
curl_cffi

lexiforest

Oct 27, 2026

CONTENTS:

1	Quick Start	1
1.1	Install	1
1.2	requests-like	1
1.3	Asyncio	10
1.4	WebSockets	11
2	Fingerprint Updates	13
2.1	Feature matrix	13
2.2	Configure access	13
2.3	Update fingerprints	13
2.4	Load local fingerprints	14
2.5	Load and edit fingerprints	14
2.6	Storage paths	14
3	CLI	15
3.1	Install	15
3.2	Usage	15
4	Impersonate Guide	25
4.1	What is TLS and http/2, http/3 fingerprinting?	25
4.2	JA4 support in curl-ffi	25
4.3	Supported browser impersonate targets	28
4.4	Which target version to use?	30
4.5	How to use my own fingerprints? e.g. okhttp	30
4.6	TLS PSK(41) Extension	35
4.7	Impersonation FAQ	36
5	Advanced Topics	39
5.1	DNS servers	39
5.2	Proxies	39
5.3	Response cache	40
5.4	Low-level curl API	42
5.5	Using CURLOPT_* in requests API	43
5.6	Selecting http version	43
5.7	Keeping session alive in http/2	43
5.8	Connection reuse across threads	44
6	Compatibility with requests	45
6.1	Transports and Adapters	45
7	Cookies	47

7.1	Save and load cookies	47
7.2	Discard cookies when using Session	47
8	Community	49
8.1	Scrapy integrations	49
8.2	Using with eventlet/gevent	49
8.3	As a urllib3/requests adapter	49
8.4	As a httpx transport	49
9	API References	51
9.1	curl low level APIs	51
9.2	requests-like API	55
9.3	Asyncio	68
9.4	WebSocket	68
9.5	Exceptions and Warnings	79
10	Security	83
10.1	SSRF protection	83
11	FAQ	85
11.1	What does the pro version offer? Is the open source project still maintained?	85
11.2	Why does the JA3 fingerprints change for Chrome 110+ impersonation?	85
11.3	Can I bypass Cloudflare with this project? or any other specific site.	85
11.4	I'm getting certs errors	86
11.5	ErrCode: 77, Reason: error setting certificate verify locations	86
11.6	How to use with fiddler/charles to intercept content	86
11.7	ErrCode: 92, Reason: 'HTTP/2 stream 0 was not closed cleanly: PROTOCOL_ERROR (err 1)'	86
11.8	Packaging with PyInstaller	87
11.9	How to change the order of headers?	87
11.10	How to deal with encoding/decoding errors?	87
12	Change Log	89
13	Development and contributing	93
13.1	macOS	93
13.2	Contributing	94
14	Sponsors	95
14.1	Bypass Cloudflare with API	95
14.2	Cloro	95
15	Features	97
16	Install	99
17	Documentation	101
18	Indices and tables	103
	Index	105

QUICK START

1.1 Install

Note: We only support Python 3.10 and above. Python 3.9 has reached its end of life.

1.1.1 Via pip

The simplest way is to install from PyPI:

```
pip install curl_cffi --upgrade
```

We have sdist(source distribution) and bdist(binary distribution) on PyPI. This should work on Linux, macOS and Windows out of the box.

If it does not work on your platform, you may need to compile and install `curl-impersonate` first and set some environment variables like `LD_LIBRARY_PATH`.

1.1.2 Beta versions

To install beta releases:

```
pip install curl_cffi --upgrade --pre
```

Note the `--pre` option here means pre-releases.

1.1.3 Latest

To install the latest unstable version from GitHub:

```
git clone https://github.com/lexiforest/curl_cffi/  
cd curl_cffi  
make preprocess  
pip install .
```

Or you can download the wheels from github actions artifacts.

1.2 requests-like

`curl_cffi` tries to follow the `requests` API when possible, if you are already familiar with requests, read the warning part in this page, skip other parts and head over to the [Compatibility with requests](#).

1.2.1 Basic GET requests

Basic GET request with the impersonate parameter.

```
import curl_cffi

url = "https://tls.browerleaks.com/json"

# Notice the impersonate parameter
r = curl_cffi.get("https://tls.browerleaks.com/json", impersonate="chrome110")

print(r.json())
# output: {..., "ja3n_hash": "aa56c057ad164ec4fdcb7a5a283be9fc", ...}
# the ja3n fingerprint should be the same as target browser

# To keep using the latest browser version as `curl_cffi` updates,
# simply set impersonate="chrome" without specifying a version.
# Other similar values are: "safari" and "safari_ios"
r = curl_cffi.get("https://tls.browerleaks.com/json", impersonate="chrome")

# http/socks proxies are supported
proxies = {"https": "http://localhost:3128"}
r = curl_cffi.get("https://tls.browerleaks.com/json", impersonate="chrome110",
↳proxies=proxies)

proxies = {"https": "socks://localhost:3128"}
r = curl_cffi.get("https://tls.browerleaks.com/json", impersonate="chrome110",
↳proxies=proxies)
```

1.2.2 URL params

Passing query params with params:

```
import curl_cffi

>>> params = {"foo": "bar"}
>>> r = curl_cffi.get("http://httpbin.org/get", params=params)
>>> r.url
'http://httpbin.org/get?foo=bar'

>>> params = {'key1': 'value1', 'key2': ['value2', 'value3']}
>>> import curl_cffi
>>> r = curl_cffi.get('https://httpbin.org/get', params=params)
>>> r.url
'https://httpbin.org/get?key1=value1&key2=value2&key2=value3'
```

1.2.3 Headers

Additional headers can be overridden by headers=...

```
headers = {"User-Agent": "curl_cffi/0.11.2"}
r = curl_cffi.get("http://example.com", headers=headers)
```

Warning

In curl_cffi, if you set `impersonate=...`, by default, the corresponding headers will be added, you can:

1. Add your headers to override them.
2. Use `default_headers=False` to completely turn off the default headers.
3. Use `curl_cffi.get_fingerprint(...)` and pass the result to `impersonate=...` for fully editable custom fingerprints.

```
fingerprint = curl_cffi.get_fingerprint("edge_146_macos_26")
fingerprint.headers["User-Agent"] = "..."
r = curl_cffi.get(
    "https://httpbin.org/headers",
    impersonate=fingerprint,
)
```

```
>>> r = curl_cffi.get("https://httpbin.org/headers", impersonate="chrome")
>>> print(r.text)
{
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
    ↳webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7",
    "Accept-Encoding": "gzip, deflate, br, zstd",
    "Accept-Language": "en-US,en;q=0.9",
    "Host": "httpbin.org",
    "Priority": "u=0, i",
    "Sec-Ch-Ua": "\"Chromium\";v=\"136\", \"Google Chrome\";v=\"136\", \"Not.A/Brand\";
    ↳v=\"99\"",
    "Sec-Ch-Ua-Mobile": "?0",
    "Sec-Ch-Ua-Platform": "\"macOS\"",
    "Sec-Fetch-Dest": "document",
    "Sec-Fetch-Mode": "navigate",
    "Sec-Fetch-Site": "none",
    "Sec-Fetch-User": "?1",
    "Upgrade-Insecure-Requests": "1",
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36_
    ↳(KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36",
    "X-Amzn-Trace-Id": "Root=1-68452cc9-7287427f222e720c57971297"
  }
}

>>> r = curl_cffi.get("https://httpbin.org/headers", impersonate="chrome", default_
    ↳headers=False)
>>> print(r.text)
{
  "headers": {
    "Accept": "*//*",
    "Accept-Encoding": "gzip, deflate, br",
    "Host": "httpbin.org",
    "X-Amzn-Trace-Id": "Root=1-68452d20-2cf4cf00201987301c476c06"
  }
}
```

(continues on next page)

```
}
```

1.2.4 Reading Response

Like requests, you can read the response content in the following ways:

Reading the binary content as bytes:

```
>>> r = curl_cffi.get("https://example.com")
>>> r.content
b'<!doctype html>\n<html>\n<head>\n...'
```

Reading the decoded content as str:

```
>>> r = curl_cffi.get("https://example.com")
>>> r.text
'<!doctype html>\n<html>\n<head>\n...'
```

By default, `curl_cffi` first uses the `encoding` attribute if given, then tries to use the `Content-Type` header to decode the content. If not found, will fallback to `default_encoding`, then to “utf-8”.

```
# force override with .encoding
>>> r.encoding = 'latin-1'
```

1.2.5 POST and uploads

Of course, we also support POST, PUT, DELETE etc.

Form submit

Use the `data={...}` option.

Note

The “application/x-www-form-urlencoded” will be automatically added.

```
>>> r = curl_cffi.post("https://httpbin.org/post", data={"name": "Luke"})
>>> print(r.text)
{
  "args": {},
  "form": {
    "name": "Luke"
  },
  ...
}
```

Binary data

Use the `content=b"..."` option. Passing bytes through `data=` remains supported for compatibility.

```
>>> r = curl_cffi.post("https://httpbin.org/post", content=b"LukeSkywalker")
>>> print(r.text)
{
  "args": {},
  "data": "LukeSkywalker",
  "files": {},
  "form": {},
  ...
}
```

Streaming request bodies

Pass a byte iterable or binary file through `content=` to upload it without buffering the entire body. If no Content-Length header is supplied, libcurl uses streaming transfer framing appropriate for the negotiated HTTP version.

```
def chunks():
    yield b"first chunk"
    yield b"second chunk"

r = curl_cffi.post("https://httpbin.org/post", content=chunks())
```

With `AsyncSession`, `content=` also accepts an async byte iterable:

```
async def chunks():
    yield b"first chunk"
    await get_more_data()
    yield b"second chunk"

async with curl_cffi.AsyncSession() as session:
    r = await session.post("https://httpbin.org/post", content=chunks())
```

One-shot iterators cannot be replayed. If a retry or redirect needs to resend the body, `curl_cffi` raises `UnrewindableBodyError`. Seekable binary files are rewound automatically.

Posting JSON

Use the `json=...` option.

Note

The “application/json” will be automatically added.

```
>>> r = curl_cffi.post("https://httpbin.org/post", json={"name": "Luke"})
>>> print(r.text)
{
  "args": {},
  "data": "{\"name\": \"Luke\"}",
  "files": {},
  "form": {},
  ...
}
```

Uploads

Warning

curl_cffi does not support the `files=...` API. Use `multipart=...` instead.

For uploading files, the requests API is horrible, we provide a similar but cleaner way:

```
mp = curl_cffi.CurlMime()

mp.addpart(
    name="attachment",          # field name in the form
    content_type="image/png",   # mime type
    filename="image.png",      # filename seen by remote server
    local_path="./image.png",   # local file to upload
    data=file.read(),          # if you already have the data in memory
)

r = curl_cffi.post("https://httpbin.org/post", data={"foo": "bar"}, multipart=mp)
print(r.json())
```

All the fields in the API are explicit. For advanced usage: see [examples](#).

1.2.6 Compressed response

Currently, we forcefully decode compressed response, but this may be changed in the future.

curl_cffi supports gzip/brotli/zstd natively.

If the response content is json, you can parse them directly:

```
>>> r = curl_cffi.get("https://httpbin.org/headers")
>>> r.json()
{'headers': {'Accept': '*//*', 'Accept-Encoding': 'gz...'}}
```

Install `curl_cffi[extra]` to select part of the response using JSONPath:

```
users = r.json(path=".data.users", default=[])
first_name = r.json(path="$.data.users[*].name")
```

Path selection returns the first matched value. If there are no matches, it returns `default`, which is `None` when omitted. Matched values such as `None` or empty lists are returned unchanged. Invalid JSON and invalid paths raise errors rather than returning the default. Calling `r.json()` returns the entire parsed response and does not require the extra dependencies.

1.2.7 Response status

```
>>> r = curl_cffi.get('https://httpbin.org/get')
>>> r.status_code
200

>>> r = curl_cffi.get("https://httpbin.org/status/404")
>>> r.status_code
404
```

(continues on next page)

(continued from previous page)

```
>>> r.raise_for_status()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/.../repos/curl_cffi/curl_cffi/requests/models.py", line 167, in raise_for_
    ↪ status
    raise HTTPError(f"HTTP Error {self.status_code}: {self.reason}", 0, self)
curl_cffi.requests.exceptions.HTTPError: HTTP Error 404:
```

1.2.8 Response headers

Response headers are stored as a case-insensitive dict.

```
>>> r.headers
Headers({'date': 'Sun, 08 Jun 2025 06:58:15 GMT', 'content-type': 'application/json',
    ↪ 'content-length': '184', 'server': 'unicorn/19.9.0', 'access-control-allow-origin': '*'
    ↪, 'access-control-allow-credentials': 'true'})
>>> r.headers["content-type"]
'application/json'
```

For a complete list of response attributes, see the [API References](#).

1.2.9 Streaming response

For compatibility, curl_cffi supports the `stream=True` option, and a `stream` method, with iterative-style content streaming.

But when possible, you should choose the native `content_callback` option.

```
r = curl_cffi.get(

>>> r = curl_cffi.get("https://httpbin.org/stream/20", stream=True)
>>> for chunk in r.iter_content():
...     print("CHUNK", chunk)
...
CHUNK b'{"url": "https://httpbin.org/stream/20",...}'
CHUNK b'{"url": "https://httpbin.org/stream/20",...}'
```

For more examples, see the [examples on GitHub](#)

Warning

Natively, libcurl only support a callback-style API, i.e. you pass a callback function for processing the streamed response content. In curl_cffi, we use a internal queue to convert the callback to a interative API.

Because of the limitation, when a request is sent, the response will start to be streamed to your client at once. You need to start consuming the content immediately, otherwise, it would be store in your memory, thus OOM could be triggered.

Alternatively, you should use the native `content_callback` API.

```
>>> def callback(chunk):
...     print("CHUNK", chunk)
```

(continues on next page)

(continued from previous page)

```
...
>>> r = curl_cffi.get("https://httpbin.org/stream/20", content_callback=callback)
CHUNK b'{"url": "https://httpbin.org/stream/20"...'
CHUNK b'{"url": "https://httpbin.org/stream/20"...'
```

1.2.10 Redirection and history

curl_cffi automatically follows redirects, use `allow_redirects=False` to disable it.

```
>>> r = curl_cffi.get("https://httpbin.org/redirect-to?url=/")
>>> r.url
'https://httpbin.org/'

>>> r = requests.get("https://httpbin.org/redirect-to?url=/", allow_redirects=False)
>>> r.url
'https://httpbin.org/redirect-to?url=/'
>>> r.status_code
302
```

Redirect responses are available in request order. Their status, URL, and headers are populated, but their response bodies are not available.

```
>>> r = curl_cffi.get("https://httpbin.org/redirect-to?url=/")
>>> r.history[0].status_code
302
```

1.2.11 Authenticate

You can use the `auth` parameter or URL to add http basic auth credentials.

```
>>> curl_cffi.get("https://example.com", auth=("my_user", "password123"))

>>> curl_cffi.get("https://user:password@example.com")
```

Digest auth is dangerous and `deprecated`, we do not support that. Although, you should be able to use it with low-level curl options.

1.2.12 Sessions and cookies

We also provide Session to persist cookies and reuse connections.

Note

You should always use a session whenever possible.

```
s = curl_cffi.Session()

# Cookies from server are stored in session
s.get("https://httpbin.org/cookies/set/foo/bar")
```

(continues on next page)

(continued from previous page)

```

print(s.cookies)
# <Cookies[<Cookie foo=bar for httpbin.org />]>

# Cookies are used in next request
r = s.get("https://httpbin.org/cookies")
print(r.json())
# {'cookies': {'foo': 'bar'}}

# It's preferred to use a context manager
with curl_cffi.Session() as s:
    r = s.get("https://example.com")

```

If you want to set a global option in Session, you can use the same parameter in request.

```

with curl_cffi.Session(headers={"User-Agent": "curl_cffi/0.11"}) as s:
    r = s.get("https://example.com")

```

For a complete list, see *API References*

Caching

Session can cache successful responses, which is useful in tests where you want to avoid repeated network calls or mock a stable upstream response.

```

from datetime import timedelta
from curl_cffi import Session

with Session(cache=timedelta(minutes=5)) as s:
    r = s.get("https://example.com/api")

```

For more control over cache location and behavior, see *Advanced Topics*.

Retries

Use retry to automatically re-run failed requests.

```

from curl_cffi import Session, RetryStrategy

# Simple count-based retries
with Session(retry=2) as s:
    r = s.get("https://example.com")

# Custom strategy with delay/backoff/jitter
strategy = RetryStrategy(count=3, delay=0.2, jitter=0.1, backoff="exponential")
with Session(retry=strategy) as s:
    r = s.get("https://example.com")

```

Response vs Session cookies

The response.cookies object contains only cookies from current request. Be aware, if you hit a redirect, response cookies may be incomplete, it's almost always better to use a session.

```
import curl_cffi
r = curl_cffi.get("https://httpbin.org/redirect")

# Cookie from previous redirect may be lost.
do_something(r.cookies)

s = curl_cffi.Session()
r = s.get("https://httpbin.org/redirect")

# Use a session instead, to retrieve all cookies in the session
do_something(s.cookies)
```

Use session without cookies

If you want to use a session, but somehow you need to discard all the cookies. You can use the `discard_cookies` option to discard cookies in session.

```
s = curl_cffi.Session(discard_cookies=True)
```

1.3 Asyncio

Besides the regular sync API, `curl_cffi` also provides a very similar `asyncio` API.

```
# You must use a session for asyncio
async with curl_cffi.AsyncSession() as s:
    r = await s.get("https://example.com")
```

The benefit of `asyncio` is an easier way to implement more concurrency:

```
import asyncio
from curl_cffi import AsyncSession

urls = [
    "https://google.com/",
    "https://facebook.com/",
    "https://apple.com/",
]

async with AsyncSession() as s:
    tasks = []
    for url in urls:
        task = s.get(url)
        tasks.append(task)
    results = await asyncio.gather(*tasks)
```

For detailed `asyncio` guide, see `asyncio`.

1.4 WebSockets

curl_cffi supports both sync and async API for websockets.

```
from curl_cffi import Session, WebSocket

def on_message(ws: WebSocket, message):
    print(message)

with Session() as session:
    ws = session.ws_connect(
        "wss://api.gemini.com/v1/marketdata/BTCUSD",
        on_message=on_message,
    )
    ws.run_forever()

# asyncio
import asyncio
from curl_cffi import AsyncSession

async with AsyncSession() as session:
    async with session.ws_connect("wss://echo.websocket.org") as ws:
        await asyncio.gather(*[ws.send_str("Hello, World!") for _ in range(10)])
        async for message in ws:
            print(message)
```

For detailed websocket guide, see [websockets](#).

FINGERPRINT UPDATES

`curl_cffi` provides fingerprint updates for professional users via impersonate.pro.

2.1 Feature matrix

Table 1: Feature matrix

Edition	HTTP/2	HTTP/3	Fingerprint updates
Open source			Major browser releases
Commercial update			Weekly updates

2.2 Configure access

Set your API key once:

```
curl-cffi config --api-key imp_XXXXXXX
```

or in Python:

```
from curl_cffi.fingerprints import FingerprintManager

FingerprintManager.set_api_key("imp_XXXXXXX")
```

Or override the configured key at runtime:

```
export IMPERSONATE_API_KEY=imp_XXXXXXX
```

2.3 Update fingerprints

You don't need to update fingerprints every time you run your script. Put *curl-cffi update* in your cron job for a daily run is sufficient.

Updating in the CLI is preferred for its simplicity.

From CLI:

```
curl-cffi update
```

From Python:

```
from curl_cffi.fingerprints import FingerprintManager

updated = FingerprintManager.update_fingerprints()
print(updated)
```

2.4 Load local fingerprints

```
from curl_cffi.fingerprints import FingerprintManager

fingerprints = FingerprintManager.load_fingerprints()
print(len(fingerprints))
print(fingerprints["chrome136"])
```

2.5 Load and edit fingerprints

For fingerprint-backed targets, you can load a fresh editable Fingerprint object and pass it back to a request with the `impersonate=...` parameter.

```
import curl_cffi

fingerprint = curl_cffi.get_fingerprint("edge_146_macos_26")
fingerprint.headers["User-Agent"] = "..."

r = curl_cffi.get(
    "https://example.com",
    impersonate=fingerprint,
)
```

2.6 Storage paths

By default, local files are written to a platform-native config directory:

- Linux: `$XDG_CONFIG_HOME/impersonate` or `~/.config/impersonate`
- macOS: `~/.config/impersonate`
- Windows: `%APPDATA%\impersonate`

Files stored there:

- `config.json`: API key config
- `fingerprints.json`: cached fingerprints

Environment overrides:

- `IMPERSONATE_API_KEY`: override the API key loaded from `config.json`.
- `IMPERSONATE_CONFIG_DIR`: override the config/cache directory.

`curl-cffi` ships with a friendly command-line interface for making HTTP requests with browser impersonation.

3.1 Install

The CLI is included when you install `curl_cffi`:

```
pip install curl_cffi
```

Install the optional `cli` extra for syntax highlighting and download progress bars:

```
pip install 'curl_cffi[cli]'
```

On macOS, you can also install via Homebrew:

```
brew install lexiforest/tap/curl-cffi
```

The command `curl-cffi` will be available in your shell. If it's not in your `PATH` for some reason, you can:

1. Add it to your `PATH`
2. Try something like `uv run curl-cffi` or `python -m curl_cffi`

3.2 Usage

```
curl-cffi METHOD URL [REQUEST_ITEMS...] [FLAGS]
```

- **METHOD** is required. Use `get`, `post`, `put`, `delete`, `patch`, `head`, `options`, `trace`, or `query` (case-insensitive).
- **URL** is required. Bare domains default to `https://` (e.g. `example.com` becomes `https://example.com`). A leading colon is a localhost shortcut (e.g. `:3000` becomes `http://localhost:3000`). When an explicit port other than 443 is given, `http://` is used instead (e.g. `example.com:8080` becomes `http://example.com:8080`).
- **REQUEST_ITEMS** use special separators to specify headers, query parameters, body fields, and cookies (see `request_items`).

