PyPAC Documentation

Release 0.13.0

Carson Lam

Sep 24, 2019

Contents

1 Features **3**

2 Installation **5**

3 Quickstart **7**

4 Documentation **9**
- 4.1 Proxy auto-configuration (PAC) files . . . 9
- 4.2 User guide . . . 12
- 4.3 Developer interface . . . 15
- 4.4 PyPAC licence . . . 24
- 4.5 Changelog . . . 24

5 Indices and tables **27**

Python Module Index **29**

Index **31**

Release v0.13.0.

PyPAC is a Python library for finding *proxy auto-config (PAC)* files and making HTTP requests that respect them. PAC files are often used in organizations that need fine-grained and centralized control of proxy settings. *Are you using one?*

PyPAC provides `PACSession`, a drop-in subclass of requests.Session, so you can start transparently finding and obeying PAC files immediately.

CHAPTER 1

Features

- The same Requests API that you already know and love
- Honour PAC setting from Windows Internet Options
- Follow DNS Web Proxy Auto-Discovery protocol
- Proxy authentication pass-through
- Proxy failover and load balancing
- Generic components for adding PAC support to other code

PyPAC supports Python 2.7 and 3.4+.

CHAPTER 2

Installation

Install PyPAC using pip:

```
$ pip install pypac
```

The source is also available on GitHub.

CHAPTER 3

Quickstart

The quickest way to get started is to use a *PACSession*:

```
>>> from pypac import PACSession
>>> session = PACSession()
>>> session.get('http://example.org')
<Response [200]>
```

Behind the scenes, the first request made with the session will trigger the PAC auto-discovery process. This process first looks for a PAC URL setting in Windows, and if not found, moves on to the *DNS WPAD protocol*.

Once a PAC file is found, it will be automatically consulted for every request. If a PAC wasn't found, then `PACSession` acts just like a requests.Session.

If you're looking to add *basic* PAC functionality to a library that you're using, try the *pac_context_for_url* context manager:

```
from pypac import pac_context_for_url
import boto3

with pac_context_for_url('https://example.amazonaws.com'):
    client = boto3.client('sqs')
    client.list_queues()
```

This sets up proxy environment variables at the start of the scope, based on any auto-discovered PAC and the given URL. *pac_context_for_url* should work for any library that honours proxy environment variables.

CHAPTER 4

Documentation

4.1 Proxy auto-configuration (PAC) files

An introduction to PAC files and why you may (or may not) care about them in your internet-accessing Python code.

4.1.1 What they are

A proxy auto-config (PAC) file is a text file that defines a JavaScript function: `FindProxyForURL(url, host)`. For every URL accessed by a client, the function is executed in order to determine the proxy server (if any) to use for that URL.

The PAC specification was created by Netscape in 1996. The specification defines a set of JavaScript functions that are available in the execution context of a PAC file. These functions enable a lot of flexibility in proxy rules: configurations can vary based on any part of the URL, the IPs that domains resolve to, the current date and time, and more.

The alternative to PAC files is to configure each client with a single proxy server to use for all URLs, plus a list of whitelisted hostnames that should bypass the proxy and be accessed directly. The `HTTP_PROXY`, `HTTPS_PROXY`, and `NO_PROXY` environment variables are recognized by Requests for this purpose. This is simpler than a PAC file, but also less powerful.

Wikipedia has further details regarding PAC files.

4.1.2 Who uses them

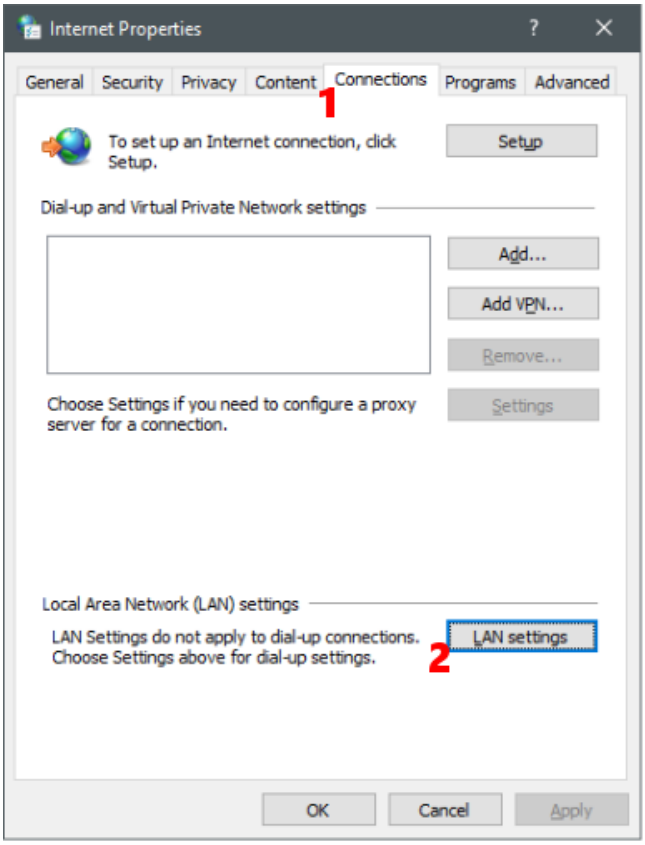
PAC files are used by organizations that require fine-grained and centralized control of proxy configuration. They're commonly deployed in government and institutional networks. If your organization is behind a firewall such as Forcepoint WebShield (Websense) or Symantec Web Gateway, then it's likely that PAC files are in use. These web security firewalls provide access to the internet via proxies that are specified through a PAC file.

