

---

# **qbittorrent-api**

***Release v2023.4.45***

**Russell Martin**

**Apr 14, 2023**



# CONTENTS

<b>1</b>	<b>Introduction</b>	<b>1</b>
	<b>Python Module Index</b>	<b>57</b>
	<b>Index</b>	<b>59</b>



## INTRODUCTION

Python client implementation for qBittorrent Web API.

Currently supports qBittorrent [v4.5.2](#) (Web API v2.8.19) released on Feb 27, 2023.

### 1.1 Features

- The entire qBittorrent [Web API](#) is implemented.
- qBittorrent version checking for an endpoint's existence/features is automatically handled.
- All Python versions are supported.
- If the authentication cookie expires, a new one is automatically requested in line with any API call.

### 1.2 Installation

- Install via pip from [PyPI](#):

```
pip install qbittorrent-api
```

- Install a specific release (e.g. [v2022.8.34](#)):

```
pip install git+https://github.com/rmartin16/qbittorrent-api.git@v2022.8.34  
↪#egg=qbittorrent-api
```

- Install direct from main:

```
pip install git+https://github.com/rmartin16/qbittorrent-api.git@main#egg=qbittorrent-api
```

- Enable WebUI in qBittorrent: Tools -> Preferences -> Web UI
- If the Web API will be exposed to the Internet, follow the [recommendations](#).

## 1.3 Getting Started

```
import qbittorrentapi

# instantiate a Client using the appropriate WebUI configuration
qbt_client = qbittorrentapi.Client(
    host='localhost',
    port=8080,
    username='admin',
    password='adminadmin'
)

# the Client will automatically acquire/maintain a logged in state in line with any
# request.
# therefore, this is not necessary; however, you may want to test the provided login
# credentials.
try:
    qbt_client.auth_log_in()
except qbittorrentapi.LoginFailed as e:
    print(e)

# display qBittorrent info
print(f'qBittorrent: {qbt_client.app.version}')
print(f'qBittorrent Web API: {qbt_client.app.web_api_version}')
for k,v in qbt_client.app.build_info.items(): print(f'{k}: {v}')

# retrieve and show all torrents
for torrent in qbt_client.torrents_info():
    print(f'{torrent.hash[-6:]}: {torrent.name} ({torrent.state})')

# pause all torrents
qbt_client.torrents.pause.all()
```

## 1.4 Usage

First, the Web API endpoints are organized in to eight namespaces.

- Authentication (auth)
- Application (app)
- Log (log)
- Sync (sync)
- Transfer (transfer)
- Torrent Management (torrents)
- RSS (rss)
- Search (search)

Second, this client has two modes of interaction with the qBittorrent Web API.

Each Web API endpoint is implemented one-to-one as a method of the instantiated client.

```
import qbittorrentapi
qbt_client = qbittorrentapi.Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
qbt_client.app_version()
qbt_client.rss_rules()
qbt_client.torrents_info()
qbt_client.torrents_resume(torrent_hashes='...')
# and so on
```

However, a more robust interface to the endpoints is available via each namespace. This is intended to provide a more seamless and intuitive interface to the Web API.

```
import qbittorrentapi
qbt_client = qbittorrentapi.Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
# changing a preference
is_dht_enabled = qbt_client.app.preferences.dht
qbt_client.app.preferences = dict(dht=not is_dht_enabled)
# stopping all torrents
qbt_client.torrents.pause.all()
# retrieve different views of the log
qbt_client.log.main.warning()
qbt_client.log.main.normal()
```

Finally, some of the objects returned by the client support methods of their own. This is most pronounced for torrents themselves.

```
import qbittorrentapi
qbt_client = qbittorrentapi.Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')

for torrent in qbt_client.torrents.info.active():
    torrent.set_location(location='/home/user/torrents/')
    torrent.reannounce()
    torrent.upload_limit = -1
```

## 1.4.1 Introduction

Python client implementation for qBittorrent Web API.

Currently supports qBittorrent v4.5.2 (Web API v2.8.19) released on Feb 27, 2023.

## Features

- The entire qBittorrent [Web API](#) is implemented.
- qBittorrent version checking for an endpoint's existence/features is automatically handled.
- All Python versions are supported.
- If the authentication cookie expires, a new one is automatically requested in line with any API call.

## Installation

- Install via pip from [PyPI](#):

```
pip install qbittorrent-api
```

- Install a specific release (e.g. v2022.8.34):

```
pip install git+https://github.com/rmartin16/qbittorrent-api.git@v2022.8.34
↪#egg=qbittorrent-api
```

- Install direct from main:

```
pip install git+https://github.com/rmartin16/qbittorrent-api.git@main#egg=qbittorrent-api
```

- Enable WebUI in qBittorrent: Tools -> Preferences -> Web UI
- If the Web API will be exposed to the Internet, follow the [recommendations](#).

## Getting Started

```
import qbittorrentapi

# instantiate a Client using the appropriate WebUI configuration
qbt_client = qbittorrentapi.Client(
    host='localhost',
    port=8080,
    username='admin',
    password='adminadmin'
)

# the Client will automatically acquire/maintain a logged in state in line with any
↪request.
# therefore, this is not necessary; however, you may want to test the provided login
↪credentials.
try:
    qbt_client.auth_log_in()
except qbittorrentapi.LoginFailed as e:
    print(e)

# display qBittorrent info
print(f'qBittorrent: {qbt_client.app.version}')
print(f'qBittorrent Web API: {qbt_client.app.web_api_version}')
for k,v in qbt_client.app.build_info.items(): print(f'{k}: {v}')
```

(continues on next page)



(continued from previous page)

```
# retrieve and show all torrents
for torrent in qbt_client.torrents_info():
    print(f'{torrent.hash[-6:]}: {torrent.name} ({torrent.state})')

# pause all torrents
qbt_client.torrents.pause.all()
```

## Usage

First, the Web API endpoints are organized in to eight namespaces.

- Authentication (auth)
- Application (app)
- Log (log)
- Sync (sync)
- Transfer (transfer)
- Torrent Management (torrents)
- RSS (rss)
- Search (search)

Second, this client has two modes of interaction with the qBittorrent Web API.

Each Web API endpoint is implemented one-to-one as a method of the instantiated client.

```
import qbittorrentapi
qbt_client = qbittorrentapi.Client(host='localhost:8080', username='admin', password=
↪ 'adminadmin')
qbt_client.app_version()
qbt_client.rss_rules()
qbt_client.torrents_info()
qbt_client.torrents_resume(torrent_hashes='...')
# and so on
```

However, a more robust interface to the endpoints is available via each namespace. This is intended to provide a more seamless and intuitive interface to the Web API.

```
import qbittorrentapi
qbt_client = qbittorrentapi.Client(host='localhost:8080', username='admin', password=
↪ 'adminadmin')
# changing a preference
is_dht_enabled = qbt_client.app.preferences.dht
qbt_client.app.preferences = dict(dht=not is_dht_enabled)
# stopping all torrents
qbt_client.torrents.pause.all()
# retrieve different views of the log
qbt_client.log.main.warning()
qbt_client.log.main.normal()
```

Finally, some of the objects returned by the client support methods of their own. This is most pronounced for torrents themselves.

```
import qbittorrentapi
qbt_client = qbittorrentapi.Client(host='localhost:8080', username='admin', password=
    ↪ 'adminadmin')

for torrent in qbt_client.torrents.info.active():
    torrent.set_location(location='/home/user/torrents/')
    torrent.reannounce()
    torrent.upload_limit = -1
```

## 1.4.2 Behavior & Configuration

### Untrusted WebUI Certificate

- qBittorrent allows you to configure HTTPS with an untrusted certificate; this commonly includes self-signed certificates.
- When using such a certificate, instantiate `Client` with `VERIFY_WEBUI_CERTIFICATE=False` or set environment variable `QBITTORRENTAPI_DO_NOT_VERIFY_WEBUI_CERTIFICATE` to a non-null value.
- Failure to do this for will cause connections to qBittorrent to fail.
- As a word of caution, doing this actually does turn off certificate verification. Therefore, for instance, potential man-in-the-middle attacks will not be detected and reported (since the error is suppressed). However, the connection will remain encrypted.

```
qbt_client = Client(..., VERIFY_WEBUI_CERTIFICATE=False)
```

### Host, Username and Password

- The authentication credentials can be provided when instantiating `Client`:

```
qbt_client = Client(host="localhost:8080", username='...', password='...')
```

- The credentials can also be specified after `Client` is created but calling `auth_log_in()` is not strictly necessary to authenticate the client; this will happen automatically for any API request.

```
qbt_client.auth_log_in(username='...', password='...')
```

- Alternatively, the credentials can be specified in environment variables:
  - `QBITTORRENTAPI_HOST`
  - `QBITTORRENTAPI_USERNAME`
  - `QBITTORRENTAPI_PASSWORD`

