opener.open("http://example.com/")
```

The next example illustrates the use of `DefaultCookiePolicy`. Turn on RFC 2965 cookies, be more strict about domains when setting and returning Netscape cookies, and block some domains from setting cookies or having them returned:

```
import urllib2
from cookielib import CookieJar, DefaultCookiePolicy
policy = DefaultCookiePolicy(
   rfc2965=True, strict_ns_domain=DefaultCookiePolicy.DomainStrict,
   blocked_domains=["ads.net", ".ads.net"])
cj = CookieJar(policy)
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))
r = opener.open("http://example.com/")
```

Table Of Contents

Previous topic

21.20. `CGIHTTPServer` — CGI-capable HTTP request handler

Next topic

21.22. `Cookie` — HTTP state management

This Page

- Show Source

Navigation