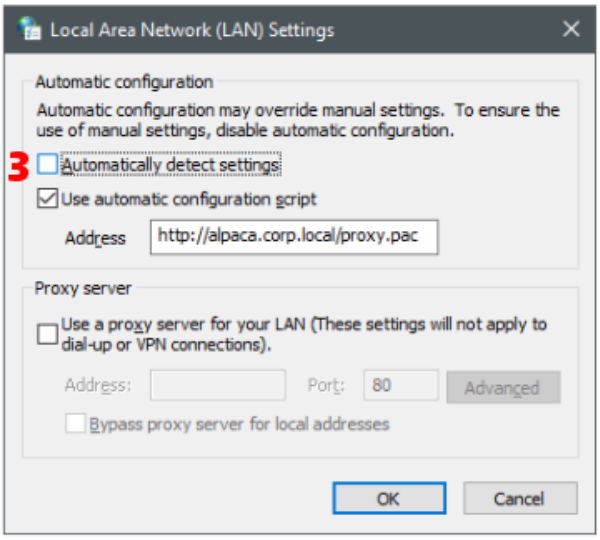
4.1.3 Am I using a PAC?

If you're behind a network where internet access is only possible through a proxy, there's a good chance that a PAC is in use.

Windows

If your network uses a PAC, it's usually the case that clients are running Windows. On Windows, proxy settings are under Internet Options > Connections > LAN Settings:



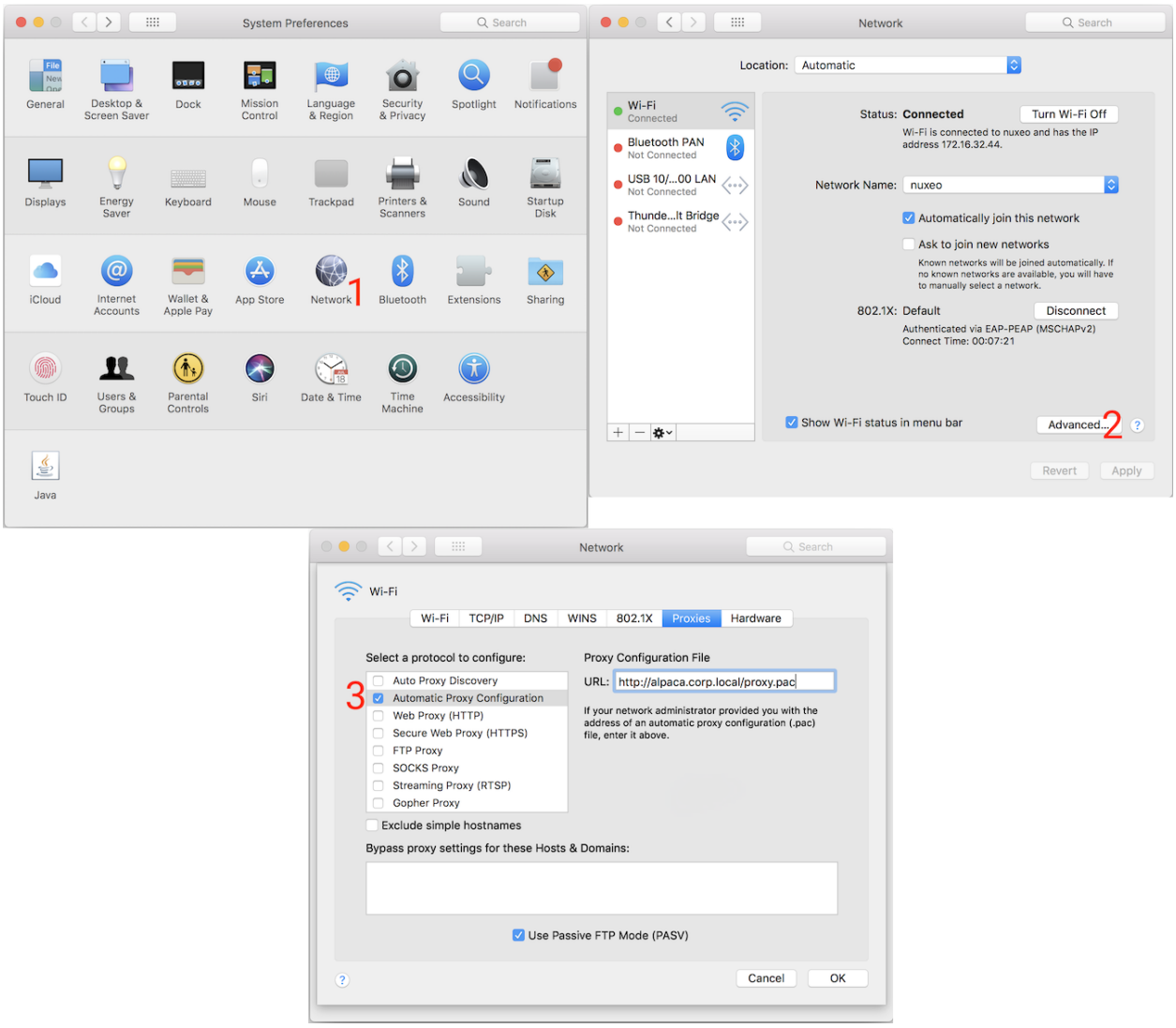


If "automatically detect settings" is enabled, then Web Proxy Auto-Discovery is in use (see next section). If "use automatic configuration script" is enabled, the address field contains the PAC URL.