## Requests Configuration

- The `Requests` package is used to issue HTTP requests to qBittorrent to facilitate this API.
- Much of `Requests` configuration for making HTTP calls can be controlled with parameters passed along with the request payload.
- For instance, HTTP Basic Authorization credentials can be provided via `auth`, timeouts via `timeout`, or Cookies via `cookies`. See [Requests documentation](#) for full details.
- These parameters are exposed here in two ways; the examples below tell `Requests` to use a connect timeout of 3.1 seconds and a read timeout of 30 seconds.
- When you instantiate `Client`, you can specify the parameters to use in all HTTP requests to qBittorrent:

```
qbt_client = Client(..., REQUESTS_ARGS={'timeout': (3.1, 30)})
```

- Alternatively, parameters can be specified for individual requests:

```
qbt_client.torrents_info(..., requests_args={'timeout': (3.1, 30)})
```

## Additional HTTP Headers

- For consistency, HTTP Headers can be specified using the method above; for backwards compatability, the methods below are supported as well.
- Either way, these additional headers will be incorporated (using clobbering) into the rest of the headers to be sent.
- To send a custom HTTP header in all requests made from an instantiated client, declare them during instantiation:

```
qbt_client = Client(..., EXTRA_HEADERS={'X-My-Fav-Header': 'header value'})
```

- Alternatively, you can send custom headers in individual requests:

```
qbt_client.torrents.add(..., headers={'X-My-Fav-Header': 'header value'})
```

## Unimplemented API Endpoints

- Since the qBittorrent Web API has evolved over time, some endpoints may not be available from the qBittorrent host.
- By default, if a call is made to endpoint that doesn't exist for the version of the qBittorrent host (e.g., the Search endpoints were introduced in Web API v2.1.1), there's a debug logger output and `None` is returned.
- To raise `NotImplementedError` instead, instantiate `Client` with:

```
qbt_client = Client(..., RAISE_NOTIMPLEMENTEDERROR_FOR_UNIMPLEMENTED_API_ENDPOINTS=True)
```

## qBittorrent Version Checking

- It is also possible to either raise an Exception for qBittorrent hosts that are not “fully” supported or manually check for support.
- The most likely situation for this to occur is if the qBittorrent team publishes a new release but its changes have not been incorporated in to this client yet.
- Instantiate Client like below to raise `UnsupportedQbittorrentVersion` exception for versions not fully supported:

```
qbt_client = Client(..., RAISE_ERROR_FOR_UNSUPPORTED_QBITTORRENT_VERSIONS=True)
```

- Additionally, the `qbittorrentapi.Version` class can be used for manual introspection of the versions.

```
Version.is_app_version_supported(qbt_client.app.version)
```

## Disable Logging Debug Output

- Instantiate Client with `DISABLE_LOGGING_DEBUG_OUTPUT=True` or manually disable logging for the relevant packages:

```
logging.getLogger('qbittorrentapi').setLevel(logging.INFO)
logging.getLogger('requests').setLevel(logging.INFO)
logging.getLogger('urllib3').setLevel(logging.INFO)
```

## 1.4.3 Performance

By default, complex objects are returned from some endpoints. These objects allow for accessing the response’s items as attributes and include methods for contextually relevant actions (such as `start()` and `stop()` for a torrent, for example).

This comes at the cost of performance, though. Generally, this cost isn’t large; however, some endpoints, such as `torrents_files()`, may need to convert a large payload and the cost can be significant.

This client can be configured to always return only the simple JSON if desired. Simply set `SIMPLE_RESPONSES=True` when instantiating the client.

```
qbt_client = qbittorrentapi.Client(
    host='localhost:8080',
    username='admin',
    password='adminadmin',
    SIMPLE_RESPONSES=True,
)
```

Alternatively, `SIMPLE_RESPONSES` can be set to `True` to return the simple JSON only for an individual method call.

```
qbt_client.torrents.files(torrent_hash='...', SIMPLE_RESPONSES=True)
```

### 1.4.4 Exceptions

**exception** qbittorrentapi.exceptions.**APIError**

Bases: [Exception](#)

Base error for all exceptions from this Client.

**exception** qbittorrentapi.exceptions.**UnsupportedQbittorrentVersion**

Bases: [APIError](#)

Connected qBittorrent is not fully supported by this Client.

**exception** qbittorrentapi.exceptions.**FileError**

Bases: [OSError](#), [APIError](#)

Base class for all exceptions for file handling.

**exception** qbittorrentapi.exceptions.**TorrentFileError**

Bases: [FileError](#)

Base class for all exceptions for torrent files.

**exception** qbittorrentapi.exceptions.**TorrentFileNotFoundError**

Bases: [TorrentFileError](#)

Specified torrent file does not appear to exist.

**exception** qbittorrentapi.exceptions.**TorrentFilePermissionError**

Bases: [TorrentFileError](#)

Permission was denied to read the specified torrent file.

**exception** qbittorrentapi.exceptions.**APIConnectionError**(\*args, \*\*kwargs)

Bases: [RequestException](#), [APIError](#)

Base class for all communications errors including HTTP errors.

**exception** qbittorrentapi.exceptions.**LoginFailed**(\*args, \*\*kwargs)

Bases: [APIConnectionError](#)

This can technically be raised with any request since log in may be attempted for any request and could fail.

**exception** qbittorrentapi.exceptions.**HTTPError**(\*args, \*\*kwargs)

Bases: [HTTPError](#), [APIConnectionError](#)

Base error for all HTTP errors.

All errors following a successful connection to qBittorrent are returned as HTTP statuses.

**http\_status\_code** = None

**exception** qbittorrentapi.exceptions.**HTTP4XXError**(\*args, \*\*kwargs)

Bases: [HTTPError](#)

Base error for all HTTP 4XX statuses.

**exception** qbittorrentapi.exceptions.**HTTP5XXError**(\*args, \*\*kwargs)

Bases: [HTTPError](#)

Base error for all HTTP 5XX statuses.

**exception** qbittorrentapi.exceptions.**HTTP400Error**(\*args, \*\*kwargs)

Bases: [HTTP4XXError](#)

HTTP 400 Status.

**http\_status\_code** = 400

**exception** qbittorrentapi.exceptions.**HTTP401Error**(\*args, \*\*kwargs)

Bases: [HTTP4XXError](#)

HTTP 401 Status.

**http\_status\_code** = 401

**exception** qbittorrentapi.exceptions.**HTTP403Error**(\*args, \*\*kwargs)

Bases: [HTTP4XXError](#)

HTTP 403 Status.

**http\_status\_code** = 403

**exception** qbittorrentapi.exceptions.**HTTP404Error**(\*args, \*\*kwargs)

Bases: [HTTP4XXError](#)

HTTP 404 Status.

**http\_status\_code** = 404

**exception** qbittorrentapi.exceptions.**HTTP405Error**(\*args, \*\*kwargs)

Bases: [HTTP4XXError](#)

HTTP 405 Status.

**http\_status\_code** = 405

**exception** qbittorrentapi.exceptions.**HTTP409Error**(\*args, \*\*kwargs)

Bases: [HTTP4XXError](#)

HTTP 409 Status.

**http\_status\_code** = 409

**exception** qbittorrentapi.exceptions.**HTTP415Error**(\*args, \*\*kwargs)

Bases: [HTTP4XXError](#)

HTTP 415 Status.

**http\_status\_code** = 415

**exception** qbittorrentapi.exceptions.**HTTP500Error**(\*args, \*\*kwargs)

Bases: [HTTP5XXError](#)

HTTP 500 Status.

**http\_status\_code** = 500

**exception** qbittorrentapi.exceptions.**MissingRequiredParameters400Error**(\*args, \*\*kwargs)

Bases: [HTTP400Error](#)

Endpoint call is missing one or more required parameters.

**exception** qbittorrentapi.exceptions.InvalidRequest400Error(\*args, \*\*kwargs)

Bases: [HTTP400Error](#)

One or more endpoint arguments are malformed.

**exception** qbittorrentapi.exceptions.Unauthorized401Error(\*args, \*\*kwargs)

Bases: [HTTP401Error](#)

Primarily reserved for XSS and host header issues.

**exception** qbittorrentapi.exceptions.Forbidden403Error(\*args, \*\*kwargs)

Bases: [HTTP403Error](#)

Not logged in, IP has been banned, or calling an API method that isn't public.

**exception** qbittorrentapi.exceptions.NotFound404Error(\*args, \*\*kwargs)

Bases: [HTTP404Error](#)

This should mean qBittorrent couldn't find a torrent for the torrent hash.

**exception** qbittorrentapi.exceptions.MethodNotAllowed405Error(\*args, \*\*kwargs)

Bases: [HTTP405Error](#)

HTTP method is not supported for the API endpoint.

**exception** qbittorrentapi.exceptions.Conflict409Error(\*args, \*\*kwargs)

Bases: [HTTP409Error](#)

Returned if arguments don't make sense specific to the endpoint.

**exception** qbittorrentapi.exceptions.UnsupportedMediaType415Error(\*args, \*\*kwargs)

Bases: [HTTP415Error](#)

torrents/add endpoint will return this for invalid URL(s) or files.