3.2.1 Quick examples

```

# Simple GET
curl-cffi get https://httpbin.org/get

# GET with a custom header
curl-cffi get https://httpbin.org/get X-My-Header:value

# POST JSON, `:=` means interpret the value instead of string
curl-cffi post https://httpbin.org/post name=John age:=30

# POST form data
curl-cffi post --form https://httpbin.org/post name=John

# Verbose output (request + response headers and body)
curl-cffi get -v https://httpbin.org/get

# Impersonate chrome by default
curl-cffi get tls.browseleaks.com/json

# Impersonate Safari instead of Chrome
curl-cffi get -i safari tls.browseleaks.com/json

# http3
curl-cffi get --http3 https://fp.impersonate.pro/api/http3

# Localhost shortcut
curl-cffi get :8000/api/health

```

Request Parameters

Request parameters are positional arguments after the URL.

Separator	Type	Example
Header:Value	HTTP header	Content-Type:application/json
Header:	Header removal (empty value)	Accept:
param==value	Query parameter	page==2
field=value	JSON/form field (as string)	name=John
field:=json	JSON field (interpreted)	age:=30 tags:=["a","b"]
@filepath	File upload	@photo.jpg
+key=value	Cookie	+session=abc123

Content type

By default, data fields are serialized as JSON. Use flags to change this:

```

# JSON (default)
curl-cffi post https://httpbin.org/post name=test count:=5

# Form-encoded
curl-cffi post --form https://httpbin.org/post name=test

# Multipart
curl-cffi post --multipart https://httpbin.org/post name=test @file.txt

```

Cookies

Use the + separator to set cookies on the request:

```
# Set a single cookie
curl-cffi get https://httpbin.org/cookies +session=abc123

# Set multiple cookies
curl-cffi get https://httpbin.org/cookies +session=abc123 +theme=dark
```

Options

Content type flags

Flag	Description
--json, -j	Serialize data items as JSON (default)
--form, -f	Serialize data items as form fields
--multipart	Force multipart form data

Output control

When connected to a terminal, the default output includes response headers and body. If rich is installed, JSON, HTML, and XML bodies are syntax highlighted. When piped, only the body is printed as plain text.

Flag	Description
--verbose, -v	Print request headers, request body, response headers, and response body
--quiet, -q	Suppress all output except errors, including download progress
--headers	Print response headers only
--body, -b	Print response body only
--print, -p	Fine-grained control using a string of characters: H (request headers), B (request body), h (response headers), b (response body). Example: --print=Hh for headers only.

```
# Headers only
curl-cffi get --headers https://httpbin.org/get

# Body only (no headers)
curl-cffi get --body https://httpbin.org/get

# Full verbose output
curl-cffi post -v https://httpbin.org/post name=test

# No output on success
curl-cffi get -q https://httpbin.org/get

# Custom: request headers + response headers
curl-cffi get -p Hh https://httpbin.org/get
```

Download

Downloads display a progress bar with transfer speed when `rich` is installed. Without `rich`, the download is written with plain text output. Filenames are automatically sanitized to remove unsafe characters. Use `--quiet` to suppress the response headers, progress, and completion message.

Flag	Description
<code>--download, -d</code>	Download response body to a file (filename from Content-Disposition header or URL path)
<code>--output, -o</code>	Specify output file path

```
# Download a file
curl-cffi get --download https://example.com/file.zip

# Download to a specific path
curl-cffi get --download -o myfile.zip https://example.com/file.zip

# Download without headers or progress output
curl-cffi get --quiet --download -o myfile.zip https://example.com/file.zip
```

Connection options

Flag	Description
<code>--auth, -a</code>	HTTP authentication (user:password)
<code>--verify /</code> <code>--no-verify</code>	Enable/disable SSL certificate verification (default: enabled)
<code>--proxy</code>	Proxy URL
<code>--timeout</code>	Request timeout in seconds
<code>--follow /</code> <code>--no-follow</code>	Follow/don't follow redirects (default: follow)
<code>--max-redirects</code>	Maximum number of redirects (default: 30)

```
# Basic auth
curl-cffi get -a user:password https://httpbin.org/basic-auth/user/password

# Skip SSL verification
curl-cffi get --no-verify https://self-signed.example.com

# Use a proxy
curl-cffi get --proxy http://proxy:8080 https://httpbin.org/get

# Set timeout
curl-cffi get --timeout 5 https://httpbin.org/delay/3
```

Browser impersonation

Flag	Description
<code>--impersonate, -i</code>	Browser to impersonate (default: chrome)

```
# Impersonate Chrome (default)
curl-cffi get https://tls.browerleaks.com/json

# Impersonate Safari
curl-cffi get -i safari https://tls.browerleaks.com/json

# Impersonate Firefox
curl-cffi get -i firefox https://tls.browerleaks.com/json
```

Batch Execution

The run subcommand executes requests from a file. It auto-detects the format by file extension.

```
curl-cffi run requests.http
curl-cffi run session.har
```

Supported formats:

- .http / .rest – HTTP Request in Editor format ([spec](#))
- .har – HAR (HTTP Archive) format

HTTP files

The HTTP Request in Editor format lets you define multiple requests in a single file, separated by ###.

```
### Get user list
GET https://api.example.com/users

### Create a user
POST https://api.example.com/users
Content-Type: application/json

{"name": "Alice", "email": "alice@example.com"}

### Get a specific user
GET https://api.example.com/users/1
```

Features:

- Requests separated by ### (with optional comment text after)
- Request line: [METHOD] URL [HTTP/version] (method defaults to GET)
- Headers follow the request line immediately
- Body after a blank line
- Comments with # or //
- File references with < filepath to include body from a file

```
POST https://api.example.com/upload
Content-Type: application/json

< body.json
```

HAR files

HAR (HTTP Archive) files can be exported from browser DevTools. The `run` command replays all entries in the archive.

```
curl-cffi run captured.har
```

Note

Chrome exported HAR files do not contain cookies and auth headers by default. To include them, go to DevTools settings: Gear Icon > Preferences > Network > select “Allow to generate HAR with sensitive data”.

Session

By default, all requests in a batch share a single session, so cookies and connections persist across requests. To disable this and execute each request independently, use `--no-session`:

```
# Default: shared session (cookies carry over between requests)
curl-cffi run requests.http

# Independent requests (no shared state)
curl-cffi run --no-session requests.http
```

Error handling

When running multiple requests, execution continues even if individual requests fail. A summary of failures is printed at the end:

```
--- [1] GET https://api.example.com/users
--- [2] POST https://api.example.com/missing
--- [3] GET https://api.example.com/health

1 request(s) failed.
```

Doctor

The `doctor` subcommand prints diagnostic information useful for debugging:

```
curl-cffi doctor
```

Example output:

```
curl-cffi doctor
-----
python: 3.10.20
executable: /usr/bin/python3
platform: macOS-15.4-arm64-arm-64bit
machine: arm64
curl_cffi: 0.15.0b4
libcurl: libcurl/8.15.0
api_root: https://api.impersonate.pro/v1
config_path: /Users/you/.config/impersonate/config.json
```

(continues on next page)

(continued from previous page)

```
config_present: True
api_key_configured: True
fingerprint_path: /Users/you/.config/impersonate/fingerprints.json
fingerprint_present: True
fingerprint_count: 42
```

Pro (impersonate.pro)

These subcommands manage fingerprints and API access for [impersonate.pro](#).

Configure API key

Store your impersonate.pro API key in the local config file:

```
curl-cffi config --api-key imp_XXXXXXX
```

Or provide it at runtime with an environment variable:

```
export IMPERSONATE_API_KEY=imp_XXXXXXX
```

Update and list fingerprints

Download the latest fingerprints:

```
curl-cffi update
```

Successful updates print the total number of fingerprints now stored in the local cache.

List native + cached fingerprints in a table:

```
curl-cffi list
```

Output fingerprints as JSON:

```
curl-cffi list --json
```

Environment variables

- `IMPERSONATE_API_KEY`: override the API key loaded from `config.json`.
- `IMPERSONATE_API_ROOT`: override API endpoint used by `update`.
- `IMPERSONATE_CONFIG_DIR`: override local config/cache directory.

curl-cffi vs curl-impersonate vs HTTPie

Why do we need yet another http request tool?

Quick comparison

	curl-cffi	curl-impersonate	HTTPie
Browser impersonation	Yes	Yes	No
HTTP/2	Yes	Yes	No
HTTP/3	Yes	Yes	No
Easier options	Yes	No	Yes
Syntax highlighting	Yes	No	Yes
Batch execution	Yes	No	No

Note

Compared to `curl-impersonate`, `curl-cffi` CLI does not expose all impersonate options in current version, but that is to be updated soon.

curl-cffi vs curl-impersonate

`curl` is the swiss army knife of HTTP, but its syntax can be intimidating for beginners:

```
# curl
curl -X POST https://api.example.com/users \
  -H "Content-Type: application/json" \
  -d '{"name": "Alice", "age": 30}'

# curl-cffi
curl-cffi post https://api.example.com/users name=Alice age:=30
```

`curl-cffi` uses a more readable syntax with request items (`key=value`, `key:=json`, `Header:Value`) instead of flags like `-H` and `-d`.

That said, if you are already comfortable with `curl` syntax, you can use `curl-impersonate` directly for browser impersonation with familiar flags.

curl-cffi vs HTTPie

Syntax

HTTPie has a friendly CLI syntax for HTTP requests, and `curl-cffi` shares many of its ideas. However, there are some key differences:

Explicit methods. `curl-cffi` requires the HTTP method as a subcommand, while HTTPie guesses it based on whether data is present:

```
# HTTPie -- implicit POST because of data fields
http example.com/api name=test

# curl-cffi -- method is always explicit
curl-cffi post example.com/api name=test
```

This explicitness is intentional. It follows the Python philosophy of “explicit is better than implicit”, and it makes `curl-cffi` commands unambiguous when used by AI agents or in scripts, where guessing intent from context is unreliable.

HTTP/2 and HTTP/3

HTTPie only supports HTTP/1.1. curl-cffi supports HTTP/2 and HTTP/3 out of the box, which is critical for browser impersonation and for testing modern APIs.

Browser impersonation

curl-cffi can impersonate browsers' TLS fingerprints, which HTTPie cannot do.

Batch execution

curl-cffi has a builtin `run` command that can execute multiple requests from a single file, supporting both `.http` and `.har` formats.

.http files – define multiple requests in a human-readable format, separated by `###`. Supported by JetBrains IDEs and VS Code REST Client:

```
### Get users
GET https://api.example.com/users

### Create user
POST https://api.example.com/users
Content-Type: application/json

{"name": "Alice"}
```

```
curl-cffi run requests.http
```

.har files – replay HTTP traffic exported from Chrome DevTools. This is especially useful for reproducing real-world requests, including headers, cookies, and POST bodies:

```
curl-cffi run session.har
```

Neither curl nor HTTPie supports batch execution natively. With curl you would need a shell loop or `xargs`; with HTTPie you would need a wrapper script. `curl-cffi run` handles sequencing, error reporting, and a summary of failures out of the box.

See [Batch Execution](#) for full details on supported file formats and options.

When to use which

- **curl-cffi** – when you need browser impersonation, HTTP/2/3 support, or a CLI that works well with AI agents and scripts.
- **curl / curl-impersonate** – when you are already fluent in curl syntax and want the full power of libcurl.
- **HTTPie** – when you only need HTTP/1.1 and prefer implicit method detection.

IMPERSONATE GUIDE

You probably come across `curl_cffi` for its ability to impersonate browsers. Here is a tutorial on how to better impersonate using `curl_cffi`.

4.1 What is TLS and http/2, http/3 fingerprinting?

4.1.1 TLS and http/2

TLS is the `s` in `https`. `https` has been uniformly deployed across the world. There are many extension and cipher suites a implementation can choose to use. According to the RFC, there are many valid combinations. But in reality, browser vendors tend to use fixed combinations, and these combinations can be used identify if the request is from a certain browser or an automated script. The digest of this combination is called a TLS fingerprints. The most common digesting method is called JA3.

Similar to TLS, there are a few settings in `http/2` connection can be used to identify the source of a request. The most commonly used digesting method is proposed by Akamai, and called the Akamai `http2` fingerprint.

To learn the details of TLS and `http2` fingerprinting, you can read these great articles from `lwthiker`:

1. <https://lwthiker.com/networks/2022/06/17/tls-fingerprinting.html>
2. <https://lwthiker.com/networks/2022/06/17/http2-fingerprinting.html>

The format of JA3 and Akamai digest is briefly discussed below.

4.1.2 http/3

As of `http/3`, the newest version of `http`. Basically, it's `http/2` reimplemented over QUIC, thus it can be fingerprinted in a similar way with `http/2`.

`Http/3` fingerprints has not yet been publicly exploited and reported. But given the rapidly increasing marketshare of `http/3`(35% of internet traffic), it is expected that some strict WAF vendors have begun to utilize `http/3` fingerprinting.

It has also been noticed by many users, that, for a lot of sites, there is less or even none detection when using `http/3`.

To check your browser's `http3` & `quic` fingerprints, you can visit our [http/3 & quic fingerprints API](#) page.

`curl_cffi` provides TLS and `http/2` impersonation and `http/3` protocol. `Http/3` fingerprints support and UDP proxy support has been added in `v0.15`.

4.2 JA4 support in curl-cffi

This page is intentionally opinionated.

JA4 is often presented as the natural successor to JA3, but for browser impersonation and practical protocol debugging, it is mostly a repackaging of things we already had, plus a few design choices that make analysis less clear, not more.

The name is misleading, it has nothing to do with JA3 or its creators. If your goal is exact impersonation, reproducibility, and debugging of TLS client hellos, simply set the `ja3=...` and `extra_fp=...` parameters in `curl_cffi`, which already covers what `ja4` claims to cover.

This page assumes that you have learned what JA4 is. If otherwise, just don't use it.

4.2.1 What JA3 Already Gives You

JA3 is not comprehensive, but it does its job: a digest of TLS client hello.

The original JA3 method from Salesforce takes a client hello and builds a canonical text value from:

- TLS version
- cipher suites
- extension list
- supported groups
- EC point formats

That text value is then hashed with MD5 easier matching.

This matters because JA3 has a clean separation between two different use cases:

1. The raw text value is for humans, debugging, and reproduction.
2. The hash is for compact indexing and correlation.

That split is simple and useful.

- When a request is blocked, you can compare the JA3 text values, it's easier for human eyes.
- When you need a small key for a database, the hash is there.

However, JA3 is not complete, and it never claimed to be. As time goes by, some extra bits in the TLS hello packet have been utilized to distinguish between bots and humans. And that's why we have an `extra_fp` parameter.

4.2.2 What JA4 Adds, and Why That Is Not Very Useful

The official JA4 material says the new format is more human readable, supports QUIC, exposes ALPN, and allows partial matching via the `a_b_c` structure. Those are real changes, but they are much less valuable than the marketing around them suggests.

The JA4 propaganda says that JA3 is obsolete because Chrome introduces the extension permutation in version 110+, so JA3 does not make sense anymore.

This is simply overstating. To counter TLS extension permutation, just use `ja3n`, sorting the extension is not a big deal as JA4 authors said.

Being able to say “these two fingerprints share the same a+c parts” may be useful in a threat-hunting product. It is not especially useful if you are trying to answer questions like:

- Why did this browser fingerprint stop working?
- Which extension changed?
- Did the ALPN change, or the signature algorithms, or both?
- What exact client hello do I need to reproduce?

In those situations, the right answer is not another derived label. The right answer is the actual structured handshake data.

So JA4 is not even an upgrade from JA3. It is mostly a different tradeoff, presented as a universal improvement.

4.2.3 Mixing Text and Hashes Is a Bad Design

The worst design choice in JA4 is that it mixes readable text and hashed segments into a single first-class value.

The result looks tidy:

```
t13d1516h2_8daaf6152771_02713d6af862
```

But this tidiness is deceptive.

The first segment is somewhat interpretable by a human. The later segments are opaque hashes. So the value is neither properly human readable nor properly machine-minimal. It sits in the worst middle ground:

- too encoded for direct debugging
- too lossy for reproduction
- too clever for casual inspection

Compare that with JA3:

- raw text for inspection
- hash for indexing

That design is boring, and that is exactly why it is good.

If you see two JA3 text values, you can compare them directly. If you see two JA4 values and one hashed segment changes, you learn almost nothing without a second decoding step or an external lookup table.

This is especially annoying in debugging. Suppose a target starts rejecting a fingerprint after a browser update. With JA3-style raw text, you can look at the changed cipher list or extension list and reason about the delta. With JA4, the pretty-looking value hides the interesting part behind hashes.

That is not “more readable”. It is more confusing, for both humans and machines.

The sane design would have been:

1. define a normalized text representation and expose that text directly,
2. optionally hash it for compact storage.

JA3 already had that basic idea right, and for permutation, just use ja3n

JA4 instead tries to make one value serve as both explanation and identifier. That is usually a mistake. Explanation and identification are different jobs.

4.2.4 Conclusion

JA4 is not worthless. If your only goal is coarse clustering in a defensive analytics pipeline, some of its normalization choices may be convenient.

But convenience for clustering is not the same as usefulness for impersonation, protocol debugging, or exact client reproduction.

JA3 already gave us a simple and honest model:

- here is the concrete handshake text
- here is a compact hash of it

JA4 adds structure, but forcing human to read hashes is simply not a very clever abstraction.

If you want to understand or reproduce a TLS fingerprint, you want the handshake packet, not a buzzword. JA3 is a useful human-readable format, and there is not point in using the JA4 string.

`curl_cffi` focuses on the impersonation of the entire hello packet, not a representation string. So to answer is `ja4` supported in `curl_cffi`?

- Yes, by supporting the entire packet, `ja4` is just part of it.
- No, we do not accept `ja4` as a input string, because it's a hash and not parsable.

4.2.5 References

- [JA3: A method for profiling SSL/TLS Clients](#)
- [TLS Fingerprinting with JA3 and JA3S](#)
- [JA4+ Network Fingerprinting](#)
- [FoxIO JA4 repository](#)

4.3 Supported browser impersonate targets

`curl_cffi` supports the same browser versions as supported by our [fork of curl-impersonate](#).

Browser versions will be added **only** when their fingerprints change. If you see a version, e.g. `chrome122`, was skipped, you can simply impersonate it with your own headers and the previous version.

If you are too busy to look up those details, you can try our commercial version at [impersonate.pro](#), which has a weekly updated list of browser profiles and even more browser types.

If you are trying to impersonate a target other than a browser, use `ja3=...`, `akamai=...` and `extra_fp=...` to specify your own customized fingerprints. See below for details.

- `chrome99`
- `chrome100`
- `chrome101`
- `chrome104`
- `chrome107`
- `chrome110`
- `chrome116` ¹
- `chrome119` ¹
- `chrome120` ¹
- `chrome123` ³
- `chrome124` ³
- `chrome131` ⁵
- `chrome133a` ^{5 6}
- `chrome136` ⁸
- `chrome142` ¹¹
- `chrome145` ^{13 14}
- `chrome146` ^{13 14}
- `chrome150` ^{14 15}
- `chrome151` ^{14 16}

- chrome151_macos ^{14 16}
- chrome152 ^{14 16}
- chrome152_macos ^{14 16}
- chrome99_android
- chrome131_android ⁵
- edge99
- edge101
- safari153 ^{2 9}
- safari155 ^{2 9}
- safari170 ^{1 9}
- safari172_ios ^{1 9}
- safari180 ^{4 9}
- safari180_ios ^{4 9}
- safari184 ^{8 9}
- safari184_ios ^{8 9}
- safari260 ^{9 10}
- safari260_ios ^{9 10}
- safari2601 ^{9 11}
- safari270 ^{14 16}
- safari270_ios ^{14 16}
- firefox133 ⁵
- firefox135 ⁷
- firefox144 ^{11 12}
- firefox147 ^{13 14}
- firefox153 ¹⁶
- firefox154 ¹⁶
- firefox155 ¹⁶
- tor145 ⁸

Notes:

1. Added in version 0.6.0.
2. Fixed in version 0.6.0, previous http2 fingerprints were `not correct`.
3. Added in version 0.7.0.
4. Added in version 0.8.0.
5. Added in version 0.9.0.
6. The version suffix a (e.g. chrome133a) means that this is an alternative version, i.e. the fingerprint has not been officially updated by browser, but has been observed because of A/B testing.

7. Added in version 0.10.0.
8. Added in version 0.11.0
9. Since 0.11.0, the format safari184_ios is preferred over safari18_4_ios, both are supported, but the latter is quite confusing and hard to parse.
10. Added in version 0.12.0
11. Added in version 0.14.0.
12. Fixed in version 0.15.0, previous User-Agent header was `not correct`.
13. Added in version 0.15.0.
14. http3 support included.
15. Added in version 0.16.1.
16. Added in version 0.17.0b1 (curl-impersonate 2.3.0).

4.4 Which target version to use?

Generally speaking, you should use the latest Chrome or Safari versions. Currently, they're `chrome152`, `safari270` and `safari270_ios`. To always impersonate the latest available browser versions, you can simply use `chrome`, `firefox`, `safari` and `chrome_android`, `safari_ios`.

```
import curl_cffi

curl_cffi.get(url, impersonate="chrome")
```

Chrome 151 and 152 have separate Windows targets (`chrome151`, `chrome152`) and macOS targets (`chrome151_macos`, `chrome152_macos`). The `chrome` alias selects the Windows profile. Firefox 153, 154 and 155 do not include dedicated HTTP/3 fingerprints.

Tips:

iOS has restrictions on WebView and TLS libs, so `safari*_ios` should work for a lot of apps. If you encountered an android app with custom fingerprints, you can try the `safari_ios` fingerprints, given that this app should have an iOS version.