These settings are often enforced through Group Policy, which means that any changes will be overwritten with the official values from the Group Policy a few minutes later.

macOS/OSX

On macOS, proxy settings are under System Preferences > Network > Advanced... > Proxies:



If “Auto Proxy Discovery” is enabled, then Web Proxy Auto-Discovery is in use (see next section). If ” Automatic Proxy Configuration” is enabled, the address field contains the PAC URL.

Other operating systems

Other operating systems don’t have a setting for the PAC URL. Instead, the Web Proxy Auto-Discovery (WPAD) protocol is used to find the PAC. This protocol defines two discovery methods: DNS and DHCP.

- The DNS method involves the client looking for a valid PAC file via a pattern of URLs based on the client’s hostname.
- The DHCP method involves the client network’s DHCP server providing a value for DHCP Option 252.

PyPAC currently only supports the DNS method.

4.1.4 How PAC files are served

PAC files are typically hosted within the intranet on a web server that’s anonymously accessible by all clients.

PAC files intended to be discovered via WPAD are named `wpad.dat` by convention. Otherwise, they're typically named `proxy.pac`.

To comply with the PAC specification, the file should be served with a Content-Type of `application/x-ns-proxy-autoconfig`. Otherwise, PyPAC will ignore them. However, this behaviour can be configured.

4.1.5 How PyPAC works

PyPAC implements PAC file discovery via Windows Internet Options and the DNS portion of the WPAD protocol. It implements all of the functions necessary to execute a PAC file, and uses the dukpy library to parse and execute the JavaScript. On top of this, PyPAC implements tools to track proxy failover state, and to convert the return value of the JavaScript function into a form understood by the Requests API.

Every request made using a *`PACSession`* consults the PAC file in order to determine the proxy configuration for the given URL.

DHCP WPAD is currently not implemented.

4.1.6 Can't I just hard-code the proxy?

One way to avoid the burden and complexity of using a PAC file is to open the PAC file, figure out which proxy to use, and then hard-code it somewhere, such as an environment variable, or as a constant for the `proxies` argument. In many cases this is good enough, but there are scenarios where it can be fragile:

- Execution on machines you do not control, or portability across network environments both inside and outside your firewall
- Valid proxy hosts changing days, months, or years after being hard-coded
- PAC file contains datetime-based logic that must be followed to yield a valid proxy
- `NO_PROXY` environment variable not granular enough for your use case
- Your application needs to access arbitrary URLs
- Your application's users have a PAC, but don't have the expertise to be aware of its existence or to implement workarounds

In these scenarios, the effort of automatically searching for and honouring a PAC file (if any) adds flexibility and resilience to your application.

4.2 User guide

Learn how to get started with PyPAC. This guide assumes familiarity with the Requests library and its API.

4.2.1 Basic usage

The quickest way to get started is to use a *`PACSession`*:

```
>>> from pypac import PACSession
>>> session = PACSession()
>>> session.get('http://example.org')
<Response [200]>
```

Behind the scenes, the first request made with the session will trigger the PAC auto-discovery process. This process first looks for a PAC URL setting in Windows, and if not found, moves on to the *DNS WPAD protocol*.

Once a PAC file is found, it will be automatically consulted for every request. If a PAC wasn't found, then `PACSession` acts just like a requests.Session.

4.2.2 Specify URL to your PAC

The *`get_pac()`* function encapsulates the PAC file discovery process and returns a *`PACFile`* object upon success. Instead of auto-discovery, this function can be used to get and parse a PAC file from a given URL, which can then be passed to `PACSession`:

```
from pypac import PACSession, get_pac
pac = get_pac(url='http://foo.corp.local/proxy.pac')
session = PACSession(pac)
```

This is useful if you already know the URL for the PAC file to use, and want to skip auto-discovery.

Note that by default, PyPAC requires that PAC files be served with a content-type of either `application/x-ns-proxy-autoconfig` or `application/x-javascript-config`. Files served with other types are excluded from consideration as a PAC file. This behaviour can be customized using the `allowed_content_types` keyword:

```
pac = get_pac(url='http://foo.corp.local/proxy.txt',
              allowed_content_types=['text/plain'])
```

4.2.3 Load PAC from a string or file

This is an unusual scenario, but also supported. Just instantiate your own *`PACFile`*, passing it a string containing the PAC JavaScript. For instance, to load a local PAC file and use it with a `PACSession`:

```
from pypac import PACSession
from pypac.parser import PACFile

with open('proxy.pac') as f:
   pac = PACFile(f.read())

session = PACSession(pac)
```

4.2.4 Proxy authentication

Proxy servers specified by a PAC file typically allow anonymous access. However, PyPAC supports including Basic proxy authentication credentials:

```
from pypac import PACSession
from requests.auth import HTTPProxyAuth
session = PACSession(proxy_auth=HTTPProxyAuth('user', 'pwd'))
# or alternatively...
session.proxy_auth = HTTPProxyAuth('user', 'pwd')
```

NTLM authentication for proxies may also be supported. Refer to the requests-ntlm project.

4.2.5 Custom proxy failover criteria

You can decide when a proxy from the PAC file should be considered unusable. When a proxy is considered unusable, it's blacklisted, and the next proxy specified by the PAC file is used. `PACSession` can be configured with callables that define the criteria for failover.

