wayback Documentation

Release 0.5.0

Contributors

May 22, 2026

CONTENTS

1 Installation **3**

2 Usage **5**
- 2.1 Tutorial . . . 5
- 2.2 API Documentation . . . 6

3 Release History **19**
- 3.1 v0.5.0 (2026-05-22) . . . 19
- 3.2 v0.4.5 (2024-02-01) . . . 21
- 3.3 v0.4.4 (2023-11-27) . . . 21
- 3.4 v0.4.3 (2023-09-26) . . . 21
- 3.5 v0.4.3a1 (2023-09-22) . . . 22
- 3.6 v0.4.2 (2023-05-29) . . . 22
- 3.7 v0.4.1 (2023-03-07) . . . 22
- 3.8 v0.4.0 (2022-11-10) . . . 23
- 3.9 v0.3.3 (2022-09-30) . . . 24
- 3.10 v0.3.2 (2021-11-16) . . . 24
- 3.11 v0.3.1 (2021-10-14) . . . 24
- 3.12 v0.3.0 (2021-03-19) . . . 24
- 3.13 v0.2.6 (2021-03-18) . . . 27
- 3.14 v0.3.0 Beta 1 (2021-03-15) . . . 27
- 3.15 v0.3.0 Alpha 3 (2020-11-05) . . . 27
- 3.16 v0.3.0 Alpha 2 (2020-11-04) . . . 28
- 3.17 v0.3.0 Alpha 1 (2020-10-20) . . . 28
- 3.18 v0.2.5 (2020-10-19) . . . 30
- 3.19 v0.2.4 (2020-09-07) . . . 30
- 3.20 v0.2.3 (2020-03-25) . . . 31
- 3.21 v0.2.2 (2020-02-13) . . . 31
- 3.22 v0.2.1 (2019-12-01) . . . 31
- 3.23 v0.2.0 (2019-11-26) . . . 31
- 3.24 v0.1 . . . 31

Index **33**

Wayback is A Python API to the Internet Archive's Wayback Machine. It gives you tools to search for and load mementos (historical copies of web pages).

The Internet Archive maintains an official "internetarchive" Python package, but it does not focus on the Wayback Machine. Instead, it is mainly concerned with the APIs and tools that manage the Internet Archive as a whole: managing items and collections. These are how e-books, audio recordings, movies, and other content in the Internet Archive are managed. It doesn't, however, provide particularly good tools for finding or loading historical captures of specific URLs (i.e. the part of the Internet Archive called the "Wayback Machine"). That's what the *wayback* package does.

CHAPTER

ONE

INSTALLATION

Wayback is meant to be used as a Python library, and is best installed via *pip* on the command line:

```
$ pip install wayback
```

CHAPTER

TWO

USAGE

Search for historical mementos (archived copies) of a URL. Download metadata about the mementos and/or the memento content itself.

2.1 Tutorial

2.1.1 What is the earliest memento of nasa.gov?

Instantiate a `WaybackClient`.

```
In [1]: from wayback import WaybackClient

In [2]: client = WaybackClient()
```

Search for all Wayback's records for nasa.gov.

```
In [3]: results = client.search('nasa.gov')
```

This statement should execute fairly quickly because it doesn't actually do much work. The object we get back, `results`, is a *generator*, a "lazy" object from which we can pull results, one at a time. As we pull items out of it, it loads them as needed from the Wayback Machine in chronological order. We can see that `results` by itself is not informative:

```
In [4]: results
Out[4]: <generator object WaybackClient.search at 0x5bf62b6ce200>
```

There are couple ways to pull items out of generator like `results`. One simple way is to use the built-in Python function `next()`, like so:

```
In [5]: record = next(results)
```

This takes a moment to run because, now that we've asked to see the first item in the generator, this lazy object goes to fetch a chunk of results from the Wayback Machine. Looking at the record in detail,

```
In [6]: record
Out[6]: CdxRecord(urlkey='gov,nasa)/', timestamp=datetime.datetime(1996, 12, 31, 23, 58,␣
↪47, tzinfo=datetime.timezone.utc), original='http://www.nasa.gov/', mimetype='text/html
↪', statuscode=200, digest='MGIGF4GRGGF5GKV6VNCBAXOE3OR5BTZC', length=1811)
```

we can find our answer: Wayback's first memento of nasa.gov was in 1996. We can use dot access on `record` to access the timestamp specifically.

```
In [7]: record.timestamp
Out[7]: datetime.datetime(1996, 12, 31, 23, 58, 47, tzinfo=datetime.timezone.utc)
```

2.1.2 How many times does the word 'mars' appear on nasa.gov?

Above, we access the metadata for the oldest memento on nasa.gov, stored in the variable `record`. Starting from where we left off, we'll access the *content* of the memento and do a very simple analysis.

The Wayback Machine provides multiple *playback modes* to view the data it has captured. The *wayback.Mode.view* mode is a copy edited for human viewers on the web, and the *wayback.Mode.original* mode is the original copy of what was captured when the page was scraped. For analysis purposes, we generally want `original`. (Check the documentation of *wayback.Mode* for a few other, less commonly used modes.)

Let's download the original content using `WaybackClient`. (You could download the content directly with an HTTP library like `requests`, but `WaybackClient` adds extra tools for dealing with Wayback Machine servers.)

```
In [8]: from wayback import Mode

# `Mode.original` is the default and doesn't need to be explicitly set;
# we've set it here to show how you might choose other modes.
In [9]: response = client.get_memento(record, mode=Mode.original)

In [10]: content = response.content.decode()
```

We can use the built-in method `count` on strings to count the number of times that `'mars'` appears in the content.

```
In [11]: content.count('mars')
Out[11]: 30
```

This is case-sensitive, so to be more accurate we should convert the content to lowercase first.

```
In [12]: content.lower().count('mars')
Out[12]: 39
```

We picked up a couple additional occurrences that the original count missed.

2.2 API Documentation

The Wayback Machine exposes its data through two different mechanisms, implementing two different standards for archival data, the CDX API and the Memento API. We implement a Python client that can speak both.

class wayback.**WaybackClient**(*session=None*)

A client for retrieving data from the Internet Archive's Wayback Machine.

You can use a WaybackClient as a context manager. When exiting, it will close the session it's using (if you've passed in a custom session, make sure not to use the context manager functionality unless you want to live dangerously).

Parameters

session
[*WaybackSession*, optional]

search(*url*, ** (Keyword-only parameters separator (PEP 3102))*, *match_type=None*, *limit=1000*, *offset=None*, *fast_latest=None*, *from_date=None*, *to_date=None*, *filter_field=None*, *collapse=None*, *resolve_revisits=True*, *skip_malformed_results=True*, *matchType=None*, *fastLatest=None*, *resolveRevisits=None*)

Search archive.org's CDX API for all captures of a given URL. This returns an iterator of *CdxRecord* objects. The *StopIteration* value is the total count of found captures.

Results include captures with similar, but not exactly matching URLs. They are matched by a SURT-formatted, canonicalized URL that:

- Does not differentiate between HTTP and HTTPS,
- Is not case-sensitive, and
- Treats `www.` and `www*.` subdomains the same as no subdomain at all.

This will automatically page through all results for a given search. If you want fewer results, you can stop iterating early:

```
from itertools import islice
first10 = list(islice(client.search(...), 10))
```

Parameters

url
[str] The URL to search for captures of.