4.5 How to use my own fingerprints? e.g. okhttp

Use `ja3=...`, `akamai=...` and `extra_fp=...`.

You can retrieve the JA3 and Akamai strings using tools like WireShark or from TLS fingerprinting sites.

```
# OKHTTP impersonation examples
# credits: https://github.com/bogdanfinn/tls-client/blob/master/profiles/contributed_
↳ custom_profiles.go

url = "https://tls.browseleaks.com/json"

okhttp4_android10_ja3 = ",".join(
    [
        "771",
        "4865-4866-4867-49195-49196-52393-49199-49200-52392-49171-49172-156-157-47-53",
        "0-23-65281-10-11-35-16-5-13-51-45-43-21",
```

(continues on next page)

(continued from previous page)

```

        "29-23-24",
        "0",
    ]
)

okhttp4_android10_akamai = "4:16777216|16711681|0|m,p,a,s"

extra_fp = {
    "tls_signature_algorithms": [
        "ecdsa_secp256r1_sha256",
        "rsa_pss_rsae_sha256",
        "rsa_pkcs1_sha256",
        "ecdsa_secp384r1_sha384",
        "rsa_pss_rsae_sha384",
        "rsa_pkcs1_sha384",
        "rsa_pss_rsae_sha512",
        "rsa_pkcs1_sha512",
        "rsa_pkcs1_sha1",
    ]
    # other options:
    # tls_min_version: int = CurlSslVersion.TLSv1_2
    # tls_grease: bool = False
    # tls_permute_extensions: bool = False
    # tls_cert_compression: Literal["zlib", "brotli"] = "brotli"
    # tls_signature_algorithms: Optional[List[str]] = None
    # http2_stream_weight: int = 256
    # http2_stream_exclusive: int = 1

    # See requests/impersonate.py and tests/unittest/test_impersonate.py for more
    # examples
}

r = curl_cffi.get(
    url, ja3=okhttp4_android10_ja3, akamai=okhttp4_android10_akamai, extra_fp=extra_fp
)
print(r.json())

```

4.5.1 JA3 and Akamai String Format

A JA3 string is a simple, comma-separated representation of the key fields in a TLS ClientHello. It consists of five parts:

- SSL/TLS Version, The numeric version the client requests (e.g. 771 for TLS 1.2).
- Cipher Suites, A hyphen-separated list of all cipher suite IDs offered by the client (e.g. 4865-4866-4867-49195-49196).
- Extension IDs, A hyphen-separated list of all TLS extension numbers the client includes (e.g. 0-11-10-35-16-5).
- Supported Groups (aka “Elliptic Curves”), A hyphen-separated list of curve IDs the client supports for ECDHE (e.g. 29-23-24).
- EC Point Formats, A hyphen-separated list of the point-format IDs (almost always just 0 for “uncompressed”) (e.g. 0).

They're concatenated in that exact order, with commas between fields. For example:

```
771,4865-4866-4867-49195-49196,0-11-10-35-16-5,29-23-24,0
```

Note that Chrome permutes the extension order on each request, so there is a new format called JA3N, which uses sorted `extension_id` list.

The Akamai HTTP/2 fingerprint string encodes four client-controlled protocol parameters, joined by the pipe character (`|`):

- **SETTINGS**, A semicolon-separated list of ID:value pairs from the client's initial SETTINGS frame. Each ID is a standard HTTP/2 setting identifier (e.g. 1 for `HEADER_TABLE_SIZE`, 4 for `INITIAL_WINDOW_SIZE`), and value is the client's chosen value for that setting
- **WINDOW_UPDATE**, A single integer: the value the client sends in its first WINDOW_UPDATE frame (or 0 if none was sent)
- **PRIORITY**, Zero or more priority-frame tuples, each formatted as `StreamID:ExclusiveBit:DependentStreamID:Weight`. Multiple tuples are comma-separated. This captures any PRIORITY frames the client issues before sending headers
- **Pseudo-Header Order**, The sequence in which the client sends HTTP/2 pseudo-headers in its request HEADERS frame, encoded as comma-separated single-letter codes:

Putting it all together, an example fingerprint might look like:

```
1:65536;4:131072;5:16384|12517377|3:0:0:201|m,p,a,s
```

where:

```
SETTINGS = 1:65536;4:131072;5:16384
```

```
WINDOW_UPDATE = 12517377
```

```
PRIORITY = 3:0:0:201
```

```
Pseudo-Header Order = m,p,a,s
```

Although JA3 and Akamai fingerprint string already captures many of the aspects of a Hello Packet, there are still some fields are not covered and can be used to detect you. This is when the `extra_fp` option comes in, each field of this dict is pretty easy to understand. You should first set the `ja3` and `akamai` string, then check if you have the identical fingerprint like your target. If not, use the `extra_fp` to further refine your impersonation.

For example, to customize regular HTTP header ordering:

```
extra_fp = {  
    "header_order": "User-Agent,Host,Connection"  
}
```

4.5.2 Using CURLOPTs

The other way is to use the `curl_opt`s to specify exactly which options you want to change.

To modify them, use `curl.setopt(Curlopt, value)`, for example:

```
import curl_cffi  
from curl_cffi import Curl, Curlopt  
  
c = Curl()  
c.setopt(Curlopt.HTTP2_PSEUDO_HEADERS_ORDER, "masp")
```

(continues on next page)

(continued from previous page)

```
# or
curl_cffi.get(url, curl_options={CurlOpt.HTTP2_PSEUDO_HEADERS_ORDER, "masp"})
```

Here are a list of options:

For TLS/JA3 fingerprints:

- https://curl.se/libcurl/c/CURLOPT_SSL_CIPHER_LIST.html

and non-standard TLS options created for this project:

- CURLOPT_SSL_ENABLE_ALPS
- CURLOPT_SSL_SIG_HASH_ALGS
- CURLOPT_SSL_CERT_COMPRESSION
- CURLOPT_SSL_ENABLE_TICKET
- CURLOPT_SSL_PERMUTE_EXTENSIONS

For Akamai http2 fingerprints, you can fully customize the 3 parts:

- CURLOPT_HTTP2_PSEUDO_HEADERS_ORDER, sets http2 pseudo header order, for example: masp (non-standard HTTP/2 options created for this project).
- CURLOPT_HTTP2_SETTINGS sets the settings frame values, for example 1:65536;3:1000;4:6291456;6:262144 (non-standard HTTP/2 options created for this project).
- CURLOPT_HTTP2_WINDOW_UPDATE sets initial window update value for http2, for example 15663105 (non-standard HTTP/2 options created for this project).

For a complete list of options and explanation, see the [curl-impersonate README](#).

4.5.3 QUIC and HTTP/3 options

These options customize HTTP/3 requests and their QUIC connections. Select `http_version="v3"` to allow HTTP/3 with fallback, or `http_version="v3only"` to require HTTP/3. Set the options through `curl_options` or `curl.setopt(CurlOpt.OPTION, value)`.

QUIC connection options:

- `CurlOpt.QUIC_TRANSPORT_PARAMETERS` (string) sets transport parameter IDs, values, and their order, using semicolon-separated decimal `id:value` pairs, for example "3:1200;4:1048576;15:AUTO". 15:AUTO inserts the connection's initial source connection ID; 15: sends an empty source connection ID. A GREASE token inserts a randomized GREASE parameter. Parameter order is randomized when HTTP/3 TLS extension permutation is enabled.
- `CurlOpt.QUIC_CID_LENGTH` (string) selects initial connection ID lengths. "webkit" uses an 8-byte destination connection ID and an empty source connection ID. "firefox" randomizes the destination connection ID length from 8 to 20 bytes and uses a 3-byte source connection ID.
- `CurlOpt.QUIC_INITIAL_PACKET_NUMBER` (integer) sets the initial packet number for each new QUIC connection. Values from 0 to 2147483647 select a fixed number; -1 uses Firefox-style randomization with Neqo's biased distribution from 1 to 1024. The libcurl default is 0, which an impersonation profile may override. Values outside this range raise `CurlError` with code `CurlErrorCode.BAD_FUNCTION_ARGUMENT`.

HTTP/3 TLS options:

- `CurlOpt.HTTP3_SIG_HASH_ALGS` (string) sets comma-separated signature algorithms, for example "rsa_pss_rsae_sha256,ecdsa_secp256r1_sha256". When set, it overrides the shared TLS signature algorithms for QUIC connections.

- `CurlOpt.HTTP3_TLS_EXTENSION_ORDER` (string) sets a hyphen-separated sequence of TLS extension IDs, for example "0-10-13-16-43-45-51-57". When set, it overrides `CurlOpt.TLS_EXTENSION_ORDER` for QUIC connections.
- `CurlOpt.HTTP3_SSL_EC_CURVES` (string) sets colon-separated TLS key exchange groups, for example "X25519:P-256". When set, it overrides `CurlOpt.SSL_EC_CURVES` for QUIC connections.
- `CurlOpt.HTTP3_SSL_PERMUTE_EXTENSIONS` (integer) controls HTTP/3 TLS extension and QUIC transport parameter permutation: 0 disables it, 1 enables it, and -1 follows `CurlOpt.SSL_PERMUTE_EXTENSIONS`. The default is -1.

HTTP/3 settings and headers:

- `CurlOpt.HTTP3_SETTINGS` (string) sets HTTP/3 SETTINGS frame IDs and values, using semicolon-separated id:value pairs, for example "1:65536;6:262144".
- `CurlOpt.HTTP3_PSEUDO_HEADERS_ORDER` (string) sets pseudo-header order using m (:method), a (:authority), s (:scheme), and p (:path), for example "masp".
- `CurlOpt.HTTP3_HTTPHEADER` (list of bytes) sets HTTP/3-specific headers, for example [b"User-Agent: custom-agent", b"Accept: */*"]. When set, this list replaces `CurlOpt.HTTPHEADER` for HTTP/3 requests.
- `CurlOpt.HTTP3_HTTPHEADER_ORDER` (string) sets comma-separated regular header order, for example "User-Agent,Accept". When set, it overrides `CurlOpt.HTTPHEADER_ORDER` for HTTP/3 requests.

For example, to customize an impersonation profile's QUIC connection behavior:

```
from curl_cffi import CurlOpt, requests

response = requests.get(
    url,
    impersonate="firefox",
    http_version="v3only",
    curl_options={
        CurlOpt.QUIC_CID_LENGTH: "firefox",
        CurlOpt.QUIC_INITIAL_PACKET_NUMBER: -1,
        CurlOpt.HTTP3_SSL_PERMUTE_EXTENSIONS: 0,
    },
)
```

The `perk` request parameter combines HTTP/3 settings, pseudo-header order, and QUIC transport parameters as `settings|header_order|transport_parameters`, for example "1:65536;6:262144|m,a,s,p|3:1200;4:1048576;15:AUTO".

`Fingerprint` exposes these options through `http3_settings`, `http3_pseudo_headers_order`, `http3_tls_extension_order`, `http3_headers`, `http3_header_order`, `http3_tls_supported_groups`, `quic_transport_parameters`, and `quic_cid_length`. The `quic_initial_packet_number` field is available on both `Fingerprint` and `ExtraFingerprints`, and in `extra_fp` dictionaries:

```
extra_fp = {"quic_initial_packet_number": -1}
```

Leaving `quic_initial_packet_number` unset or setting it to `None` preserves the impersonation profile's default. `ExtraFingerprints` and `extra_fp` also accept `http3_sig_hash_algs` and `http3_tls_extension_order`.

4.5.4 How to toggle firefox-specific extensions?

There are a few special extensions from firefox that you should add extra options by `extra_fp`:

Extension 34: delegated credentials

```
extra_fp = {
    "tls_delegated_credential": "ecdsa_secp256r1_sha256:ecdsa_secp384r1_sha384:ecdsa_
↪secp521r1_sha512:ecdsa_sha1"
}

# Note that the ja3 string also includes extension: 34
ja3 = "771,4865-4867-4866-49195-49199-52393-52392-49196-49200-49162-49161-49171-49172-
↪156-157-47-53,0-23-65281-10-11-35-16-5-34-18-51-43-13-45-28-27-65037,4588-29-23-24-25-
↪256-257,0"

r = curl_cffi.get(url, ja3=ja3, extra_fp=extra_fp)
```

Extension 28: record size limit

```
extra_fp = {
    "tls_record_size_limit": 4001
}

# Note that the ja3 string also includes extension: 28
ja3 = "771,4865-4867-4866-49195-49199-52393-52392-49196-49200-49162-49161-49171-49172-
↪156-157-47-53,0-23-65281-10-11-35-16-5-34-18-51-43-13-45-28-27-65037,4588-29-23-24-25-
↪256-257,0"

r = curl_cffi.get(url, ja3=ja3, extra_fp=extra_fp)
```

4.6 TLS PSK(41) Extension

What is the TLS PSK(41) extension, how to deal with it?

PSK is short for Pre-Shared Key, as defined in [RFC 8446](#),

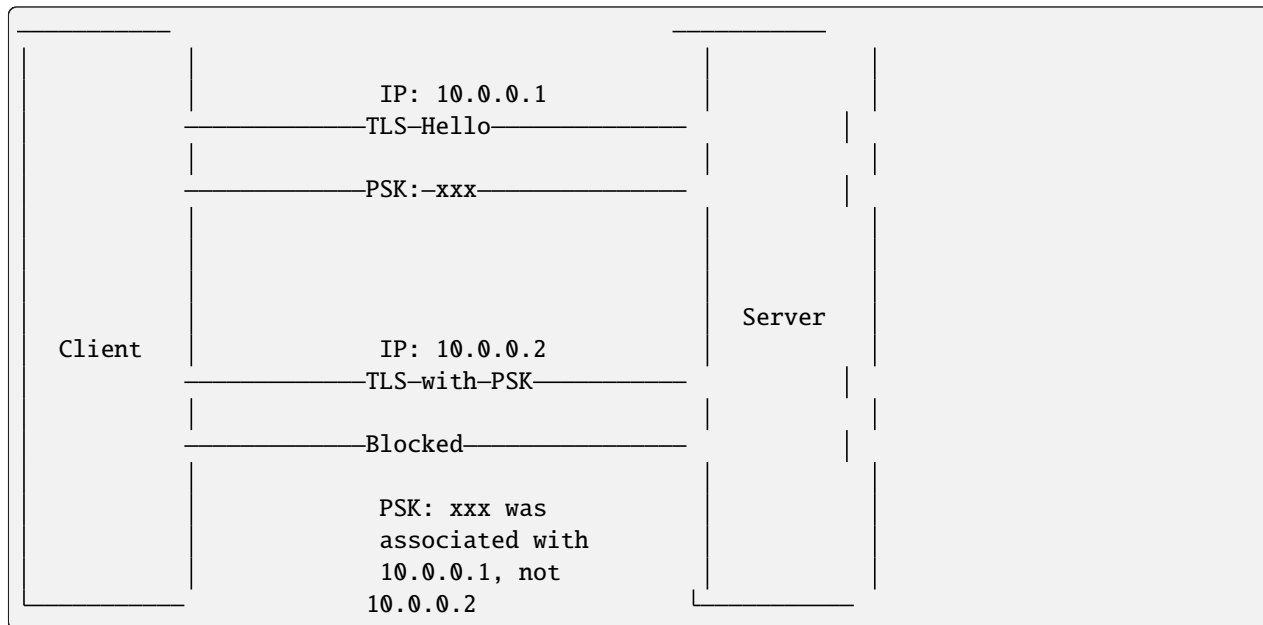
Once a handshake has completed, the server can send the client a PSK identity that corresponds to a unique key derived from the initial handshake (see Section 4.6.1). The client can then use that PSK identity in future handshakes to negotiate the use of the associated PSK.

Usually, when you first visit a website, the PSK extension is not present in the extension list. But when you visit the same website for the second time, in a relatively short time, the client may offer a PSK extension with the key from the server.

For example, you can visit <https://tls.peet.ws/api/all>, and then refresh the page, the PSK extension will be there.

To correctly implement the PSK extension, the client must have some kind of a session cache held in memory or persisted on disk. All the major browsers have this feature for a very long time. `curl_cffi` added this feature in version 0.11.0, with libcurl 8.13.0.

The mechanism and behavior of a PSK looks like an http session cookie, where the server sent a cryptographic value as a key to resume a previous disconnected session. When the server generates a PSK, it is possible that the server keeps the mapping between the incoming IP and key. Thus, it can be problematic if you reuse a TLS session with rotating proxies.



Luckily, since curl_cffi 0.12.0, we added a new option called `proxy_credential_no_reuse`, when enabled, the TLS session cache will be bound based on the proxy username and IP, such that the session can only be reused when the proxy username and IP matches. From the server's viewpoint, the Pre-Shared Key will be locked to the same source IP, not bouncing around among different exit nodes.

```
# Python example to be added.
# We might enable this by default when proxies are used.
```

4.6.1 How do I enable PSK extension anyway?

You don't. If you haven't, please read the explanation above first. Generally speaking, the client should manage this extension, and it should automatically offer this extension on the second request.

From the server's perspective, if you forcefully add a PSK extension with random value, it's an obvious sign that you are not a valid visitor, just like you providing an invalid cookie value.

However, it's reasonable that you don't want the PSK extension to be sent, i.e. pretending to be a first time visitor. We don't support this for now, your option is to use an older version of curl_cffi or create a new session on each request.

Note, some other impersonation-oriented http clients give you the control over adding the PSK or not, but you should let the client decide, if you are trying to impersonating browsers.

References:

1. <https://github.com/icing/blog/blob/main/curl-sessions-earlydata.md>

4.7 Impersonation FAQ

4.7.1 How to check if my impersonation is working?

The most reliable way is to use WireShark, and compare packets from curl_cffi and your targets.

If it's challenging for you to use WireShark, you can use the following sites for JA3 and Akamai fingerprints:

- <https://fp.impersonate.pro>

There are other sites that do the similar thing, but we cannot guarantee their terms work for curl_cffi.

1. <https://tls.browerleaks.com/json>
2. <https://tls.peet.ws/api/all>
3. <https://scrapfly.io/web-scraping-tools/browser-fingerprint>

For http/3 fingerprints, use our service, too:

1. <https://fp.impersonate.pro/api/http3>

4.7.2 I'm still being detected even if I impersonated correctly

First, JA3 and akamai fingerprints are not comprehensive, there are other fields that can be detected, we have a few more options listed in `extra_fp`. Be sure to also check them.

Note

Since `curl-impersonate` was posted on [Hacker News](#), some features and behaviors of `curl_cffi` is being detected by professional players. If we continue to fix these niche behavior in public, it would soon be noticed by those providers.

In short, if you are using `curl_cffi` in production and you are sure about being blocked by TLS or http detection, try the `curl_cffi pro` version.

Since 0.15.1, use the new `Fingerprint` class to set every bit we supported.

4.7.3 Should I randomize my fingerprints for each request?

You can choose a random version from the list above, like:

```
random.choice(["chrome119", "chrome120", ...])
```

However, be aware of the browser market share, very old versions are not good choices.

Generally, **you should not try to generate a customized random fingerprints**. The reason is that, for a given browser version, the fingerprints are fixed. If you create a new random fingerprints, the server is easy to know that you are not using a typical browser.

If you were thinking about `ja3`, and not `ja3n`, then the fingerprints is already randomized, due to the `extension permutation` feature introduced in Chrome 110.

As far as we know, most websites use an allowlist, not a blocklist to filter out bot traffic. So do not expect random `ja3` fingerprints would work in the wild.

Moreover, **do not generate random ja3 strings**. There are certain limits for a valid `ja3` string. For example:

- TLS 1.3 ciphers must be at the front.
- GREASE extension must be the first.
- etc.

You should copy `ja3` strings from sniffing tools, or from <https://fp.impersonate.pro> as we mentioned above. Do not generate them, unless you can make sure all the requirements are met.

4.7.4 Can I change JavaScript fingerprints with this library?

No, you can not. As the name suggests, JavaScript fingerprints are generated using JavaScript APIs provided by real browsers. `curl_cffi` is a python binding to a C library, with no browser or JavaScript runtime under the hood.

If you need to impersonate browsers on the JavaScript perspective, you can search for “Anti-detect Browser”, “Playwright stealth” and similar keywords. Or simply use a commercial plan from our sponsors.

4.7.5 Why are all the User-Agents macOS?

Simple, because I primarily use macOS and I copied the headers from my own browser. Fingerprints are generally the same across desktop OSes, if you want it to look like Windows, just update the user-agent and other related headers to Windows.

ADVANCED TOPICS

5.1 DNS servers

Use `dns` to select a DNS server or a list of servers instead of the system resolver. This requires libcurl to be built with c-ares; it is unavailable in builds that use a threaded or system resolver. `AsyncSession` accepts the same session-level and request-level `dns` option.

```
from curl_cffi import Session

with Session(dns=["8.8.8.8", "8.8.4.4"]) as session:
    response = session.get("https://example.com")
    response = session.get("https://example.com", dns="1.1.1.1")
```

An empty string or list uses the system resolver. Server addresses may include a port, for example `"8.8.8.8:5353"`. This option can boost your performance, according to some users who have already used this before.

5.2 Proxies

You can use the `proxy` parameter:

```
import curl_cffi

curl_cffi.get(url, proxy="http://user:pass@example.com:3128")
```

You can also use the `http_proxy`, `https_proxy`, and `ws_proxy`, `wss_proxy` environment variables, respectively. For HTTP requests, set `trust_env=False` on `Session` or `AsyncSession` to ignore proxy environment variables:

```
from curl_cffi import Session

with Session(trust_env=False) as session:
    response = session.get("https://example.com")
```

Explicit `proxy=` and `proxies=` settings still apply. With `trust_env=False`, `no_proxy` also cannot bypass an explicit proxy.

Warning

For beginners, a very common mistake is to add `https://` prefix to the `https` proxy.
For explanation of differences between `http_proxy` and `https_proxy`, please see #6.