One way to decide when to fail over is by inspecting the response to a request. By default, PyPAC does not do this, but you may find it useful in case a failing proxy interjects with an unusual response. Another use case is to skip proxies upon an HTTP 407 response:

```
from pypac import PACSession
import requests

def failover_criteria(response):
    return response.status_code == requests.codes.proxy_authentication_required

session = PACSession(response_proxy_fail_filter=failover_criteria)
```

Another way to decide proxy failover is based on any exception raised while making the request. This can be configured by passing a callable for the `exception_proxy_fail_filter` keyword in the `PACSession` constructor. This callable takes an exception object as an argument, and returns true if failover should occur. The default behaviour is to trigger proxy failover upon encountering `requests.exceptions.ConnectTimeout` or `requests.exceptions.ProxyError`.

If all proxies specified by the PAC file have been blacklisted, and the PAC didn't return a final instruction to go `DIRECT`, then *`ProxyConfigExhaustedError`* is raised.

4.2.6 Errors and exceptions

PyPAC defines some exceptions that can occur in the course of PAC auto-discovery, parsing, and execution.

`MalformedPacError` PyPAC failed to parse a file that claims to be a PAC.

`ProxyConfigExhaustedError` All proxy servers for the given URL have been marked as failed, and the PAC file did not specify a final instruction to go `DIRECT`.

4.2.7 Security considerations

Supporting and using PAC files comes with some security implications that are worth considering.

PAC discovery and parsing

PAC files are JavaScript. PyPAC uses dukpy to parse and execute JavaScript. dukpy was not designed for handling untrusted JavaScript, and so it is unclear whether the handling of PAC files is sufficiently sandboxed to prevent untrusted Python code execution.

When looking for a PAC file using DNS WPAD, the local machine's fully-qualified hostname is checked against the Mozilla Public Suffix List to prevent requesting any PAC files outside the scope of the organization. If the hostname's TLD isn't in the Public Suffix List, then everything up to the final node is used in the search path. For example, a hostname of `foo.bar.local` will result in a search for a PAC file from `wpad.bar.local` and `wpad.local`.

PyPAC uses the tld library to match TLDs.

HTTPS-decrypting proxies

Proxies operated by a firewall or web security gateway may may be configured with a man-in-the-middle (MITM) certificate to allow decrypting HTTPS traffic for inspection. Your organization may then provision its client machines with this certificate trusted. Browsers such as Internet Explorer and Chrome, which honour the operating system's certificate store, will accept the proxy's certificate. However, Requests defaults to its own bundled CA certificates, and thus SSL certificate verification will fail when using such a proxy.

A quick solution is to make your requests with the `verify=False` option. Understand that this is an overly broad solution: while it allows your request to proceed and be decrypted for inspection by your network proxy (an entity that you ostensibly trust), it also disables SSL certificate verification entirely. This means requests may be vulnerable to MITM attacks.

4.2.8 What's missing

The DHCP portion of the Web Proxy Auto-Discovery (WPAD) protocol is not implemented.

PyPAC currently works with Requests by including a subclass of requests.Session. No ready-to-use solutions are included for other HTTP libraries, though PyPAC has all the building blocks needed to make one easily.

Pull requests to add these features are welcome.

4.3 Developer interface

These are the interfaces that PyPAC exposes to developers.

4.3.1 Main interface

These are the most commonly used components of PyPAC.

class pypac.**PACSession**(*pac=None*, *proxy_auth=None*, *pac_enabled=True*, *response_proxy_fail_filter=None*, *exception_proxy_fail_filter=None*, *socks_scheme='socks5'*, ***kwargs*)

A PAC-aware Requests Session that discovers and complies with a PAC file, without any configuration necessary. PAC file discovery is accomplished via the Windows Registry (if applicable), and the Web Proxy Auto-Discovery (WPAD) protocol. Alternatively, a PAC file may be provided in the constructor.

Parameters

- **pac** (PACFile) – The PAC file to consult for proxy configuration info. If not provided, then upon the first request, *get_pac()* is called with default arguments in order to find a PAC file.
- **proxy_auth** (*requests.auth.HTTPProxyAuth*) – Username and password proxy authentication.
- **pac_enabled** (*bool*) – Set to `False` to disable all PAC functionality, including PAC auto-discovery.
- **response_proxy_fail_filter** – Callable that takes a `requests.Response` and returns a boolean for whether the response means the proxy used for the request should no longer be used. By default, the response is not inspected.
- **exception_proxy_fail_filter** – Callable that takes an exception and returns a boolean for whether the exception means the proxy used for the request should no longer

be used. By default, `requests.exceptions.ConnectTimeout` and `requests.exceptions.ProxyError` are matched.

- **socks_scheme** (*str*) – Scheme to use when PAC file returns a SOCKS proxy. *socks5* by default.

do_proxy_failover(*proxy_url*, *for_url*)

Parameters

- **proxy_url** (*str*) – Proxy to ban.
- **for_url** (*str*) – The URL being requested.

Returns The next proxy config to try, or 'DIRECT'.

Raises ***ProxyConfigExhaustedError*** – If the PAC file provided no usable proxy configuration.

get_pac(***kwargs*)

Search for, download, and parse PAC file if it hasn't already been done. This method is called upon the first use of *request()*, but can also be called manually beforehand if desired. Subsequent calls to this method will only return the obtained PAC file, if any.

Returns The obtained PAC file, if any.