Special patterns in `url` imply a value for the `match_type` parameter and match multiple URLs:

- If the URL starts with *. (e.g. `*.epa.gov`) OR `match_type='domain'`, the search will include all URLs at the given domain and its subdomains.
- If the URL ends with /* (e.g. `https://epa.gov/*`) OR `match_type='prefix'`, the search will include all URLs that start with the text up to the *.
- Otherwise, this returns matches just for the requeted URL.

match_type
[str, optional] Determines how to interpret the `url` parameter. It must be one of the following:

- `exact` (default) returns results matching the requested URL (see notes about SURT above; this is not an exact string match of the URL you pass in).
- `prefix` returns results that start with the requested URL.
- `host` returns results from all URLs at the host in the requested URL.
- `domain` returns results from all URLs at the domain or any subdomain of the requested URL.

The default value is calculated based on the format of `url`.

limit
[int, default: 1000] Maximum number of results per request to the API (not the maximum number of results this function yields).

Negative values return the most recent N results.

Positive values are complicated! The search server will only scan so much data on each query, and if it finds fewer than `limit` results before hitting its own internal limits, it will behave as if if there are no more results, even though there may be.

Unfortunately, ideal values for `limit` aren't very predicatable because the search server combines data from different sources, and they do not all behave the same. Their parameters may also be changed over time.

In general…

- The default value should work well in typical cases.
- For frequently captured URLs, you may want to set a higher value (e.g. 12,000) for more efficient querying.
- For infrequently captured URLs, you may want to set a lower value (e.g. 100 or even 10) to ensure that your query does not hit internal limits before returning.
- For extremely infrequently captured URLs, you may simply want to call `search()` multiple times with different, close together `from_date` and `to_date` values.

offset
[`int`, optional] Skip the first N results.

fast_latest
[bool, optional] Get faster results when using a negative value for `limit`. It may return a variable number of results that doesn't match the value of `limit`. For example, `search('http://epa.gov', limit=-10, fast_latest=True)` may return any number of results between 1 and 10.

from_date
[`datetime` or `date`, optional] Only include captures after this date. Equivalent to the *from* argument in the CDX API. If it does not have a time zone, it is assumed to be in UTC.

to_date
[`datetime` or `date`, optional] Only include captures before this date. Equivalent to the *to* argument in the CDX API. If it does not have a time zone, it is assumed to be in UTC.

filter_field
[`str` or `list of str` or `tuple of str`, optional] A filter or list of filters for any field in the results. Equivalent to the `filter` argument in the CDX HTTP API. Format: `[!]field:regex` or `~[!]field:substring`, e.g. `'!statuscode:200'` to select only captures with a non-2xx status code, or `'~urlkey:feature'` to select only captures where the SURT-formatted URL key has "feature" somewhere in it.

To apply multiple filters, use a list or tuple of strings instead of a single string, e.g. `['statuscode:200', 'urlkey:.*feature.*']`.

Regexes are matched against the *entire* field value. For example, `'statuscode:2'` will never match, because all `statuscode` values are three characters. Instead, to match all status codes with a "2" in them, use `'statuscode:.*2.*'`. Add a `!` at before the field name to negate the match.

Valid field names are: `urlkey`, `timestamp`, `original`, `mimetype`, `statuscode`, `digest`, `length`.

collapse
[`str`, optional] Collapse consecutive results that match on a given field. (format: *fieldname* or *fieldname:N* – N is the number of chars to match.)

resolve_revisits
[bool, default: `True`] Attempt to resolve `warc/revisit` records to their actual content type and response code. Not supported on all CDX servers.

skip_malformed_results
[bool, default: `True`] If true, don't yield records that look like they have no actual memento associated with them. Some crawlers will erroneously attempt to capture bad URLs like `http://mailto:someone@domain.com` or `http://...` and so on. This is a filter performed client side and is not a CDX API argument.

Yields

version: *CdxRecord*
A *CdxRecord* encapsulating one capture or revisit

Raises

UnexpectedResponseFormat
If the CDX response was not parseable.

Notes

Several CDX API parameters are not relevant or handled automatically by this function. This does not support: *output*, *fl*, *showDupeCount*, *showSkipCount*, *lastSkipTimestamp*, *showNumPages*, *showPagedIndex*.

It also does not support *page* and *pageSize* for pagination because they work differently from the *resumeKey* method this uses, and results do not include recent captures when using them.

References

- HTTP API Docs: https://github.com/internetarchive/wayback/tree/master/wayback-cdx-server
- SURT formatting: http://crawler.archive.org/articles/user_manual/glossary.html#surt
- SURT implementation: https://github.com/internetarchive/surt

get_memento(*url*, *timestamp=None*, *mode=Mode.original*, ***, *exact=True*, *exact_redirects=None*, *target_window=datetime.timedelta(days=1)*, *follow_redirects=True*, *datetime=None*)

Fetch a memento (an archived HTTP response) from the Wayback Machine.

Not all mementos can be successfully fetched (or "played back" in Wayback terms). In this case, `get_memento` can load the next-closest-in-time memento or it will raise *wayback.exceptions.MementoPlaybackError* depending on the value of the `exact` and `exact_redirects` parameters (see more details below).

Parameters

url
[`str` or *CdxRecord*] URL to retrieve a memento of. This can be any of:

- A normal URL (e.g. `http://www.noaa.gov/`). When using this form, you must also specify `timestamp`.
- A `CdxRecord` retrieved from *wayback.WaybackClient.search()*.
- A URL of the memento in Wayback, e.g. `https://web.archive.org/web/20180816111911id_/http://www.noaa.gov/`

timestamp
[`datetime.datetime` or `datetime.date` or `str`, optional] The time at which to retrieve a memento of `url`. If `url` is a *wayback.CdxRecord* or full memento URL, this parameter can be omitted.

mode
[*wayback.Mode* or `str`, default: *wayback.Mode.original*] The playback mode of the memento. This determines whether the content of the returned memento is exactly as originally captured (the default) or modified in some way. See *wayback.Mode* for a description of possible values.

For more details, see: https://archive-access.sourceforge.net/projects/wayback/administrator_manual.html#Archival_URL_Replay_Mode

exact
[bool, default: True] If false and the requested memento either doesn't exist or can't be played back, this returns the closest-in-time memento to the requested one, so long as it is within `target_window`. If there was no memento in the target window or if `exact=True`, then this will raise *wayback.exceptions.MementoPlaybackError*.

exact_redirects
[bool, optional] If false and the requested memento is a redirect whose *target* doesn't exist or can't be played back, this returns the closest-in-time memento to the intended target, so long as it is within `target_window`. If unset, this will be the same as `exact`.

target_window
[`int` or `datetime.timedelta`, default: 86400] If the memento is of a redirect, allow up to this amount of time (in seconds if an integer) between the capture of the redirect and the capture of the redirect's target URL. This window also applies to the first memento if `exact=False` and the originally requested memento was not available. Defaults to 86,400 (24 hours).

follow_redirects
[bool, default: True] If true (the default), `get_memento` will follow historical redirects to return the content that a web browser would have ultimately displayed at the requested URL and time, rather than the memento of an HTTP redirect response (i.e. a 3xx status code). That is, if `http://example.com/a` redirected to `http://example.com/b`, then this method returns the memento for `/a` when `follow_redirects=False` and the memento for `/b` when `follow_redirects=True`.

Returns

Memento
A *Memento* object with information about the archived HTTP response.

class wayback.**CdxRecord**(*urlkey: str*, *timestamp: datetime*, *original: str*, *mimetype: str*, *statuscode: int | None*, *digest: str*, *length: int | None*)

Represents an entry from Wayback's "CDX" index of mementos (archived HTTP responses). These entries contain some metadata about the memento. You can also pass a `CdxRecord` to *WaybackClient.get_memento()* to retrieve the corresponding memento.

In general, you should not create new instances of `CdxRecord` yourself, but should get them by calling *WaybackClient.search()*.