For compatibility with `requests`, we also support using dicts.

```
import curl_cffi

proxies = {
    "http": "http://localhost:3128",
    "https": "http://localhost:3128"
}
curl_cffi.get(url, proxies=proxies)
```

Note

Prefer the single *proxy* parameter, unless you do have different proxies for http and https

5.2.1 HTTP/3 over proxies

With `http_version="v3"` or `"v3only"`, a `socks5://` proxy carries QUIC via UDP ASSOCIATE, and an `https://` proxy via MASQUE CONNECT-UDP.

To talk HTTP/3 to the proxy itself, set its type to `CurlProxyType.HTTPS3`:

```
import curl_cffi
from curl_cffi import CurlOpt, CurlProxyType

curl_cffi.get(
    url,
    proxy="https://proxy.example.com:443",
    http_version="v3only",
    curl_options={CurlOpt.PROXYTYPE: CurlProxyType.HTTPS3},
)
```

5.3 Response cache

`Session` can cache successful responses on disk. This is especially useful for tests and local mocks: the first request records a real response, and later requests with the same method, URL and request body are served from the cache.

The simplest form is to pass a number of seconds or a `timedelta`:

```
from datetime import timedelta
from curl_cffi import Session

with Session(cache=timedelta(minutes=10)) as s:
    first = s.get("https://example.com/api/users")
    second = s.get("https://example.com/api/users") # served from cache
```

By default, cached files are stored in a `curl_cffi_cache` directory under the system temporary directory. Use `FileCacheBackend` when you want a stable cache directory, for example inside a test fixture:

```
from datetime import timedelta
from curl_cffi import FileCacheBackend, Session

cache = FileCacheBackend(
```

(continues on next page)

(continued from previous page)

```

    path="tests/.cache/http",
    expires=timedelta(hours=1),
)

with Session(cache=cache) as s:
    response = s.get("https://api.example.test/items")

```

Only successful responses are stored by default. Cached responses preserve the response body, status code, headers, cookies and common curl metadata. On a cache hit, response cookies are also merged into the session cookie jar, matching normal Session behavior.

5.3.1 Ignoring query parameters

You can ignore volatile query parameters when building cache keys. This is useful for analytics, cache-busting or nonce parameters that should not create separate cache entries.

```

from datetime import timedelta
from curl_cffi import FileCacheBackend, Session

cache = FileCacheBackend(
    expires=timedelta(minutes=10),
    ignored=["utm_source", "cache_buster"],
)

with Session(cache=cache) as s:
    s.get("https://example.com/items?item=1&utm_source=a")
    s.get("https://example.com/items?item=1&utm_source=b") # same cache entry

```

5.3.2 Clearing cache files

Call `clear` to remove cache entries managed by that backend:

```
cache.clear()
```

`FileCacheBackend.clear` only removes cache-looking files from its own cache directory.

When the session creates the backend from a duration, access it through the read-only `cache_backend` property:

```

from curl_cffi import Session

with Session(cache=600) as session:
    response = session.get("https://example.com")
    session.cache_backend.clear()

```

`cache_backend` is `None` when response caching is disabled. This cache stores HTTP responses; `CurlShare` shares DNS and TLS session data separately.

5.3.3 Custom backends

To store responses somewhere other than the filesystem, subclass `CacheBackend` and implement the storage methods:

```

from datetime import timedelta
from curl_cffi import CacheBackend, Session

```

(continues on next page)

(continued from previous page)

```
class DictCache(CacheBackend):
    def __init__(self):
        super().__init__(expires=timedelta(minutes=10))
        self.data = {}

    def _read_payload(self, key):
        return self.data.get(key)

    def _write_payload(self, key, payload):
        self.data[key] = payload

    def _delete_payload(self, key):
        self.data.pop(key, None)

    def clear(self):
        self.data.clear()

with Session(cache=DictCache()) as s:
    response = s.get("https://example.com/api")
```

5.3.4 Limitations

- The built-in cache currently works with synchronous `Session` only. `AsyncSession` rejects the cache option because the backend API is blocking.
- Streamed responses and requests using `content_callback` are not cached.
- The default key uses request method, normalized URL and request body. Be careful with authenticated or cookie-specific responses; use a custom backend or a separate cache directory when different users may receive different content for the same URL.

5.4 Low-level curl API

Although we provide an easy to use `requests`-like API, sometimes, you may prefer to use the `curl`-like API.

The `curl` API is very much like what you may have used – `pycurl`, with extra impersonation support.

```
from curl_cffi import Curl, CurlOpt
from io import BytesIO

buffer = BytesIO()
c = Curl()
c.setopt(CurlOpt.URL, b'https://tls.browserleaks.com/json')
c.setopt(CurlOpt.WRITEDATA, buffer)

c.impersonate("chrome124")

c.perform()
c.close()
body = buffer.getvalue()
print(body.decode())
```

For a complete list of options, see [API References](#)

5.5 Using CURLOPT_* in requests API

Sometimes, you know an option from libcurl, but we haven't exposed it in the requests API. You can simply add the `curl_options` dict to apply the option.

```
from curl_cffi import CurlHttpVersion, CurlOpt, requests

response = requests.get(
    "https://example.com",
    curl_options={CurlOpt.HTTP_VERSION: CurlHttpVersion.V1_1},
)
```

Note

Using `curl_options` is preferred over using `session.curl.setopt`, the latter may get overridden internally, while the former is executed after all options have been set.

5.6 Selecting http version

The recommended and default http version is http/2, the present and most widely used http version as of 2025.

According to [Wikipedia](#), the marketshare is:

- HTTP/1.1, 33.8%
- HTTP/2, 35.3%
- HTTP/3, 30.9%

To change http versions, use the `http_version` parameter.

```
import curl_cffi
curl_cffi.get("https://cloudflare-quic.com", http_version="v3")
```

Common values are: `v1`, `v2`, `v3` and `v3only`.

To get the actual used http version, you need to compare the response field with `const` from libcurl:

```
>>> from curl_cffi import CurlHttpVersion
>>> r = curl_cffi.get("https://example.com", http_version="v2")
>>> r.http_version == CurlHttpVersion.V2_0
True
```

5.7 Keeping session alive in http/2

With http/2, you can optionally send a ping frame to keep the connection alive when not actively using it.

```
from curl_cffi import Session

s = Session()
s.get("https://example.com")
s.upkeep()
```

`AsyncSession` also exposes `upkeep` for idle connections in its pool:

```
from curl_cffi import AsyncSession

async with AsyncSession() as s:
    await s.get("https://example.com")
    await s.upkeep()
```

5.8 Connection reuse across threads

A synchronous `Session` keeps one curl handle per thread (`use_thread_local_curl`, on by default), because a curl handle is not safe to use from several threads at once. The connection pool lives inside each handle, so connections — and options that act on the pool such as `MAXCONNECTS` — are only reused *within* a thread. Reusing the same `Session` from a thread pool therefore opens at least one connection per thread.

libcurl does **not** support sharing the connection cache between concurrent threads, so there is no safe way to pool the actual connections across the thread-local handles. If you need real connection reuse across workers, use one `Session` per worker, or the `AsyncSession`, whose connections are shared within its event loop.

What you *can* share across threads is the DNS cache and the TLS session cache, which cuts the cost of reconnecting to the same host (DNS lookups and TLS handshakes). Pass a `CurlShare` to the session:

```
from curl_cffi import CurlShare, Session

share = CurlShare() # shares DNS + TLS sessions, safe across threads
with Session(curl_share=share) as s:
    s.get("https://example.com")
share.close()
```

The share is attached to every handle the session creates, including the ones used for streaming and WebSocket responses. `AsyncSession` accepts the same `curl_share=` argument, sharing the DNS and TLS caches across the handles in its pool.

Keep the share alive until all attached handles have closed. `share.close()` raises `CurlError` if any handle is still using it; close those handles before retrying cleanup.

`CurlShare` can also share the connection cache (`CurlShare(connect=True)`), but that is only safe when the handles are used serially, never from multiple concurrent threads.

COMPATIBILITY WITH REQUESTS

Although we try our best to mimic the requests API, some functionality is not easy to implement and left out. Here are a list of known incompatibilities:

- files API are slightly different, but more error-proof.
- empty-domains cookies may lost during redirects, tracked in [#55](#).
- streaming response objects cannot be pickled; non-streaming responses can.
- The `requests` proxies dict is supported, but we prefer `proxy=...`, unless you really use different proxies for http and https.
- You can use `transports/adapters`, instead, you can use `curl_cffi` as adapter for `requests`.

6.1 Transports and Adapters

`curl_cffi` is deeply coupled with `libcurl-impersonate`. Unlike `requests` or `httpx`, There is no way to use a different networking library or mount different adapters.

Alternatively, you can use `curl_cffi` as a requests adapter via [curl-adapter](#). In this way, you get the full functionality of requests.

COOKIES

7.1 Save and load cookies

Do not use `get_dict` to dump and load cookies. Cookies are more than just plain key-value pairs.

Using pickle:

```
# example from: https://github.com/encode/httpx/issues/895
import pickle
# import httpx
import curl_cffi

def save_cookies(client):
    with open("cookies.pk", "wb") as f:
        pickle.dump(client.cookies.jar._cookies, f)

def load_cookies():
    if not os.path.isfile("cookies.pk"):
        return None
    with open("cookies.pk", "rb") as f:
        return pickle.load(f)

# client = httpx.Client(cookies=load_cookies())
client = curl_cffi.Session()
client.get("https://httpbin.org/cookies/set/foo/bar")
save_cookies(client)

client = curl_cffi.Session()
client.cookies.jar._cookies.update(load_cookies())
print(client.cookies.get("foo"))
```

Using mozilla cookie jar:

See: https://github.com/lexiforest/curl_cffi/issues/381

TODO: expose libcurl's native cookies.txt support.

7.2 Discard cookies when using Session

You may need to discard cookies when using sessions, especially when using `AsyncSession`.

Use the `discard_cookies` option.

```
s = requests.Session()

set_url = "https://httpbin.org/cookies"

r = s.get(set_url)
assert r.cookies["foo"] == s.cookies["foo"]
old_cookie = r.cookies["foo"]

# Let's start discarding cookies
s.discard_cookies = True
r = s.get(set_url)
assert r.cookies["foo"] != s.cookies["foo"]
assert old_cookie == s.cookies["foo"]

# The behavior can be reverted
s.discard_cookies = False
r = s.get(set_url)
assert r.cookies["foo"] == s.cookies["foo"]
old_cookie = r.cookies["foo"]

# Also works as request parameter
r = s.get(set_url, discard_cookies=True)
assert r.cookies["foo"] != s.cookies["foo"]
assert old_cookie == s.cookies["foo"]
```

8.1 Scrapy integrations

If you are using scrapy, check out these middlewares:

- [divtipty/scrapy-curl-cffi](https://github.com/divtipty/scrapy-curl-cffi)
- [tieyongjie/scrapy-fingerprint](https://github.com/tieyongjie/scrapy-fingerprint)
- [jxlil/scrapy-impersonate](https://github.com/jxlil/scrapy-impersonate)

8.2 Using with eventlet/gevent

Just set `thread` to `eventlet` or `gevent`.

```
from curl_cffi import requests

s = requests.Session(thread="eventlet")
s.get(url)
```

8.3 As a urllib3/requests adapter

You can also use `curl-cffi` as a requests adapter via `curl-adapter`. In this way, you get the full functionality of requests.

```
import requests
from curl_adapter import CurlCffiAdapter

session = requests.Session()
session.mount("http://", CurlCffiAdapter())
session.mount("https://", CurlCffiAdapter())

# just use requests session like you normally would
session.get("https://example.com")
```

8.4 As a httpx transport

You can also use `curl-cffi` as a httpx transport via `httpx-curl-cffi`. With this, you get the full functionality of httpx.

```
from httpx import Client, AsyncClient
from httpx_curl_cffi import CurlTransport, AsyncCurlTransport, CurlOpt
```

(continues on next page)

(continued from previous page)

```
client = Client(transport=CurlTransport(impersonate="chrome", default_headers=True))
client.get("https://tools.scrapfly.io/api/fp/ja3")

async_client = AsyncClient(transport=AsyncCurlTransport(
    impersonate="chrome",
    default_headers=True,
    # required for parallel requests, see curl_cffi issues below
    curl_options={CurlOpt.FRESH_CONNECT: True}
))
```

API REFERENCES

9.1 curl low level APIs

9.1.1 Curl

class `curl_cffi.Curl`(*cacert*: *str* = "", *debug*: *bool* = *False*, *handle*=*None*)

Wrapper for `curl_easy_*` functions of libcurl.

__init__(*cacert*: *str* = "", *debug*: *bool* = *False*, *handle*=*None*) → *None*

Parameters

- **cacert** – CA cert path to use, by default, certs from `certifi` are used.
- **debug** – whether to show curl debug messages.
- **handle** – a curl handle instance from `curl_easy_init`.

debug() → *None*

Set debug to True

setopt(*option*: `CurlOpt`, *value*: *Any*) → *int*

Wrapper for `curl_easy_setopt`.

Parameters

- **option** – option to set, using constants from `CurlOpt` enum
- **value** – value to set, strings will be handled automatically

Returns

0 if no error, see `CurlECode`.

getinfo(*option*: `CurlInfo`) → *bytes* | *int* | *float* | *list*[*str* | *int*]

Wrapper for `curl_easy_getinfo`. Gets information in response after `curl.perform`.

Parameters

option – option to get info of, using constants from `CurlInfo` enum

Returns

value retrieved from last perform.

version() → *bytes*

Get the underlying libcurl version.

impersonate(*target*: *str*, *default_headers*: *bool* = *True*) → *int*

Set the browser type to impersonate.

Parameters

- **target** – browser to impersonate.
- **default_headers** – whether to add default headers, like User-Agent.

Returns

0 if no error.

perform(*clear_headers: bool = True, clear_resolve: bool = True*) → None

Wrapper for `curl_easy_perform`, performs a curl request.

Parameters

- **clear_headers** – clear header slist used in this perform
- **clear_resolve** – clear resolve slist used in this perform

Raises

CurlError – if the perform was not successful.

duphandle() → *Curl*

Wrapper for `curl_easy_duphandle`.

This is not a full copy of entire curl object in python. For example, headers handle is not copied, you have to set them again.

reset() → None

Reset all curl options, wrapper for `curl_easy_reset`.

parse_cookie_headers(*headers: list[bytes]*) → SimpleCookie

Extract cookies.SimpleCookie from header lines.

Parameters

headers – list of headers in bytes.

Returns

A parsed cookies.SimpleCookie instance.

static get_reason_phrase(*status_line: bytes*) → bytes

Extract reason phrase, like OK, Not Found from response status line.

static parse_status_line(*status_line: bytes*) → tuple[*CurlHttpVersion*, int, bytes]

Parse status line.

Returns

http_version, status_code, and reason phrase

close() → None

Close and cleanup curl handle, wrapper for `curl_easy_cleanup`.

ws_recv() → tuple[bytes, CurlWsFrame]

Receive a frame from a websocket connection.

Returns

a tuple of frame content and curl frame meta struct. Content is decompressed when permessage-deflate is negotiated. bytesleft may be -1 while the decoded size is unknown; 0 marks the end of the frame.

Raises

CurlError – if failed.

ws_send(payload: bytes | bytearray | memoryview, flags: CurlWsFlag | int = CurlWsFlag.BINARY) → int

Send data to a websocket connection.

Parameters

- **payload** – content to send.
- **flags** – websocket flag to set for the frame, default: binary.

Returns

The number of bytes sent.

Raises

CurlError – if failed.

Notes

Memoryview payloads must be byte-format for `len()` to work correctly.

ws_close(code: int = 1000, message: bytes = b'') → int

Close a websocket connection. Shorthand for `ws_send()` with close code and message. Note that to completely close the connection, you must close the curl handle after this call with `close()`.

Parameters

- **code** – close code.
- **message** – close message.

Returns

0 if no error.

Raises

CurlError – if failed.

9.1.2 AsyncCurl

class curl_cffi.AsyncCurl(cacert: str = "", loop=None)

Wrapper around curl_multi handle to provide asyncio support. It uses the libcurl socket_action APIs.

__init__(cacert: str = "", loop=None)

Parameters

- **cacert** – CA cert path to use, by default, certs from `certifi` are used.
- **loop** – EventLoop to use.

add_handle(curl: Curl)

Add a curl handle to be managed by curl_multi. This is the equivalent of *perform* in the async world.

remove_handle(curl: Curl)

Cancel a future for given curl handle.

set_result(curl: Curl)

Mark a future as done for given curl handle.

set_exception(curl: Curl, exception)

Raise exception of a future for given curl handle.

setopt(option, value)

Wrapper around curl_multi_setopt.

socket_action(sockfd: int, ev_bitmask: int) → int

wrapper for curl_multi_socket_action, returns the number of running curl handles.

process_data(sockfd: int, ev_bitmask: int)

Call curl_multi_info_read to read data for given socket.

async close()

Close and cleanup running timers, readers, writers and handles.

9.1.3 CurlMime

class curl_cffi.CurlMime(curl: Curl | None = None)

Wrapper for the curl_mime_ API.

__init__(curl: Curl | None = None)

Parameters

curl – Curl instance to use.

addpart(name: str, *(Keyword-only parameters separator (PEP 3102)), content_type: str | None = None, filename: str | None = None, local_path: str | bytes | Path | None = None, data: bytes | None = None) → None

Add a mime part for a multipart html form.

Note: You can only use either local_path or data, not both.

Parameters

- **name** – name of the field.
- **content_type** – content_type for the field. for example: image/png.
- **filename** – filename for the server.
- **local_path** – file to upload on local disk.
- **data** – file content to upload.

classmethod from_list(files: list[dict])

Create a multipart instance from a list of dict, for keys, see addpart

attach(curl: Curl | None = None) → None

Attach the mime instance to a curl instance.

close() → None

Close the mime instance and underlying files. This method must be called after perform or request.

9.1.4 CurlShare

class curl_cffi.CurlShare(connect: bool = False, dns: bool = True, ssl_session: bool = True)

Wrapper for the curl_share_* API.

__init__(connect: bool = False, dns: bool = True, ssl_session: bool = True) → None

Parameters

- **connect** – share the connection cache. Only safe for serialized, single-threaded use; libcurl does not support sharing connections between concurrent threads.
- **dns** – share the DNS cache.

- **ssl_session** – share the TLS session cache.

share(data: *CurlLockData*) → None

Start sharing the given *CurlLockData* across attached handles.

unshare(data: *CurlLockData*) → None

Stop sharing the given *CurlLockData*.

close() → None

Cleanup the share handle, wrapper for `curl_share_cleanup`.

Only call this once every attached easy handle has been closed, otherwise raises *CurlError* with `CurlShareCode.IN_USE` and keeps the share open so cleanup can be retried after closing those handles.

9.1.5 Constants

Enum values used by `setopt` and `getinfo` can be accessed from *CurlOpt* and *CurlInfo*.

class *curl_cffi.CurlOpt*(value)

CULROPT_ constants extracted from libcurl, see: https://curl.se/libcurl/c/curl_easy_setopt.html

class *curl_cffi.CurlInfo*(value)

CURLINFO_ constants extracted from libcurl, see: https://curl.se/libcurl/c/curl_easy_getinfo.html

class *curl_cffi.CurlMopt*(value)

CURLMOPT_ constants extracted from libcurl, see: https://curl.se/libcurl/c/curl_multi_setopt.html

class *curl_cffi.CurlECode*(value)

CURLECODE_ constants extracted from libcurl, see: <https://curl.se/libcurl/c/libcurl-errors.html>

class *curl_cffi.CurlHttpVersion*(value)

CURL_HTTP_VERSION constants from libcurl, see comments for details.

class *curl_cffi.CurlProxyType*(value)

CURLPROXY constants from libcurl.

class *curl_cffi.CurlWsFlag*(value)

CURL_WS_FLAG constants from libcurl, see comments for details.

class *curl_cffi.CurlSslVersion*(value)

CURL_SSLVERSION constants from libcurl, see comments for details.

9.2 requests-like API

9.2.1 request method

`requests.get`, `requests.post`, etc are just aliases of `.request(METHOD, ...)`

curl_cffi.requests.request(method: *Literal*['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS', 'HEAD', 'TRACE', 'PATCH', 'QUERY'], url: *str*, thread: *Literal*['eventlet', 'gevent'] | None = None, curl_options: *dict* | None = None, debug: *bool* | None = None, **kwargs: None) → *Response*

Send an http request.

Parameters

- **method** – http method for the request: GET/POST/PUT/DELETE etc.
- **url** – url for the requests.

- **params** – query string for the requests.
- **data** – form values(dict/list/tuple) or binary data to use in body, *Content-Type: application/x-www-form-urlencoded* will be added if a dict is given.
- **content** – raw request body as str, bytes, a byte iterable, or a binary file. *AsyncSession* also accepts an async byte iterable.
- **json** – json values to use in body, *Content-Type: application/json* will be added automatically.
- **headers** – headers to send.
- **cookies** – cookies to use.
- **files** – not supported, use *multipart* instead.
- **auth** – HTTP basic auth, a tuple of (username, password), only basic auth is supported.
- **timeout** – how many seconds to wait before giving up.
- **allow_redirects** – whether to allow redirection.
- **max_redirects** – max redirect counts, default 30, use -1 for unlimited.
- **proxies** – dict of proxies to use, prefer to use *proxy* if they are the same. format: {"http": proxy_url, "https": proxy_url}.
- **proxy** – proxy to use, format: "http://user@pass:proxy_url". Can't be used with *proxies* parameter.
- **proxy_auth** – HTTP basic auth for proxy, a tuple of (username, password).
- **verify** – whether to verify https certs.
- **referer** – shortcut for setting referer header.
- **accept_encoding** – shortcut for setting accept-encoding header.
- **content_callback** – a callback function to receive response body. `def callback(chunk: bytes) -> None:`
- **impersonate** – which browser version or fingerprint to impersonate.
- **ja3** – ja3 string to impersonate.
- **akamai** – akamai string to impersonate.
- **extra_fp** – extra fingerprints options, in complement to ja3 and akamai strings.
- **thread** – thread engine to use for working with other thread implementations. choices: eventlet, gevent.
- **default_headers** – whether to set default browser headers when impersonating.
- **default_encoding** – encoding for decoding response content if charset is not found in headers. Defaults to "utf-8". Can be set to a callable for automatic detection.
- **quote** – Set characters to be quoted, i.e. percent-encoded. Default safe string is !#\$%&'()*+,-/,:;=?@[~. If set to a sting, the character will be removed from the safe string, thus quoted. If set to False, the url will be kept as is, without any automatic percent-encoding, you must encode the URL yourself.
- **curl_options** – extra curl options to use.
- **http_version** – limiting http version, defaults to http2.
- **debug** – print extra curl debug info.