Return type PACFile|None

Raises ***MalformedPacError*** – If something that claims to be a PAC file was downloaded but could not be parsed.

pac_enabled = None

Set to `False` to disable all PAC functionality, including PAC auto-discovery.

proxy_auth

Proxy authentication object.

request(*method*, *url*, *proxies=None*, ***kwargs*)

Raises

- ***ProxyConfigExhaustedError*** – If the PAC file provided no usable proxy configuration.
- ***MalformedPacError*** – If something that claims to be a PAC file was downloaded but could not be parsed.

pypac.**get_pac**(*url=None*, *js=None*, *from_os_settings=True*, *from_dns=True*, *timeout=2*, *allowed_content_types=None*, *session=None*, ***kwargs*)

Convenience function for finding and getting a parsed PAC file (if any) that's ready to use.

Parameters

- **url** (*str*) – Download PAC from a URL. If provided, *from_os_settings* and *from_dns* are ignored.
- **js** (*str*) – Parse the given string as a PAC file. If provided, *from_os_settings* and *from_dns* are ignored.
- **from_os_settings** (*bool*) – Look for a PAC URL or filesystem path from the OS settings, and use it if present. Doesn't do anything on non-Windows or non-macOS/OSX platforms.
- **from_dns** (*bool*) – Look for a PAC file using the WPAD protocol.
- **timeout** – Time to wait for host resolution and response for each URL.

- **allowed_content_types** – If the response has a `Content-Type` header, then consider the response to be a PAC file only if the header is one of these values. If not specified, the allowed types are `application/x-ns-proxy-autoconfig` and `application/x-javascript-config`.

Returns The first valid parsed PAC file according to the criteria, or *None* if nothing was found.

Return type PACFile|None

Raises ***MalformedPacError*** – If something that claims to be a PAC file was obtained but could not be parsed.

`pypac.`**collect_pac_urls**(*from_os_settings=True*, *from_dns=True*, ***kwargs*)

Get all the URLs that potentially yield a PAC file.

Parameters

- **from_os_settings** (*bool*) – Look for a PAC URL from the OS settings. If a value is found and is a URL, it comes first in the returned list. Doesn't do anything on non-Windows or non-macOS/OSX platforms.
- **from_dns** (*bool*) – Assemble a list of PAC URL candidates using the WPAD protocol.

Returns A list of URLs that should be tried in order.

Return type list[str]

`pypac.`**download_pac**(*candidate_urls*, *timeout=1*, *allowed_content_types=None*, *session=None*)

Try to download a PAC file from one of the given candidate URLs.

Parameters

- **candidate_urls** (*list[str]*) – URLs that are expected to return a PAC file. Requests are made in order, one by one.
- **timeout** – Time to wait for host resolution and response for each URL. When a timeout or DNS failure occurs, the next candidate URL is tried.
- **allowed_content_types** – If the response has a `Content-Type` header, then consider the response to be a PAC file only if the header is one of these values. If not specified, the allowed types are `application/x-ns-proxy-autoconfig` and `application/x-javascript-config`.

Returns Contents of the PAC file, or *None* if no URL was successful.

Return type str|None

`pypac.`**pac_context_for_url**(*url*, *proxy_auth=None*)

This context manager provides a simple way to add rudimentary PAC functionality to code that cannot be modified to use *PACSession*, but obeys the `HTTP_PROXY` and `HTTPS_PROXY` environment variables.

Upon entering this context, PAC discovery occurs with default parameters. If a PAC is found, then it's asked for the proxy to use for the given URL. The proxy environment variables are then set accordingly.

Note that this provides a very simplified PAC experience that's insufficient for some scenarios.

Parameters

- **url** – Consult the PAC for the proxy to use for this URL.
- **proxy_auth** (*requests.auth.HTTPProxyAuth*) – Username and password proxy authentication.

4.3.2 PAC parsing and execution

Functions and classes for parsing and executing PAC files.

class pypac.parser.**PACFile**(*pac_js*, ***kwargs*)
Represents a PAC file.

JavaScript parsing and execution is handled by the dukpy library.

Load a PAC file from a given string of JavaScript. Errors during parsing and validation may raise a specialized exception.

Parameters **pac_js** (*str*) – JavaScript that defines the FindProxyForURL() function.

Raises ***MalformedPacError*** – If the JavaScript could not be parsed, does not define FindProxyForURL(), or is otherwise invalid.

find_proxy_for_url(*url*, *host*)
Call `FindProxyForURL()` in the PAC file with the given arguments.

Parameters

- **url** (*str*) – The full URL.
- **host** (*str*) – The URL's host.

Returns Result of evaluating the `FindProxyForURL()` JavaScript function in the PAC file.

Return type str

pypac.parser.**parse_pac_value**(*value*, *socks_scheme=None*)
Parse the return value of `FindProxyForURL()` into a list. List elements will either be the string "DIRECT" or a proxy URL.

For example, the result of parsing `PROXY example.local:8080; DIRECT` is a list containing strings `http://example.local:8080` and `DIRECT`.

Parameters

- **value** (*str*) – Any value returned by `FindProxyForURL()`.
- **socks_scheme** (*str*) – Scheme to assume for SOCKS proxies. `socks5` by default.

Returns Parsed output, with invalid elements ignored. Warnings are logged for invalid elements.

Return type list[str]

pypac.parser.**proxy_url**(*value*, *socks_scheme=None*)
Parse a single proxy config value from FindProxyForURL() into a more usable element.

Parameters

- **value** (*str*) – Value to parse, e.g.: `DIRECT`, `PROXY example.local:8080`, or `SOCKS example.local:8080`.
- **socks_scheme** (*str*) – Scheme to assume for SOCKS proxies. `socks5` by default.