Attributes

urlkey: `str`
SURT-formatted URL.

timestamp: `datetime`
The capture time represented as a `datetime.datetime`, such as `datetime.datetime(1996, 12, 31, 23, 58, 47, tzinfo=timezone.utc)`.

original: `str`
The URL that was captured by this record, such as `'http://www.nasa.gov/'`.

mimetype: `str`
MIME type of record, such as `'text/html'`, `'warc/revisit'` or `'unk'` ("unknown") if this information was not captured.

statuscode: `int | None`
Status code returned by the server when the record was captured, such as `200`. This is may be `None` if the record is a revisit record.

digest: str

The base 32-encoded SHA-1 hash of the archived HTTP response body. This can be useful for comparing to other `CdxRecord` instances or avoiding duplicate requests for mementos.

Please keep in mind that this digest is generally computed based on the response body *as stored on disk* (usually the exact bytes originally received when saving the response), so is not useful for validation or fixity checks against a memento loaded with *WaybackClient.get_memento()*. For example, if the response body was stored in brotli-compressed form but *transferred to you* in gzip-compressed form, your bytes will not match this digest.

For revisit records, this is the digest of the originally received HTTP response body as it *would have been stored*, so you can use it to match a non-revisit record containing the same response body. (But keep in mind this is just the body. It does not include HTTP headers, which may have been different for two records with the same digest.)

length: int | None

Size (in bytes) of the archived data *as stored on disk*. Like *digest*, this usually will not be useful for external users, since it does not reflect the actual archived HTTP response body size. For example, revisit records will generally be small because the archived data on disk is just a pointer to a different record that was saved previously.

raw_url: str

The URL to the raw captured content, such as `'https://web.archive.org/web/19961231235847id_/http://www.nasa.gov/'`.

view_url: str

The URL to the public view on Wayback Machine. In this view, the links and some subresources in the document are rewritten to point to Wayback URLs. There is also a navigation panel around the content. Example URL: `'https://web.archive.org/web/19961231235847/http://www.nasa.gov/'`.

key: str

Deprecated since version 0.5.0: This attribute was renamed to *urlkey*. This name will be removed in a future release.

url: str

Deprecated since version 0.5.0: This attribute was renamed to *original*. This name will be removed in a future release.

mime_type: str

Deprecated since version 0.5.0: This attribute was renamed to *mimetype*. This name will be removed in a future release.

status_code: str

Deprecated since version 0.5.0: This attribute was renamed to *statuscode*. This name will be removed in a future release.

class wayback.**Memento**(*, *url*, *timestamp*, *mode*, *memento_url*, *status_code*, *headers*, *encoding*, *raw*, *raw_headers*, *links*, *history*, *debug_history*)

Represents a memento (an archived HTTP response). This object is similar to a response object from the popular "Requests" package, although it has some differences designed to differentiate historical information vs. current metadata about the stored memento (for example, the `headers` attribute lists the headers recorded in the memento, and does not include additional headers that provide metadata about the Wayback Machine).

Note that, like an HTTP response, this object represents a potentially open network connection to the Wayback Machine. Reading the `content` or `text` attributes will read all the data being received and close the connection automatically, but if you do not read those properties, you must make sure to call `close()` to close to connection. Alternatively, you can use a Memento as a context manager. The connection will be closed for you when the context ends:

```
>>> with a_memento:
>>>     do_something()
>>> # Connection is automatically closed here.
```

Fields

encoding: str

The text encoding of the response, e.g. `'utf-8'`.

headers: dict

A dict representing the headers of the archived HTTP response. The keys are case-insensitive. If you iterate over it, you will receive the header names as they were originally sent. However, you can look them up via strings that vary in upper/lower-case. For example:

```
list(memento.headers) == ['Content-Type', 'Date']
memento.headers['Content-Type'] == memento.headers['content-type']
```

history: tuple[*wayback.Memento*]

A list of *wayback.Memento* objects that were redirects and were followed to produce this memento.

debug_history: tuple[str]

List of all URLs redirects followed in order to produce this memento. These are "memento URLs" – that is, they are absolute URLs to the Wayback machine like `https://web.archive.org/web/20180816111911id_/http://www.noaa.gov/`, rather than URLs of captured redirects, like `http://www.noaa.gov`. Many of the URLs in this list do not represent actual mementos.

status_code: int

The HTTP status code of the archived HTTP response.

mode: str

The playback mode used to produce the Memento.

timestamp: datetime.datetime

The time the memento was originally captured. This includes `tzinfo`, and will always be in UTC.

url: str

The URL that the memento represents, e.g. `http://www.noaa.gov`.

memento_url: str

The URL at which the memento was fetched from the Wayback Machine, e.g. `https://web.archive.org/web/20180816111911id_/http://www.noaa.gov/`.

ok: bool

Whether the response had an non-error status (i.e. < 400).

is_redirect: bool

Whether the response was a redirect (i.e. had a 3xx status).

content: bytes

The body of the archived HTTP response in bytes.

text: str

The body of the archived HTTP response decoded as a string.

links: dict of (str, dict of (str, str))

Related links to this Memento (e.g. the previous and/or next Memento in time). The keys are the relationship (e.g. `'prev memento'`) as a string and the values are dicts where the keys and values are strings.

In each entry, the `'url'` key is the URL of the related link, the `'rel'` key is the relationship (the same as the key in the top-level dict), and the rest of the keys will be any other attributes that are relevant for that link (e.g. `'datetime'` or `'type'`).

For example:

```
{
    'original': {
        'url': 'https://www.fws.gov/birds/',
        'rel': 'original'
    },
    'first memento': {
        'url': 'https://web.archive.org/web/20050323155300/http://www.fws.
↪gov:80/birds',
        'rel': 'first memento',
        'datetime': 'Wed, 23 Mar 2005 15:53:00 GMT'
    },
    'prev memento': {
        'url': 'https://web.archive.org/web/20210125125216/https://www.fws.gov/
↪birds/',
        'rel': 'prev memento',
        'datetime': 'Mon, 25 Jan 2021 12:52:16 GMT'
    },
    'next memento': {
        'url': 'https://web.archive.org/web/20210321180831/https://www.fws.gov/
↪birds',
        'rel': 'next memento',
        'datetime': 'Sun, 21 Mar 2021 18:08:31 GMT'
    },
    'last memento': {
        'url': 'https://web.archive.org/web/20221006031005/https://fws.gov/birds
↪',
        'rel': 'last memento',
        'datetime': 'Thu, 06 Oct 2022 03:10:05 GMT'
    }
}
```

Links to other mementos use the same mode as the memento object this `links` attribute belongs to. For example:

```
raw_memento = client.get_memento('https://fws.gov/birds', '20210318004901')
raw_memento.links['next memento']['url'] == 'https://web.archive.org/web/
↪20210321180831id_/https://fws.gov/birds'
# The "id_" after the timestamp means "original" mode ---------------------------
↪-------^^^

view_memento = client.get_memento('https://fws.gov/birds', '20210318004901',␣
↪mode=Mode.view)
view_memento.links['next memento']['url'] == 'https://web.archive.org/web/
↪20210321180831/https://fws.gov/birds'
```

(continues on next page)

(continued from previous page)

```
# Nothing after the timestamp for "view" mode ----------------------------------
↪-------^
```

`close()`

Close the HTTP response for this Memento. This happens automatically if you read `content` or `text`, and if you use the memento as a context manager. This method is always safe to call – it does nothing if the response has already been closed.

`classmethod parse_memento_headers(raw_headers, url='https://web.archive.org/')`

Extract historical headers from the Memento HTTP response's headers.

Parameters

raw_headers
[`dict`] A dict of HTTP headers from the Memento's HTTP response.

url
[`str`, optional] The URL of the resource the headers are being parsed for. It's used when header data contains relative/incomplete URL information.