- **interface** – interface name or local IP to bind to (bare IP = source address).
- **dns** – DNS server IP address or list of addresses. Requires c-ares.
- **cert** – a tuple of (cert, key) filenames for client cert.
- **stream** – streaming the response, default False.
- **max_recv_speed** – maximum receive speed, bytes per second.
- **multipart** – upload files using the multipart format, see examples for details.
- **discard_cookies** – discard cookies from server. Default to False.

Returns

A Response object.

9.2.2 Sessions

```
class curl_cffi.requests.Session(curl: Curl | None = None, thread: ThreadType | None = None,
                                use_thread_local_curl: bool = True, curl_share: CurlShare | None =
                                None, **kwargs: Unpack[BaseSessionParams[R]])
```

A request session, cookies and connections will be reused. This object is thread-safe, but it's recommended to use a separate session for each thread.

```
__init__(curl: Curl | None = None, thread: ThreadType | None = None, use_thread_local_curl: bool = True,
          curl_share: CurlShare | None = None, **kwargs: Unpack[BaseSessionParams[R]]) → None
```

Parameters set in the `__init__` method will be overridden by the same parameter in request method.

Parameters

- **curl** – curl object to use in the session. If not provided, a new one will be created. Also, a fresh curl object will always be created when accessed from another thread.
- **thread** – thread engine to use for working with other thread implementations. choices: eventlet, gevent.
- **curl_share** – a `CurlShare` attached to every (thread-local) handle, to share the DNS and TLS session caches across threads. See the note in the docs on connection reuse across threads.
- **headers** – headers to use in the session.
- **cookies** – cookies to add in the session.
- **auth** – HTTP basic auth, a tuple of (username, password), only basic auth is supported.
- **proxies** – dict of proxies to use, prefer to use proxy if they are the same. format: {"http": proxy_url, "https": proxy_url}.
- **proxy** – proxy to use, format: "http://proxy_url". Cannot be used with the above parameter.
- **proxy_auth** – HTTP basic auth for proxy, a tuple of (username, password).
- **base_url** – absolute url to use as base for relative urls.
- **params** – query string for the session.
- **verify** – whether to verify https certs.
- **timeout** – how many seconds to wait before giving up.
- **trust_env** – use http_proxy/https_proxy and other environments, default True.

- **allow_redirects** – whether to allow redirection. Can be a bool, a `CurlFollow` value, or the string "safe". Use `CurlFollow.SAFE` or "safe" to reject redirects to internal/private IP addresses (SSRF protection).
- **max_redirects** – max redirect counts, default 30, use -1 for unlimited.
- **retry** – number of retries or `RetryStrategy` for failed requests.
- **impersonate** – which browser version or fingerprint to impersonate in the session.
- **ja3** – ja3 string to impersonate in the session.
- **akamai** – akamai string to impersonate in the session.
- **perk** – perk string to impersonate in the session.
- **extra_fp** – extra fingerprints options, in complement to ja3 and akamai str.
- **interface** – interface name or local IP to bind to (bare IP = source address).
- **dns** – DNS server IP address or list of addresses. Requires c-ares.
- **doh_url** – DNS-over-HTTPS server url, e.g. <https://1.1.1.1/dns-query>.
- **default_encoding** – encoding for decoding response content if charset is not found in headers. Defaults to "utf-8". Can be set to a callable for automatic detection.
- **cert** – a tuple of (cert, key) filenames for client cert.
- **response_class** – A customized subtype of `Response` to use.
- **raise_for_status** – automatically raise an `HTTPError` for 4xx and 5xx status codes.

Notes

This class can be used as a context manager.

```
from curl_cffi.requests import Session

with Session() as s:
    r = s.get("https://example.com")
```

request(method: `HttpMethod`, url: `str`, params: `dict` | `list` | `tuple` | `None` = `None`, data: `RequestData` | `None` = `None`, content: `SyncRequestContent` | `None` = `None`, json: `dict` | `list` | `None` = `None`, headers: `HeaderTypes` | `None` = `None`, cookies: `CookieTypes` | `None` = `None`, files: `dict` | `None` = `None`, auth: `tuple`[`str`, `str`] | `None` = `None`, timeout: `float` | `tuple`[`float`, `float`] | `object` | `None` = `NOT_SET`, allow_redirects: `bool` | `CurlFollow` | `str` | `None` = `None`, max_redirects: `int` | `None` = `None`, proxies: `ProxySpec` | `None` = `None`, proxy: `str` | `None` = `None`, proxy_auth: `tuple`[`str`, `str`] | `None` = `None`, verify: `bool` | `None` = `None`, referer: `str` | `None` = `None`, accept_encoding: `str` | `None` = 'gzip, deflate, br', content_callback: `Callable` | `None` = `None`, impersonate: `BrowserTypeLiteral` | `Fingerprint` | `None` = `None`, ja3: `str` | `None` = `None`, akamai: `str` | `None` = `None`, perk: `str` | `None` = `None`, extra_fp: `ExtraFingerprints` | `ExtraFpDict` | `None` = `None`, default_headers: `bool` | `None` = `None`, default_encoding: `str` | `Callable`[[`bytes`], `str`] = 'utf-8', quote: `str` | `Literal`[`False`] = "", http_version: `CurlHttpVersion` | `HttpVersionLiteral` | `None` = `None`, interface: `str` | `None` = `None`, dns: `str` | `list`[`str`] | `None` = `None`, doh_url: `str` | `None` = `None`, cert: `str` | `tuple`[`str`, `str`] | `None` = `None`, stream: `bool` | `None` = `None`, max_recv_speed: `int` = 0, multipart: `CurlMime` | `None` = `None`, discard_cookies: `bool` = `False`) → `R`

Send the request, see `requests.request` for details on parameters.

stream(method: `Literal`['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS', 'HEAD', 'TRACE', 'PATCH', 'QUERY'], url: `str`, **kwargs: `None`) → `Generator`[`R`, `None`, `None`]

Equivalent to with `request(..., stream=True)` as `r`:

```
ws_connect(url: str, on_message: Callable[[WebSocket, bytes | str], None] | None = None, on_error:
    Callable[[WebSocket, Exception], None] | None = None, on_open: Callable[[WebSocket],
    None] | None = None, on_close: Callable[[WebSocket, int, str], None] | None = None, on_data:
    Callable[[WebSocket, bytes, CurlWsFrame], None] | None = None, *, autoclose: bool = True,
    skip_utf8_validation: bool = False, ws_retry: WebSocketRetryStrategy | None = None,
    max_message_size: int = 4194304, params: dict[str, object] | list[object] | tuple[str, int |
    list[str] | dict[str, str | int]] | None = None, headers: HeaderTypes | None = None, cookies:
    CookieTypes | None = None, auth: tuple[str, str] | None = None, timeout: float | tuple[float,
    float] | NotSetType | None = NOT_SET, allow_redirects: bool | CurlFollow | str | None = None,
    max_redirects: int | None = None, proxies: ProxySpec | None = None, proxy: str | None = None,
    proxy_auth: tuple[str, str] | None = None, verify: bool | str | None = None, referer: str | None =
    None, accept_encoding: str | None = 'gzip, deflate, br', impersonate: BrowserTypeLiteral | str |
    Fingerprint | None = None, ja3: str | None = None, akamai: str | None = None, perk: str | None =
    None, extra_fp: ExtraFingerprints | ExtraFpDict | None = None, default_headers: bool | None =
    None, quote: str | Literal[False] = '', http_version: CurlHttpVersion | HttpVersionLiteral |
    None = None, interface: str | None = None, dns: str | list[str] | None = None, doh_url: str |
    None = None, cert: str | tuple[str, str] | None = None, max_recv_speed: int = 0, curl_options:
    dict[CurlOpt, Any] | None = None) → WebSocket
```

Connects to a WebSocket URL using the session's configuration.

libcurl automatically handles pings and pongs under the hood. Ref: <https://curl.se/libcurl/c/libcurl-ws.html>

Parameters

- **url** – The WebSocket URL to connect to.
- **on_message** – Message received callback: `def on_message(ws, msg).`
- **on_error** – Error callback: `def on_error(ws, error).`
- **on_open** – Open callback: `def on_open(ws).`
- **on_close** – Close callback: `def on_close(ws, code, reason).`
- **on_data** – Raw data frame callback: `def on_data(ws, bytes, frame).`
- **autoclose** – Whether to automatically close on CLOSE frame.
- **skip_utf8_validation** – Skips UTF-8 validation of text frames in `run_forever()`, `recv_str()`, and `recv_json()`. When set, raise but don't close connection per RFC 6455 §8.1.
- **ws_retry** – Custom retry strategy for failed network receives.
- **max_message_size** – Maximum allowed message size in bytes (default 4 MiB).
- **params** – Query string parameters to attach to the handshake URL.
- **headers** – Handshake request headers (merges with session defaults).
- **cookies** – Handshake request cookies (merges with session defaults).
- **auth** – HTTP basic auth, a tuple of (username, password).
- **timeout** – Handshake connection phase timeout in seconds. `None` is clamped to `MAX_HANDSHAKE_SECS` (30s) because the handshake cannot be interrupted once started.
- **allow_redirects** – Allow redirects. Can be a bool, `CurlFollow`, or "safe". Defaults to session's value (True).

- **max_redirects** – Max redirect counts. Defaults to the session's configured value (30 by default).
- **proxies** – Dictionary of proxies, e.g. {"http": proxy_url}.
- **proxy** – Proxy URL. Cannot be used concurrently with *proxies*.
- **proxy_auth** – HTTP basic auth for proxy, a tuple of (username, password).
- **verify** – Whether to verify SSL/TLS certificates.
- **referer** – Shortcut for setting the HTTP Referer header.
- **accept_encoding** – Shortcut for setting the Accept-Encoding header.
- **impersonate** – Which browser version or fingerprint to impersonate.
- **ja3** – JA3 TLS client fingerprint string to impersonate.
- **akamai** – Akamai HTTP/2 fingerprint string to impersonate.
- **perk** – Perk fingerprint option to impersonate.
- **extra_fp** – Extra fingerprint options complementing ja3 and akamai.
- **default_headers** – Whether to set default browser headers. Defaults to session's value (True).
- **quote** – Quote (percent-encode) characters. Default safe string is !#\$%&'()*+,-/;:=?@[]~. Set to a string to quote more, or False to bypass automatic encoding.
- **http_version** – WebSockets are always bootstrapped over HTTP/1.1 (RFC 6455), so this option has no effect.
- **interface** – Interface name or local IP to bind to.
- **dns** – DNS server IP address or list of addresses. Requires c-ares.
- **doh_url** – DNS-over-HTTPS server url, e.g. <https://1.1.1.1/dns-query>.
- **cert** – Tuple of (cert, key) filenames for client certificate auth.
- **max_recv_speed** – Maximum receive speed in bytes per second.
- **curl_options** – Dictionary of extra low-level curl options to apply.

Returns

A fully-connected synchronous WebSocket instance.

Return type

WebSocket

Note

The `WebSocket` is NOT thread-safe, add your own locking or use `AsyncWebSocket` for concurrent sending and receiving.

```
class curl_cffi.requests.AsyncSession(*, loop: asyncio.AbstractEventLoop | None = None, async_curl:
    AsyncCurl | None = None, max_clients: int = 10, curl_share:
    CurlShare | None = None, **kwargs:
    Unpack[BaseSessionParams[R]])
```

An async request session, cookies and connections will be reused.

```
__init__(*, loop: asyncio.AbstractEventLoop | None = None, async_curl: AsyncCurl | None = None,
max_clients: int = 10, curl_share: CurlShare | None = None, **kwargs:
Unpack[BaseSessionParams[R]]) → None
```

Parameters set in the `__init__` method are overridden by the same parameter in request method.

Parameters

- **loop** – loop to use, if not provided, the running loop will be used.
- **async_curl** – [AsyncCurl](/api/curl_cffi#curl_cffi.AsyncCurl) object to use.
- **max_clients** – maximum curl handles to use in the session, this will affect the concurrency ratio. WebSockets count against this limit for their connection lifetime.
- **curl_share** – a CurlShare attached to every pooled handle, to share the DNS and TLS session caches across the handles in the pool.
- **headers** – headers to use in the session.
- **cookies** – cookies to add in the session.
- **auth** – HTTP basic auth, a tuple of (username, password), only basic auth is supported.
- **proxies** – dict of proxies to use, prefer to use proxy if they are the same. format: {"http": proxy_url, "https": proxy_url}.
- **proxy** – proxy to use, format: "http://proxy_url". Cannot be used with the above parameter.
- **proxy_auth** – HTTP basic auth for proxy, a tuple of (username, password).
- **base_url** – absolute url to use for relative urls.
- **params** – query string for the session.
- **verify** – whether to verify https certs.
- **timeout** – how many seconds to wait before giving up.
- **trust_env** – use http_proxy/https_proxy and other environments, default True.
- **allow_redirects** – whether to allow redirection. Can be a bool, a CurlFollow value, or the string "safe". Use CurlFollow.SAFE or "safe" to reject redirects to internal/private IP addresses (SSRF protection).
- **max_redirects** – max redirect counts, default 30, use -1 for unlimited.
- **retry** – number of retries or RetryStrategy for failed requests.
- **impersonate** – which browser version or fingerprint to impersonate in the session.
- **ja3** – ja3 string to impersonate in the session.
- **akamai** – akamai string to impersonate in the session.
- **perk** – perk string to impersonate in the session.
- **extra_fp** – extra fingerprints options, in complement to ja3 and akamai str.
- **default_encoding** – encoding for decoding response content if charset is not found in headers. Defaults to "utf-8". Can be set to a callable for automatic detection.
- **cert** – a tuple of (cert, key) filenames for client cert.
- **response_class** – A customized subtype of Response to use.
- **raise_for_status** – automatically raise an HTTPError for 4xx and 5xx status codes.

Notes

This class can be used as a context manager, and it's recommended to use via `async with`. However, unlike `aiohttp`, it is not required to use `with`.

```
from curl_cffi.requests import AsyncSession

# recommended.
async with AsyncSession() as s:
    r = await s.get("https://example.com")

s = AsyncSession() # it also works.
```

async request(method: *HttpMethod*, url: *str*, params: *dict[str, str] | list[tuple[str, str]] | tuple[tuple[str, str], ...] | None* = None, data: *RequestData* | *None* = None, content: *RequestContent* | *None* = None, json: *dict[str, Any] | list[Any] | None* = None, headers: *HeaderTypes* | *None* = None, cookies: *CookieTypes* | *None* = None, files: *dict[str, Any] | None* = None, auth: *tuple[str, str] | None* = None, timeout: *float | tuple[float, float] | object* | *None* = *NOT_SET*, allow_redirects: *bool | CurlFollow* | *str* | *None* = None, max_redirects: *int* | *None* = None, proxies: *ProxySpec* | *None* = None, proxy: *str* | *None* = None, proxy_auth: *tuple[str, str] | None* = None, verify: *bool* | *None* = None, referer: *str* | *None* = None, accept_encoding: *str* | *None* = 'gzip, deflate, br', content_callback: *Callable[..., Any] | None* = None, impersonate: *BrowserTypeLiteral | Fingerprint* | *None* = None, ja3: *str* | *None* = None, akamai: *str* | *None* = None, perk: *str* | *None* = None, extra_fp: *ExtraFingerprints | ExtraFpDict* | *None* = None, default_headers: *bool* | *None* = None, default_encoding: *str* | *Callable[[bytes], str]* = 'utf-8', quote: *str* | *Literal[False]* = '', http_version: *CurlHttpVersion | HttpVersionLiteral* | *None* = None, interface: *str* | *None* = None, dns: *str* | *list[str]* | *None* = None, doh_url: *str* | *None* = None, cert: *str* | *tuple[str, str]* | *None* = None, stream: *bool* | *None* = None, max_recv_speed: *int* = 0, multipart: *CurlMime* | *None* = None, discard_cookies: *bool* = False) → *R*

Send the request, see `curl_cffi.requests.request` for details on args.

stream(method: *Literal['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS', 'HEAD', 'TRACE', 'PATCH', 'QUERY']*, url: *str*, **kwargs: *None*) → *AsyncGenerator[R, None]*

Equivalent to `async with request(..., stream=True) as r:`

async close() → *None*

Close the session.

ws_connect(url: *str*, autoclose: *bool* = True, params: *dict[str, object] | list[object] | tuple[object, ...] | None* = None, headers: *HeaderTypes* | *None* = None, cookies: *CookieTypes* | *None* = None, auth: *tuple[str, str] | None* = None, timeout: *float | tuple[float, float] | NotSetType* | *None* = *NOT_SET*, allow_redirects: *bool | CurlFollow* | *str* | *None* = None, max_redirects: *int* | *None* = None, proxies: *ProxySpec* | *None* = None, proxy: *str* | *None* = None, proxy_auth: *tuple[str, str] | None* = None, verify: *bool* | *str* | *None* = None, referer: *str* | *None* = None, accept_encoding: *str* | *None* = 'gzip, deflate, br', impersonate: *BrowserTypeLiteral* | *str* | *Fingerprint* | *None* = None, ja3: *str* | *None* = None, akamai: *str* | *None* = None, perk: *str* | *None* = None, extra_fp: *ExtraFingerprints | ExtraFpDict* | *None* = None, default_headers: *bool* | *None* = None, quote: *str* | *Literal[False]* = '', http_version: *CurlHttpVersion | HttpVersionLiteral* | *None* = None, interface: *str* | *None* = None, dns: *str* | *list[str]* | *None* = None, doh_url: *str* | *None* = None, cert: *str* | *tuple[str, str]* | *None* = None, max_recv_speed: *int* = 0, recv_queue_size: *int* = 64, send_queue_size: *int* = 32, max_send_batch_size: *int* = 64, coalesce_frames: *bool* = False, ws_retry: *WebSocketRetryStrategy* | *None* = None, recv_time_slice: *float* = 0.01, send_time_slice: *float* = 0.01, max_message_size: *int* = 4194304, drain_on_error: *bool* = False, block_on_recv_queue_full: *bool* = True, curl_options: *dict[CurlOpt, Any]* | *None* = None, *, skip_utf8_validation: *bool* = False) → *AsyncWebSocketContext*

Connects to a WebSocket.

Parameters

- **url** – The WebSocket URL to connect to (e.g. `wss://echo.websocket.org`).
- **autoclose** – Whether to automatically close on CLOSE frame.
- **skip_utf8_validation** – Skips UTF-8 validation of text frames in `recv_str()`, and `recv_json()`. When set, raise an error but don't close the connection as per RFC 6455 §8.1.
- **params** – Query string parameters to attach to the handshake URL.
- **headers** – Handshake request headers (merges with session defaults).
- **cookies** – Handshake request cookies (merges with session defaults).
- **auth** – HTTP basic auth, a tuple of (username, password). Only basic auth is supported.
- **timeout** – Handshake connection phase timeout in seconds. `None` is clamped to `MAX_HANDSHAKE_SECS` (30s) because the handshake cannot be interrupted once started.
- **allow_redirects** – Whether to allow redirection. Can be a bool, a `CurlFollow` value, or the string "safe". Use `CurlFollow.SAFE` or "safe" to reject redirects to internal/private IP addresses (SSRF protection).
- **max_redirects** – Max redirect counts, default 30, use -1 for unlimited.
- **proxies** – Dict of proxies to use, prefer to use `proxy` if they are the same. format: `{"http": proxy_url, "https": proxy_url}`.
- **proxy** – Proxy to use, format: `http://user@pass:proxy_url`. Can't be used with *proxies* parameter.
- **proxy_auth** – HTTP basic auth for proxy, a tuple of (username, password).
- **verify** – Whether to verify SSL/TLS certificates.
- **referer** – Shortcut for setting the HTTP Referer header.
- **accept_encoding** – Shortcut for setting the Accept-Encoding header.
- **impersonate** – Which browser version or fingerprint to impersonate.
- **ja3** – JA3 string to impersonate.
- **akamai** – Akamai HTTP/2 fingerprint string to impersonate.
- **perk** – Perk fingerprint option to impersonate.
- **extra_fp** – Extra fingerprints options, in complement to ja3 and akamai str.
- **default_headers** – Whether to set default browser headers.
- **quote** – Quote (percent-encode) characters. Default safe string is `!#$%&'()*+,-/;=?@[]~`. Set to a string to quote more, or `False` to bypass automatic encoding.
- **http_version** – WebSockets are always bootstrapped over HTTP/1.1 (RFC 6455), so this option has no effect.
- **interface** – Interface name or local IP to bind to (bare IP = source address).
- **dns** – DNS server IP address or list of addresses. Requires c-ares.
- **doh_url** – DNS-over-HTTPS server url, e.g. <https://1.1.1.1/dns-query>.