**exception** qbittorrentapi.exceptions.InternalServerError500Error(\*args, \*\*kwargs)

Bases: [HTTP500Error](#)

Returned if qBittorrent errors internally while processing the request.

## 1.4.5 API Reference

### Application

**class** qbittorrentapi.app.AppAPIMixin(host="", port=None, username=None, password=None, \*\*kwargs)

Bases: [AuthAPIMixin](#)

Implementation of all Application API methods.

#### Usage

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> client.app_version()
>>> client.app_preferences()
```

**app\_build\_info(\*\*kwargs)**

Retrieve build info.

**Returns**

*BuildInfoDictionary* - [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#get-build-info](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-build-info)

**app\_default\_save\_path(\*\*kwargs)**

Retrieves the default path for where torrents are saved.

**Returns**

string

**app\_preferences(\*\*kwargs)**

Retrieve qBittorrent application preferences.

**Returns**

*ApplicationPreferencesDictionary* - [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#get-application-preferences](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-application-preferences)

**app\_set\_preferences(prefs=None, \*\*kwargs)**

Set one or more preferences in qBittorrent application.

**Parameters**

**prefs** – dictionary of preferences to set

**Returns**

None

**app\_shutdown(\*\*kwargs)**

Shutdown qBittorrent.

**app\_version(\*\*kwargs)**

Retrieve application version.

**Returns**

string

**app\_web\_api\_version(\*\*kwargs)**

Retrieve web API version.

**Returns**

string

**class qbittorrentapi.app.Application(\*args, \*\*kwargs)**

Allows interaction with Application API endpoints.

**Usage**

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> # these are all the same attributes that are available as named in_
↳ the
>>> # endpoints or the more pythonic names in Client (with or without
↳ 'app_' prepended)
>>> webapiVersion = client.application.webapiVersion
>>> web_api_version = client.application.web_api_version
>>> app_web_api_version = client.application.web_api_version
```

(continues on next page)



(continued from previous page)

```

>>> # access and set preferences as attributes
>>> is_dht_enabled = client.application.preferences.dht
>>> # supports sending a just subset of preferences to update
>>> client.application.preferences = dict(dht=(not is_dht_enabled))
>>> prefs = client.application.preferences
>>> prefs['web_ui_clickjacking_protection_enabled'] = True
>>> client.app.preferences = prefs
>>>
>>> client.application.shutdown()

```

**property build\_info**Implements `app_build_info()`**property default\_save\_path**Implements `app_default_save_path()`**property preferences**Implements `app_preferences()` and `app_set_preferences()`**set\_preferences**(prefs=None, \*\*kwargs)Implements `app_set_preferences()`**shutdown()**Implements `app_shutdown()`**property version**Implements `app_version()`**property web\_api\_version**Implements `app_web_api_version()`**class** qbittorrentapi.app.ApplicationPreferencesDictionary(data=None, client=None)Bases: `Dictionary`Response for `app_preferences()`**class** qbittorrentapi.app.BuildInfoDictionary(data=None, client=None)Bases: `Dictionary`Response for `app_build_info()`**AttrDict (internal)**

Copyright (c) 2013 Brendan Curran-Johnson

**class** qbittorrentapi.\_attrdict.AttrDict(\*args, \*\*kwargs)Bases: `dict`, `MutableAttr`A dict that implements `MutableAttr`.**class** qbittorrentapi.\_attrdict.MutableAttrBases: `Attr`, `MutableMapping`

A mixin class for a mapping that allows for attribute-style access of values.

**class** qbittorrentapi.\_attrdict.**Attr**

Bases: [Mapping](#)

A mixin class for a mapping that allows for attribute-style access of values.

A key may be used as an attribute if:

- It is a string
- It matches `^[A-Za-z][A-Za-z0-9_]*$` (i.e., a public attribute)
- The key doesn't overlap with any class attributes (for `Attr`, those would be `get`, `items`, `keys`, `values`, `mro`, and `register`).

If a value which is accessed as an attribute is a Sequence-type (and is not a string/bytes), it will be converted to a `_sequence_type` with any mappings within it converted to `Attrs`.

**NOTE:**

This means that if `_sequence_type` is not `None`, then a sequence accessed as an attribute will be a different object than if accessed as an attribute than if it is accessed as an item.

## Authentication

**class** qbittorrentapi.auth.**AuthAPIMixin**(*host=""*, *port=None*, *username=None*, *password=None*,  
\*\**kwargs*)

Bases: [Request](#)

Implementation of all Authorization API methods.

**Usage**

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↪ 'adminadmin')
>>> _ = client.is_logged_in
>>> client.auth_log_in(username='admin', password='adminadmin')
>>> client.auth_log_out()
```

**auth\_log\_in**(*username=None*, *password=None*, \*\**kwargs*)

Log in to qBittorrent host.

**Raises**

- **LoginFailed** – if credentials failed to log in
- **Forbidden403Error** – if user is banned...or not logged in

**Parameters**

- **username** – username for qBittorrent client
- **password** – password for qBittorrent client

**Returns**

`None`

**auth\_log\_out**(\*\**kwargs*)

End session with qBittorrent.

**property is\_logged\_in**

Returns True if low-overhead API call succeeds; False otherwise.

There isn't a reliable way to know if an existing session is still valid without attempting to use it. qBittorrent invalidates cookies when they expire.

**Returns**

True/False if current auth cookie is accepted by qBittorrent.

**class qbittorrentapi.auth.Authorization(\*args, \*\*kwargs)**

Allows interaction with the Authorization API endpoints.

**Usage**

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> is_logged_in = client.auth.is_logged_in
>>> client.auth.log_in(username='admin', password='adminadmin')
>>> client.auth.log_out()
```

**property is\_logged\_in**

Implements [is\\_logged\\_in\(\)](#)

**log\_in**(username=None, password=None, \*\*kwargs)

Implements [auth\\_log\\_in\(\)](#)

**log\_out**(\*\*kwargs)

Implements [auth\\_log\\_out\(\)](#)

**Client**

**class qbittorrentapi.client.Client**(host="", port=None, username=None, password=None, EXTRA\_HEADERS=None, REQUESTS\_ARGS=None, VERIFY\_WEBUI\_CERTIFICATE=True, FORCE\_SCHEME\_FROM\_HOST=False, RAISE\_NOTIMPLEMENTEDERROR\_FOR\_UNIMPLEMENTED\_API\_ENDPOINTS=False, RAISE\_ERROR\_FOR\_UNSUPPORTED\_QBITTORRENT\_VERSIONS=False, VERBOSE\_RESPONSE\_LOGGING=False, SIMPLE\_RESPONSES=False, DISABLE\_LOGGING\_DEBUG\_OUTPUT=False, \*\*kwargs)

Bases: [LogAPIMixin](#), [SyncAPIMixin](#), [TransferAPIMixin](#), [TorrentsAPIMixin](#), [RSSAPIMixin](#), [SearchAPIMixin](#)

Initialize API for qBittorrent client.

Host must be specified. Username and password can be specified at login. A call to [auth\\_log\\_in\(\)](#) is not explicitly required if username and password are provided during Client construction.

**Usage**

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> torrents = client.torrents_info()
```

**Parameters**

- **host** – hostname for qBittorrent Web API (e.g. `[http[s]://]localhost[:8080]`)
- **port** – port number for qBittorrent Web API (note: only used if host does not contain a port)
- **username** – username for qBittorrent client
- **password** – password for qBittorrent client
- **SIMPLE\_RESPONSES** – By default, complex objects are returned from some endpoints. These objects will allow for accessing responses' items as attributes and include methods for contextually relevant actions. This comes at the cost of performance. Generally, this cost isn't large; however, some endpoints, such as `torrents_files()` method, may need to convert a large payload. Set this to `True` to return the simple JSON back. Alternatively, set this to `True` only for an individual method call. For instance, when requesting the files for a torrent: `client.torrents_files(hash='...', SIMPLE_RESPONSES=True)`.
- **VERIFY\_WEBUI\_CERTIFICATE** – Set to `False` to skip verify certificate for HTTPS connections; for instance, if the connection is using a self-signed certificate. Not setting this to `False` for self-signed certs will cause a `qbittorrentapi.exceptions.APIConnectionError` exception to be raised.
- **EXTRA\_HEADERS** – Dictionary of HTTP Headers to include in all requests made to qBittorrent.
- **REQUESTS\_ARGS** – Dictionary of configuration for Requests package: <https://requests.readthedocs.io/en/latest/api/#requests.request>
- **FORCE\_SCHEME\_FROM\_HOST** – If a scheme (i.e. `http` or `https`) is specified in host, it will be used regardless of whether qBittorrent is configured for HTTP or HTTPS communication. Normally, this client will attempt to determine which scheme qBittorrent is actually listening on... but this can cause problems in rare cases.
- **RAISE\_NOTIMPLEMENTEDERROR\_FOR\_UNIMPLEMENTED\_API\_ENDPOINTS** – Some Endpoints may not be implemented in older versions of qBittorrent. Setting this to `True` will raise a `NotImplementedError` instead of just returning `None`.
- **RAISE\_ERROR\_FOR\_UNSUPPORTED\_QBITTORRENT\_VERSIONS** – raise the `UnsupportedQbittorrentVersion` exception if the connected version of qBittorrent is not fully supported by this client.
- **DISABLE\_LOGGING\_DEBUG\_OUTPUT** – Turn off debug output from logging for this package as well as Requests & urllib3.

## Definitions

