

---

# **pycaching Documentation**

***Release 4.4.0***

**Tomáš Bedřich**

**Sep 20, 2023**



---

## Contents

---

<b>1</b>	<b>Features</b>	<b>3</b>
<b>2</b>	<b>Documentation contents</b>	<b>5</b>
2.1	Quickstart . . . . .	5
2.2	API reference . . . . .	8
2.3	Contributing . . . . .	21
2.4	Appendix . . . . .	23
	<b>Python Module Index</b>	<b>25</b>
	<b>Index</b>	<b>27</b>



Source codes can be found at [GitHub repository](#).



# CHAPTER 1

---

## Features

---

- **login** to Geocaching.com
- **search** caches
  - normal search (unlimited number of caches from any point)
  - quick search (all caches inside some area) - currently not working, see below
- **get cache** and its details
  - normal loading (can load all details)
  - quick loading (can load just basic info but very quickly)
  - load logbook for given cache
- **get trackable** details by tracking code
- **post log** for a cache or a trackable
- **geocode** given location





## 2.1 Quickstart

### 2.1.1 Installation

Stable version - using pip:

```
pip install pycaching
```

Dev version - manually from GIT:

```
git clone https://github.com/tomasbedrich/pycaching.git
cd pycaching
pip install .
```

Pycaching has following requirements:

```
Python>=3.5
requests>=2.8
beautifulsoup4>=4.9
geopy>=1.11
```

Pycaching tests have the following additional requirements:

```
betamax >=0.8, <0.9
betamax-serializers >=0.2, <0.3
```

### 2.1.2 Examples

#### Login

Simply call `pycaching.login()` method and it will do everything for you.

```
import pycaching
geocaching = pycaching.login("user", "pass")
```

If you won't provide an username or password, pycaching will try to load `.gc_credentials` file from current directory or home folder. It will try to parse it as JSON and use the keys `username` and `password` from that file as login credentials.

```
{ "username": "myusername", "password": "mypassword" }
```

You can also provide multiple username and password tuples in a file as login credentials. The tuple to be used can be chosen by providing its username when calling `pycaching.login()`, e.g. `pycaching.login("myusername2")`. The first username and password tuple specified will be used as default if `pycaching.login()` is called without providing a username.

```
[ { "username": "myusername1", "password": "mypassword1" },
  { "username": "myusername2", "password": "mypassword2" } ]
```

```
import pycaching
geocaching = pycaching.login() # assume the .gc_credentials file is presented
```

In case you have a password manager in place featuring a command line interface (e.g. [GNU pass](#)) you may specify a password retrieval command using the `password_cmd` key instead of `password`.

```
{ "username": "myusername", "password_cmd": "pass geocaching.com/myUsername" }
```

Note that the `password` and `password_cmd` keys are mutually exclusive.

## Load a cache details

```
cache = geocaching.get_cache("GC1PAR2")
print(cache.name) # cache.load() is automatically called
print(cache.location) # stored in cache, printed immediately
```

This uses lazy loading, so the `Cache` object is created immediately and the page is loaded when needed (accessing the name).

You can use a different method of loading cache details. It will be much faster, but it will load less details:

```
cache = geocaching.get_cache("GC1PAR2")
cache.load_quick() # takes a small while
print(cache.name) # stored in cache, printed immediately
print(cache.location) # NOT stored in cache, will trigger full loading
```

You can also load a logbook for cache:

```
for log in cache.load_logbook(limit=200):
    print(log.visited, log.type, log.author, log.text)
```

Or its trackables:

```
for trackable in cache.load_trackables(limit=5):
    print(trackable.name)
```

## Post a log to cache

```
geocaching.post_log("GC1PAR2", "Found cache in the rain. Nice place, TFTC!")
```

It is also possible to call `post_log` on `Cache` object, but you would have to create `Log` object manually and pass it to this method.

## Search for all traditional caches around

```
from pycaching import Point
from pycaching.cache import Type

point = Point(56.25263, 15.26738)

for cache in geocaching.search(point, limit=50):
    if cache.type == Type.traditional:
        print(cache.name)
```

Notice the `limit` in the search function. It is because `geocaching.search()` returns a generator object, which would fetch the caches forever in case of a simple loop.

## Geocode address and search around

```
point = geocaching.geocode("Prague")

for cache in geocaching.search(point, limit=10):
    print(cache.name)
```

## Find caches in some area

```
from pycaching import Point, Rectangle

rect = Rectangle(Point(60.15, 24.95), Point(60.17, 25.00))

for cache in geocaching.search_rect(rect):
    print(cache.name)
```

If you want to search in a larger area, you could use the `limit` parameter as described above.

## Load trackable details

```
trackable = geocaching.get_trackable("TB3ZGT2")
print(trackable.name, trackable.goal, trackable.description, trackable.location)
```

## Post a log for trackable

```
from pycaching.log import Log, Type as LogType
import datetime

log = Log(type=LogType.discovered_it, text="Nice TB!", visited=datetime.date.today())
tracking_code = "ABCDEF"
trackable.post_log(log, tracking_code)
```

## Get geocaches by log type

```
from pycaching.log import Type as LogType

for find in geocaching.my_finds(limit=5):
    print(find.name)

for dnf in geocaching.my_dnfs(limit=2):
    print(dnf.name)

for note in geocaching.my_logs(LogType.note, limit=6):
    print(note.name)
```

## 2.2 API reference

Here you can find an overview of all available classes and methods.