- **cert** – A tuple of (cert, key) filenames for client cert.
- **max_recv_speed** – Maximum receive speed, bytes per second.
- **recv_queue_size** – The maximum number of incoming WebSocket messages to buffer internally. This queue stores messages received by the Curl socket that are waiting to be consumed on calling `recv()`.
- **send_queue_size** – The maximum number of outgoing WebSocket messages to buffer before applying network backpressure. When you call `send()` the message is placed in this queue and transmitted when the Curl socket is next available for sending.
- **coalesce_frames** – When enabled, multiple outgoing messages in the send queue can be merged into a single message for improved throughput. Defaults to `False`. This is an optional power-user optimization, which breaks frame boundaries and merges multiple frame payloads together. It should only be used when the application protocol is designed to handle concatenated data streams. See online docs before use.
- **max_send_batch_size** – The max batch size when `coalesce_frames` is used.
- **ws_retry** (*WebSocketRetryStrategy*) – Retry policy for WebSocket messages.
- **recv_time_slice** – The maximum duration (in seconds) to process incoming messages before yielding to the event loop. Defaults to `0.01` (10ms).
- **send_time_slice** – The maximum duration (in seconds) to process outgoing messages before yielding to the event loop. Defaults to `0.01` (10ms).
- **max_message_size** – Maximum allowed message size in bytes (default 4 MiB).
- **drain_on_error** – When a connection error occurs, attempt to consume all the buffered received messages first, before raising the error. Otherwise, raise it immediately, discarding the buffered messages (the default).
- **block_on_recv_queue_full** – When the receive queue is full, receiving is paused for backpressure. When this option is disabled, the connection is immediately failed instead. The message that caused the overflow is not delivered; any messages already buffered can still be drained if `drain_on_error` is also set.
- **curl_options** – Dictionary of extra low-level curl options to apply.

Documentation:

<https://curl-cffi.readthedocs.io/en/latest/websockets.html>

9.2.3 Cache

```
class curl_cffi.requests.CacheBackend(*, expires: timedelta, methods: Sequence[str] | None = None,
                                     ignored: Sequence[str] | None = None)

    __init__(*, expires: timedelta, methods: Sequence[str] | None = None, ignored: Sequence[str] | None =
        None) → None

    should_cache_request(request: Request, *, stream: bool = False, content_callback: Any = None) → bool
        XXX: streamed content is not cached.

    should_store_response(response: Response) → bool

    get(request: Request, response_class: type[Response] = <class 'curl_cffi.requests.models.Response'>) →
        Response | None
```

```

set(request: Request, response: Response) → None

delete(request: Request) → None

abstractmethod _read_payload(key: str) → dict[str, Any] | None

abstractmethod _write_payload(key: str, payload: dict[str, Any]) → None

abstractmethod _delete_payload(key: str) → None

abstractmethod clear() → None

class curl_cffi.requests.FileCacheBackend(* , expires: timedelta, path: str | PathLike[str] | None = None,
                                         methods: Sequence[str] | None = None, ignored:
                                         Sequence[str] | None = None)

    __init__(* , expires: timedelta, path: str | PathLike[str] | None = None, methods: Sequence[str] | None =
        None, ignored: Sequence[str] | None = None) → None

    clear() → None

```

9.2.4 Headers

```

class curl_cffi.requests.Headers(headers: Headers | Mapping[str, str | None] | Mapping[bytes, bytes |
                                         None] | Sequence[tuple[str, str]] | Sequence[tuple[bytes, bytes]] |
                                         Sequence[str | bytes] | None = None, encoding: str | None = None)

```

HTTP headers, as a case-insensitive multi-dict.

property encoding: str

Header encoding is mandated as ascii, but we allow fallbacks to utf-8 or iso-8859-1.

raw()

Returns a list of the raw header items, as byte pairs.

multi_items() → list[tuple[str, str | None]]

Return a list of (*key*, *value*) pairs of headers. Allow multiple occurrences of the same key without concatenating into a single comma separated value.

get(key: str, default: Any = None) → Any

Return a header value. If multiple occurrences of the header occur then concatenate them together with commas.

get_list(key: str, split_commas: bool = False) → list[str | None]

Return a list of all header values for a given key. If *split_commas=True* is passed, then any comma separated header values are split into multiple return strings.

update([*E*,]***F*) → None. Update D from mapping/iterable E and F.

If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

__getitem__(key: str) → str | None

Return a single header value. If there are multiple headers with the same key, then we concatenate them with commas. See: <https://tools.ietf.org/html/rfc7230#section-3.2.2>

__setitem__(key: str, value: str | None) → None

Set the header *key* to *value*, removing any duplicate entries. Retains insertion order.

__delitem__(key: str) → None

Remove the header *key*.

9.2.5 Cookies

class `curl_cffi.requests.Cookies`(*cookies*: `Cookies` | `CookieJar` | `dict[str, str]` | `list[tuple[str, str]]` | `None` = `None`)

HTTP Cookies, as a mutable mapping.

set(*name*: `str`, *value*: `str`, *domain*: `str` = "", *path*: `str` = '/', *secure*=`False`) → `None`

Set a cookie value by name. May optionally include domain and path.

get(*name*: `str`, *default*: `str` | `None` = `None`, *domain*: `str` | `None` = `None`, *path*: `str` | `None` = `None`) → `str` | `None`

Get a cookie by name. May optionally include domain and path in order to specify exactly which cookie to retrieve.

delete(*name*: `str`, *domain*: `str` | `None` = `None`, *path*: `str` | `None` = `None`) → `None`

Delete a cookie by name. May optionally include domain and path in order to specify exactly which cookie to delete.

clear(*domain*: `str` | `None` = `None`, *path*: `str` | `None` = `None`) → `None`

Delete all cookies. Optionally include a domain and path in order to only delete a subset of all the cookies.

update(`[E, ]**F`) → `None`. Update D from mapping/iterable E and F.

If E present and has a `.keys()` method, does: for k in E: D[k] = E[k] If E present and lacks `.keys()` method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

__getitem__(*name*: `str`) → `str`

__setitem__(*name*: `str`, *value*: `str`) → `None`

__delitem__(*name*: `str`) → `None`

9.2.6 Request, Response

class `curl_cffi.requests.Request`(*url*: `str`, *headers*: `Headers`, *method*: `str`, *body*: `bytes` | `None` = `None`)

Representing a sent request.

url

request url.

headers

request headers.

method

request http method.

body

request body as bytes, or `None` if not provided.

class `curl_cffi.requests.Response`(*curl*: `Curl` | `None` = `None`, *request*: `Request` | `None` = `None`)

Contains information the server sends.

url

url used in the request.

content

response body in bytes.

text

response body in str.

status_code

http status code.

reason

http response reason, such as OK, Not Found.

ok

is status_code in [200, 400)?

headers

response headers.

cookies

response cookies.

elapsed

timedelta of the request duration.

encoding

http body encoding.

charset

alias for encoding.

primary_ip

primary ip of the server.

primary_port

primary port of the server.

local_ip

local ip used in this connection.

local_port

local port used in this connection.

charset_encoding

encoding specified by the Content-Type header.

default_encoding

encoding for decoding response content if charset is not found in headers. Defaults to “utf-8”. Can be set to a callable for automatic detection.

redirect_count

how many redirects happened.

redirect_url

the final redirected url.

http_version

http version used.

history

redirect responses, in request order. Response bodies are not available.

download_size

total downloaded bytes (body).

upload_size

total uploaded bytes (body).

header_size

total header size.

request_size

request size.

response_size

download_size + header_size

raise_for_status()

Raise an error if status code is not in [200, 400)

iter_lines(*chunk_size=None, decode_unicode=False, delimiter=None*)

iterate streaming content line by line, separated by \n.

Copied from: https://requests.readthedocs.io/en/latest/_modules/requests/models/ which is under the License: Apache 2.0

iter_content(*chunk_size=None, decode_unicode=False*)

iterate streaming content chunk by chunk in bytes.

json(**, path: str | None = None, default: Any = None, **kw: Any*) → Any

Parse JSON, optionally returning the first JSONPath match.

Path selection requires `curl_cffi[extra]`. Return `default` (None if omitted) only when no matches exist; decoding and path errors raise. Additional keyword arguments are passed to the JSON decoder.

close()

Close the streaming connection, only valid in stream mode.

async aiter_lines(*chunk_size=None, decode_unicode=False, delimiter=None*)

iterate streaming content line by line, separated by \n.

Copied from: https://requests.readthedocs.io/en/latest/_modules/requests/models/ which is under the License: Apache 2.0

async aiter_content(*chunk_size=None, decode_unicode=False*)

iterate streaming content chunk by chunk in bytes.

async atext() → str

Return a decoded string.

async acontent() → bytes

wait and read the streaming content in one bytes object.

async aclose()

Close the streaming connection, only valid in stream mode.

9.3 Asyncio

9.4 WebSocket

```

final class curl_cffi.requests.WebSocket(curl: Curl | NotSetType = NOT_SET, *, autoclose: bool =
    True, skip_utf8_validation: bool = False, debug: bool = False,
    on_open: OnOpenType | None = None, on_close:
    OnCloseType | None = None, on_data: OnDataType | None =
    None, on_message: OnMessageType | None = None, on_error:
    OnErrorType | None = None, ws_retry:
    WebSocketRetryStrategy | None = None, max_message_size:
    int = 4194304)

```

A high-performance synchronous WebSocket implementation using libcurl.

```

__init__(curl: Curl | NotSetType = NOT_SET, *, autoclose: bool = True, skip_utf8_validation: bool =
    False, debug: bool = False, on_open: OnOpenType | None = None, on_close: OnCloseType | None
    = None, on_data: OnDataType | None = None, on_message: OnMessageType | None = None,
    on_error: OnErrorType | None = None, ws_retry: WebSocketRetryStrategy | None = None,
    max_message_size: int = 4194304) → None

```

Parameters

- **curl** (Curl | NotSetType, optional) – Curl handle.
- **autoclose** (bool, optional) – Whether to close the WebSocket after receiving a close frame.
- **skip_utf8_validation** (bool, optional) – Skips UTF-8 validation of text frames in `run_forever()`, `recv_str()`, and `recv_json()`. When set, raise error but don't close connection as per RFC 6455 §8.1.
- **debug** (bool, optional) – Print extra curl debug info.
- **on_open** (OnOpenType | None, optional) – Open callback.
- **on_close** (OnCloseType | None, optional) – Close callback.
- **on_data** (OnDataType | None, optional) – Data received callback. The frame attribute is a view into libcurl's internal state and is only valid for the duration of the callback. Read what you need from it; do not retain it — the next received frame overwrites the same memory.
- **on_message** (OnMessageType | None, optional) – Message received callback.
- **on_error** (OnErrorType | None, optional) – Error callback.
- **ws_retry** (WebSocketRetryStrategy | None, optional) – Retry policy.
- **max_message_size** (int, optional) – Message size limit.

```

connect(url: str, params: dict[str, object] | list[object] | tuple[str, int | list[str] | dict[str, str | int]] | None =
    None, headers: HeaderTypes | None = None, cookies: CookieTypes | None = None, auth: tuple[str,
    str] | None = None, timeout: float | tuple[float, float] | NotSetType | None = NOT_SET,
    allow_redirects: bool | CurlFollow | str = True, max_redirects: int = 30, proxies: ProxySpec | None
    = None, proxy: str | None = None, proxy_auth: tuple[str, str] | None = None, verify: bool | str | None
    = None, referer: str | None = None, accept_encoding: str | None = 'gzip, deflate, br', impersonate:
    BrowserTypeLiteral | str | Fingerprint | None = None, ja3: str | None = None, akamai: str | None =
    None, perk: str | None = None, extra_fp: ExtraFingerprints | ExtraFpDict | None = None,
    default_headers: bool = True, quote: str | Literal[False] = "", http_version: CurlHttpVersion |
    HttpVersionLiteral | None = None, interface: str | None = None, cert: str | tuple[str, str] | None =
    None, max_recv_speed: int = 0, curl_options: dict[CurlOpt, Any] | None = None, *, base_url: str |
    None = None, base_params: dict[str, object] | list[object] | tuple[str, int | list[str] | dict[str, str |
    int]] | None = None, base_proxies: ProxySpec | None = None, base_verify: bool | str | None = True,
    dns: str | list[str] | None = None, doh_url: str | None = None) → WebSocket

```

Connect to the WebSocket.

libcurl automatically handles pings and pongs. ref: <https://curl.se/libcurl/c/libcurl-ws.html>

Parameters

- **url** – WebSocket connection URL.
- **base_url** – Base URL to use for relative paths.
- **base_params** – Session-level params applied before the request-level params.
- **base_proxies** – Session-level proxies, used when the request sets none.
- **base_verify** – Session-level verification settings to inherit.
- **params** – query string for the requests.
- **headers** – headers to send.
- **cookies** – cookies to use.
- **auth** – HTTP basic auth, a tuple of (username, password), only basic auth is supported.
- **timeout** – how many seconds to wait before giving up. `None` waits indefinitely; prefer `Session.ws_connect()`, which clamps it.
- **allow_redirects** – whether to allow redirection. Can be a bool, a `CurlFollow` value, or the string "safe".
- **max_redirects** – max redirect counts, default 30, use -1 for unlimited.
- **proxies** – dict of proxies to use, prefer to use `proxy` if they are the same. format: `{"http": proxy_url, "https": proxy_url}`.
- **proxy** – proxy to use, format: `"http://user@pass:proxy_url"`. Can't be used with *proxies* parameter.
- **proxy_auth** – HTTP basic auth for proxy, a tuple of (username, password).
- **verify** – whether to verify https certs.
- **referer** – shortcut for setting referer header.
- **accept_encoding** – shortcut for setting accept-encoding header.
- **impersonate** – which browser version or fingerprint to impersonate.
- **ja3** – ja3 string to impersonate.
- **akamai** – akamai string to impersonate.
- **perk** – perk string to impersonate.
- **extra_fp** – extra fingerprints options, in complement to ja3 and akamai str.
- **default_headers** – whether to set default browser headers.
- **quote** – Set characters to be quoted (percent-encoded). Default safe string is `!#$%&'()*+,-/;:;=?@[]~`. If set to a string, the characters will be removed from the safe string. If set to `False`, the URL is used as-is (you must encode it yourself).
- **http_version** – WebSockets are bootstrapped over HTTP/1.1, this has no effect.
- **interface** – interface name or local IP to bind to (bare IP = source address).
- **dns** – DNS server IP address or list of addresses. Requires c-ares.

- **doh_url** – DNS-over-HTTPS server url, e.g. <https://1.1.1.1/dns-query>.
- **cert** – a tuple of (cert, key) filenames for client cert.
- **max_recv_speed** – maximum receive speed, bytes per second.
- **curl_options** – extra curl options to use.

Note

The `WebSocket` is NOT thread-safe, add your own locking or use `AsyncWebSocket` for concurrent sending and receiving.

recv(**, timeout: float | None = None*) → tuple[bytes, int]

Receive a `WebSocket` message.

This method waits for and returns the next complete `WebSocket` message. Attempts to read immediately and only polls the socket if it returns `EAGAIN`.

Parameters

timeout – How many seconds to wait for a message if socket is blocked.

Returns

A tuple with the received payload and flags.

Return type

tuple[bytes, int]

Raises

- **WebSocketTimeout** – If the timeout expires.
- **WebSocketClosed** – If the connection is closed.
- **WebSocketError** – `WebSocket` connection specific error.
- **CurlError** – If a network-level transport error occurs.

Note

This method is NOT thread-safe.

Message fragmentation and reassembly are handled automatically by the implementation, so callers always receive complete messages.

recv_str(**, timeout: float | None = None*) → str

Receive a text frame.

Parameters

timeout – How many seconds to wait before raising a timeout.

Raises

- **WebSocketClosed** – A close frame was received.
- **WebSocketError** – The payload contains invalid UTF-8.
- **WebSocketError** – The received frame is not a text frame.

Returns

The text frame payload string.

Return type

str

Note

This method checks if the message payload is valid UTF-8, unless `skip_utf8_validation` is set at connection time.

recv_json(* , loads: ~collections.abc.Callable[[str], ~curl_cffi.requests.websockets.T] = <function loads>, timeout: float | None = None) → T

Receive a JSON frame.

Parameters

- **loads** – JSON decoder. Defaults to `json.loads()`.
- **timeout** – How many seconds to wait before raising a timeout.

Raises

JSONDecodeError – The default decoder could not decode the message. Exceptions from a custom decoder propagate unchanged.

Returns

Deserialized JSON object.

Return type

T

Note

This method checks if the message payload is valid UTF-8, unless `skip_utf8_validation` is set at connection time.

This method inherits the same exceptions from `recv_str()`.

send(payload: str | bytes | bytearray | memoryview, flags: CurlWsFlag | int = CurlWsFlag.BINARY, *, timeout: float | None = None) → int

Send a WebSocket message.

Parameters

- **payload** – Data to send (str/bytes/bytearray/memoryview).
- **flags** – Frame type flags (e.g. `CurlWsFlag.TEXT` / `CurlWsFlag.BINARY`).
- **timeout** – Max seconds to wait if the socket is blocked.

Returns

The number of bytes successfully written to the socket.

Return type

int

Raises

- **CurlError** – Network related exception occurred.
- **WebSocketClosed** – The WebSocket has been closed.
- **WebSocketTimeout** – The send operation timed out.

- **WebSocketError** – The socket stalled while sending.

Note

This method is NOT thread-safe.

There are no limits on the size of the message that can be sent. Large outbound messages are seamlessly broken down into optimal fragments using the `CURLWS_CONT` flag, arriving as a single logical message to the server.

Non-contiguous (i.e. strided) memoryviews are copied as they cannot be cast into a contiguous format required by `ffi.from_buffer()`.

send_binary(*payload: bytes | bytearray | memoryview*, *, *timeout: float | None = None*) → int

Send a binary frame.

For more info, see the docstring for `send()`.

send_bytes(*payload: bytes | bytearray | memoryview*, *, *timeout: float | None = None*) → int

Send a binary frame, alias of `send_binary()`.

For more info, see the docstring for `send()`.

send_str(*payload: str*, *, *timeout: float | None = None*) → int

Send a text frame.

For more info, see the docstring for `send()`.

send_json(*payload: object*, *, *dumps: ~collections.abc.Callable[[...], str | bytes] = <function dumps>*, *timeout: float | None = None*) → int

Send a JSON frame.

Parameters

- **payload** – data to send.
- **dumps** – JSON encoder, default is `json.dumps`. The encoder may return `str` or `bytes`; bytes must be UTF-8, since the payload is sent as a text frame.
- **timeout** – Max seconds to wait if the socket is blocked.

For more info, see the docstring for `send()`.

ping(*payload: str | bytes = b''*, *, *timeout: float | None = None*) → int

Send a ping frame.

Parameters

- **payload** (*str | bytes*, *optional*) – Ping frame payload. Defaults to `b''`.
- **timeout** (*float | None*, *optional*) – Max seconds to wait if socket is blocked.

Raises

WebSocketError – The payload length is outside specification (>125 bytes).

Returns

The number of bytes successfully written to the socket.

Return type

int

run_forever(url: str = "", **kwargs: Any) → None

Run the WebSocket forever. See `connect()` for details on parameters.

The connection is always terminated when this method returns, including when it exits by exception.

libcurl automatically handles pings and pongs. ref: <https://curl.se/libcurl/c/libcurl-ws.html>

Raises

- **WebSocketError** – The selectors were not initialized by `connect()`.
- **WebSocketClosed** – The server dropped the connection without a close frame (EOF).
- **CurlError** – An unrecoverable transport error, after any configured retries are exhausted.

close(code: int | WsCloseCode = WsCloseCode.OK, message: str | bytes = b'', timeout: float = 3.0) → None

Close the WebSocket connection.

Sends a close frame to the peer and shuts down the connection.

Parameters

- **code** (int | WsCloseCode, optional) – Close code.
- **message** (str | bytes, optional) – Close reason. Defaults to b''.
- **timeout** (float, optional) – How long (in seconds) to wait for the connection to close gracefully before force-terminating.

Raises

WebSocketError – The close reason is invalid UTF-8.

Note

WebSocket control frames are limited to 125-bytes. The close code requires 2 bytes, therefore 123-bytes of UTF-8 are available for the close reason. If the close reason is longer than this, it is implicitly truncated.

```
final class curl_cffi.requests.AsyncWebSocket(session: AsyncSession, curl: Curl, *, autoclose: bool =
    True, skip_utf8_validation: bool = False, debug: bool =
    False, recv_queue_size: int = 64, send_queue_size: int
    = 32, max_send_batch_size: int = 64, coalesce_frames:
    bool = False, ws_retry: WebSocketRetryStrategy | None
    = None, recv_time_slice: float = 0.01, send_time_slice:
    float = 0.01, max_message_size: int = 4194304,
    drain_on_error: bool = False,
    block_on_recv_queue_full: bool = True)
```

A high-performance asynchronous WebSocket implementation using libcurl.

```
__init__(session: AsyncSession, curl: Curl, *, autoclose: bool = True, skip_utf8_validation: bool = False,
    debug: bool = False, recv_queue_size: int = 64, send_queue_size: int = 32, max_send_batch_size:
    int = 64, coalesce_frames: bool = False, ws_retry: WebSocketRetryStrategy | None = None,
    recv_time_slice: float = 0.01, send_time_slice: float = 0.01, max_message_size: int = 4194304,
    drain_on_error: bool = False, block_on_recv_queue_full: bool = True) → None
```

Initializes an Async WebSocket session.

Do not instantiate this class directly. Use `AsyncSession.ws_connect`.