```
class qbittorrentapi.definitions.APINames(value, names=None, *, module=None, qualname=None,
                                         type=None, start=1, boundary=None)
```

Bases: `Enum`

API namespaces for API endpoints.

e.g torrents in `http://localhost:8080/api/v2/torrents/addTrackers`

**Application** = 'app'

**Authorization** = 'auth'

**EMPTY** = ''

**Log** = 'log'

RSS = 'rss'

Search = 'search'

Sync = 'sync'

Torrents = 'torrents'

Transfer = 'transfer'

**class** qbittorrentapi.definitions.**ClientCache**(\*args, \*\*kwargs)

Bases: `object`

Caches the client.

Subclass this for any object that needs access to the Client.

**class** qbittorrentapi.definitions.**Dictionary**(data=None, client=None)

Bases: `ClientCache`, `AttrDict`

Base definition of dictionary-like objects returned from qBittorrent.

**class** qbittorrentapi.definitions.**List**(list\_entries=None, entry\_class=None, client=None)

Bases: `ClientCache`, `UserList`

Base definition for list-like objects returned from qBittorrent.

**class** qbittorrentapi.definitions.**ListEntry**(data=None, client=None)

Bases: `Dictionary`

Base definition for objects within a list returned from qBittorrent.

**class** qbittorrentapi.definitions.**TorrentState**(value, names=None, \*, module=None, qualname=None, type=None, start=1, boundary=None)

Bases: `Enum`

Torrent States as defined by qBittorrent.

#### Definitions:

- wiki: [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#get-torrent-list](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-torrent-list)
- code: <https://github.com/qbittorrent/qBittorrent/blob/5dcc14153f046209f1067299494a82e5294d883a/src/base/bittorrent/torrent.h#L73>

#### Usage

```
>>> from qbittorrentapi import Client, TorrentStates
>>> client = Client()
>>> # print torrent hashes for torrents that are downloading
>>> for torrent in client.torrents_info():
>>>     # check if torrent is downloading
>>>     if torrent.state_enum.is_downloading:
>>>         print(f'{torrent.hash} is downloading...')
>>>     # the appropriate enum member can be directly derived
>>>     state_enum = TorrentStates(torrent.state)
>>>     print(f'{torrent.hash}: {state_enum.value}')
```

ALLOCATING = 'allocating'

```
CHECKING_DOWNLOAD = 'checkingDL'
CHECKING_RESUME_DATA = 'checkingResumeData'
CHECKING_UPLOAD = 'checkingUP'
DOWNLOADING = 'downloading'
ERROR = 'error'
FORCED_DOWNLOAD = 'forcedDL'
FORCED_METADATA_DOWNLOAD = 'forcedMetaDL'
FORCED_UPLOAD = 'forcedUP'
METADATA_DOWNLOAD = 'metaDL'
MISSING_FILES = 'missingFiles'
MOVING = 'moving'
PAUSED_DOWNLOAD = 'pausedDL'
PAUSED_UPLOAD = 'pausedUP'
QUEUED_DOWNLOAD = 'queuedDL'
QUEUED_UPLOAD = 'queuedUP'
STALLED_DOWNLOAD = 'stalledDL'
STALLED_UPLOAD = 'stalledUP'
UNKNOWN = 'unknown'
UPLOADING = 'uploading'
```

**property is\_checking**

Returns True if the State is categorized as Checking.

**property is\_complete**

Returns True if the State is categorized as Complete.

**property is\_downloading**

Returns True if the State is categorized as Downloading.

**property is\_errored**

Returns True if the State is categorized as Errored.

**property is\_paused**

Returns True if the State is categorized as Paused.

**property is\_uploading**

Returns True if the State is categorized as Uploading.

```
class qbittorrentapi.definitions.TrackerStatus(value, names=None, *, module=None,
                                              qualname=None, type=None, start=1,
                                              boundary=None)
```

Bases: [Enum](#)

Tracker Statuses as defined by qBittorrent.

#### Definitions:

- wiki: [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#get-torrent-trackers](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-torrent-trackers)
- code: <https://github.com/qbittorrent/qBittorrent/blob/5dcc14153f046209f1067299494a82e5294d883a/src/base/bittorrent/trackerentry.h#L42>

#### Usage

```
>>> from qbittorrentapi import Client, TrackerStatus
>>> client = Client()
>>> # print torrent hashes for torrents that are downloading
>>> for torrent in client.torrents_info():
>>>     for tracker in torrent.trackers:
>>>         # display status for each tracker
>>>         print(f"{torrent.hash[-6:]}: {TrackerStatus(tracker.status).
↪display:>13} :{tracker.url}")
```

DISABLED = 0

NOT\_CONTACTED = 1

NOT\_WORKING = 4

UPDATING = 3

WORKING = 2

property display

Returns a descriptive display value for status.

## Log

```
class qbittorrentapi.log.LogAPIMixin(host="", port=None, username=None, password=None, **kwargs)
```

Bases: [AppAPIMixin](#)

Implementation of all Log API methods.

#### Usage

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↪'adminadmin')
>>> client.log_main(info=False)
>>> client.log_peers()
```

**log\_main**(*normal=None, info=None, warning=None, critical=None, last\_known\_id=None, \*\*kwargs*)

Retrieve the qBittorrent log entries. Iterate over returned object.

**Parameters**

- **normal** – False to exclude normal entries
- **info** – False to exclude info entries
- **warning** – False to exclude warning entries
- **critical** – False to exclude critical entries
- **last\_known\_id** – only entries with an ID greater than this value will be returned

**Returns**

*LogMainList*

**log\_peers**(*last\_known\_id=None, \*\*kwargs*)

Retrieve qBittorrent peer log.

**Parameters**

**last\_known\_id** – only entries with an ID greater than this value will be returned

**Returns**

*LogPeersList*

**class** qbittorrentapi.log.**Log**(*client*)

Allows interaction with Log API endpoints.