Returns

`dict`

`class wayback.WaybackSession(retries=6, backoff=2, timeout=60, user_agent=None, search_calls_per_second=<wayback._utils.RateLimit object>, memento_calls_per_second=<wayback._utils.RateLimit object>, timemap_calls_per_second=<wayback._utils.RateLimit object>)`

A custom session object that pools network connections and resources for requests to the Wayback Machine.

Parameters

retries
[`int`, default: 6] The maximum number of retries for requests.

backoff
[`int` or `float`, default: 2] Number of seconds from which to calculate how long to back off and wait when retrying requests. The first retry is always immediate, but subsequent retries increase by powers of 2:

seconds = backoff * 2 ^ (retry number - 1)

So if this was *4*, retries would happen after the following delays: 0 seconds, 4 seconds, 8 seconds, 16 seconds, …

timeout
[`int` or `float` or `tuple of` (`int` or `float`, `int` or `float`), default: 60] A timeout to use for all requests. See the Requests docs for more: https://docs.python-requests.org/en/master/user/advanced/#timeouts

user_agent
[`str`, optional] A custom user-agent string to use in all requests. Defaults to: *wayback/{version} (+https://github.com/edgi-govdata-archiving/wayback)*

search_calls_per_second
[*`wayback.RateLimit`* or `int` or `float`, default: 0.8] The maximum number of calls per second made to the CDX search API. To disable the rate limit, set this to 0.

To have multiple sessions share a rate limit (so requests made by one session count towards the limit of the other session), use a single *`wayback.RateLimit`* instance and pass it to

each `WaybackSession` instance. If you do not set a limit, the default limit is shared globally across all sessions.

memento_calls_per_second
[*wayback.RateLimit* or `int` or `float`, default: 8] The maximum number of calls per second made to the memento API. To disable the rate limit, set this to 0.

To have multiple sessions share a rate limit (so requests made by one session count towards the limit of the other session), use a single *wayback.RateLimit* instance and pass it to each `WaybackSession` instance. If you do not set a limit, the default limit is shared globally across all sessions.

timemap_calls_per_second
[*wayback.RateLimit* or `int` or `float`, default: 1.33] The maximum number of calls per second made to the timemap API (the Wayback Machine's new, beta CDX search is part of the timemap API). To disable the rate limit, set this to 0.

To have multiple sessions share a rate limit (so requests made by one session count towards the limit of the other session), use a single *wayback.RateLimit* instance and pass it to each `WaybackSession` instance. If you do not set a limit, the default limit is shared globally across all sessions.

reset()
Reset any network connections the session is using.

2.2.1 Utilities

wayback.**memento_url_data**(*memento_url*)
Get the original URL, time, and mode that a memento URL represents a capture of.

Returns

url
[`str`] The URL that the memento is a capture of.

time
[`datetime.datetime`] The time the memento was captured in the UTC timezone.

mode
[`str`] The playback mode.

Examples

Extract original URL, time and mode.

```
>>> url = ('https://web.archive.org/web/20170813195036id_/'
...        'https://arpa-e.energy.gov/?q=engage/events-workshops')
>>> memento_url_data(url)
('https://arpa-e.energy.gov/?q=engage/events-workshops',
 datetime.datetime(2017, 8, 13, 19, 50, 36, tzinfo=timezone.utc),
 'id_')
```

class wayback.**Mode**(**values*)
An enum describing the playback mode of a memento. When requesting a memento (e.g. with *wayback.WaybackClient.get_memento()*), you can use these values to determine how the response body should be formatted.

For more details, see: https://archive-access.sourceforge.net/projects/wayback/administrator_manual.html#Archival_URL_Replay_Mode

Examples

```
>>> waybackClient.get_memento('https://noaa.gov/',
>>>                           timestamp=datetime.datetime(2018, 1, 2),
>>>                           mode=wayback.Mode.view)
```

Values

original

Returns the HTTP response body as originally captured.

view

Formats the response body so it can be viewed with a web browser. URLs for links and subresources like scripts, stylesheets, images, etc. will be modified to point to the equivalent memento in the Wayback Machine so that the resulting page looks as similar as possible to how it would have appeared when originally captured. It's mainly meant for use with HTML pages. This is the playback mode you typically use when browsing the Wayback Machine with a web browser.

javascript

Formats the response body by updating URLs, similar to `Mode.view`, but designed for JavaScript instead of HTML.

css

Formats the response body by updating URLs, similar to `Mode.view`, but designed for CSS instead of HTML.

image

formats the response body similar to `Mode.view`, but designed for image files instead of HTML.

class wayback.**RateLimit**(*per_second: int | float*)

`RateLimit` is a simple locking mechanism that can be used to enforce rate limits and is safe to use across multiple threads. It can also be used as a context manager.

Calling *rate_limit_instance.wait()* blocks until a minimum time has passed since the last call. Using *with rate_limit_instance:* blocks entries to the context until a minimum time since the last context entry.

Parameters

per_second
[`int` or `float`] The maximum number of calls per second that are allowed. If 0, a call to *wait()* will never block.

Examples

Slow down a tight loop to only occur twice per second:

```
>>> limit = RateLimit(per_second=2)
>>> for x in range(10):
>>>     with limit:
>>>         print(x)
```

2.2.2 Exception Classes

class wayback.exceptions.**WaybackException**

Base exception class for all Wayback-specific errors.

class wayback.exceptions.**UnexpectedResponseFormat**

Raised when data returned by the Wayback Machine is formatted in an unexpected or unparseable way.

class wayback.exceptions.**BlockedByRobotsError**

Raised when a URL can't be queried in Wayback because it was blocked by a site's *robots.txt* file.

class wayback.exceptions.**BlockedSiteError**

Raised when a URL has been blocked from access or querying in Wayback. This is often because of a takedown request. (URLs that are blocked because of `robots.txt` get a `BlockedByRobotsError` instead.)

class wayback.exceptions.**MementoPlaybackError**

Raised when a Memento can't be 'played back' (loaded) by the Wayback Machine for some reason. This is a server-side issue, not a problem in parsing data from Wayback.

class wayback.exceptions.**NoMementoError**

Raised when there was no memento available for a given URL. This might mean the given URL has no mementos at all or that none that are available for playback.

This also means you should *not* try to request a memento of the same URL in a different timeframe. If there may be other mementos of the URL available, you'll get a different error.

class wayback.exceptions.**RateLimitError**(*response*, *retry_after*)

Raised when the Wayback Machine responds with a 429 (too many requests) status code. In general, this package's built-in limits should help you avoid ever hitting this, but if you are running multiple processes in parallel, you could go overboard.

Attributes

retry_after
[`int`, optional] Recommended number of seconds to wait before retrying. If the Wayback Machine does not include it in the HTTP response, it will be set to `None`.

class wayback.exceptions.**WaybackRetryError**(*retries*, *total_time*, *causal_error*)

Raised when a request to the Wayback Machine has been retried and failed too many times. The number of tries before this exception is raised generally depends on your *WaybackSession* settings.

Attributes

retries
[`int`] The number of retries that were attempted.

cause
[`Exception`] The actual, underlying error that would have caused a retry.

time
[`int`] The total time spent across all retried requests, in seconds.

class wayback.exceptions.**SessionClosedError**

Raised when a Wayback session is used to make a request after it has been closed and disabled.

CHAPTER

THREE

RELEASE HISTORY

3.1 v0.5.0 (2026-05-22)

The main focus of this release is continued refinement of rate limiting. You can now share or *not* share rate limits across sessions in a clearer way. Server-side rate limiting errors (if you set the rate limit too high in this package) are also now raised directly to your code instead of causing many-minute-long pauses.

There are number of other useful improvements here, too! Read on for details.

3.1.1 Breaking Changes

- Wayback now requires Python 3.8 or newer. Going forward, we intend to drop support for older Python releases once they hit end-of-life (i.e. they no longer receive any security updates or support from the Core Python team). These updates will always be noted as breaking changes.
- Wayback will no longer automatically pause and retry when the server returns rate-limit errors. Instead, it will raise a *wayback.exceptions.WaybackRetryError* exception (which includes how long you should probably pause for in the `time` attribute).