This class implements an async context manager, closing the connection automatically on exit:

```

async with AsyncSession() as session:
    async with session.ws_connect("wss://api.example.com") as ws:
        await ws.send("Hello")
        msg = await ws.recv()

```

Parameters

- **session** (*AsyncSession*) – The parent session object.
- **curl** (*Curl*) – The underlying Curl handle.
- **autoclose** (*bool*) – Automatically close on receiving a close frame.
- **skip_utf8_validation** – Skips UTF-8 validation of text frames in `recv_str()`, and `recv_json()`. When set, raise an error but don't close the connection as per RFC 6455 §8.1.
- **debug** (*bool*) – Enable verbose debug logging.
- **recv_queue_size** (*int*) – Max number of incoming messages to buffer.
- **send_queue_size** (*int*) – Max number of outgoing messages to buffer.
- **max_send_batch_size** (*int*) – Max frames to coalesce per transmission.
- **coalesce_frames** (*bool*) – Combine small frame payloads to improve throughput.
- **ws_retry** (*WebSocketRetryStrategy*) – Retry configuration for failed receives.
- **recv_time_slice** (*float*) – Max seconds to read messages before yielding.
- **send_time_slice** (*float*) – Max seconds to write messages before yielding.
- **max_message_size** (*int*) – Max size (bytes) of a single received message.
- **drain_on_error** (*bool*) – Yield buffered messages before raising errors.
- **block_on_recv_queue_full** (*bool*) – Behavior when the receive queue is full. If True (default), the reader blocks (may cause timeouts). If False, the connection fails immediately to prevent data loss.

Note

Architecture: This uses a background I/O model. Network operations run in background tasks. Network errors are raised in calls to `send()` or `recv()`.

Performance: Time slices default to 10ms. Increase these values to allocate more CPU time to I/O operations at the cost of event loop latency.

Increasing queue sizes reduces backpressure and buffers more messages in memory, increasing throughput - but increases latency and bufferbloat.

See also

- https://curl.se/libcurl/c/curl_ws_recv.html
- https://curl.se/libcurl/c/curl_ws_send.html

async recv(*, *timeout: float | None = None*) → tuple[bytes, int]

Receive a WebSocket message.

This method waits for and returns the next complete WebSocket message.

Parameters

timeout – How many seconds to wait for a message before raising a timeout.

Returns

A tuple with the received payload and flags.

Return type

tuple[bytes, int]

Raises

- **WebSocketTimeout** – If the timeout expires.
- **WebSocketClosed** – If the connection is closed.
- **WebSocketError** – WebSocket connection specific error.
- **CurlError** – If a network-level transport error occurs.

Note

Message fragmentation and reassembly are handled automatically by the implementation, so callers always receive complete messages.

WebSocketError exceptions may have originated from prior `send()` or `recv()` operations, since all operations share the same transport state once a failure occurs.

This method does not wait for additional messages after a transport error is detected. If `drain_on_error=True`, subsequent calls to `recv()` will return any messages that were buffered in the receive queue at the time the reader failed, before the connection error is raised.

Concurrent calls to `recv()` are supported and safe; each caller awaits the next available message and will receive distinct messages in FIFO order.

If this coroutine is cancelled while a message is being received, that message may be dropped. Cancellation is treated as abandoning the receive operation.

async recv_str(*, *timeout: float | None = None*) → str

Receive a text frame.

Parameters

timeout – How many seconds to wait before raising a timeout.

Raises

- **WebSocketClosed** – A close frame was received.
- **WebSocketError** – The payload contains invalid UTF-8.
- **WebSocketError** – The received frame is not a text frame.

Returns

The text frame payload string.

Return type

str

Note

This method checks if the message payload is valid UTF-8, unless `skip_utf8_validation` is set at connection time.

```
async recv_json(* , loads: ~collections.abc.Callable[[str], ~curl_cffi.requests.websockets.T] = <function loads>, timeout: float | None = None) → T
```

Receive a JSON frame.

Parameters

- **loads** – JSON decoder. Defaults to `json.loads()`.
- **timeout** – How many seconds to wait before raising a timeout.

Raises

- **WebSocketError** – The JSON data was empty.
- **WebSocketError** – The JSON was invalid.

Returns

Deserialized JSON object.

Return type

T

Note

This method checks if the message payload is valid UTF-8, unless `skip_utf8_validation` is set at connection time.

This method inherits the same exceptions from `recv_str()`.

```
async send(payload: str | bytes | bytearray | memoryview, flags: CurlWsFlag | int = CurlWsFlag.BINARY, timeout: float | None = None) → None
```

Send a WebSocket message.

Parameters

- **payload** – Data to send (str/bytes/bytearray/memoryview).
- **flags** – Frame type flags (e.g., `CurlWsFlag.TEXT` / `CurlWsFlag.BINARY`).
- **timeout** – Max seconds to wait if the send queue is full.

Raises

- **CurlError** – Network related exception occurred.
- **WebSocketClosed** – The WebSocket has been closed.
- **WebSocketTimeout** – The send operation timed out.
- **WebSocketError** – Writer stalled or socket closed while sending.

Note

There are no limits on the size of the message that can be sent. Large outbound messages are seamlessly broken down into optimal fragments using the `CURLWS_CONT` flag, arriving as a single logical message to the server.

Non-contiguous (i.e. strided) memoryviews are copied as they cannot be cast into a contiguous format required by `ffi.from_buffer()`.

Warning

This method is non-blocking. It queues the message for immediate transmission. Call `await ws.flush()` after sending if you need to guarantee that the data has actually reached the socket.

async send_binary(*payload: bytes | bytearray | memoryview, *, timeout: float | None = None*) → None
Send a binary frame.

Parameters

- **payload** – Binary data to send.
- **timeout** – Max seconds to wait if the send queue is full.

For more info, see the docstring for [send\(\)](#).

async send_bytes(*payload: bytes | bytearray | memoryview, *, timeout: float | None = None*) → None
Send a binary frame, alias of [send_binary\(\)](#).

Parameters

- **payload** – Binary data to send.
- **timeout** – Max seconds to wait if the send queue is full.

For more info, see the docstring for [send\(\)](#).

async send_str(*payload: str, *, timeout: float | None = None*) → None
Send a text frame.

Parameters

- **payload** – Text data to send.
- **timeout** – Max seconds to wait if the send queue is full.

For more info, see the docstring for [send\(\)](#).

async send_json(*payload: object, *, dumps: ~collections.abc.Callable[[...], str | bytes] = <function dumps>, timeout: float | None = None*) → None

Send a JSON frame.

Parameters

- **payload** – Data to send.
- **dumps** – JSON encoder, default is `json.dumps()`. The encoder may return `str` or `bytes`; bytes must be UTF-8, since the payload is sent as a text frame.
- **timeout** – Max seconds to wait if the send queue is full.

For more info, see the docstring for [send\(\)](#).

async ping(payload: str | bytes = b'', *, timeout: float | None = None) → None

Send a ping frame.

Parameters

- **payload** – Ping payload to send. Defaults to b''.
- **timeout** – Max seconds to wait if the send queue is full.

Raises

WebSocketError – The payload length is outside specification (>125 bytes).

For more info, see the docstring for [send\(\)](#).

async close(code: int | WsCloseCode = WsCloseCode.OK, message: str | bytes = b'', timeout: float = 3.0) → None

Close the WebSocket connection.

Sends a close frame to the peer, waits for queued outgoing messages to be sent, and then shuts down the connection.

Parameters

- **code** (int | WsCloseCode, optional) – Close code.
- **message** (str | bytes, optional) – Close reason. Defaults to b''.
- **timeout** (float, optional) – How long (in seconds) to wait for the connection to close gracefully before force-terminating.

Raises

WebSocketError – The close reason is invalid UTF-8.

Note

WebSocket control frames are limited to 125-bytes. The close code requires 2 bytes, therefore 123-bytes of UTF-8 are available for the close reason. If the close reason is longer than this, it is implicitly truncated.

Transport errors during the closing handshake are suppressed. The connection tears down regardless. The teardown also runs if the calling task is cancelled.

9.5 Exceptions and Warnings

9.5.1 Exceptions

We try to follow the *requests* exception hierarchy, however, some are missing, while some are added.

If an exception is marked as “not used”, please catch the base exception.

```
class curl_cffi.requests.exceptions.RequestException(msg, code: CurlECode | Literal[0] = 0,
                                                    response=None, *args: Any, **kwargs: Any)
```

Base exception for curl_cffi.requests package

```
class curl_cffi.requests.exceptions.CookieConflict(msg, code: CurlECode | Literal[0] = 0,
                                                    response=None, *args: Any, **kwargs: Any)
```

Same cookie exists for different domains.

```
class curl_cffi.requests.exceptions.SessionClosed(msg, code: CurlECode | Literal[0] = 0,  
response=None, *args: Any, **kwargs: Any)
```

The session has already been closed.

```
class curl_cffi.requests.exceptions.ImpersonateError(msg, code: CurlECode | Literal[0] = 0,  
response=None, *args: Any, **kwargs: Any)
```

The impersonate config was wrong or impersonate failed.

```
class curl_cffi.requests.exceptions.InvalidJSONError(msg, code: CurlECode | Literal[0] = 0,  
response=None, *args: Any, **kwargs: Any)
```

A JSON error occurred. not used

```
class curl_cffi.requests.exceptions.HTTPError(msg, code: CurlECode | Literal[0] = 0, response=None,  
*args: Any, **kwargs: Any)
```

An HTTP error occurred.

```
class curl_cffi.requests.exceptions.IncompleteRead(msg, code: CurlECode | Literal[0] = 0,  
response=None, *args: Any, **kwargs: Any)
```

Incomplete read of content

```
class curl_cffi.requests.exceptions.ConnectionError(msg, code: CurlECode | Literal[0] = 0,  
response=None, *args: Any, **kwargs: Any)
```

A Connection error occurred.

```
class curl_cffi.requests.exceptions.DNSError(msg, code: CurlECode | Literal[0] = 0, response=None,  
*args: Any, **kwargs: Any)
```

Could not resolve

```
class curl_cffi.requests.exceptions.ProxyError(msg, code: CurlECode | Literal[0] = 0,  
response=None, *args: Any, **kwargs: Any)
```

A proxy error occurred.

```
class curl_cffi.requests.exceptions.SSLError(msg, code: CurlECode | Literal[0] = 0, response=None,  
*args: Any, **kwargs: Any)
```

An SSL error occurred.

```
class curl_cffi.requests.exceptions.CertificateVerifyError(msg, code: CurlECode | Literal[0] = 0,  
response=None, *args: Any, **kwargs: Any)
```

Raised when certificate validated has failed

```
class curl_cffi.requests.exceptions.Timeout(msg, code: CurlECode | Literal[0] = 0, response=None,  
*args: Any, **kwargs: Any)
```

The request timed out.

```
class curl_cffi.requests.exceptions.ConnectTimeout(msg, code: CurlECode | Literal[0] = 0,  
response=None, *args: Any, **kwargs: Any)
```

The request timed out while trying to connect to the remote server.

Requests that produced this error are safe to retry.

not used

```
class curl_cffi.requests.exceptions.ReadTimeout(msg, code: CurlECode | Literal[0] = 0,  
response=None, *args: Any, **kwargs: Any)
```

The server did not send any data in the allotted amount of time. not used

```
class curl_cffi.requests.exceptions.URLRequired(msg, code: CurlECode | Literal[0] = 0,
                                              response=None, *args: Any, **kwargs: Any)
    A valid URL is required to make a request. not used
```

```
class curl_cffi.requests.exceptions.TooManyRedirects(msg, code: CurlECode | Literal[0] = 0,
                                                    response=None, *args: Any, **kwargs: Any)
    Too many redirects.
```

```
class curl_cffi.requests.exceptions.MissingSchema(msg, code: CurlECode | Literal[0] = 0,
                                                  response=None, *args: Any, **kwargs: Any)
    The URL scheme (e.g. http or https) is missing. not used
```

```
class curl_cffi.requests.exceptions.InvalidSchema(msg, code: CurlECode | Literal[0] = 0,
                                                  response=None, *args: Any, **kwargs: Any)
    The URL scheme provided is either invalid or unsupported. not used
```

```
class curl_cffi.requests.exceptions.InvalidURL(msg, code: CurlECode | Literal[0] = 0,
                                              response=None, *args: Any, **kwargs: Any)
    The URL provided was somehow invalid.
```

```
class curl_cffi.requests.exceptions.InvalidHeader(msg, code: CurlECode | Literal[0] = 0,
                                                  response=None, *args: Any, **kwargs: Any)
    The header value provided was somehow invalid. not used
```

```
class curl_cffi.requests.exceptions.InvalidProxyURL(msg, code: CurlECode | Literal[0] = 0,
                                                    response=None, *args: Any, **kwargs: Any)
    The proxy URL provided is invalid. not used
```

```
class curl_cffi.requests.exceptions.ChunkedEncodingError(msg, code: CurlECode | Literal[0] = 0,
                                                         response=None, *args: Any, **kwargs:
                                                         Any)
    The server declared chunked encoding but sent an invalid chunk. not used
```

```
class curl_cffi.requests.exceptions.ContentDecodingError(msg, code: CurlECode | Literal[0] = 0,
                                                         response=None, *args: Any, **kwargs:
                                                         Any)
    Failed to decode response content. not used
```

```
class curl_cffi.requests.exceptions.StreamConsumedError(msg, code: CurlECode | Literal[0] = 0,
                                                         response=None, *args: Any, **kwargs:
                                                         Any)
    The content for this response was already consumed. not used
```

```
class curl_cffi.requests.exceptions.RetryError(msg, code: CurlECode | Literal[0] = 0,
                                              response=None, *args: Any, **kwargs: Any)
    Custom retries logic failed. not used
```

```
class curl_cffi.requests.exceptions.UnrewindableBodyError(msg, code: CurlECode | Literal[0] = 0,
                                                         response=None, *args: Any, **kwargs:
                                                         Any)
    The request body could not be rewound for a redirect or retry.
```

```
class curl_cffi.requests.exceptions.InterfaceError(msg, code: CurlECode | Literal[0] = 0,
                                                    response=None, *args: Any, **kwargs: Any)
    A specified outgoing interface could not be used.
```

9.5.2 Warnings

class curl_cffi.requests.exceptions.**RequestsWarning**

Base warning for Requests. not used

class curl_cffi.requests.exceptions.**FileModeWarning**

A file was opened in text mode, but Requests determined its binary length. not used

class curl_cffi.requests.exceptions.**RequestsDependencyWarning**

An imported dependency doesn't match the expected version range.

10.1 SSRF protection

Server-Side Request Forgery (SSRF) is an attack where an attacker can induce a server-side application to make HTTP requests to an unintended destination. A common vector is **open redirects**: a request to a public URL returns a 3xx redirect pointing to an internal or private IP address (e.g. 127.0.0.1, 10.x.x.x, 169.254.169.254). If the client blindly follows the redirect, the attacker gains access to internal services.

By default, `curl_cffi` follows redirects (`allow_redirects=True`) without restricting the target, which mirrors the behavior of `requests` and `httpx`. This is fine for desktop applications and scripts, but **dangerous in server-side contexts** where user-supplied URLs are fetched.

10.1.1 Example: vulnerable FastAPI app

Consider a preview service that fetches a URL on behalf of the user:

```
from fastapi import FastAPI
from curl_cffi import requests

app = FastAPI()

@app.get("/preview")
def preview(url: str):
    # VULNERABLE: follows redirects to any destination
    r = requests.get(url)
    return {"body": r.text}
```

An attacker can exploit this by supplying a URL that redirects to an internal service:

```
GET /preview?url=https://evil.com/redirect
```

Where `https://evil.com/redirect` returns:

```
HTTP/1.1 302 Found
Location: http://169.254.169.254/latest/meta-data/iam/security-credentials/
```

The application follows the redirect and returns AWS instance credentials (or any other internal resource) to the attacker.

The fix is to use `CurlFollow.SAFE`:

```
from fastapi import FastAPI
from curl_cffi import CurlFollow, requests
```

(continues on next page)

(continued from previous page)

```

app = FastAPI()

@app.get("/preview")
def preview(url: str):
    # SAFE: rejects redirects to internal/private IPs
    r = requests.get(url, allow_redirects=CurlFollow.SAFE)
    return {"body": r.text}

```

10.1.2 Using CurlFollow.SAFE

curl_cffi exposes a CurlFollow.SAFE option backed by curl-impersonate that rejects redirects to internal/private IP addresses:

```

from curl_cffi import CurlFollow, requests

# Recommended for server-side code that fetches user-supplied URLs
r = requests.get(url, allow_redirects=CurlFollow.SAFE)

# The string "safe" is also accepted
r = requests.get(url, allow_redirects="safe")

# Session-level setting
s = requests.Session(allow_redirects=CurlFollow.SAFE)

```

With CurlFollow.SAFE, redirects to the following address ranges are rejected:

- Loopback (127.0.0.0/8, ::1)
- Private networks (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16)
- Link-local (169.254.0.0/16, fe80::/10)

10.1.3 allow_redirects values

Value	Behavior
True (default)	Follow all redirects (same as CurlFollow.ALL)
False	Do not follow any redirects
CurlFollow.SAFE or "safe"	Follow redirects, but reject redirects to internal/private IPs
CurlFollow.ALL	Follow all redirects
CurlFollow.OBEYCODE	Follow redirects, reset custom method on 301/302/303
CurlFollow.FIRSTONLY	Only use custom method for the first request, reset for subsequent redirects

10.1.4 Recommendation

If your application fetches URLs provided by users or external systems, always use CurlFollow.SAFE:

```

from curl_cffi import CurlFollow, requests

session = requests.Session(allow_redirects=CurlFollow.SAFE)

```

This protects against SSRF without disabling redirects entirely.

11.1 What does the pro version offer? Is the open source project still maintained?

Yes, the open source project is maintained as before.

In the [pro version](#), we provide:

- weekly update of targets
- profiles for mobile browsers and apps
- some private detection fields
- http/3 fingerprints and proxy support

And, a better financial situation will help the open source version better maintained.

11.2 Why does the JA3 fingerprints change for Chrome 110+ impersonation?

This is intended.

Chrome introduces `ClientHello` permutation in version 110, which means the order of extensions will be random, thus JA3 fingerprints will be random. So, when comparing JA3 fingerprints of `curl_cffi` and a browser, they may differ. However, this does not mean that TLS fingerprints will not be a problem, `ClientHello` extension order is just one factor of how servers can tell automated requests from browsers.

Roughly, this can be mitigated like:

```
ja3 = md5(list(extensions), ...other arguments)
ja3n = md5(set(extensions), ...other arguments)
```

See more from [this article](#) and [curl-impersonate notes](#).

11.3 Can I bypass Cloudflare with this project? or any other specific site.

Short answer is: it depends.

TLS and http2 fingerprints are just one of the many factors Cloudflare considers. Other factors include but are not limited to: IP quality, request rate, JS fingerprints, etc.

There are different protection levels for website owners to choose. For the most basic ones, TLS fingerprints alone maybe enough, but for higher levels, you may need to find a better proxy IP provider and use browser automation tools like playwright.

If you are in a hurry or just want the professionals to take care of the hard parts, you can consider the commercial solutions from our sponsors:

- [Yescaptcha](#), captcha resolver and proxy service for bypassing Cloudflare.
- [ScrapeNinja](#), Managed web scraping API.

For details, see the *Sponsor* section on front page.

11.4 I'm getting certs errors

The simplest way is to turn off cert verification by `verify=False`:

```
r = curl_cffi.get("https://example.com", verify=False)
```

11.5 ErrCode: 77, Reason: error setting certificate verify locations

On Windows, if your Python environment or CA bundle path contains non-ASCII characters (e.g. accents), libcurl may fail to open the CA file when passed as a narrow `char*`. `curl_cffi` now encodes file-path options (e.g. `CAINFO`, `PROXY_CAINFO`, `SSLCERT`) using the system's preferred ANSI code page on Windows to ensure correct file access. This fixes most occurrences of error 77.

11.6 How to use with fiddler/charles to intercept content

Fiddler and Charles uses man-in-the-middle self-signed certs to intercept TLS traffic, to use with them, simply set `verify=False`.

11.7 ErrCode: 92, Reason: 'HTTP/2 stream 0 was not closed cleanly: PROTOCOL_ERROR (err 1)'

This error(http/2 stream 0) has been reported many times ever since `curl_cffi` was published, but I still can not find a reproducible way to trigger it. Given that the majority users are behind proxies, the situation is even more difficult to deal with.

I'm even not sure it's a bug introduced in libcurl, curl-impersonate or `curl_cffi`, or it's just a server error. Depending on your context, here are some general suggestions for you:

- First, try removing the `Content-Length` header from you request.
- Try to see if this error was caused by proxies, if so, use better proxies.
- If it stops working after a while, maybe you're just being blocked by, such as, Akamai.
- Force http/1.1 mode. Some websites' h2 implementation is simply broken.
- See if the url works in your real browser.
- Find a stable way to reproduce it, so we can finally fix, or at least bypass it.

To force curl to use http 1.1 only.

```
import curl_cffi

r = curl_cffi.get("https://postman-echo.com", http_version=curl_cffi.CurlHttpVersion.V1_
↪1)
```

Related issues:

- [#19](#),
- [#42](#),
- [#79](#),
- [#165](#),

11.8 Packaging with PyInstaller

If you encountered any issue with PyInstaller, here are a list of options provided by the community:

Add the `--hidden-import` option.

```
pyinstaller -F .\example.py --hidden-import=_cffi_backend --collect-all curl_cffi
```

Add other paths:

```
pyinstaller --noconfirm --onefile --console \
    --paths "C:/Users/Administrator/AppData/Local/Programs/Python/Python39" \
    --add-data "C:/Users/Administrator/AppData/Local/Programs/Python/Python39/Lib/site-
↪packages/curl_cffi.libs/libcurl-cbb416caa1dd01638554eab3f38d682d.dll;." \
    --collect-data "curl_cffi" \
    "C:/Users/Administrator/Desktop/test_script.py"
```

See also:

- [#5](#)
- [#48](#)

11.9 How to change the order of headers?

By default, setting `impersonate` parameter will bring the corresponding headers. If you want to change the order or use your own headers, you need to turn off that and bring your own headers. For stored fingerprint targets, you can also load and edit a `Fingerprint` object directly.

```
fingerprint = curl_cffi.get_fingerprint("edge_146_macos_26")
fingerprint.headers["User-Agent"] = "...
requests.get(url, impersonate=fingerprint)
```

11.10 How to deal with encoding/decoding errors?

Use `chardet` or `cchardet`

```
>>> import curl_cffi
>>> r = curl_cffi.get("https://example.com/messy_codec.html")
```

(continues on next page)

(continued from previous page)

```
>>> import chardet
>>> chardet.detect(r.content)
{'encoding': 'GB2312', 'confidence': 0.99, 'language': 'Chinese'}
```

Or use regex or lxml to parse the meta header:

```
<meta http-equiv="Content-Type" content="text/html; charset=gbk" />
```

CHANGE LOG

This doc is no longer maintained.