**Usage**

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> # this is all the same attributes that are available as named in the
>>> # endpoints or the more pythonic names in Client (with or without
↳ 'log_' prepended)
>>> log_list = client.log.main()
>>> peers_list = client.log.peers(hash='...')
>>> # can also filter log down with additional attributes
>>> log_info = client.log.main.info(last_known_id='...')
>>> log_warning = client.log.main.warning(last_known_id='...')
```

**peers**(*last\_known\_id=None, \*\*kwargs*)

Implements *log\_peers()*

**class** qbittorrentapi.log.**LogPeersList**(*list\_entries, client*)

Bases: *List*

Response for *log\_peers()*

**class** qbittorrentapi.log.**LogPeer**(*data=None, client=None*)

Bases: *ListEntry*

Item in *LogPeersList*

**class** qbittorrentapi.log.**LogMainList**(*list\_entries, client*)

Bases: *List*

Response to *log\_main()*



**class** qbittorrentapi.log.**LogEntry**(data=None, client=None)

Bases: [ListEntry](#)

Item in [LogMainList](#)

## Request (internal)

**class** qbittorrentapi.request.**Request**(host="", port=None, username=None, password=None, \*\*kwargs)

Bases: [object](#)

Facilitates HTTP requests to qBittorrent's Web API.

**\_cast**(response, response\_class, \*\*response\_kwargs)

Returns the API response casted to the requested class.

### Parameters

- **response** – requests Response from API
- **response\_class** – class to return response as; if none, response is returned
- **response\_kwargs** – request-specific configuration for response

### Returns

API response as type of response\_class

**\_get**(\_name=APINames.EMPTY, \_method="", requests\_args=None, requests\_params=None, headers=None, params=None, data=None, files=None, response\_class=None, \*\*kwargs)

Send GET request.

### Parameters

- **api\_namespace** – the namespace for the API endpoint (e.g. [APINames](#) or torrents)
- **api\_method** – the name for the API endpoint (e.g. add)
- **kwargs** – see [\\_request\(\)](#)

### Returns

Requests [Response](#)

**static** **\_get\_data**(http\_method, params=None, data=None, files=None, \*\*kwargs)

Determine data, params, and files for the Requests call.

### Parameters

- **http\_method** – get or post
- **params** – key value pairs to send with GET calls
- **data** – key value pairs to send with POST calls
- **files** – dictionary of files to send with request

### Returns

final dictionaries of data to send to qBittorrent

**static** **\_get\_headers**(headers=None, more\_headers=None)

Determine headers specific to this request. Request headers can be specified explicitly or with the requests kwargs. Headers specified in self.\_EXTRA\_HEADERS are merged in Requests itself.

### Parameters

- **headers** – headers specified for this specific request

- **more\_headers** – headers from requests\_kwargs config

**Returns**

final dictionary of headers for this specific request

**\_get\_requests\_kwargs**(*requests\_args=None, requests\_params=None*)

Determine the requests\_kwargs for the call to Requests. The global configuration in self.\_REQUESTS\_ARGS is updated by any arguments provided for a specific call.

**Parameters**

- **requests\_args** – default location to expect Requests requests\_kwargs
- **requests\_params** – alternative location to expect Requests requests\_kwargs

**Returns**

final dictionary of Requests requests\_kwargs

**static \_get\_response\_kwargs**(*kwargs*)

Determine the kwargs for managing the response to return.

**Parameters**

**kwargs** – extra keywords arguments to be passed along in request

**Returns**

sanitized arguments

**static \_handle\_error\_responses**(*data, params, response*)

Raise proper exception if qBittorrent returns Error HTTP Status.

**\_initialize\_context**()

Initialize and/or reset communications context with qBittorrent.

This is necessary on startup or when the auth cookie needs to be replaced...perhaps because it expired, qBittorrent was restarted, significant settings changes, etc.

**\_initialize\_lessor**(*EXTRA\_HEADERS=None, REQUESTS\_ARGS=None, VERIFY\_WEBUI\_CERTIFICATE=True, FORCE\_SCHEME\_FROM\_HOST=False, RAISE\_UNIMPLEMENTEDERROR\_FOR\_UNIMPLEMENTED\_API\_ENDPOINTS=False, RAISE\_NOTIMPLEMENTEDERROR\_FOR\_UNIMPLEMENTED\_API\_ENDPOINTS=False, RAISE\_ERROR\_FOR\_UNSUPPORTED\_QBITTORRENT\_VERSIONS=False, VERBOSE\_RESPONSE\_LOGGING=False, PRINT\_STACK\_FOR\_EACH\_REQUEST=False, SIMPLE\_RESPONSES=False, DISABLE\_LOGGING\_DEBUG\_OUTPUT=False, MOCK\_WEB\_API\_VERSION=None*)

Initialize lesser used configuration.

**classmethod \_list2string**(*input\_list=None, delimiter='|'*)

Convert entries in a list to a concatenated string.

**Parameters**

- **input\_list** – list to convert
- **delimiter** – delimiter for concatenation

**Returns**

if input is a list, concatenated string...else whatever the input was

**\_post**(*\_name=APINames.EMPTY, \_method="", requests\_args=None, requests\_params=None, headers=None, params=None, data=None, files=None, response\_class=None, \*\*kwargs*)

Send POST request.

**Parameters**

- **api\_namespace** – the namespace for the API endpoint (e.g. [APINames](#) or torrents)
- **api\_method** – the name for the API endpoint (e.g. add)
- **kwargs** – see [\\_request\(\)](#)

**Returns**Requests [Response](#)

**\_request**(*http\_method, api\_namespace, api\_method, requests\_args=None, requests\_params=None, headers=None, params=None, data=None, files=None, response\_class=None, \*\*kwargs*)

Meat and potatoes of sending requests to qBittorrent.

**Parameters**

- **http\_method** – get or post
- **api\_namespace** – the namespace for the API endpoint (e.g. [APINames](#) or torrents)
- **api\_method** – the name for the API endpoint (e.g. add)
- **requests\_args** – default location for Requests kwargs
- **requests\_params** – alternative location for Requests kwargs
- **headers** – HTTP headers to send with the request
- **params** – key/value pairs to send with a GET request
- **data** – key/value pairs to send with a POST request
- **files** – files to be sent with the request
- **response\_class** – class to use to cast the API response
- **kwargs** – arbitrary keyword args to send to qBittorrent with the request

**Returns**Requests [Response](#)

**\_request\_manager**(*http\_method, api\_namespace, api\_method, \_retries=1, \_retry\_backoff\_factor=0.3, requests\_args=None, requests\_params=None, headers=None, params=None, data=None, files=None, response\_class=None, \*\*kwargs*)

Wrapper to manage request retries and severe exceptions.

This should retry at least once to account for the Web API switching from HTTP to HTTPS. During the second attempt, the URL is rebuilt using HTTP or HTTPS as appropriate.

**property \_session**

Create or return existing HTTP session.

**Returns**Requests [Session](#) object**\_trigger\_session\_initialization()**

Effectively resets the HTTP session by removing the reference to it.

During the next request, a new session will be created.

**\_verbose\_logging**(*http\_method, url, data, params, requests\_kwargs, response*)

Log verbose information about request.

Can be useful during development.

**class** qbittorrentapi.request.URL(*client*)

Bases: `object`

Management for the qBittorrent Web API URL.

**build\_base\_url**(*headers, requests\_kwargs=None*)

Determine the Base URL for the Web API endpoints.

A URL is only actually built here if it's the first time here or the context was re-initialized. Otherwise, the most recently built URL is used.

If the user doesn't provide a scheme for the URL, it will try HTTP first and fall back to HTTPS if that doesn't work. While this is probably backwards, qBittorrent or an intervening proxy can simply redirect to HTTPS and that'll be respected.

Additionally, if users want to augment the path to the API endpoints, any path provided here will be preserved in the returned Base URL and prefixed to all subsequent API calls.

#### Parameters

- **headers** – HTTP headers for request
- **requests\_kwargs** – additional params from user for HTTP HEAD request

#### Returns

base URL as a string for Web API endpoint

**build\_url**(*api\_namespace, api\_method, headers, requests\_kwargs*)

Create a fully qualified URL for the API endpoint.

#### Parameters

- **api\_namespace** – the namespace for the API endpoint (e.g. `torrents`)
- **api\_method** – the specific method for the API endpoint (e.g. `info`)
- **headers** – HTTP headers for request
- **requests\_kwargs** – kwargs for any calls to Requests

#### Returns

fully qualified URL string for endpoint

**build\_url\_path**(*api\_namespace, api\_method*)

Determine the full URL path for the API endpoint.

#### Parameters

- **api\_namespace** – the namespace for the API endpoint (e.g. `torrents`)
- **api\_method** – the specific method for the API endpoint (e.g. `info`)

#### Returns

entire URL string for API endpoint (e.g. `http://localhost:8080/api/v2/torrents/info` or `http://example.com/qbt/api/v2/torrents/info`)

**detect\_scheme**(*base\_url, default\_scheme, alt\_scheme, headers, requests\_kwargs*)

Determine if the URL endpoint is using HTTP or HTTPS.

#### Parameters

- **base\_url** – urllib `ParseResult` URL object
- **default\_scheme** – default scheme to use for URL
- **alt\_scheme** – alternative scheme to use for URL if default doesn't work

- **headers** – HTTP headers for request
- **requests\_kwargs** – kwargs for calls to Requests

**Returns**

scheme (ie HTTP or HTTPS)

**RSS**

**class** qbittorrentapi.rss.RSSAPIMixin(*host="", port=None, username=None, password=None, \*\*kwargs*)

Bases: [AppAPIMixin](#)

Implementation of all RSS API methods.

**Usage**

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> rss_rules = client.rss_rules()
>>> client.rss_set_rule(rule_name="...", rule_def={...})
```

**rss\_add\_feed**(*url=None, item\_path=None, \*\*kwargs*)

Add new RSS feed. Folders in path must already exist.

**Raises**

[Conflict409Error](#) –

**Parameters**

- **url** – URL of RSS feed (e.g. <https://distrowatch.com/news/torrents.xml>)
- **item\_path** – Name and/or path for new feed (e.g. Folder\Subfolder\FeedName)

**Returns**

None

**rss\_add\_folder**(*folder\_path=None, \*\*kwargs*)

Add an RSS folder. Any intermediate folders in path must already exist.

**Raises**

[Conflict409Error](#) –

**Parameters**

**folder\_path** – path to new folder (e.g. Linux\ISOs)

**Returns**

None

**rss\_items**(*include\_feed\_data=None, \*\*kwargs*)

Retrieve RSS items and optionally feed data.

**Parameters**

**include\_feed\_data** – True or false to include feed data

**Returns**

[RSSItemsDictionary](#)

**rss\_mark\_as\_read**(*item\_path=None, article\_id=None, \*\*kwargs*)

Mark RSS article as read. If article ID is not provided, the entire feed is marked as read.

**Raises**

*NotFound404Error* –

**Parameters**

- **item\_path** – path to item to be refreshed (e.g. Folder\Subfolder\ItemName)
- **article\_id** – article ID from *rss\_items()*

**Returns**

None

**rss\_matching\_articles**(*rule\_name=None, \*\*kwargs*)

Fetch all articles matching a rule.

**Parameters**

**rule\_name** – Name of rule to return matching articles

**Returns**

*RSSItemsDictionary*

**rss\_move\_item**(*orig\_item\_path=None, new\_item\_path=None, \*\*kwargs*)

Move/rename an RSS item (folder, feed, etc).