Returns Parsed value, e.g.: `DIRECT`, `http://example.local:8080`, or `socks5://example.local:8080`.

Return type str

Raises **ValueError** – If input value is invalid.

class pypac.parser.**MalformedPacError**(*msg=None*, *original_exc=None*)

PAC JavaScript functions

Python implementations of JavaScript functions needed to execute a PAC file.

These are injected into the JavaScript execution context. They aren't meant to be called directly from Python, so the function signatures may look unusual.

Most docstrings below are adapted from http://findproxyforurl.com/netscape-documentation/.

pypac.parser_functions.**alert**(_)
No-op. PyPAC ignores JavaScript alerts.

pypac.parser_functions.**dateRange**(**args*)
Accepted forms:

- `dateRange(day)`
- `dateRange(day1, day2)`
- `dateRange(mon)`
- `dateRange(month1, month2)`
- `dateRange(year)`
- `dateRange(year1, year2)`
- `dateRange(day1, month1, day2, month2)`
- `dateRange(month1, year1, month2, year2)`
- `dateRange(day1, month1, year1, day2, month2, year2)`
- `dateRange(day1, month1, year1, day2, month2, year2, gmt)`

day is the day of month between 1 and 31 (as an integer).

month

is one of the month strings: `JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC`

year is the full year number, for example 1995 (but not 95). Integer.

gmt is either the string "GMT", which makes time comparison occur in GMT timezone; if left unspecified, times are taken to be in the local timezone.

Even though the above examples don't show, the "GMT" parameter can be specified in any of the 9 different call profiles, always as the last parameter.

If only a single value is specified (from each category: `day`, `month`, `year`), the function returns a true value only on days that match that specification. If both values are specified, the result is true between those times, including bounds.

Return type bool

pypac.parser_functions.**dnsDomainIs**(*host*, *domain*)

Parameters

- **host** (*str*) – is the hostname from the URL.
- **domain** (*str*) – is the domain name to test the hostname against.

Returns true iff the domain of hostname matches.

Return type bool

pypac.parser_functions.**dnsDomainLevels**(*host*)

Parameters **host** (*str*) – is the hostname from the URL.

Returns the number (integer) of DNS domain levels (number of dots) in the hostname.

Return type int

pypac.parser_functions.**dnsResolve**(*host*)

Resolves the given DNS hostname into an IP address, and returns it in the dot separated format as a string. Returns an empty string if there is an error

Parameters **host** (*str*) – hostname to resolve

Returns Resolved IP address, or empty string if resolution failed.

Return type str

pypac.parser_functions.**isInNet**(*host*, *pattern*, *mask*)

Pattern and mask specification is done the same way as for SOCKS configuration.

Parameters

- **host** (*str*) – a DNS hostname, or IP address. If a hostname is passed, it will be resolved into an IP address by this function.
- **pattern** (*str*) – an IP address pattern in the dot-separated format
- **mask** (*str*) – mask for the IP address pattern informing which parts of the IP address should be matched against. 0 means ignore, 255 means match.

Returns True iff the IP address of the host matches the specified IP address pattern.

Return type bool

pypac.parser_functions.**isPlainHostName**(*host*)

Parameters **host** (*str*) – the hostname from the URL (excluding port number).

Returns True iff there is no domain name in the hostname (no dots).

Return type bool

pypac.parser_functions.**isResolvable**(*host*)

Tries to resolve the hostname.

Parameters **host** (*str*) – is the hostname from the URL.

Returns true if succeeds.

Return type bool

pypac.parser_functions.**localHostOrDomainIs**(*host*, *hostdom*)

Parameters

- **host** (*str*) – the hostname from the URL.
- **hostdom** (*str*) – fully qualified hostname to match against.

Returns true if the hostname matches exactly the specified hostname, or if there is no domain name part in the hostname, but the unqualified hostname matches.

Return type bool

pypac.parser_functions.**myIpAddress**()

Returns the IP address of the host that the Navigator is running on, as a string in the dot-separated integer format.

Return type str

pypac.parser_functions.**shExpMatch**(*host*, *pattern*)
Case-insensitive host comparison using a shell expression pattern.

Parameters

- **host** (*str*) –
- **pattern** (*str*) – Shell expression pattern to match against.

Return type bool

pypac.parser_functions.**timeRange**(**args*)
Accepted forms:

- `timeRange(hour)`
- `timeRange(hour1, hour2)`
- `timeRange(hour1, min1, hour2, min2)`
- `timeRange(hour1, min1, sec1, hour2, min2, sec2)`
- `timeRange(hour1, min1, sec1, hour2, min2, sec2, gmt)`

hour is the hour from 0 to 23. (0 is midnight, 23 is 11 pm.)

min minutes from 0 to 59.

sec seconds from 0 to 59.

gmt either the string "GMT" for GMT timezone, or not specified, for local timezone. Again, even though the above list doesn't show it, this parameter may be present in each of the different parameter profiles, always as the last parameter.

Returns True during (or between) the specified time(s).

Return type bool

pypac.parser_functions.**weekdayRange**(*start_day*, *end_day=None*, *gmt=None*)
Accepted forms:

- `weekdayRange(wd1)`
- `weekdayRange(wd1, gmt)`
- `weekdayRange(wd1, wd2)`
- `weekdayRange(wd1, wd2, gmt)`

If only one parameter is present, the function yields a true value on the weekday that the parameter represents. If the string "GMT" is specified as a second parameter, times are taken to be in GMT, otherwise in local timezone.