  The previous behavior helped solve some issues with the way rate limits were implemented on the client side that have since been fixed. It was also designed around users who had custom limits on Internet Archive servers, which is an unusual situation. The new approach is safer and can help prevent your IP address from getting blocked. If you need to retry after rate limit errors, make sure your code handles the exception and pauses all requests on all client instances for an appropriate amount of time.

3.1.2 Features

- Rate limiting has been significantly updated under the hood. Separate sessions or clients with customized limits now operate completely independently (their limits previously overlapped in a hard-to-predict way).

  If you need to make multiple *wayback.WaybackSession* instances that share limits, you can create an instance of *wayback.RateLimit* and pass it to each session via the `*_calls_per_second` arguments instead of passing an integer or float. For example:

```
# Each of these sessions will be limited to 1 search request every 2
# seconds. During a period where only one of these sessions is in use, it
# will continue to operate at 1 request every 2 seconds:
session1 = WaybackSession(search_calls_per_second=0.5)
session2 = WaybackSession(search_calls_per_second=0.5)

# These sessions will be limited to 1 search request each second as a
# combined total (what you usually want). During a period where only one of
# these sessions is in use, it will make 1 request per second, but if both
```

(continues on next page)

(continued from previous page)

```
# are actively in use, each will make 1 request every 2 seconds:
rate = RateLimit(per_second=1)
session1 = WaybackSession(search_calls_per_second=rate)
session2 = WaybackSession(search_calls_per_second=rate)
```

Sessions using the default limits share them via this same mechanism.

- Several attributes of *wayback.CdxRecord* have been renamed to match their names in Wayback's actual CDX API. The old names still work for now, but using them will log a deprecation warning. We plan to remove them entirely in a future release. (Issue #187)

  The renamed attributes are:

  - `key` (now `urlkey`).
  - `url` (now `original`).
  - `mime_type` (now `mimetype`).
  - `status_code` (now `statuscode`).

- You can now use a `datetime.timedelta` instance for the `target_window` argument to *wayback.WaybackClient.get_memento()* (an integer indicating a number of seconds still works, too). (Issue #193)

  For example, to get the content of `epa.gov` from anytime within 5 days of January 1, 2008:

```
client.get_memento(
  'https://epa.gov/',
  datetime(2008, 1, 1),
  exact=False,
  target_window=timedelta(days=5)
)
# <wayback.Memento url="http://www.epa.gov/" timestamp="2008-01-03T00:07:13+00:00">
```

3.1.3 Fixes & Maintenance

- The default rate limits have been further tweaked since v0.4.4 based on closer collaboration with Internet Archive staff. Rate limits are also now more accurately applied to each individual request (they were previously applied more roughly, without respect to retries and redirects).
- Updated license identifier to follow PEP 639 style. This should make sure the license is surfaced better and more clearly on PyPI. (Issue #186)
- *wayback.Memento* now has a nicer, more informative `repr` when you are using a notebook or Python REPL. (Issue #192)
- Tests where rate limits don't matter now run faster. (Issue #194)
- Source code is now autoformatted, type-checked, and more comprehensively covered by lint and style rules. (Issue #185)

3.1.4 New Contributors

A huge thanks to three new contributors in this release!

- Derzan Chiang
- Beckett Frey
- @Adeelp1

3.2 v0.4.5 (2024-02-01)

In v0.4.4, we broke *archived mementos* of rate limit errors — they started raising exceptions instead of returning the actual memento. We now correctly return mementos of rate limit errors while still raising exceptions for actual live rate limit errors from the Wayback Machine itself. (Issue #158)

3.3 v0.4.4 (2023-11-27)

This release makes some small fixes to rate limits and retries in order to better match with the current behavior of Wayback Machine servers:

- Updated the *wayback.WaybackClient.search()* rate limit to 1 call per second (it was previously 1.5 per second). (Issue #140)
- Delayed retries for 60 seconds when receiving rate limit errors from the server. (Issue #142)
- Added more logging around requests and rate limiting. This should make it easier to debug future rate limit issues. (Issue #139)
- Fixed calculation of the `time` attribute on *wayback.exceptions.WaybackRetryError*. It turns out it was only accounting for the time spent waiting between retries and skipping the time waiting for the server to respond! (Issue #142)
- Fixed some spots where we leaked HTTP connections during retries or during exception handling. (Issue #142)

The next minor release (v0.5) will almost certainly include some bigger changes to how rate limits and retries are handled.

3.4 v0.4.3 (2023-09-26)

This is mainly a compatibility release: it adds support for urllib3 v2.x and the next upcoming major release of Python, v3.12.0. It also adds support for multiple filters in *wayback.WaybackClient.search()*. There are no breaking changes.

3.4.1 Features

You can now apply multiple filters to a search by using a list or tuple for the `filter_field` parameter of *wayback.WaybackClient.search()*. (Issue #119)

For example, to search for all captures at `nasa.gov` with a 404 status and "feature" somewhere in the URL:

```
client.search('nasa.gov/',
              match_type='prefix',
              filter_field=['statuscode:404',
                            'urlkey:.*feature.*'])
```

3.4.2 Fixes & Maintenance

- Add support for Python 3.12.0. (Issue #123)
- Add support for urllib3 v2.x (urllib3 v1.20+ also still works). (Issue #116)

3.5 v0.4.3a1 (2023-09-22)

This is a test release for properly supporting the upcoming release of Python 3.12.0. Please file an issue if you encounter issues using on Python 3.12.0rc3 or later. (Issue #123)

3.6 v0.4.2 (2023-05-29)

Wayback is not compatible with urllib3 v2, and this release updates the package's requirements to make sure Pip and other package managers install compatible versions of Wayback and urllib3. There are no other fixes or new features.

3.7 v0.4.1 (2023-03-07)

3.7.1 Features

wayback.Memento now has a `links` property with information about other URLs that are related to the memento, such as the previous or next mementos in time. It's a dict where the keys identify the relationship (e.g. `'prev memento'`) and the values are dicts with additional information about the link. (Issue #57) For example:

```
{
    'original': {
        'url': 'https://www.fws.gov/birds/',
        'rel': 'original'
    },
    'first memento': {
        'url': 'https://web.archive.org/web/20050323155300id_/http://www.fws.gov:80/birds
↪',
        'rel': 'first memento',
        'datetime': 'Wed, 23 Mar 2005 15:53:00 GMT'
    },
    'prev memento': {
        'url': 'https://web.archive.org/web/20210125125216id_/https://www.fws.gov/birds/
↪',
        'rel': 'prev memento',
        'datetime': 'Mon, 25 Jan 2021 12:52:16 GMT'
    },
    'next memento': {
        'url': 'https://web.archive.org/web/20210321180831id_/https://www.fws.gov/birds',
        'rel': 'next memento',
        'datetime': 'Sun, 21 Mar 2021 18:08:31 GMT'
    },
    'last memento': {
        'url': 'https://web.archive.org/web/20221006031005id_/https://fws.gov/birds',
        'rel': 'last memento',
        'datetime': 'Thu, 06 Oct 2022 03:10:05 GMT'
    }
}
```

One use for these is to iterate through additional mementos. For example, to get the previous memento:

```
client.get_memento(memento.links['prev memento']['url'])
```

3.7.2 Fixes & Maintenance

- Fix an issue where the `Memento.url` attribute might be slightly off from the exact URL that was captured (it could have a different protocol, different upper/lower-casing, etc.). (Issue #99)
- Fix an error when getting a memento for a redirect in `view` mode. If you called *wayback.WaybackClient.get_memento()* with a URL that turned out to be a redirect at the given time and set the `mode` option to *wayback.Mode.view*, you'd get an exception saying "Memento at {url} could not be played." Now this works just fine. (Issue #109)

3.8 v0.4.0 (2022-11-10)

3.8.1 Breaking Changes

This release includes a significant overhaul of parameters for *wayback.WaybackClient.search()*.