**Raises**

*Conflict409Error* –

**Parameters**

- **orig\_item\_path** – path to item to be removed (e.g. Folder\Subfolder\ItemName)
- **new\_item\_path** – path to item to be removed (e.g. Folder\Subfolder\ItemName)

**Returns**

None

**rss\_refresh\_item**(*item\_path=None, \*\*kwargs*)

Trigger a refresh for an RSS item.

**Parameters**

**item\_path** – path to item to be refreshed (e.g. Folder\Subfolder\ItemName)

**Returns**

None

**rss\_remove\_item**(*item\_path=None, \*\*kwargs*)

Remove an RSS item (folder, feed, etc).

NOTE: Removing a folder also removes everything in it.

**Raises**

*Conflict409Error* –

**Parameters**

**item\_path** – path to item to be removed (e.g. Folder\Subfolder\ItemName)

**Returns**

None

**rss\_remove\_rule**(*rule\_name=None, \*\*kwargs*)

Delete a RSS auto-downloading rule.

**Parameters**

**rule\_name** – Name of rule to delete

**Returns**

None

**rss\_rename\_rule**(*orig\_rule\_name=None, new\_rule\_name=None, \*\*kwargs*)

Rename an RSS auto-download rule.

Note: this endpoint did not work properly until qBittorrent v4.3.0

**Parameters**

- **orig\_rule\_name** – current name of rule
- **new\_rule\_name** – new name for rule

**Returns**

None

**rss\_rules**(*\*\*kwargs*)

Retrieve RSS auto-download rule definitions.

**Returns**

*RSSRulesDictionary*

**rss\_setFeedURL**(*url=None, item\_path=None, \*\*kwargs*)

Update the URL for an existing RSS feed.

**Raises**

*Conflict409Error* –

**Parameters**

- **url** – URL of RSS feed (e.g <https://distrowatch.com/news/torrents.xml>)
- **item\_path** – Name and/or path for feed (e.g. Folder\Subfolder\FeedName)

**Returns**

None

**rss\_set\_feed\_url**(*url=None, item\_path=None, \*\*kwargs*)

Update the URL for an existing RSS feed.

**Raises**

*Conflict409Error* –

**Parameters**

- **url** – URL of RSS feed (e.g <https://distrowatch.com/news/torrents.xml>)
- **item\_path** – Name and/or path for feed (e.g. Folder\Subfolder\FeedName)

**Returns**

None

**rss\_set\_rule**(*rule\_name=None, rule\_def=None, \*\*kwargs*)

Create a new RSS auto-downloading rule.

**Parameters**

- **rule\_name** – name for new rule

- **rule\_def** – dictionary with rule fields - [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#set-auto-downloading-rule](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#set-auto-downloading-rule)

**Returns**

None

**class** qbittorrentapi.rss.RSS(*client*)

Allows interaction with RSS API endpoints.

**Usage**

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> # this is all the same attributes that are available as named in the
>>> # endpoints or the more pythonic names in Client (with or without
↳ 'log_' prepended)
>>> rss_rules = client.rss.rules
>>> client.rss.addFolder(folder_path="TPB")
>>> client.rss.addFeed(url='...', item_path="TPB\Top100")
>>> client.rss.remove_item(item_path="TPB") # deletes TPB and Top100
>>> client.rss.set_rule(rule_name="...", rule_def={...})
>>> items = client.rss.items.with_data
>>> items_no_data = client.rss.items.without_data
```

**add\_feed**(*url=None, item\_path=None, \*\*kwargs*)

Implements `rss_add_feed()`

**add\_folder**(*folder\_path=None, \*\*kwargs*)

Implements `rss_add_folder()`

**mark\_as\_read**(*item\_path=None, article\_id=None, \*\*kwargs*)

Implements `rss_mark_as_read()`

**matching\_articles**(*rule\_name=None, \*\*kwargs*)

Implements `rss_matching_articles()`

**move\_item**(*orig\_item\_path=None, new\_item\_path=None, \*\*kwargs*)

Implements `rss_move_item()`

**refresh\_item**(*item\_path=None*)

Implements `rss_refresh_item()`

**remove\_item**(*item\_path=None, \*\*kwargs*)

Implements `rss_remove_item()`

**remove\_rule**(*rule\_name=None, \*\*kwargs*)

Implements `rss_remove_rule()`

**rename\_rule**(*orig\_rule\_name=None, new\_rule\_name=None, \*\*kwargs*)

Implements `rss_rename_rule()`

**property rules**

Implements `rss_rules()`

**setFeedURL**(*url=None, item\_path=None, \*\*kwargs*)

Implements `rss_set_feed_url()`



**set\_feed\_url**(url=None, item\_path=None, \*\*kwargs)

Implements [rss\\_set\\_feed\\_url\(\)](#)

**set\_rule**(rule\_name=None, rule\_def=None, \*\*kwargs)

Implements [rss\\_set\\_rule\(\)](#)

**class** qbittorrentapi.rss.RSSItemsDictionary(data=None, client=None)

Bases: [Dictionary](#)

Response for [rss\\_items\(\)](#)

**class** qbittorrentapi.rss.RSSRulesDictionary(data=None, client=None)

Bases: [Dictionary](#)

Response for [rss\\_rules\(\)](#)

## Search

**class** qbittorrentapi.search.SearchAPIMixin(host="", port=None, username=None, password=None, \*\*kwargs)

Bases: [AppAPIMixin](#)

Implementation for all Search API methods.

### Usage

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> search_job = client.search_start(pattern='Ubuntu', plugins='all',
↳ category='all')
>>> client.search_stop(search_id=search_job.id)
>>> # or
>>> search_job.stop()
>>>
```

**search\_categories**(plugin\_name=None, \*\*kwargs)

Retrieve categories for search.

Note: endpoint was removed in qBittorrent v4.3.0

### Parameters

**plugin\_name** – Limit categories returned by plugin(s) (supports all and enabled)

### Returns

[SearchCategoriesList](#)

**search\_delete**(search\_id=None, \*\*kwargs)

Delete a search job.

### Raises

[NotFound404Error](#) –

### Parameters

**search\_id** – ID of search to delete

### Returns

None

**search\_enable\_plugin**(*plugins=None, enable=None, \*\*kwargs*)

Enable or disable search plugin(s).

**Parameters**

- **plugins** – list of plugin names
- **enable** – True or False

**Returns**

None

**search\_install\_plugin**(*sources=None, \*\*kwargs*)

Install search plugins from either URL or file.

**Parameters**

**sources** – list of URLs or filepaths

**Returns**

None

**search\_plugins**(*\*\*kwargs*)

Retrieve details of search plugins.

**Returns**

[SearchPluginsList](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-search-plugins) - [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#get-search-plugins](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-search-plugins)

**search\_results**(*search\_id=None, limit=None, offset=None, \*\*kwargs*)

Retrieve the results for the search.

**Raises**

- [NotFound404Error](#) –
- [Conflict409Error](#) –

**Parameters**

- **search\_id** – ID of search job
- **limit** – number of results to return
- **offset** – where to start returning results

**Returns**

[SearchResultsDictionary](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-search-results) - [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#get-search-results](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-search-results)

**search\_start**(*pattern=None, plugins=None, category=None, \*\*kwargs*)

Start a search. Python must be installed. Host may limit number of concurrent searches.

**Raises**

[Conflict409Error](#) –

**Parameters**

- **pattern** – term to search for
- **plugins** – list of plugins to use for searching (supports ‘all’ and ‘enabled’)
- **category** – categories to limit search; dependent on plugins. (supports ‘all’)

**Returns**

[SearchJobDictionary](#)

**search\_status**(search\_id=None, \*\*kwargs)

Retrieve status of one or all searches.

**Raises**

*NotFound404Error* –

**Parameters**

**search\_id** – ID of search to get status; leave empty for status of all jobs

**Returns**

*SearchStatusesList* – [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#get-search-status](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-search-status)

**search\_stop**(search\_id=None, \*\*kwargs)

Stop a running search.

**Raises**

*NotFound404Error* –

**Parameters**

**search\_id** – ID of search job to stop

**Returns**

None

**search\_uninstall\_plugin**(names=None, \*\*kwargs)

Uninstall search plugins.

**Parameters**

**names** – names of plugins to uninstall

**Returns**

None

**search\_update\_plugins**(\*\*kwargs)

Auto update search plugins.

**Returns**

None

**class** qbittorrentapi.search.**Search**(\*args, \*\*kwargs)

Allows interaction with Search API endpoints.