**Warning:** Deprecated methods will be removed in next minor version.

For example: if you rely on some non-deprecated method in version 3.3, then it's fine to update once to 3.4. If the method gets deprecated in 3.4, then it will be removed in 3.5!

`pycaching.login(username=None, password=None)`

A shortcut for user login.

Create a *Geocaching* instance and try to login a user. See *Geocaching.login()*.

**Returns** Created *Geocaching* instance.

### 2.2.1 Geocaching

**class** `pycaching.geocaching.Geocaching(*, session=None)`

Provides some basic methods for communicating with geocaching.com website.

Provides methods to login and search. There are also some shortcut methods in this class to make working with pycaching more convinient.

**advanced\_search** (*options: dict, limit: int = inf, per\_query: int = 200, wait\_sleep: bool = True*) → Generator[Optional[pycaching.cache.Cache], None, None]

Perform an advanced search for geocaches with specific search criteria.

The search is performed using the options provided in the *options* parameter. Example of the *options* parameter:

```
# https://www.geocaching.com/play/search?owner[0]=Geocaching%20HQ&a=0
options = {"owner[0]": "Geocaching HQ", "a": "0"}
```

#### Parameters

- **options** – A dictionary of search options.
- **limit** – The maximum number of caches to load. Defaults to infinity.
- **per\_query** – The number of caches to request in each query. Defaults to 200.
- **wait\_sleep** – In case of rate limits exceeding, wait appropriate time if set to `True`, otherwise just yield `None`. Defaults to `True`.

**Returns** A generator that yields `Cache` objects.

**geocode** (*location*)

Return a `Point` object from geocoded location.

**Parameters** **location** (*str*) – Location to geocode.

**get\_cache** (*wp=None, guid=None*)

Return a `Cache` object by its waypoint or GUID.

#### Parameters

- **wp** (*str*) – Cache waypoint.
- **guid** (*str*) – Cache GUID.

---

**Note:** Provide only the GUID or the waypoint, not both.

---

**get\_logged\_user** (*login\_page=None*)

Return the name of currently logged user.

**Parameters** **login\_page** (*bs4.BeautifulSoup*) – Object containing already loaded page.

**Returns** User's name or `None`, if no user is logged in.

**Return type** `str` or `None`

**get\_trackable** (*tid*)

Return a `Trackable` object by its trackable ID.

**Parameters** **tid** (*str*) – Trackable ID.

**login** (*username=None, password=None*)

Log in the user for this instance of Geocaching.

If username or password is not set, try to load credentials from file. Then load login page and do some checks about currently logged user. As a last thing post the login form and check result.

#### Parameters

- **username** (*str*) – User's username or `None` to use data from credentials file.
- **password** (*str*) – User's password or `None` to use data from credentials file.

**Raises** `LoginFailedException` – If login fails either because of bad credentials or non-existing credentials file.

**logout** ()

Log out the user for this instance.

**my\_dnfs** (*limit=inf*)

Get an iterable of the logged-in user's DNFs.

**Parameters** **limit** – The maximum number of results to return (default: infinity).

**my\_finds** (*limit=inf*)

Get an iterable of the logged-in user's finds.

**Parameters** **limit** – The maximum number of results to return (default: infinity).

**my\_logs** (*log\_type=None, limit=inf*)

Get an iterable of the logged-in user's logs.

**Parameters**

- **log\_type** – The log type to search for. Use a *Type* value. If set to None, all logs will be returned (default: None).
- **limit** – The maximum number of results to return (default: infinity).

**post\_log** (*wp, text, type=<Type.found\_it: '2'>, date=None*)

Post a log for cache.

**Parameters**

- **wp** (*str*) – Cache waypoint.
- **text** (*str*) – Log text.
- **type** (*log.Type*) – Type of log.
- **date** (*datetime.date*) – Log date. If set to None, *datetime.date.today()* is used instead.

**search** (*point: pycaching.geo.Point, limit: int = inf, \*, sort\_by: Union[str, pycaching.geocaching.SortOrder] = <SortOrder.date\_last\_visited: 'datelastvisited'>, reverse: bool = False, per\_query: int = 200, wait\_sleep: bool = True*) → Generator[Optional[pycaching.cache.Cache], None, None]

Search for caches around a specified location using a search API.

**Parameters**

- **point** – The *geo.Point* object representing the center point of the search.
- **limit** – The maximum number of caches to load. Defaults to infinity.
- **sort\_by** – The criterion to sort the caches by. Defaults to *SortOrder.date\_last\_visited*.
- **reverse** – If True, the order of the results is reversed. Defaults to False.
- **per\_query** – The number of caches to request in each query. Defaults to 200.
- **wait\_sleep** – In case of rate limits exceeding, wait appropriate time if set to True, otherwise just yield None. Defaults to True.

**Returns** A generator that yields *Cache* objects.

**search\_quick** (*area*)

Search for caches in a specified *Rectangle* area.