- Removed parameters that did nothing, could break search, or that were for internal use only: `gzip`, `showResumeKey`, `resumeKey`, `page`, `pageSize`, `previous_result`.
- Removed support for extra, arbitrary keyword parameters that could be added to each request to the search API.
- All parameters now use snake_case. (Previously, parameters that were passed unchanged to the HTTP API used camelCase, while others used snake_case.) The old, non-snake-case names are deprecated, but still work. They'll be completely removed in v0.5.0.
  - `matchType` → `match_type`
  - `fastLatest` → `fast_latest`
  - `resolveRevisits` → `resolve_revisits`
- The `limit` parameter now has a default value. There are very few cases where you should not set a `limit` (not doing so will typically break pagination), and there is now a default value to help prevent mistakes. We've also added documentation to explain how and when to adjust this value, since it is pretty complex. (Issue #65)
- Expanded the method documentation to explain things in more depth and link to more external references.

While we were at it, we also renamed the `datetime` parameter of *wayback.WaybackClient.get_memento()* to `timestamp` for consistency with *wayback.CdxRecord* and *wayback.Memento*. The old name still works for now, but it will be fully removed in v0.5.0.

3.8.2 Features

- *wayback.Memento.headers* is now case-insensitive. The keys of the `headers` dict are returned with their original case when iterating, but lookups are performed case-insensitively. For example:

```
list(memento.headers) == ['Content-Type', 'Date']
memento.headers['Content-Type'] == memento.headers['content-type']
```

  (Issue #98)

- There are now built-in rate limits for calls to `search()` and `get_memento()`. The default values should keep you from getting temporarily blocked by the Wayback Machine servers, but you can also adjust them when instantiating *wayback.WaybackSession*:

```
# Limit get_memento() calls to 2 per second (or one every 0.5 seconds):
client = WaybackClient(WaybackSession(memento_calls_per_second=2))

# These now take a minimum of 0.5 seconds, even if the Wayback Machine
```

(continues on next page)

(continued from previous page)

```
# responds instantly (there's no delay on the first call):
client.get_memento('http://www.noaa.gov/', timestamp='20180816111911')
client.get_memento('http://www.noaa.gov/', timestamp='20180829092926')
```

A huge thanks to @LionSzl for implementing this. (Issue #12)

3.8.3 Fixes & Maintenance

- All API requests to archive.org now use HTTPS instead of HTTP. Thanks to @sundhaug92 for calling this out. (Issue #81)
- Headers from the original archived response are again included in *wayback.Memento.headers*. As part of this, the `headers` attribute is now case-insensitive (see new features above), since the Internet Archive servers now return headers with different cases depending on how the request was made. (Issue #98)

3.9 v0.3.3 (2022-09-30)

This release extends the timestamp parsing fix from version 0.3.2 to handle a similar problem, but with the month portion of timestamps in addition to the day. It also implements a small performance improvement in timestamp parsing. Thanks to @edsu for discovering this issue and addressing this. (Issue #88)

3.10 v0.3.2 (2021-11-16)

Some Wayback CDX records have invalid timestamps with `"00"` for the day-of-month portion. *wayback.WaybackClient.search()* previously raised an exception when parsing CDX records with this issue, but now handles them safely. Thanks to @8W9aG for discovering this issue and addressing it. (Issue #85)

3.11 v0.3.1 (2021-10-14)

Some Wayback CDX records have no `length` information, and previously caused *wayback.WaybackClient.search()* to raise an exception. These records will have their `length` property set to `None` instead of a number. Thanks to @8W9aG for discovering this issue and addressing it. (Issue #83)

3.12 v0.3.0 (2021-03-19)

This release marks a *major* update we're really excited about: *wayback.WaybackClient.get_memento()* no longer returns a `Response` object from the Requests package that takes a lot of extra work to interpret correctly. Instead, it returns a new *wayback.Memento* object. It's really similar to the `Response` we used to return, but doesn't mix up current and historical data — it represents the historical, archived HTTP response that is stored in the Wayback Machine. This is a big change to the API, so we've bumped the version number to `0.3.x`.

3.12.1 Notable Changes

- **Breaking change:** *wayback.WaybackClient.get_memento()* takes new parameters and has a new return type. More details below.
- **Breaking change:** *wayback.memento_url_data()* now returns 3 values instead of 2. The last value is a string representing the playback mode (see below description of the new `mode` parameter on *wayback.WaybackClient.get_memento()* for more about playback modes).
- Requests to the Wayback Machine now have a default timeout of 60 seconds. This was important because we've seen many recent issues where the Wayback Machine servers don't always close connections.

If needed, you can disable this by explicitly setting `timeout=None` when creating a *wayback.WaybackSession*. Please note this is *not* a timeout on how long a whole request takes, but on the time between bytes received.

- *wayback.WaybackClient.get_memento()* now raises *wayback.exceptions.NoMementoError* when the requested URL has never been archived by the WaybackMachine. It no longer raises `requests.exceptions.HTTPError` under any circumstances.

You may notice that removing APIs from the Requests package is a theme here. Under the hood, *Wayback* still uses *Requests* for HTTP requests, but we expect to change that in order to ensure this package is thread-safe. We will bump the version to v0.4.x when doing so.

3.12.2 get_memento() Parameters

The parameters in *wayback.WaybackClient.get_memento()* have been re-organized. The method signature is now:

```
def get_memento(self,
                url,                          # Accepts new types of values.
                datetime=None,                # New parameter.
                mode=Mode.original,           # New parameter.
                *,                            # Everything below is keyword-only.
                exact=True,
                exact_redirects=None,
                target_window=24 * 60 * 60,
                follow_redirects=True)        # New parameter.
```

- All parameters except `url` (the first parameter) from v0.2.x must now be specified with keywords, and cannot be specified positionally.

  If you previously used keywords, your code will be fine and no changes are necessary:

  ```
  # This still works great!
  client.get_memento('http://web.archive.org/web/20180816111911id_/http://www.noaa.
  ↪gov/',
                     exact=False,
                     exact_redirects=False,
                     target_window=3600)
  ```

  However, positional parameters like the following will now cause problems, and you should switch to the above keyword form:

  ```
  # This will now cause you some trouble :(
  client.get_memento('http://web.archive.org/web/20180816111911id_/http://www.noaa.
  ↪gov/',
                     False,
                     False,
                     3600)
  ```

- The `url` parameter can now be a normal, non-Wayback URL or a *wayback.CdxRecord*, and new `datetime` and `mode` parameters have been added.

  Previously, if you wanted to get a memento of what `http://www.noaa.gov/` looked like on August 1, 2018, you would have had to construct a complex string to pass to `get_memento()`:

  ```
  client.get_memento('http://web.archive.org/web/20180801000000id_/http://www.noaa.
  ↪gov/')
  ```

  Now you can pass the URL and time you want as separate parameters:

```
client.get_memento('http://www.noaa.gov/', datetime.datetime(2018, 8, 1))
```

If the `datetime` parameter does not specify a timezone, it will be treated as UTC (*not* local time).

You can also pass a *wayback.CdxRecord* that you received from *wayback.WaybackClient.search()* instead of a URL and time:

```
for record in client.search('http://www.noaa.gov/'):
    client.get_memento(record)
```

Finally, you can now specify the *playback mode* of a memento using the `mode` parameter:

```
client.get_memento('http://www.noaa.gov/',
                   datetime=datetime.datetime(2018, 8, 1),
                   mode=wayback.Mode.view)
```

The default mode is *wayback.Mode.original*, which returns the exact HTTP response body as was originally archived. Other modes reformat the response body so it's more friendly for browsing by changing the URLs of links, images, etc. and by adding informational content to the page about the memento you are viewing. They are the modes typically used when you view the Wayback Machine in a web browser.