**Usage**

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> # this is all the same attributes that are available as named in the
>>> # endpoints or the more pythonic names in Client (with or without
↳ 'search_' prepended)
>>> # initiate searches and retrieve results
>>> search_job = client.search.start(pattern='Ubuntu', plugins='all',
↳ category='all')
>>> status = search_job.status()
>>> results = search_job.result()
>>> search_job.delete()
>>> # inspect and manage plugins
>>> plugins = client.search.plugins
```

(continues on next page)

(continued from previous page)

```
>>> cats = client.search.categories(plugin_name='...')
>>> client.search.install_plugin(sources='...')
>>> client.search.update_plugins()
```

**categories**(*plugin\_name=None, \*\*kwargs*)

Implements [search\\_categories\(\)](#)

**delete**(*search\_id=None, \*\*kwargs*)

Implements [search\\_delete\(\)](#)

**enable\_plugin**(*plugins=None, enable=None, \*\*kwargs*)

Implements [search\\_enable\\_plugin\(\)](#)

**install\_plugin**(*sources=None, \*\*kwargs*)

Implements [search\\_install\\_plugin\(\)](#)

**property plugins**

Implements [search\\_plugins\(\)](#)

**results**(*search\_id=None, limit=None, offset=None, \*\*kwargs*)

Implements [search\\_results\(\)](#)

**start**(*pattern=None, plugins=None, category=None, \*\*kwargs*)

Implements [search\\_start\(\)](#)

**status**(*search\_id=None, \*\*kwargs*)

Implements [search\\_status\(\)](#)

**stop**(*search\_id=None, \*\*kwargs*)

Implements [search\\_stop\(\)](#)

**uninstall\_plugin**(*sources=None, \*\*kwargs*)

Implements [search\\_uninstall\\_plugin\(\)](#)

**update\_plugins**(*\*\*kwargs*)

Implements [search\\_update\\_plugins\(\)](#)

**class** qbittorrentapi.search.SearchJobDictionary(*data, client*)

Bases: [Dictionary](#)

Response for [search\\_start\(\)](#)

**delete**(*\*\*kwargs*)

Implements [search\\_delete\(\)](#)

**results**(*limit=None, offset=None, \*\*kwargs*)

Implements [search\\_results\(\)](#)

**status**(*\*\*kwargs*)

Implements [search\\_status\(\)](#)

**stop**(*\*\*kwargs*)

Implements [search\\_stop\(\)](#)

**class** qbittorrentapi.search.SearchResultsDictionary(*data=None, client=None*)

Bases: [Dictionary](#)

Response for [search\\_results\(\)](#)

```
class qbittorrentapi.search.SearchStatusesList(list_entries, client)
```

Bases: *List*

Response for *search\_status()*

```
class qbittorrentapi.search.SearchStatus(data=None, client=None)
```

Bases: *ListEntry*

Item in *SearchStatusesList*

```
class qbittorrentapi.search.SearchCategoriesList(list_entries, client)
```

Bases: *List*

Response for *search\_categories()*

```
class qbittorrentapi.search.SearchCategory(data=None, client=None)
```

Bases: *ListEntry*

Item in *SearchCategoriesList*

```
class qbittorrentapi.search.SearchPluginsList(list_entries, client)
```

Bases: *List*

Response for *search\_plugins()*

```
class qbittorrentapi.search.SearchPlugin(data=None, client=None)
```

Bases: *ListEntry*

Item in *SearchPluginsList*

## Sync

```
class qbittorrentapi.sync.SyncAPIMixin(host="", port=None, username=None, password=None,
                                       **kwargs)
```

Bases: *AppAPIMixin*

Implementation of all Sync API Methods.

### Usage

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> maindata = client.sync_maindata(rid="...")
>>> torrent_peers = client.sync_torrent_peers(torrent_hash="...", rid=
↳ '...')
```

```
sync_maindata(rid=0, **kwargs)
```

Retrieves sync data.

### Parameters

**rid** – response ID

### Returns

*SyncMainDataDictionary* - [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#get-main-data](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-main-data)

**sync\_torrent\_peers**(torrent\_hash=None, rid=0, \*\*kwargs)

Retrieves torrent sync data.

**Raises**

*NotFound404Error* –

**Parameters**

- **torrent\_hash** – hash for torrent
- **rid** – response ID

**Returns**

*SyncTorrentPeersDictionary* – [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#get-torrent-peers-data](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-torrent-peers-data)

**class** qbittorrentapi.sync.Sync(client)

Allows interaction with the Sync API endpoints.

**Usage:**

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> # these are all the same attributes that are available as named in the
>>> # endpoints or the more pythonic names in Client (with or without 'sync_'
↳ prepended)
>>> maindata = client.sync.maindata(rid="...")
>>> # for use when continuously calling maindata for changes in torrents
>>> # this will automatically request the changes since the last call
>>> md = client.sync.maindata.delta()
>>> #
>>> torrentPeers = client.sync.torrentPeers(hash="...", rid='...')
>>> torrent_peers = client.sync.torrent_peers(hash="...", rid='...')
```

**class** qbittorrentapi.sync.SyncMainDataDictionary(data=None, client=None)

Bases: *Dictionary*

Response for *sync\_maindata()*

**class** qbittorrentapi.sync.SyncTorrentPeersDictionary(data=None, client=None)

Bases: *Dictionary*

Response for *sync\_torrent\_peers()*

## Torrent States

qbittorrentapi.definitions.TorrentStates

alias of *TorrentState*

## Torrents

**class** qbittorrentapi.torrents.**TorrentsAPIMixin**(host="", port=None, username=None, password=None, *\*\*kwargs*)

Bases: [AppAPIMixin](#)

Implementation of all Torrents API methods.

### Usage

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> client.torrents_add(urls='...')
>>> client.torrents_reannounce()
```

**torrents\_add**(urls=None, torrent\_files=None, save\_path=None, cookie=None, category=None, is\_skip\_checking=None, is\_paused=None, is\_root\_folder=None, rename=None, upload\_limit=None, download\_limit=None, use\_auto\_torrent\_management=None, is\_sequential\_download=None, is\_first\_last\_piece\_priority=None, tags=None, content\_layout=None, ratio\_limit=None, seeding\_time\_limit=None, download\_path=None, use\_download\_path=None, stop\_condition=None, *\*\*kwargs*)

Add one or more torrents by URLs and/or torrent files.

### Raises

- [UnsupportedMediaType415Error](#) – if file is not a valid torrent file
- [TorrentFileNotFoundError](#) – if a torrent file doesn't exist
- [TorrentFilePermissionError](#) – if read permission is denied to torrent file

### Parameters

- **urls** – single instance or an iterable of URLs (<http://>, <https://>, magnet: and bc://bt/)
- **torrent\_files** – several options are available to send torrent files to qBittorrent:
  - single instance of bytes: useful if torrent file already read from disk or downloaded from internet.
  - single instance of file handle to torrent file: use `open(<filepath>, 'rb')` to open the torrent file.
  - single instance of a filepath to torrent file: e.g. `'/home/user/torrent_filename.torrent'`
  - an iterable of the single instances above to send more than one torrent file
  - dictionary with key/value pairs of torrent name and single instance of above object Note: The torrent name in a dictionary is useful to identify which torrent file errored. qBittorrent provides back that name in the error text. If a torrent name is not provided, then the name of the file will be used. And in the case of bytes (or if filename cannot be determined), the value `'torrent__n'` will be used
- **save\_path** – location to save the torrent data
- **cookie** – cookie to retrieve torrents by URL
- **category** – category to assign to torrent(s)
- **is\_skip\_checking** – skip hash checking
- **is\_paused** – True to start torrent(s) paused
- **is\_root\_folder** – True or False to create root folder (superseded by `content_layout` with v4.3.2)
- **rename** – new name for torrent(s)

- **upload\_limit** – upload limit in bytes/second
- **download\_limit** – download limit in bytes/second
- **use\_auto\_torrent\_management** – True or False to use automatic torrent management
- **is\_sequential\_download** – True or False for sequential download
- **is\_first\_last\_piece\_priority** – True or False for first and last piece download priority
- **tags** – tag(s) to assign to torrent(s) (added in Web API 2.6.2)
- **content\_layout** – Original, Subfolder, or NoSubfolder to control filesystem structure for content (added in Web API 2.7)
- **ratio\_limit** – share limit as ratio of upload amt over download amt; e.g. 0.5 or 2.0 (added in Web API 2.8.1)
- **seeding\_time\_limit** – number of minutes to seed torrent (added in Web API 2.8.1)
- **download\_path** – location to download torrent content before moving to **save\_path** (added in Web API 2.8.4)
- **use\_download\_path** – True or False whether **download\_path** should be used...defaults to True if **download\_path** is specified (added in Web API 2.8.4)
- **stop\_condition** – MetadataReceived or FilesChecked to stop the torrent when started automatically (added in Web API 2.8.15)

**Returns**

Ok. for success and Fails. for failure

**torrents\_add\_peers**(*peers=None, torrent\_hashes=None, \*\*kwargs*)

Add one or more peers to one or more torrents.

**Raises**

[\*InvalidRequest400Error\*](#) – for invalid peers

**Parameters**

- **peers** – one or more peers to add. each peer should take the form 'host:port'
- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or *all* for all torrents.