**Parameters** **rect** (*geo.Rectangle*) – The *Rectangle* object representing the search area.

**Returns** A generator that yields [Cache](#) objects.

**Return type** Generator[Optional[[Cache](#)], None, None]

**search\_rect** (*rect*: [pycaching.geo.Rectangle](#), *limit*: *int* = *inf*, \*, *sort\_by*: *Union*[*str*, [pycaching.geocaching.SortOrder](#)] = *<SortOrder.date\_last\_visited: 'datelastvisited'>*, *reverse*: *bool* = *False*, *per\_query*: *int* = 200, *origin*: *Optional*[[pycaching.geo.Point](#)] = *None*, *wait\_sleep*: *bool* = *True*) → Generator[Optional[[pycaching.cache.Cache](#)], None, None]

Search for caches in a specified [Rectangle](#) area using a search API.

**Parameters**

- **rect** – The [Rectangle](#) object representing the search area.
- **limit** – The maximum number of caches to load. Defaults to infinity.
- **sort\_by** – The criterion to sort the caches by. Defaults to [SortOrder.date\\_last\\_visited](#).
- **reverse** – If *True*, the order of the results is reversed. Defaults to *False*.
- **per\_query** – The number of caches to request in each query. Defaults to 200.
- **origin** – The origin point for search by distance, required when sorting by distance.
- **wait\_sleep** – In case of rate limits exceeding, wait appropriate time if set to *True*, otherwise just yield *None*. Defaults to *True*.

**Returns** A generator that yields [Cache](#) objects.

**class** [pycaching.geocaching.SortOrder](#)

Enum of possible cache sort orderings returned in Groundspeak API.

## 2.2.2 Cache

**class** [pycaching.cache.Cache](#) (*geocaching*, *wp*, *\*\*kwargs*)

Represents a geocache with its properties and methods for loading them.

Provides some getters and setters for geocache properties like name, size, terrain, etc. Also contains two possible methods to load cache details and ensures, that these methods are called when accessing a property which hasn't been filled yet.

There are also methods for posting and loadings logs here. For more detail about Logs, please refer to [Log](#).

In summary, this class contains everything, which is possible to see or do on geocache page on [geocaching.com](#).

**classmethod** [from\\_block](#) (*block*)

Return [Cache](#) instance from [Block](#).

Used during quick search. The Cache will have only GC code, name and approximate location filled in.

**Parameters** **block** ([Block](#)) – Source block

**classmethod** [from\\_trackable](#) (*trackable*)

Return [Cache](#) instance from [Trackable](#).

This only makes sense, if trackable is currently placed in cache. Otherwise it will have unexpected behavior.

**Parameters** **trackable** ([Trackable](#)) – Source trackable.

### `load()`

Load all possible cache details.

Use full cache details page. Therefore all possible properties are filled in, but the loading is a bit slow.

If you want to load basic details about a PM only cache, the `PMOnlyException` is still thrown, but available details are filled in. If you know, that the cache you are loading is PM only, please consider using `load_quick()` as it will load the same details, but quicker.

---

**Note:** This method is called automatically when you access a property which isn't yet filled in (so-called "lazy loading"). You don't have to call it explicitly.

---

### Raises

- `PMOnlyException` – If cache is PM only and current user is basic member.
- `LoadError` – If cache loading fails (probably because of not existing cache).

### `load_by_guid()`

Load cache details using the GUID to request and parse the caches 'print-page'. Loading as many properties as possible except the following ones, since they are not present on the 'print-page':

- `original_location`
- `state`
- `found`
- `pm_only`

**Raises** `PMOnlyException` – If the PM only warning is shown on the page

### `load_logbook(limit=inf)`

Return a generator of logs for this cache.

Yield instances of `Log` filled with log data.

**Parameters** `limit(int)` – Maximum number of logs to generate.

### `load_quick()`

Load basic cache details.

Use information from geocaching map tooltips. Therefore loading is very quick, but the only loaded properties are: *name*, *type*, *size*, *difficulty*, *terrain*, *hidden*, *author*, *favorites* and *pm\_only*. It also loads *status*, but only for enabled caches. For other states, it can't be completely determined (can't distinguish between archived and locked caches), so lazy loading is used.

**Raises** `LoadError` – If cache loading fails (probably because of not existing cache).

### `load_trackables(limit=inf)`

Return a generator of trackables in this cache.

Yield instances of `Trackable` filled with trackable data.

**Parameters** `limit(int)` – Maximum number of trackables to generate.

### `post_log(log)`

Post a log for this cache.

**Parameters** `log(Log)` – Previously created `Log` filled with data.



**attributes**

The cache attributes.

**Setter** Set a cache attributes. Walk through passed `dict` and use `str` keys as attribute names and `bool` values as positive / negative attributes. Unknown attributes are ignored with warning (you can find possible attribute keys in `Cache._possible_attributes`).

**Type** `dict`

**author**

The cache author.

**Type** `str`

**description**

The cache long description.

**Type** `str`

**description\_html**

The cache long description in raw HTML.

**Type** `str`

**difficulty**

The cache difficulty.

**Setter** Set a cache difficulty. It must be in a common range - 1 to 5 in 0.5 steps.

**Type** `float`

**favorites**

The cache favorite points.