If both `wd1` and wd2'' are defined, the condition is true if the current weekday is in between those two weekdays. Bounds are inclusive. If the `gmt` parameter is specified, times are taken to be in GMT, otherwise the local timezone is used.

Weekday arguments are one of `MON TUE WED THU FRI SAT SUN`.

Parameters

- **start_day** (*str*) – Weekday string.
- **end_day** (*str*) – Weekday string.
- **gmt** (*str*) – is either the string: GMT or is left out.

Return type bool

4.3.3 Proxy resolution

Tools for working with a given PAC file and its return values.

class pypac.resolver.**ProxyResolver**(*pac*, *proxy_auth=None*, *socks_scheme='socks5'*)
Handles the lookup of the proxy to use for any given URL, including proxy failover logic.

Parameters

- **pac** (pypac.parser.PACFile) – Parsed PAC file.
- **proxy_auth** (*requests.auth.HTTPProxyAuth*) – Username and password proxy authentication. If provided, then all proxy URLs returned will include these credentials.
- **socks_scheme** (*str*) – Scheme to assume for SOCKS proxies. *socks5* by default.

pypac.resolver.**add_proxy_auth**(*possible_proxy_url*, *proxy_auth*)
Add a username and password to a proxy URL, if the input value is a proxy URL.

Parameters

- **possible_proxy_url** (*str*) – Proxy URL or DIRECT.
- **proxy_auth** (*requests.auth.HTTPProxyAuth*) – Proxy authentication info.

Returns Proxy URL with auth info added, or DIRECT.

Return type str

pypac.resolver.**proxy_parameter_for_requests**(*proxy_url_or_direct*)

Parameters **proxy_url_or_direct** (*str*) – Proxy URL, or DIRECT. Cannot be empty.

Returns Value for use with the proxies parameter in Requests.

Return type dict

class pypac.resolver.**ProxyConfigExhaustedError**(*for_url*)

4.3.4 WPAD functions

Tools for the Web Proxy Auto-Discovery Protocol.

pypac.wpad.**proxy_urls_from_dns**(*local_hostname=None*)
Generate URLs from which to look for a PAC file, based on a hostname. Fully-qualified hostnames are checked against the Mozilla Public Suffix List to ensure that generated URLs don't go outside the scope of the organization. If the fully-qualified hostname doesn't have a recognized TLD, such as in the case of intranets with '.local' or '.internal', the TLD is assumed to be the part following the rightmost dot.

Parameters **local_hostname** (*str*) – Hostname to use for generating the WPAD URLs. If not provided, the local hostname is used.

Returns PAC URLs to try in order, according to the WPAD protocol. If the hostname isn't qualified or is otherwise invalid, an empty list is returned.

Return type list[str]

pypac.wpad.**wpad_search_urls**(*subdomain_or_host*, *fld*)
Generate URLs from which to look for a PAC file, based on the subdomain and TLD parts of a fully-qualified host name.

Parameters

- **subdomain_or_host** (*str*) – Subdomain portion of the fully-qualified host name. For foo.bar.example.com, this is foo.bar.
- **fld** (*str*) – FLD portion of the fully-qualified host name. For foo.bar.example.com, this is example.com.

Returns PAC URLs to try in order, according to the WPAD protocol.

Return type list[str]

4.3.5 OS stuff

Tools for getting the configured PAC file URL out of the OS settings.

exception pypac.os_settings.**NotDarwinError**

exception pypac.os_settings.**NotWindowsError**

pypac.os_settings.**ON_DARWIN = False**
True if running on macOS/OSX.

pypac.os_settings.**ON_WINDOWS = False**
True if running on Windows.

pypac.os_settings.**autoconfig_url_from_preferences**()
Get the PAC `AutoConfigURL` value from the macOS System Preferences. This setting is visible as the "URL" field in System Preferences > Network > Advanced... > Proxies > Automatic Proxy Configuration.

Returns The value from the registry, or None if the value isn't configured or available. Note that it may be local filesystem path instead of a URL.

Return type str|None

Raises ***NotDarwinError*** – If called on a non-macOS/OSX platform.

pypac.os_settings.**autoconfig_url_from_registry**()
Get the PAC `AutoConfigURL` value from the Windows Registry. This setting is visible as the "use automatic configuration script" field in Internet Options > Connection > LAN Settings.

Returns The value from the registry, or None if the value isn't configured or available. Note that it may be local filesystem path instead of a URL.

Return type str|None

Raises ***NotWindowsError*** – If called on a non-Windows platform.

pypac.os_settings.**file_url_to_local_path**(*file_url*)
Parse a `AutoConfigURL` value with `file://` scheme into a usable local filesystem path.

Parameters **file_url** – Must start with `file://`.

Returns A local filesystem path. It might not exist.

class pypac.os_settings.**NotWindowsError**

class pypac.os_settings.**NotDarwinError**

4.4 PyPAC licence

4.5 Changelog

4.5.1 0.13.0 (2019-09-16)

- Make it possible to configure the request for the PAC file. (#44) Thanks @SeyfSV.
- urlencode proxy username and password. (#46) Thanks @aslafy-z.

4.5.2 0.12.0 (2018-09-11)

- Fix possible error when `dnsResolve()` fails. (#34) Thanks @maximinus.

4.5.3 0.11.0 (2018-09-08)

- Require dukpy 0.2.2, to fix memory leak. (#32) Thanks @maximinus.
- Change Mac environment marker. (#30)
- Support Python 3.7.