Don't worry, though — complete Wayback URLs are still supported. This code still works fine:

```
client.get_memento('http://web.archive.org/web/20180801000000id_/http://www.noaa.
↪gov/')
```

- A new `follow_redirects` parameter specifies whether to follow *historical* redirects (i.e. redirects that happened when the requested memento was captured). It defaults to `True`, which matches the old behavior of this method.

3.12.3 get_memento() Returns a Memento Object

`get_memento()` no longer returns a response object from the Requests package. Instead it returns a specialized *wayback.Memento* object, which is similar, but provides more useful information about the Memento than just the HTTP response from Wayback. For example, `memento.url` is the original URL the memento is a capture of (e.g. `http://www.noaa.gov/`) rather than the Wayback URL (e.g. `http://web.archive.org/web/20180816111911id_/http://www.noaa.gov/`). You can still get the full Wayback URL from `memento.memento_url`.

You can check out the full API documentation for *wayback.Memento*, but here's a quick guide to what's available:

```
memento = client.get_memento('http://www.noaa.gov/home',
                             datetime(2018, 8, 16, 11, 19, 11),
                             exact=False)

# These values were previously not available except by parsing
# `memento.url`. The old `memento.url` is now `memento.memento_url`.
memento.url == 'http://www.noaa.gov/'
memento.timestamp == datetime(2018, 8, 29, 8, 8, 49, tzinfo=timezone.utc)
memento.mode == 'id_'

# Used to be `memento.url`:
memento.memento_url == 'http://web.archive.org/web/20180816111911id_/http://www.noaa.gov/
↪'
```

(continues on next page)

(continued from previous page)

```
# Used to be a list of `Response` objects, now a *tuple* of Mementos. It
# lists only the redirects that are actual Mementos and not part of
# Wayback's internal machinery:
memento.history == (Memento<url='http://noaa.gov/home'>,)

# Used to be a list of `Response` objects, now a *tuple* of URL strings:
memento.debug_history == ('http://web.archive.org/web/20180816111911id_/http://noaa.gov/
↪home',
                          'http://web.archive.org/web/20180829092926id_/http://noaa.gov/
↪home',
                          'http://web.archive.org/web/20180829092926id_/http://noaa.gov/
↪')

# Headers now only lists headers from the original archived response, not
# additional headers from the Wayback Machine itself. (If there's
# important information you needed in the headers, file an issue and let
# us know! We'd like to surface that kind of information as attributes on
# the Memento now.
memento.headers = {'header_name': 'header_value',
                   'another_header': 'another_value',
                   'and': 'so on'}

# Same as before:
memento.status_code
memento.ok
memento.is_redirect
memento.encoding
memento.content
memento.text
```

3.13 v0.2.6 (2021-03-18)

Fix a major bug where a session's `timeout` would not actually be applied to most requests. HUGE thanks to @LionSzl for discovering this issue and addressing it. (Issue #68)

3.14 v0.3.0 Beta 1 (2021-03-15)

wayback.WaybackClient.get_memento() now raises *wayback.exceptions.NoMementoError* when the requested URL has never been archived. It also now raises *wayback.exceptions.MementoPlaybackError* in all other cases where an error was returned by the Wayback Machine (so you should never see a `requests.exceptions.HTTPError`). However, you may still see other *network-level* errors (e.g. `ConnectionError`).

3.15 v0.3.0 Alpha 3 (2020-11-05)

Fixes a bug in the new *wayback.Memento* type where header parsing would fail for mementos with schemeless `Location` headers. (Issue #61)

3.16 v0.3.0 Alpha 2 (2020-11-04)

Fixes a bug in the new *wayback.Memento* type where header parsing would fail for mementos with path-based `Location` headers. (Issue #60)

3.17 v0.3.0 Alpha 1 (2020-10-20)

Breaking Changes:

This release focuses on *wayback.WaybackClient.get_memento()* and makes major, breaking changes to its parameters and return type. They're all improvements, though, we promise!

get_memento() Parameters

The parameters in *wayback.WaybackClient.get_memento()* have been re-organized. The method signature is now:

```
def get_memento(self,
                url,                          # Accepts new types of values.
                datetime=None,                # New parameter.
                mode=Mode.original,           # New parameter.
                *,                            # Everything below is keyword-only.
                exact=True,
                exact_redirects=None,
                target_window=24 * 60 * 60,
                follow_redirects=True)        # New parameter.
```

- All parameters except `url` (the first parameter) from v0.2.x must now be specified with keywords, and cannot be specified positionally.

  If you previously used keywords, your code will be fine and no changes are necessary:

  ```
  # This still works great!
  client.get_memento('http://web.archive.org/web/20180816111911id_/http://www.noaa.
  ↪gov/',
                     exact=False,
                     exact_redirects=False,
                     target_window=3600)
  ```

  However, positional parameters like the following will now cause problems, and you should switch to the above keyword form:

  ```
  # This will now cause you some trouble :(
  client.get_memento('http://web.archive.org/web/20180816111911id_/http://www.noaa.
  ↪gov/',
                     False,
                     False,
                     3600)
  ```

- The `url` parameter can now be a normal, non-Wayback URL or a *wayback.CdxRecord*, and new `datetime` and `mode` parameters have been added.

  Previously, if you wanted to get a memento of what `http://www.noaa.gov/` looked like on August 1, 2018, you would have had to construct a complex string to pass to `get_memento()`:

  ```
  client.get_memento('http://web.archive.org/web/20180801000000id_/http://www.noaa.
  ↪gov/')
  ```

Now you can pass the URL and time you want as separate parameters:

```
client.get_memento('http://www.noaa.gov/', datetime.datetime(2018, 8, 1))
```

If the `datetime` parameter does not specify a timezone, it will be treated as UTC (*not* local time).

You can also pass a *wayback.CdxRecord* that you received from *wayback.WaybackClient.search()* instead of a URL and time:

```
for record in client.search('http://www.noaa.gov/'):
    client.get_memento(record)
```

Finally, you can now specify the *playback mode* of a memento using the `mode` parameter:

```
client.get_memento('http://www.noaa.gov/',
                   datetime=datetime.datetime(2018, 8, 1),
                   mode=wayback.Mode.view)
```

The default mode is *wayback.Mode.original*, which returns the exact HTTP response body as was originally archived. Other modes reformat the response body so it's more friendly for browsing by changing the URLs of links, images, etc. and by adding informational content to the page about the memento you are viewing. They are the modes typically used when you view the Wayback Machine in a web browser.

Don't worry, though — complete Wayback URLs are still supported. This code still works fine:

```
client.get_memento('http://web.archive.org/web/20180801000000id_/http://www.noaa.
↪gov/')
```

- A new `follow_redirects` parameter specifies whether to follow *historical* redirects (i.e. redirects that happened when the requested memento was captured). It defaults to `True`, which matches the old behavior of this method.

get_memento() Returns a Memento Object

`get_memento()` no longer returns a response object from the Requests package. Instead it returns a specialized *wayback.Memento* object, which is similar, but provides more useful information about the Memento than just the HTTP response from Wayback. For example, `memento.url` is the original URL the memento is a capture of (e.g. `http://www.noaa.gov/`) rather than the Wayback URL (e.g. `http://web.archive.org/web/20180816111911id_/http://www.noaa.gov/`). You can still get the full Wayback URL from `memento.memento_url`.