**Type** `int`

**found**

The cache found status.

True if cache is found by current user, False if not.

**Type** `bool`

**geocaching**

A reference to *Geocaching* used for communicating with geocaching.com.

**Type** *Geocaching* instance

**guid**

The cache GUID. An identifier used at some places on geocaching.com

**Type** `str`

**hidden**

The cache hidden date.

**Setter** Set a cache hidden date. If `str` is passed, then `util.parse_date()` is used and its return value is stored as a date.

**Type** `datetime.date`

**hint**

The cache hint.

**Setter** Set a cache hint. Don't decode text, you have to use `util.rot13()` before.

**Type** `str`

**location**

The cache location.

**Setter** Set a cache location. If `str` is passed, then `Point.from_string()` is used and its return value is stored as a location.

**Type** `Point`

**log\_counts**

The log count for each log type.

**Setter** Store a dictionary of log counts for each type used in the logbook of the current cache.

**Type** `dict`

**name**

A human readable name of cache.

**Type** `str`

**original\_location**

The cache original location.

**Setter** Set a cache original location. If `str` is passed, then `Point.from_string()` is used and its return value is stored as a location.

**Type** `Point`

**pm\_only**

If the cache is PM only.

**Type** `bool`

**size**

The cache size.

**Setter** Set a cache size. If `str` is passed, then `cache.Size.from_string()` is used and its return value is stored as a size.

**Type** `cache.Size`

**state**

The cache status.

True if cache is enabled, otherwise False.

**Type** `bool`

**status**

The cache status (Enabled, Disabled, Archived, Unpublished, Locked).

**Type** `cache.Status`

**summary**

The cache text summary.

**Type** `str`

**terrain**

The cache terrain.

**Setter** Set a cache terrain. It must be in a common range - 1 to 5 in 0.5 steps.

**Type** `float`

**type**

The cache type.

**Setter** Set a cache type. If `str` is passed, then `cache.Type.from_string()` is used and its return value is stored as a type.

**Type** `cache.Type`

#### **visited**

The cache log date (filled by function `geocaching.my_logs()` if cache is created there)

**Setter** Set a cache log date. If `str` is passed, then `util.parse_date()` is used and its return value is stored as a date.

**Type** `datetime.date`

#### **waypoints**

Any waypoints listed in the cache.

**Setter** Store a dictionary of locations using their lookup.

**Type** `dict`

#### **wp**

The cache GC code, must start with GC.

**Type** `str`

**class** `pycaching.cache.Waypoint` (*id=None, type=None, location=None, note=None*)

Waypoint represents a waypoint related to the cache. This may be a Parking spot, a stage in a multi-cache or similar.

#### **Parameters**

- **identifier** (*str*) – the unique identifier of the location
- **type** (*str*) – type of waypoint
- **location** (*Point*) – waypoint coordinates
- **note** (*str*) – Information about the waypoint

**classmethod** `from_html` (*soup, table\_id*)

Return a dictionary of all waypoints found in the page representation

#### **Parameters**

- **soup** (*bs4.BeautifulSoup*) – parsed html document containing the waypoints table
- **table\_id** (*str*) – html id of the waypoints table

#### **identifier**

The waypoint unique identifier.

**Type** `str`

#### **location**

The waypoint location.

**Type** `Point`

#### **note**

Any additional information about the waypoint.

**Type** `str`

#### **type**

The waypoint type.

**Type** `str`

**class** pycaching.cache.Type

Enum of possible cache types.

according to

- <https://www.geocaching.com/app/ui-icons/sprites/cache-types.svg>
- [https://www.geocaching.com/about/cache\\_types.aspx](https://www.geocaching.com/about/cache_types.aspx)

cache\_in\_trash\_out\_event = '13'

cito = '13'

community\_celebration = '3653'

earthcache = '137'

event = '6'

from\_filename = <bound method Type.from\_filename of <enum 'Type'>>

from\_number = <bound method Type.from\_number of <enum 'Type'>>

from\_string = <bound method Type.from\_string of <enum 'Type'>>

geocaching\_hq = '3773'

geocaching\_hq\_block\_party = '4738'

giga\_event = '7005'

gps\_adventures\_exhibit = '1304'

gps\_maze = '1304'

groundspk\_block\_party = '4738'

groundspk\_hq = '3773'

hq\_celebration = '3774'

letterbox = '5'

locationless = '12'

lost\_and\_found\_event = '3653'

mega\_event = '453'

multicache = '3'

mystery = '8'

project\_ape = '9'

puzzle = '8'

reverse = '12'

traditional = '2'

unknown = '8'

virtual = '4'

webcam = '11'

wherigo = '1858'

```
class pycaching.cache.Size
```

Enum of possible cache sizes.

Naming follows Groundspeak image filenames, values are human readable names.

```
from_filename = <bound method Size.from_filename of <enum 'Size'>>
from_number = <bound method Size.from_number of <enum 'Size'>>
from_string = <bound method Size.from_string of <enum 'Size'>>
large = 'large'
micro = 'micro'
not_chosen = 'not chosen'
other = 'other'
regular = 'regular'
small = 'small'
virtual = 'virtual'
```

### 2.2.3 Logs

```
class pycaching.log.Log (*, uuid=None, type=None, text=None, visited=None, author=None)
```

Represents a log record with its properties.