Please see the [GitHub Releases](#) page for details.

- **v0.13**
 - Added support for Windows on Arm
 - Improved support for websockets
- **v0.12**
 - Added support for safari 26
 - Improved support for websockets
- **v0.11**
 - Added support for http3
 - Added tor145, new safari and chrome targets
- **v0.10.0**
 - Added support for using curl_cffi directly
- **v0.9.0**
 - Brought back Windows support
 - Added support for Firefox
 - Added support for Chrome 133a
- **v0.8.0**
 - Added more recent impersonate versions, Safari 18.0 for iOS and macOS, Chrome 131.
 - Added `quote` parameter for setting which letter should be quoted in URL.
 - Added `response_class` parameter for using a customized Response class.
- **v0.7.3**
 - Bugfixes.
- **v0.7.2**
 - Added requests-like exception hierarchy.
- **v0.7.1**
 - Added `Cookies.get_dict()`, for compatibility with requests.

- Fixed type conversion in C shim, by @qishipai.
- Fixed cookie `subdomains` attribute.
- **v0.7.0**
 - Added more recent impersonate versions, up to Chrome 124.
 - Upgraded `libcurl` to 8.7.1.
 - Supported custom impersonation.
 - Added support for list of tuple in post fields.
 - Updated header strategy: always exclude empty headers, never send Expect header.
 - Changed default redirect limit to 30.
 - Prefer not sending CONNECT for plain http proxy.
 - Fix Windows build.
 - Fix Safari Stream priority.

The minimum Python version is now 3.8. Windows fingerprints are wrong in 0.6.x.

- **v0.6.1**
 - `AsyncSession.close` is now a coroutine.
 - This is a bugfix release.
- **v0.6.0**
 - Added more recent impersonate versions, up to Chrome 120 and Safari 17.0
 - Upgraded `libcurl` to 8.1.1
 - Added experimental websocket support
 - Supported proactive eventloop on Windows
 - Added win32 and macOS arm64 build targets
 - Added `allow_redirects` to Session parameters
 - Use `certifi` to replace packaged `cacert.pem`
 - Improved proxy support by accepting `proxy=...`
 - Bumped minimum python version to 3.8
 - Added files support
 - Added client certs support
 - Incorporated build time files for `sdist`
 - Bugfix: async curl timer leak
- **v0.5.10**
 - Add stream support
 - Add support for secure cookies
 - Add `curl_infos` to extract extra info after performing
 - Bugfix: `timeout=None` not working
- **v0.5.9**

- Add interface support
 - Make POST work as in real world
 - Add support for custom resolve
 - Switched to libcurl's COOKIELIST to sync cookies between python and curl
 - Add default_headers option for sessions like in curl-impersonate
 - Add curl_options for extra curl_options in Session
 - Add http_version option for limiting http version to 1.1 or whatever
 - Add debug option for extra curl debug info
 - Add CurlError.code
 - Bugfix: duplicated header lines for the same header
 - Bugfix: clearing headers when request fails
 - Bugfix: fix HEAD request
 - Bugfix: reset curl options when errors occur
- **v0.5.7**
 - Refactor JSON serialization to mimic browser behavior (#66)
 - Add http options to Session classes (#72)
 - Add Windows eventloop warning
- **v0.5.6**
 - Make Session.curl a thread-local variable (#50)
 - Add support for eventlet and gevent with threadpool
 - Bugfix: Only close future if it's not done or cancelled
- **0.5.5**
 - Bugfix: Fix high CPU usage (#46)
- **0.5.4**
 - Bugfix: Fix cert and error buffer when calling curl_easy_reset
- **0.5.3**
 - Bugfix: Reset curl after performing, fix #39
- **0.5.2**
 - Bugfix: Clear headers after async perform
- **0.5.1**
 - Bugfix: Clean up timerfunction when curl already closed
- **0.5.0**
 - Added asyncio support
- **0.4.0**
 - Removed c shim callback function, use cffi native callback function
- **0.3.6**

- Updated to curl-impersonate v0.5.4, supported chrome107 and chrome110
- **0.3.0, copied more code from *httpx* to support session**
 - Add *requests.Session*
 - Breaking change: *Response.cookies* changed from *http.cookies.SimpleCookie* to *curl_cffi.requests.Cookies*
 - Using ABI3 wheels to reduce package size.

DEVELOPMENT AND CONTRIBUTING

This page documents how to compile curl-impersonate and curl-cffi from source. If binary package is not available on your platform, you may refer to this page for some inspirations.

First, you need to check if there are libcurl-impersonate binaries for you platform. If so, you can simply download and install them.

For now, a pre-compiled *libcurl-impersonate* is downloaded from github and built into a bdist wheel, which is a binary package format used by PyPI. However, the right way is to download curl and curl-impersonate sources on our side and compile them all together.

By default, source builds download the platform-specific library into a temporary directory. Set `IMPERSONATE_BUILD_DIR` to reuse an existing download or choose where it is downloaded and unpacked. The directory must contain the library filename configured for the current platform, such as `libcurl-impersonate.so` or `libcurl-impersonate.a`:

```
IMPERSONATE_BUILD_DIR=/path/to/lib pip install .
```

On macOS, static builds require `pkg-config` and the `libcurl-impersonate.pc` metadata included in curl-impersonate 2.3.0 and newer. Keep the complete release archive contents together: the metadata supplies the system libraries and frameworks needed by the static archive. For local installs, the metadata may also be in the library directory's `pkgconfig` subdirectory. Other platforms do not use this `pkg-config` lookup.

The default link type is configured per platform. Set `IMPERSONATE_LINK_TYPE` to `dynamic` or `static` to override it. A dynamic local build can be used without merging its static dependency archives:

```
IMPERSONATE_BUILD_DIR=/path/to/lib \  
IMPERSONATE_LINK_TYPE=dynamic pip install -e .
```

On macOS, the directory must contain `libcurl-impersonate.dylib` and any versioned symlinks required by its install name. The directory is recorded as a runtime search path in the extension module.

13.1 macOS

To install the local editable build:

```
# This is for using the libcurl-impersonate built by GitHub actions  
  
sudo mkdir /Users/runner  
sudo chmod 777 /Users/runner  
  
# Dependencies
```

(continues on next page)

(continued from previous page)

```
brew install libidn2 zstd pkg-config
```

```
# Then install
```

```
pip install -e .[test]
```

```
pip install -e .[dev]
```

13.2 Contributing

When opening PR, please do not use the main branch in your fork, otherwise I cannot add my modification, such as unittests.

[Discuss on Telegram](#)

curl_cffi is a Python binding for [curl-impersonate fork](#) via [cffi](#). For commercial support, visit [impersonate.pro](#).

Unlike other pure Python http clients like `httplib` or `requests`, `curl_cffi` can impersonate browsers' TLS signatures or JA3 fingerprints. If you are blocked by some website for no obvious reason, you can give this package a try.

If you are looking for Python http3 clients, `curl_cffi` added http3 support since `v0.11`.

SPONSORS

14.1 Bypass Cloudflare with API



[Yescaptcha](#) is a proxy service that bypasses Cloudflare and uses the API interface to obtain verified cookies (e.g. `cf_clearance`). Click [here](#) to register.

14.2 Cloro



[cloro](#): SERP and AI search API for Google, ChatGPT, Perplexity, Gemini, Copilot and Grok.

You can also click [here](#) to buy me a coffee.

FEATURES

- Supports JA3/TLS and http2 fingerprints impersonation, including recent browsers and custom fingerprints.
- Much faster than requests/httpx, on par with aiohttp/pycurl, see [benchmarks](#).
- Mimics requests API, no need to learn another one.
- Pre-compiled, so you don't have to compile on your machine.
- Supports `asyncio` with proxy rotation on each request.
- Supports http 2.0 & 3.0, which requests does not.
- Supports websocket.

Table 1: Feature matrix

	requests	aiohttp	httpx	pycurl	curl_cffi
http2					
http3					
sync					
async					
websocket					
native retry					
fingerprints					
speed					

Notes:

1. For pycurl, you need a http/3 enabled libcurl, while curl_cffi packages libcurl-impersonate inside Python wheels.
2. Full http/3 supported was added in v0.12.0.

INSTALL

```
pip install curl_cffi --upgrade
```

For more details, see *Quick Start*.

DOCUMENTATION

You can first check out [Quick Start](#). Then [Impersonate Guide](#).

For advanced topics, checkout [Cookies](#), [asyncio](#) and [websockets](#).

Since v0.15.1, you can update fingerprints without updating `curl_cffi`, see [Fingerprint Updates](#).

Moreover, **we offer comprehensive fingerprint suites at [impersonate.pro](#)**.

There is a CLI helper for managing fingerprints and debugging, see [CLI](#).

You can also find common use cases in the [examples](#) directory.

Finally, if something is missing from the tutorial, you can always find them in the [API References](#).

If you have any questions, be sure to check out the [FAQ](#) section before opening an issue.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

Symbols

`__delitem__()` (*curl_cffi.requests.Cookies method*), 66
`__delitem__()` (*curl_cffi.requests.Headers method*), 65
`__getitem__()` (*curl_cffi.requests.Cookies method*), 66
`__getitem__()` (*curl_cffi.requests.Headers method*), 65
`__init__()` (*curl_cffi.AsyncCurl method*), 53
`__init__()` (*curl_cffi.Curl method*), 51
`__init__()` (*curl_cffi.CurlMime method*), 54
`__init__()` (*curl_cffi.CurlShare method*), 54
`__init__()` (*curl_cffi.requests.AsyncSession method*), 60
`__init__()` (*curl_cffi.requests.AsyncWebSocket method*), 74
`__init__()` (*curl_cffi.requests.CacheBackend method*), 64
`__init__()` (*curl_cffi.requests.FileCacheBackend method*), 65
`__init__()` (*curl_cffi.requests.Session method*), 57
`__init__()` (*curl_cffi.requests.WebSocket method*), 69
`__setitem__()` (*curl_cffi.requests.Cookies method*), 66
`__setitem__()` (*curl_cffi.requests.Headers method*), 65
`_delete_payload()` (*curl_cffi.requests.CacheBackend method*), 65
`_read_payload()` (*curl_cffi.requests.CacheBackend method*), 65
`_write_payload()` (*curl_cffi.requests.CacheBackend method*), 65

A

`aclose()` (*curl_cffi.requests.Response method*), 68
`acontent()` (*curl_cffi.requests.Response method*), 68
`add_handle()` (*curl_cffi.AsyncCurl method*), 53
`addpart()` (*curl_cffi.CurlMime method*), 54
`aiter_content()` (*curl_cffi.requests.Response method*), 68
`aiter_lines()` (*curl_cffi.requests.Response method*), 68
`AsyncCurl` (*class in curl_cffi*), 53
`AsyncSession` (*class in curl_cffi.requests*), 60
`AsyncWebSocket` (*class in curl_cffi.requests*), 74
`atext()` (*curl_cffi.requests.Response method*), 68
`attach()` (*curl_cffi.CurlMime method*), 54

B

`body` (*curl_cffi.requests.Request attribute*), 66

C

`CacheBackend` (*class in curl_cffi.requests*), 64
`CertificateVerifyError` (*class in curl_cffi.requests.exceptions*), 80
`charset` (*curl_cffi.requests.Response attribute*), 67
`charset_encoding` (*curl_cffi.requests.Response attribute*), 67
`ChunkedEncodingError` (*class in curl_cffi.requests.exceptions*), 81
`clear()` (*curl_cffi.requests.CacheBackend method*), 65
`clear()` (*curl_cffi.requests.Cookies method*), 66
`clear()` (*curl_cffi.requests.FileCacheBackend method*), 65
`close()` (*curl_cffi.AsyncCurl method*), 54
`close()` (*curl_cffi.Curl method*), 52
`close()` (*curl_cffi.CurlMime method*), 54
`close()` (*curl_cffi.CurlShare method*), 55
`close()` (*curl_cffi.requests.AsyncSession method*), 62
`close()` (*curl_cffi.requests.AsyncWebSocket method*), 79
`close()` (*curl_cffi.requests.Response method*), 68
`close()` (*curl_cffi.requests.WebSocket method*), 74
`connect()` (*curl_cffi.requests.WebSocket method*), 69
`ConnectionError` (*class in curl_cffi.requests.exceptions*), 80
`ConnectTimeout` (*class in curl_cffi.requests.exceptions*), 80
`content` (*curl_cffi.requests.Response attribute*), 66
`ContentDecodingError` (*class in curl_cffi.requests.exceptions*), 81
`CookieConflict` (*class in curl_cffi.requests.exceptions*), 79
`Cookies` (*class in curl_cffi.requests*), 66
`cookies` (*curl_cffi.requests.Response attribute*), 67
`Curl` (*class in curl_cffi*), 51
`CurlECode` (*class in curl_cffi*), 55
`CurlHttpVersion` (*class in curl_cffi*), 55
`CurlInfo` (*class in curl_cffi*), 55
`CurlMime` (*class in curl_cffi*), 54
`CurlMOpt` (*class in curl_cffi*), 55

`CurlOpt` (class in `curl_cffi`), 55
`CurlProxyType` (class in `curl_cffi`), 55
`CurlShare` (class in `curl_cffi`), 54
`CurlSslVersion` (class in `curl_cffi`), 55
`CurlWsFlag` (class in `curl_cffi`), 55

D

`debug()` (`curl_cffi.Curl` method), 51
`default_encoding` (`curl_cffi.requests.Response` attribute), 67
`delete()` (`curl_cffi.requests.CacheBackend` method), 65
`delete()` (`curl_cffi.requests.Cookies` method), 66
`DNSError` (class in `curl_cffi.requests.exceptions`), 80
`download_size` (`curl_cffi.requests.Response` attribute), 67
`duphandle()` (`curl_cffi.Curl` method), 52

E

`elapsed` (`curl_cffi.requests.Response` attribute), 67
`encoding` (`curl_cffi.requests.Headers` property), 65
`encoding` (`curl_cffi.requests.Response` attribute), 67

F

`FileCacheBackend` (class in `curl_cffi.requests`), 65
`FileModeWarning` (class in `curl_cffi.requests.exceptions`), 82
`from_list()` (`curl_cffi.CurlMime` class method), 54

G

`get()` (`curl_cffi.requests.CacheBackend` method), 64
`get()` (`curl_cffi.requests.Cookies` method), 66
`get()` (`curl_cffi.requests.Headers` method), 65
`get_list()` (`curl_cffi.requests.Headers` method), 65
`get_reason_phrase()` (`curl_cffi.Curl` static method), 52
`getinfo()` (`curl_cffi.Curl` method), 51

H

`header_size` (`curl_cffi.requests.Response` attribute), 68
`Headers` (class in `curl_cffi.requests`), 65
`headers` (`curl_cffi.requests.Request` attribute), 66
`headers` (`curl_cffi.requests.Response` attribute), 67
`history` (`curl_cffi.requests.Response` attribute), 67
`http_version` (`curl_cffi.requests.Response` attribute), 67
`HTTPError` (class in `curl_cffi.requests.exceptions`), 80

I

`impersonate()` (`curl_cffi.Curl` method), 51
`ImpersonateError` (class in `curl_cffi.requests.exceptions`), 80
`IncompleteRead` (class in `curl_cffi.requests.exceptions`), 80

`InterfaceError` (class in `curl_cffi.requests.exceptions`), 81
`InvalidHeader` (class in `curl_cffi.requests.exceptions`), 81
`InvalidJSONError` (class in `curl_cffi.requests.exceptions`), 80
`InvalidProxyURL` (class in `curl_cffi.requests.exceptions`), 81
`InvalidSchema` (class in `curl_cffi.requests.exceptions`), 81
`InvalidURL` (class in `curl_cffi.requests.exceptions`), 81
`iter_content()` (`curl_cffi.requests.Response` method), 68
`iter_lines()` (`curl_cffi.requests.Response` method), 68

J

`json()` (`curl_cffi.requests.Response` method), 68

L

`local_ip` (`curl_cffi.requests.Response` attribute), 67
`local_port` (`curl_cffi.requests.Response` attribute), 67

M

`method` (`curl_cffi.requests.Request` attribute), 66
`MissingSchema` (class in `curl_cffi.requests.exceptions`), 81
`multi_items()` (`curl_cffi.requests.Headers` method), 65

O

`ok` (`curl_cffi.requests.Response` attribute), 67

P

`parse_cookie_headers()` (`curl_cffi.Curl` method), 52
`parse_status_line()` (`curl_cffi.Curl` static method), 52
`perform()` (`curl_cffi.Curl` method), 52
`ping()` (`curl_cffi.requests.AsyncWebSocket` method), 78
`ping()` (`curl_cffi.requests.WebSocket` method), 73
`primary_ip` (`curl_cffi.requests.Response` attribute), 67
`primary_port` (`curl_cffi.requests.Response` attribute), 67
`process_data()` (`curl_cffi.AsyncCurl` method), 54
`ProxyError` (class in `curl_cffi.requests.exceptions`), 80

R

`raise_for_status()` (`curl_cffi.requests.Response` method), 68
`raw()` (`curl_cffi.requests.Headers` method), 65
`ReadTimeout` (class in `curl_cffi.requests.exceptions`), 80
`reason` (`curl_cffi.requests.Response` attribute), 67
`recv()` (`curl_cffi.requests.AsyncWebSocket` method), 75
`recv()` (`curl_cffi.requests.WebSocket` method), 71

recv_json() (curl_cffi.requests.AsyncWebSocket method), 77
 recv_json() (curl_cffi.requests.WebSocket method), 72
 recv_str() (curl_cffi.requests.AsyncWebSocket method), 76
 recv_str() (curl_cffi.requests.WebSocket method), 71
 redirect_count (curl_cffi.requests.Response attribute), 67
 redirect_url (curl_cffi.requests.Response attribute), 67
 remove_handle() (curl_cffi.AsyncCurl method), 53
 Request (class in curl_cffi.requests), 66
 request() (curl_cffi.requests.AsyncSession method), 62
 request() (curl_cffi.requests.Session method), 58
 request() (in module curl_cffi.requests), 55
 request_size (curl_cffi.requests.Response attribute), 68
 RequestException (class in curl_cffi.requests.exceptions), 79
 RequestsDependencyWarning (class in curl_cffi.requests.exceptions), 82
 RequestsWarning (class in curl_cffi.requests.exceptions), 82
 reset() (curl_cffi.Curl method), 52
 Response (class in curl_cffi.requests), 66
 response_size (curl_cffi.requests.Response attribute), 68
 RetryError (class in curl_cffi.requests.exceptions), 81
 run_forever() (curl_cffi.requests.WebSocket method), 73

S

send() (curl_cffi.requests.AsyncWebSocket method), 77
 send() (curl_cffi.requests.WebSocket method), 72
 send_binary() (curl_cffi.requests.AsyncWebSocket method), 78
 send_binary() (curl_cffi.requests.WebSocket method), 73
 send_bytes() (curl_cffi.requests.AsyncWebSocket method), 78
 send_bytes() (curl_cffi.requests.WebSocket method), 73
 send_json() (curl_cffi.requests.AsyncWebSocket method), 78
 send_json() (curl_cffi.requests.WebSocket method), 73
 send_str() (curl_cffi.requests.AsyncWebSocket method), 78
 send_str() (curl_cffi.requests.WebSocket method), 73
 Session (class in curl_cffi.requests), 57
 SessionClosed (class in curl_cffi.requests.exceptions), 79
 set() (curl_cffi.requests.CacheBackend method), 64
 set() (curl_cffi.requests.Cookies method), 66
 set_exception() (curl_cffi.AsyncCurl method), 53
 set_result() (curl_cffi.AsyncCurl method), 53
 setopt() (curl_cffi.AsyncCurl method), 53
 setopt() (curl_cffi.Curl method), 51
 share() (curl_cffi.CurlShare method), 55
 should_cache_request() (curl_cffi.requests.CacheBackend method), 64
 should_store_response() (curl_cffi.requests.CacheBackend method), 64
 socket_action() (curl_cffi.AsyncCurl method), 53
 SSLError (class in curl_cffi.requests.exceptions), 80
 status_code (curl_cffi.requests.Response attribute), 66
 stream() (curl_cffi.requests.AsyncSession method), 62
 stream() (curl_cffi.requests.Session method), 58
 StreamConsumedError (class in curl_cffi.requests.exceptions), 81

T

text (curl_cffi.requests.Response attribute), 66
 Timeout (class in curl_cffi.requests.exceptions), 80
 TooManyRedirects (class in curl_cffi.requests.exceptions), 81

U

UnrewindableBodyError (class in curl_cffi.requests.exceptions), 81
 unshare() (curl_cffi.CurlShare method), 55
 update() (curl_cffi.requests.Cookies method), 66
 update() (curl_cffi.requests.Headers method), 65
 upload_size (curl_cffi.requests.Response attribute), 67
 url (curl_cffi.requests.Request attribute), 66
 url (curl_cffi.requests.Response attribute), 66
 URLRequired (class in curl_cffi.requests.exceptions), 80

V

version() (curl_cffi.Curl method), 51

W

WebSocket (class in curl_cffi.requests), 68
 ws_close() (curl_cffi.Curl method), 53
 ws_connect() (curl_cffi.requests.AsyncSession method), 62
 ws_connect() (curl_cffi.requests.Session method), 59
 ws_recv() (curl_cffi.Curl method), 52
 ws_send() (curl_cffi.Curl method), 52