4.5.4 0.10.1 (2018-08-26)

- Require tld 0.9.x. (#29)

4.5.5 0.10.0 (2018-08-26)

- Switch JavaScript interpreter to dukpy. (#24)
- Fix `pac_context_for_url()` erroring with `DIRECT` PAC setting. (#27)
- Fix warning about invalid escape sequence (#26). Thanks @BoboTiG.

4.5.6 0.9.0 (2018-06-02)

- Add macOS support for PAC in System Preferences (#23). Thanks @LKleinNux.
- The *from_registry* argument on *pypac.get_pac()* and *pypac.collect_pac_urls()* is now deprecated and will be removed in 1.0.0. Use *from_os_settings* instead.

4.5.7 0.8.1 (2018-03-01)

- Defer Js2Py import until it's needed. It uses a lot of memory. See #20 for details.

4.5.8 0.8.0 (2018-02-28)

- Add support for `file://` PAC URLs on Windows.

4.5.9 0.7.0 (2018-02-21)

- Drop support for Python 3.3.
- Add doc explaining how to use `pac_context_for_url()`.
- Internal changes to dev and test processes.

4.5.10 0.6.0 (2018-01-28)

- Add `pac_context_for_url()`, a context manager that adds basic PAC functionality through proxy environment variables.

4.5.11 0.5.0 (2018-01-18)

- Accept PAC files served with no `Content-Type` header.

4.5.12 0.4.0 (2017-11-07)

- Add `recursion_limit` keyword argument to `PACSession` and `PACFile`. The default is an arbitrarily high value (10000), which should cover most applications.
- Exclude port numbers from `host` passed to `FindProxyForURL(url, host)`.

4.5.13 0.3.1 (2017-06-23)

- Update GitHub username.

4.5.14 0.3.0 (2017-04-12)

- Windows: Get system auto-proxy config setting using `winreg` module.
- Windows: Accept local filesystem paths from system proxy auto-config setting.
- Raise `PacComplexityError` when recursion limit is hit while parsing PAC file.
- Support setting `PACSession.proxy_auth` and `ProxyResolver.proxy_auth` after constructing an instance.
- Narrative docs.

4.5.15 0.2.1 (2017-01-19)

- Require Js2Py >= 0.43 for Python 3.6 support, and to avoid needing to monkeypatch out `pyimport`.

4.5.16 0.1.0 (2016-06-12)

- First release.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

pypac.api, 15
pypac.os_settings, 23
pypac.parser, 18
pypac.parser_functions, 19
pypac.resolver, 22
pypac.wpad, 22

Index

A

add_proxy_auth() (*in module pypac.resolver*), 22
alert() (*in module pypac.parser_functions*), 19
autoconfig_url_from_preferences() (*in module pypac.os_settings*), 23
autoconfig_url_from_registry() (*in module pypac.os_settings*), 23

C

collect_pac_urls() (*in module pypac*), 17

D

dateRange() (*in module pypac.parser_functions*), 19
dnsDomainIs() (*in module pypac.parser_functions*), 19
dnsDomainLevels() (*in module pypac.parser_functions*), 19
dnsResolve() (*in module pypac.parser_functions*), 20
do_proxy_failover() (*pypac.PACSession method*), 16
download_pac() (*in module pypac*), 17

F

file_url_to_local_path() (*in module pypac.os_settings*), 23
find_proxy_for_url() (*pypac.parser.PACFile method*), 18

G

get_pac() (*in module pypac*), 16
get_pac() (*pypac.PACSession method*), 16

I

isInNet() (*in module pypac.parser_functions*), 20
isPlainHostName() (*in module pypac.parser_functions*), 20
isResolvable() (*in module pypac.parser_functions*), 20

L

localHostOrDomainIs() (*in module pypac.parser_functions*), 20

M

MalformedPacError (*class in pypac.parser*), 18
myIpAddress() (*in module pypac.parser_functions*), 20

N

NotDarwinError, 23
NotDarwinError (*class in pypac.os_settings*), 23
NotWindowsError, 23
NotWindowsError (*class in pypac.os_settings*), 23

O

ON_DARWIN (*in module pypac.os_settings*), 23
ON_WINDOWS (*in module pypac.os_settings*), 23

P

pac_context_for_url() (*in module pypac*), 17
pac_enabled (*pypac.PACSession attribute*), 16
PACFile (*class in pypac.parser*), 18
PACSession (*class in pypac*), 15
parse_pac_value() (*in module pypac.parser*), 18
proxy_auth (*pypac.PACSession attribute*), 16
proxy_parameter_for_requests() (*in module pypac.resolver*), 22
proxy_url() (*in module pypac.parser*), 18
proxy_urls_from_dns() (*in module pypac.wpad*), 22
ProxyConfigExhaustedError (*class in pypac.resolver*), 22
ProxyResolver (*class in pypac.resolver*), 22
pypac.api (*module*), 15
pypac.os_settings (*module*), 23
pypac.parser (*module*), 18
pypac.parser_functions (*module*), 19
pypac.resolver (*module*), 22

pypac.wpad (*module*), 22

R

request() (*pypac.PACSession method*), 16

S

shExpMatch() (*in module pypac.parser_functions*), 21

T

timeRange() (*in module pypac.parser_functions*), 21

W

weekdayRange() (*in module pypac.parser_functions*), 21
wpad_search_urls() (*in module pypac.wpad*), 22