**author**

The log author.

Type `str`

**text**

The log text.

Type `str`

**type**

The log type.

Type `log.Type`

**uuid**

The log unique identifier.

Type `str`

**visited**

The log date.

**Setter** Set a log date. If `str` is passed, then `util.parse_date()` is used and its return value is stored as a date.

Type `datetime.date`

```
class pycaching.log.Type
```

Enum of possible log types.

Values are log type IDs (as used in HTML `<option value=XX>` on the log page). Also the log images can be found there - [https://www.geocaching.com/images/logtypes/\[\]VALUE{}.png](https://www.geocaching.com/images/logtypes/[]VALUE{}.png)

```
announcement = '74'
```

```
archive = '5'
attended = '10'
didnt_find_it = '3'
discovered_it = '48'
enable_listing = '23'
found_it = '2'
from_filename = <bound method Type.from_filename of <enum 'Type'>>
grabbed_it = '19'
marked_missing = '16'
needs_archive = '7'
needs_maintenance = '45'
note = '4'
oc_team_comment = '83'
owner_maintenance = '46'
placed_it = '14'
post_reviewer_note = '18'
publish_listing = '24'
retract = '25'
retrieved_it = '13'
submit_for_review = '76'
temp_disable_listing = '22'
unarchive = '12'
update_coordinates = '47'
visit = '75'
webcam_photo_taken = '11'
will_attend = '9'
```

## 2.2.4 Trackables

```
class pycaching.trackable.Trackable(geocaching, tid, *, name=None, location=None,  
                                   owner=None, type=None, description=None,  
                                   goal=None, url=None)
```

Represents a trackable with its properties.

```
get_KML()
```

Return the KML route of the trackable.

Return type `str`

**load()**

Load all possible details about the trackable.

---

**Note:** This method is called automatically when you access a property which isn't yet filled in (so-called "lazy loading"). You don't have to call it explicitly.

---

**Raises** `LoadError` – If trackable loading fails (probably because of not existing trackable).

**post\_log(log, tracking\_code)**

Post a log for this trackable.

**Parameters**

- **log** (`Log`) – Previously created `Log` filled with data.
- **tracking\_code** (`str`) – A tracking code to verify current trackable holder.

**description**

The trackable long description.

**Type** `str`

**geocaching**

A reference to `Geocaching` used for communicating with geocaching.com.

**Type** `Geocaching` instance

**goal**

The trackable goal.

**Type** `str`

**location**

The trackable current location.

Can be either string with location description (eg. "in the hands of someone") or cache URL.

**Type** `str`

**name**

A human readable trackable name.

**Type** `str`

**owner**

The trackable owner.

**Type** `str`

**tid**

The trackable ID, must start with TB.

**Type** `str`

**type**

The trackable type.

**A type depends on the trackable icon. It can be either "Travel Bug Dog Tag" or specific geocoin name, eg. "Adventure Race Hracholusky 2015 Geocoin".**

**Type** `str`

## 2.2.5 Geo utilities

`pycaching.geo.to_decimal(deg, min)`

Convert coordinates from degrees minutes to decimal degrees format.

**class** `pycaching.geo.Point`

A point on earth defined by its latitude, longitude and possibly more attributes.

Subclass of `geopy.Point`.

**classmethod** `from_location(geocaching, location)`

Return a `Point` instance from geocoded location.

**Parameters**

- **geocaching** (`Geocaching`) – Reference to `Geocaching` instance, used to do a geocoding request.
- **location** (`str`) – Location to geocode.

**Raises** `GeocodeError` – If location cannot be geocoded (not found).

**classmethod** `from_string(string)`

Return a `Point` instance from coordinates in degrees minutes format.

This method can handle various malformed formats. Example inputs are:

- S 36 51.918 E 174 46.725 or
- N 6 52.861 w174 43.327

**Parameters** **string** (`str`) – Coordinates to parse.

**Raises** `ValueError` – If string cannot be parsed as coordinates.

**class** `pycaching.geo.Polygon(*points)`

Area defined by bordering `Point` instances.

Subclass of `Area`.

**bounding\_box**

Get area's bounding box (`Rectangle` computed from min and max coordinates).

**mean\_point**

Return `Point` with average latitude and longitude of all area's points.

**class** `pycaching.geo.Rectangle(point_a, point_b)`

Upright rectangle.

Subclass of `Polygon`.

**\_\_contains\_\_** (`p`)

Return if the rectangle contains a point.

**Parameters** **p** (`Point`) – Examined point.

**diagonal**

Return a length of bounding box diagonal in meters as `int`.

## 2.2.6 Errors

**exception** `pycaching.errors.BadBlockError`



**exception** `pycaching.errors.Error`

General pycaching error.

**exception** `pycaching.errors.GeocodeError`

Geocoding failed.

Probably because of non-existing location.

**exception** `pycaching.errors.LoadError`

Object loading failed.

Probably because of non-existing object or missing informations required to load it.

**exception** `pycaching.errors.LoginFailedException`

Login failed.