You can check out the full API docs for *wayback.Memento*, but here's a quick guide to what's available:

```
memento = client.get_memento('http://www.noaa.gov/home',
                             datetime(2018, 8, 16, 11, 19, 11),
                             exact=False)

# These values were previously not available except by parsing
# `memento.url`. The old `memento.url` is now `memento.memento_url`.
memento.url == 'http://www.noaa.gov/'
memento.timestamp == datetime(2018, 8, 29, 8, 8, 49, tzinfo=timezone.utc)
memento.mode == 'id_'

# Used to be `memento.url`:
memento.memento_url == 'http://web.archive.org/web/20180816111911id_/http://www.noaa.gov/
↪'
```

(continues on next page)

(continued from previous page)

```
# Used to be a list of `Response` objects, now a *tuple* of Mementos. It
# Still lists only the redirects that are actual Mementos and not part of
# Wayback's internal machinery:
memento.history == (Memento<url='http://noaa.gov/home'>,)

# Used to be a list of `Response` objects, now a *tuple* of URL strings:
memento.debug_history == ('http://web.archive.org/web/20180816111911id_/http://noaa.gov/
↪home',
                          'http://web.archive.org/web/20180829092926id_/http://noaa.gov/
↪home',
                          'http://web.archive.org/web/20180829092926id_/http://noaa.gov/
↪')

# Headers now only lists headers from the original, archived response, not
# additional headers from the Wayback Machine itself. (If there's
# important information you needed in the headers, file an issue and let
# us know! We'd like to surface that kind of information as attributes on
# the Memento now.
memento.headers = {'header_name': 'header_value',
                   'another_header': 'another_value',
                   'and': 'so on'}

# Same as before:
memento.status_code
memento.ok
memento.is_redirect
memento.encoding
memento.content
memento.text
```

Under the hood, *Wayback* still uses Requests for HTTP requests, but we expect to change that soon to ensure this package is thread-safe.

Other Breaking Changes

Finally, *wayback.memento_url_data()* now returns 3 values instead of 2. The last value is a string representing the playback mode (see above description of the new `mode` parameter on *wayback.WaybackClient.get_memento()* for more about playback modes).

3.18 v0.2.5 (2020-10-19)

This release fixes a bug where the `target_window` parameter for *wayback.WaybackClient.get_memento()* did not work correctly if the memento you were redirected to was off by more than a day from the requested time. See Issue #53 for more.

3.19 v0.2.4 (2020-09-07)

This release is focused on improved error handling.

Breaking Changes:

- The timestamps in `CdxRecord` objects returned by *wayback.WaybackClient.search()* now include time-zone information. (They are always in the UTC timezone.)

Updates:

- The `history` attribute of a memento now only includes redirects that were mementos (i.e. redirects that would have been seen when browsing the recorded site at the time it was recorded). Other redirects involved in working with the memento API are still available in `debug_history`, which includes all redirects, whether or not they were mementos.
- Wayback's CDX search API sometimes returns repeated, identical results. These are now filtered out, so repeat search results will not be yielded from *wayback.WaybackClient.search()*.
- *wayback.exceptions.RateLimitError* will now be raised as an exception any time you breach the Wayback Machine's rate limits. This would previously have been *wayback.exceptions.WaybackException*, *wayback.exceptions.MementoPlaybackError*, or regular HTTP responses, depending on the method you called. It has a `retry_after` property that indicates how many seconds you should wait before trying again (if the server sent that information, otherwise it will be `None`).
- *wayback.exceptions.BlockedSiteError* will now be raised any time you search for a URL or request a memento that has been blocked from access (for example, in situations where the Internet Archive has received a takedown notice).

3.20 v0.2.3 (2020-03-25)

This release downgrades the minimum Python version to 3.6! You can now use Wayback in places like Google Colab.

The `from_date` and `to_date` arguments for *wayback.WaybackClient.search()* can now be `datetime.date` instances in addition to `datetime.datetime`.

Huge thanks to @edsu for implementing both of these!

3.21 v0.2.2 (2020-02-13)

When errors were raised or redirects were involved in `WaybackClient.get_memento()`, it was previously possible for connections to be left hanging open. Wayback now works harder to make sure connections aren't left open.

This release also updates the default user agent string to include the repo URL. It now looks like: `wayback/0.2.2 (+https://github.com/edgi-govdata-archiving/wayback)`

3.22 v0.2.1 (2019-12-01)

All custom exceptions raised publicly and used internally are now exposed via a new module, `wayback.exceptions`.

3.23 v0.2.0 (2019-11-26)

Initial release of this project. See v0.1 below for information about a separate project with the same name that has since been removed from PyPI.

3.24 v0.1

This version number is reserved because it was the last published release of a separate Python project also named `wayback` that has since been deleted from the Python Package Index and subsequently superseded by this one. That project, which focused on the Wayback Machine's timemap API, was maintained by Jeff Goettsch (username `jgoettsch` on the Python Package Index). Its source code is still available on BitBucket at https://bitbucket.org/jgoettsch/py-wayback/.

INDEX

B

BlockedByRobotsError (*class in wayback.exceptions*), 17
BlockedSiteError (*class in wayback.exceptions*), 17

C

CdxRecord (*class in wayback*), 10
close() (*wayback.Memento method*), 14
content (*wayback.Memento attribute*), 12
css (*wayback.Mode attribute*), 16

D

debug_history (*wayback.Memento attribute*), 12
digest (*wayback.CdxRecord attribute*), 10

E

encoding (*wayback.Memento attribute*), 12

G

get_memento() (*wayback.WaybackClient method*), 9

H

headers (*wayback.Memento attribute*), 12
history (*wayback.Memento attribute*), 12

I

image (*wayback.Mode attribute*), 16
is_redirect (*wayback.Memento attribute*), 12

J

javascript (*wayback.Mode attribute*), 16

K

key (*wayback.CdxRecord attribute*), 11

L

length (*wayback.CdxRecord attribute*), 11
links (*wayback.Memento attribute*), 12

M

Memento (*class in wayback*), 11
memento_url (*wayback.Memento attribute*), 12
memento_url_data() (*in module wayback*), 15
MementoPlaybackError (*class in wayback.exceptions*), 17
mime_type (*wayback.CdxRecord attribute*), 11
mimetype (*wayback.CdxRecord attribute*), 10
Mode (*class in wayback*), 15
mode (*wayback.Memento attribute*), 12

N

NoMementoError (*class in wayback.exceptions*), 17

O

ok (*wayback.Memento attribute*), 12
original (*wayback.CdxRecord attribute*), 10
original (*wayback.Mode attribute*), 16

P

parse_memento_headers() (*wayback.Memento class method*), 14

R

RateLimit (*class in wayback*), 16
RateLimitError (*class in wayback.exceptions*), 17
raw_url (*wayback.CdxRecord attribute*), 11
reset() (*wayback.WaybackSession method*), 15

S

search() (*wayback.WaybackClient method*), 6
SessionClosedError (*class in wayback.exceptions*), 17
status_code (*wayback.CdxRecord attribute*), 11
status_code (*wayback.Memento attribute*), 12
statuscode (*wayback.CdxRecord attribute*), 10

T

text (*wayback.Memento attribute*), 12
timestamp (*wayback.CdxRecord attribute*), 10
timestamp (*wayback.Memento attribute*), 12

U

UnexpectedResponseFormat (*class in wayback.exceptions*), 16

url (*wayback.CdxRecord attribute*), 11
url (*wayback.Memento attribute*), 12
urlkey (*wayback.CdxRecord attribute*), 10

V

view (*wayback.Mode attribute*), 16
view_url (*wayback.CdxRecord attribute*), 11

W

WaybackClient (*class in wayback*), 6
WaybackException (*class in wayback.exceptions*), 16
WaybackRetryError (*class in wayback.exceptions*), 17
WaybackSession (*class in wayback*), 14