The provided credentials probably doesn't work to log in.

**exception** `pycaching.errors.NotLoggedInException`

Tried to perform an operation which requires logging in first.

**exception** `pycaching.errors.PMOnlyException`

Requested cache is PM only.

**exception** `pycaching.errors.TooManyRequestsError` (*url: str, rate\_limit\_reset: int = 0*)

Geocaching API rate limit has been reached.

**wait\_for()**

Wait enough time to release Rate Limits.

**exception** `pycaching.errors.ValueError`

Wrapper for Python's native ValueError.

Can be raised in various situations, but most commonly when unexpected property value is set.

## 2.3 Contributing

### 2.3.1 First time

1. Clone a repository from GitHub:

```
git clone https://github.com/tomasbedrich/pycaching.git
cd pycaching
```

2. Create a virtualenv:

```
python3 -m venv .venv
source .venv/bin/activate # Unix
.venv\Scripts\activate # Windows
```

3. Install Flit:

```
pip install flit
```

4. Install Pycaching package + dependencies in development mode:

```
flit install --symlink
```

## 2.3.2 Typical workflow

1. Pick an issue labeled as “[contributors friendly](#)” or create a new one. Please **notify others** that you will solve this problem (write a comment on GitHub).
2. Activate the virtualenv:

```
source .venv/bin/activate # Unix
.venv\Scripts\activate # Windows
```

3. Write some **code and tests**. After that, don’t forget to update the **docs**. See coding style below.
4. Sort imports using [isort](#), format the code using [Black](#), lint it using [Flake](#) and run the tests using [pytest](#):

```
isort .
black .
flake8
pytest
```

Make sure that:

- there are no lint errors,
- all tests are passing,
- the coverage is above 90%.

6. Push your work and create a **pull request**. Yay!

## 2.3.3 Testing

Pycaching uses [Betamax](#) for testing, which speeds it up by recording network requests so that they can be mocked.

If you haven’t written or modified any tests, tests can be run just like:

```
pytest <test folder name>
```

If you have written or modified tests, you must provide a username and password for testing. Don’t worry, these will not leave your computer. Betamax will insert a placeholder when it records any new cassettes. To run new tests, first set up the following environment variables:

```
PYCACHING_TEST_USERNAME="yourusername" PYCACHING_TEST_PASSWORD="yourpassword" pytest
↪<test folder name>
```

Substitute your username for `yourusername` and your password for `yourpassword`. This requires you to use a basic member account, otherwise you might see unexpected test failures.

To re-record a specific cassette in case of site changes, delete the corresponding JSON file and provide username and password as explained above. The missing cassette will be recorded for future usages.

## 2.3.4 Coding style

- Use `.format()` for string formatting. Black will guide you with the rest. :)
- For docs, please follow [PEP257](#).
- **Importing modules** is okay for modules from standard library. If you want to include a third-party module, please consult it on GitHub before.

- Please use regular expressions only as a last resort. When possible, use string manipulations, such as `split()` and then list operations. It is more readable.

### 2.3.5 Release process

1. Pick a suitable semantic version number. We adhere to [generic rules](#) with an exception of our [specific deprecation policy](#).
2. If the deprecation policy triggers, remove the deprecated methods. Create a separate PR in that case.
3. Bump the version number in `pycaching/__init__.py` ([example](#)). Feel free to push this bump directly to `master`, or create a regular PR.
4. Once the version bump commit equals HEAD of `master`, [draft a new release](#) using Github. Using that form, create a new tag corresponding to the version number (no prefixes). Leave release title empty, let Github generate the release notes. Update release notes manually if needed.
5. Publish the Github release. There is a Github action which publishes the release to Pypi. There are Webhooks which update Readthedocs and Coveralls.

Should there be any issue with the above (most likely stuck release pipeline), please create a Github issue.

## 2.4 Appendix

### 2.4.1 Legal notice

Be sure to read [Geocaching.com's terms of use](#). By using this piece of software you break them and your Geocaching account may be suspended or *even deleted*. To prevent this, I recommend you to load the data you really need, nothing more. This software is provided “as is” and I am not responsible for any damage possibly caused by it.

### 2.4.2 Inspiration

Original version was inspired by these packages:

- [Geocache Grabber](#) (by Fuad Tabba)
- [geocaching-py](#) (by Lev Shamardin)

Although the new version was massively rewritten, I'd like to thank to their authors.

### 2.4.3 Authors

Authors of this project are [all contributors](#). Maintainer is [Tomáš Bedřich](#).



### p

- `pycaching`, [8](#)
- `pycaching.errors`, [20](#)
- `pycaching.geo`, [20](#)
- `pycaching.geocaching`, [8](#)
- `pycaching.trackable`, [18](#)



## Symbols

`__contains__()` (*pycaching.geo.Rectangle* method), 20

## A

`advanced_search()` (*pycaching.geocaching.Geocaching* method), 8

`announcement` (*pycaching.log.Type* attribute), 17

`archive` (*pycaching.log.Type* attribute), 17

`attended` (*pycaching.log.Type* attribute), 18

`attributes` (*pycaching.cache.Cache* attribute), 12

`author` (*pycaching.cache.Cache* attribute), 13

`author` (*pycaching.log.Log* attribute), 17

## B

`BadBlockError`, 20

`bounding_box` (*pycaching.geo.Polygon* attribute), 20

## C

`Cache` (class in *pycaching.cache*), 11

`cache_in_trash_out_event` (*pycaching.cache.Type* attribute), 16

`cito` (*pycaching.cache.Type* attribute), 16

`community_celebration` (*pycaching.cache.Type* attribute), 16

## D

`description` (*pycaching.cache.Cache* attribute), 13

`description` (*pycaching.trackable.Trackable* attribute), 19

`description_html` (*pycaching.cache.Cache* attribute), 13

`diagonal` (*pycaching.geo.Rectangle* attribute), 20

`didnt_find_it` (*pycaching.log.Type* attribute), 18

`difficulty` (*pycaching.cache.Cache* attribute), 13

`discovered_it` (*pycaching.log.Type* attribute), 18

## E

`earthcache` (*pycaching.cache.Type* attribute), 16

`enable_listing` (*pycaching.log.Type* attribute), 18

`Error`, 20

`event` (*pycaching.cache.Type* attribute), 16

## F

`favorites` (*pycaching.cache.Cache* attribute), 13

`found` (*pycaching.cache.Cache* attribute), 13

`found_it` (*pycaching.log.Type* attribute), 18

`from_block()` (*pycaching.cache.Cache* class method), 11

`from_filename` (*pycaching.cache.Size* attribute), 17

`from_filename` (*pycaching.cache.Type* attribute), 16

`from_filename` (*pycaching.log.Type* attribute), 18

`from_html()` (*pycaching.cache.Waypoint* class method), 15

`from_location()` (*pycaching.geo.Point* class method), 20

`from_number` (*pycaching.cache.Size* attribute), 17

`from_number` (*pycaching.cache.Type* attribute), 16

`from_string` (*pycaching.cache.Size* attribute), 17

`from_string` (*pycaching.cache.Type* attribute), 16

`from_string()` (*pycaching.geo.Point* class method), 20

`from_trackable()` (*pycaching.cache.Cache* class method), 11

## G

`Geocaching` (class in *pycaching.geocaching*), 8

`geocaching` (*pycaching.cache.Cache* attribute), 13

`geocaching` (*pycaching.trackable.Trackable* attribute), 19

`geocaching_hq` (*pycaching.cache.Type* attribute), 16

`geocaching_hq_block_party` (*pycaching.cache.Type* attribute), 16

`geocode()` (*pycaching.geocaching.Geocaching* method), 9

`GeocodeError`, 21

`get_cache()` (*pycaching.geocaching.Geocaching* method), 9

get\_KML() (*pycaching.trackable.Trackable method*), 18  
 get\_logged\_user() (*pycaching.geocaching.Geocaching method*), 9  
 get\_trackable() (*pycaching.geocaching.Geocaching method*), 9  
 giga\_event (*pycaching.cache.Type attribute*), 16  
 goal (*pycaching.trackable.Trackable attribute*), 19  
 gps\_adventures\_exhibit (*pycaching.cache.Type attribute*), 16  
 gps\_maze (*pycaching.cache.Type attribute*), 16  
 grabbed\_it (*pycaching.log.Type attribute*), 18  
 groundspeak\_block\_party (*pycaching.cache.Type attribute*), 16  
 groundspeak\_hq (*pycaching.cache.Type attribute*), 16  
 guid (*pycaching.cache.Cache attribute*), 13

## H

hidden (*pycaching.cache.Cache attribute*), 13  
 hint (*pycaching.cache.Cache attribute*), 13  
 hq\_celebration (*pycaching.cache.Type attribute*), 16

## I

identifier (*pycaching.cache.Waypoint attribute*), 15

## L

large (*pycaching.cache.Size attribute*), 17  
 letterbox (*pycaching.cache.Type attribute*), 16  
 load() (*pycaching.cache.Cache method*), 11  
 load() (*pycaching.trackable.Trackable method*), 18  
 load\_by\_guid() (*pycaching.cache.Cache method*), 12  
 load\_logbook() (*pycaching.cache.Cache method*), 12  
 load\_quick() (*pycaching.cache.Cache method*), 12  
 load\_trackables() (*pycaching.cache.Cache method*), 12  
 LoadError, 21  
 location (*pycaching.cache.Cache attribute*), 13  
 location (*pycaching.cache.Waypoint attribute*), 15  
 location (*pycaching.trackable.Trackable attribute*), 19  
 locationless (*pycaching.cache.Type attribute*), 16  
 Log (*class in pycaching.log*), 17  
 log\_counts (*pycaching.cache.Cache attribute*), 14  
 login() (*in module pycaching*), 8  
 login() (*pycaching.geocaching.Geocaching method*), 9  
 LoginFailedException, 21

logout() (*pycaching.geocaching.Geocaching method*), 9  
 lost\_and\_found\_event (*pycaching.cache.Type attribute*), 16

## M

marked\_missing (*pycaching.log.Type attribute*), 18  
 mean\_point (*pycaching.geo.Polygon attribute*), 20  
 mega\_event (*pycaching.cache.Type attribute*), 16  
 micro (*pycaching.cache.Size attribute*), 17  
 multicache (*pycaching.cache.Type attribute*), 16  
 my\_dnfs() (*pycaching.geocaching.Geocaching method*), 10  
 my\_finds() (*pycaching.geocaching.Geocaching method*), 10  
 my\_logs() (*pycaching.geocaching.Geocaching method*), 10  
 mystery (*pycaching.cache.Type attribute*), 16

## N

name (*pycaching.cache.Cache attribute*), 14  
 name (*pycaching.trackable.Trackable attribute*), 19  
 needs\_archive (*pycaching.log.Type attribute*), 18  
 needs\_maintenance (*pycaching.log.Type attribute*), 18  
 not\_chosen (*pycaching.cache.Size attribute*), 17  
 note (*pycaching.cache.Waypoint attribute*), 15  
 note (*pycaching.log.Type attribute*), 18  
 NotLoggedInException, 21

## O

oc\_team\_comment (*pycaching.log.Type attribute*), 18  
 original\_location (*pycaching.cache.Cache attribute*), 14  
 other (*pycaching.cache.Size attribute*), 17  
 owner (*pycaching.trackable.Trackable attribute*), 19  
 owner\_maintenance (*pycaching.log.Type attribute*), 18

## P

placed\_it (*pycaching.log.Type attribute*), 18  
 pm\_only (*pycaching.cache.Cache attribute*), 14  
 PMOnlyException, 21  
 Point (*class in pycaching.geo*), 20  
 Polygon (*class in pycaching.geo*), 20  
 post\_log() (*pycaching.cache.Cache method*), 12  
 post\_log() (*pycaching.geocaching.Geocaching method*), 10  
 post\_log() (*pycaching.trackable.Trackable method*), 19  
 post\_reviewer\_note (*pycaching.log.Type attribute*), 18  
 project\_ape (*pycaching.cache.Type attribute*), 16  
 publish\_listing (*pycaching.log.Type attribute*), 18



puzzle (*pycaching.cache.Type attribute*), 16  
 pycaching (*module*), 8  
 pycaching.errors (*module*), 20  
 pycaching.geo (*module*), 20  
 pycaching.geocaching (*module*), 8  
 pycaching.trackable (*module*), 18

## R

Rectangle (*class in pycaching.geo*), 20  
 regular (*pycaching.cache.Size attribute*), 17  
 retract (*pycaching.log.Type attribute*), 18  
 retrieved\_it (*pycaching.log.Type attribute*), 18  
 reverse (*pycaching.cache.Type attribute*), 16

## S

search() (*pycaching.geocaching.Geocaching method*), 10  
 search\_quick() (*pycaching.geocaching.Geocaching method*), 10  
 search\_rect() (*pycaching.geocaching.Geocaching method*), 11  
 Size (*class in pycaching.cache*), 16  
 size (*pycaching.cache.Cache attribute*), 14  
 small (*pycaching.cache.Size attribute*), 17  
 SortOrder (*class in pycaching.geocaching*), 11  
 state (*pycaching.cache.Cache attribute*), 14  
 status (*pycaching.cache.Cache attribute*), 14  
 submit\_for\_review (*pycaching.log.Type attribute*), 18  
 summary (*pycaching.cache.Cache attribute*), 14

## T

temp\_disable\_listing (*pycaching.log.Type attribute*), 18  
 terrain (*pycaching.cache.Cache attribute*), 14  
 text (*pycaching.log.Log attribute*), 17  
 tid (*pycaching.trackable.Trackable attribute*), 19  
 to\_decimal() (*in module pycaching.geo*), 20  
 TooManyRequestsError, 21  
 Trackable (*class in pycaching.trackable*), 18  
 traditional (*pycaching.cache.Type attribute*), 16  
 Type (*class in pycaching.cache*), 15  
 Type (*class in pycaching.log*), 17  
 type (*pycaching.cache.Cache attribute*), 14  
 type (*pycaching.cache.Waypoint attribute*), 15  
 type (*pycaching.log.Log attribute*), 17  
 type (*pycaching.trackable.Trackable attribute*), 19

## U

unarchive (*pycaching.log.Type attribute*), 18  
 unknown (*pycaching.cache.Type attribute*), 16  
 update\_coordinates (*pycaching.log.Type attribute*), 18

uuid (*pycaching.log.Log attribute*), 17

## V

ValueError, 21  
 virtual (*pycaching.cache.Size attribute*), 17  
 virtual (*pycaching.cache.Type attribute*), 16  
 visit (*pycaching.log.Type attribute*), 18  
 visited (*pycaching.cache.Cache attribute*), 15  
 visited (*pycaching.log.Log attribute*), 17

## W

wait\_for() (*pycaching.errors.TooManyRequestsError method*), 21  
 Waypoint (*class in pycaching.cache*), 15  
 waypoints (*pycaching.cache.Cache attribute*), 15  
 webcam (*pycaching.cache.Type attribute*), 16  
 webcam\_photo\_taken (*pycaching.log.Type attribute*), 18  
 wherigo (*pycaching.cache.Type attribute*), 16  
 will\_attend (*pycaching.log.Type attribute*), 18  
 wp (*pycaching.cache.Cache attribute*), 15