**Returns**

[\*TorrentsAddPeersDictionary\*](#) - {<hash>: {'added': #, 'failed': #}}

**torrents\_add\_tags**(*tags=None, torrent\_hashes=None, \*\*kwargs*)

Add one or more tags to one or more torrents.

Note: Tags that do not exist will be created on-the-fly.

**Parameters**

- **tags** – tag name or list of tags
- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or *all* for all torrents.

**Returns**

None

**torrents\_add\_trackers**(*torrent\_hash=None, urls=None, \*\*kwargs*)

Add trackers to a torrent.



**Raises***NotFound404Error* –**Parameters**

- **torrent\_hash** – hash for torrent
- **urls** – tracker urls to add to torrent

**Returns**

None

**torrents\_bottom\_priority**(*torrent\_hashes=None, \*\*kwargs*)

Set torrent as lowest priority. Torrent Queuing must be enabled.

**Raises***Conflict409Error* –**Parameters****torrent\_hashes** – single torrent hash or list of torrent hashes. Or *all* for all torrents.**Returns**

None

**torrents\_categories**(*\*\*kwargs*)

Retrieve all category definitions.

Note: *torrents/categories* is not available until v2.1.0**Returns***TorrentCategoriesDictionary***torrents\_create\_category**(*name=None, save\_path=None, download\_path=None, enable\_download\_path=None, \*\*kwargs*)

Create a new torrent category.

**Raises***Conflict409Error* – if category name is not valid or unable to create**Parameters**

- **name** – name for new category
- **save\_path** – location to save torrents for this category (added in Web API 2.1.0)
- **download\_path** – download location for torrents with this category
- **enable\_download\_path** – True or False to enable or disable download path

**Returns**

None

**torrents\_create\_tags**(*tags=None, \*\*kwargs*)

Create one or more tags.

**Parameters****tags** – tag name or list of tags**Returns**

None

**torrents\_decrease\_priority**(*torrent\_hashes=None, \*\*kwargs*)

Decrease the priority of a torrent. Torrent Queuing must be enabled.

**Raises**

*Conflict409Error* –

**Parameters**

**torrent\_hashes** – single torrent hash or list of torrent hashes. Or `all` for all torrents.

**Returns**

None

**torrents\_delete**(*delete\_files=False, torrent\_hashes=None, \*\*kwargs*)

Remove a torrent from qBittorrent and optionally delete its files.

**Parameters**

- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or `all` for all torrents.
- **delete\_files** – True to delete the torrent's files

**Returns**

None

**torrents\_delete\_tags**(*tags=None, \*\*kwargs*)

Delete one or more tags.

**Parameters**

**tags** – tag name or list of tags

**Returns**

None

**torrents\_download\_limit**(*torrent\_hashes=None, \*\*kwargs*)

Retrieve the download limit for one or more torrents.

**Returns**

*TorrentLimitsDictionary* - {hash: limit} (-1 represents no limit)

**torrents\_edit\_category**(*name=None, save\_path=None, download\_path=None, enable\_download\_path=None, \*\*kwargs*)

Edit an existing category.

Note: `torrents/editCategory` was introduced in Web API 2.1.0

**Raises**

*Conflict409Error* – if category name is not valid or unable to create

**Parameters**

- **name** – category to edit
- **save\_path** – new location to save files for this category
- **download\_path** – download location for torrents with this category
- **enable\_download\_path** – True or False to enable or disable download path

**Returns**

None

**torrents\_edit\_tracker**(*torrent\_hash=None, original\_url=None, new\_url=None, \*\*kwargs*)

Replace a torrent's tracker with a different one.

**Raises**

- *InvalidRequest400Error* –

- *NotFound404Error* –
- *Conflict409Error* –

**Parameters**

- **torrent\_hash** – hash for torrent
- **original\_url** – URL for existing tracker
- **new\_url** – new URL to replace

**Returns**

None

**torrents\_export**(*torrent\_hash=None, \*\*kwargs*)

Export a .torrent file for the torrent.

**Raises**

- *NotFound404Error* – torrent not found
- *Conflict409Error* – unable to export .torrent file

**Parameters****torrent\_hash** – hash for torrent**Returns**

bytes .torrent file

**torrents\_file\_priority**(*torrent\_hash=None, file\_ids=None, priority=None, \*\*kwargs*)

Set priority for one or more files.

**Raises**

- *InvalidRequest400Error* – if priority is invalid or at least one file ID is not an integer
- *NotFound404Error* –
- *Conflict409Error* – if torrent metadata has not finished downloading or at least one file was not found

**Parameters**

- **torrent\_hash** – hash for torrent
- **file\_ids** – single file ID or a list.
- **priority** – priority for file(s) - [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#set-file-priority](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#set-file-priority)

**Returns**

None

**torrents\_files**(*torrent\_hash=None, \*\*kwargs*)

Retrieve individual torrent's files.

**Raises***NotFound404Error* –**Parameters****torrent\_hash** – hash for torrent**Returns***TorrentFilesList* - [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#get-torrent-contents](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-torrent-contents)

**torrents\_increase\_priority**(*torrent\_hashes=None, \*\*kwargs*)

Increase the priority of a torrent. Torrent Queuing must be enabled.

**Raises**

[\*Conflict409Error\*](#) –

**Parameters**

**torrent\_hashes** – single torrent hash or list of torrent hashes. Or `all` for all torrents.

**Returns**

None

**torrents\_info**(*status\_filter=None, category=None, sort=None, reverse=None, limit=None, offset=None, torrent\_hashes=None, tag=None, \*\*kwargs*)

Retrieves list of info for torrents.

**Parameters**

- **status\_filter** – Filter list by torrent status. `all`, `downloading`, `seeding`, `completed`, `paused active`, `inactive`, `resumed`, `errored` Added in Web API 2.4.1: `stalled`, `stalled_uploading`, and `stalled_downloading` Added in Web API 2.8.4: `checking` Added in Web API 2.8.18: `moving`
- **category** – Filter list by category
- **sort** – Sort list by any property returned
- **reverse** – Reverse sorting
- **limit** – Limit length of list
- **offset** – Start of list (if `< 0`, offset from end of list)
- **torrent\_hashes** – Filter list by hash (separate multiple hashes with a `'|'`) (added in Web API 2.0.1)
- **tag** – Filter list by tag (empty string means “untagged”; no “tag” param means “any tag”; added in Web API 2.8.3)

**Returns**

[\*TorrentInfoList\*](#) – [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#get-torrent-list](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-torrent-list)

**torrents\_pause**(*torrent\_hashes=None, \*\*kwargs*)

Pause one or more torrents in qBittorrent.

**Parameters**

**torrent\_hashes** – single torrent hash or list of torrent hashes. Or `all` for all torrents.

**Returns**

None

**torrents\_piece\_hashes**(*torrent\_hash=None, \*\*kwargs*)

Retrieve individual torrent’s pieces’ hashes.

**Raises**

[\*NotFound404Error\*](#) –

**Parameters**

**torrent\_hash** – hash for torrent

**Returns**

[\*TorrentPieceInfoList\*](#)

**torrents\_piece\_states**(*torrent\_hash=None, \*\*kwargs*)

Retrieve individual torrent's pieces' states.

**Raises**

*NotFound404Error* –

**Parameters**

**torrent\_hash** – hash for torrent

**Returns**

*TorrentPieceInfoList*

**torrents\_properties**(*torrent\_hash=None, \*\*kwargs*)

Retrieve individual torrent's properties.

**Raises**

*NotFound404Error* –

**Parameters**

**torrent\_hash** – hash for torrent

**Returns**

*TorrentPropertiesDictionary* - [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#get-torrent-generic-properties](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-torrent-generic-properties)

**torrents\_reannounce**(*torrent\_hashes=None, \*\*kwargs*)

Reannounce a torrent.

Note: torrents/reannounce introduced in Web API 2.0.2

**Parameters**

**torrent\_hashes** – single torrent hash or list of torrent hashes. Or *all* for all torrents.

**Returns**

None

**torrents\_recheck**(*torrent\_hashes=None, \*\*kwargs*)

Recheck a torrent in qBittorrent.

**Parameters**

**torrent\_hashes** – single torrent hash or list of torrent hashes. Or *all* for all torrents.

**Returns**

None

**torrents\_remove\_categories**(*categories=None, \*\*kwargs*)

Delete one or more categories.

**Parameters**

**categories** – categories to delete

**Returns**

None

**torrents\_remove\_tags**(*tags=None, torrent\_hashes=None, \*\*kwargs*)

Add one or more tags to one or more torrents.

**Parameters**

- **tags** – tag name or list of tags
- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or *all* for all torrents.

**Returns**

None

**torrents\_remove\_trackers**(*torrent\_hash=None, urls=None, \*\*kwargs*)

Remove trackers from a torrent.

**Raises**

- [\*NotFound404Error\*](#) –
- [\*Conflict409Error\*](#) –

**Parameters**

- **torrent\_hash** – hash for torrent
- **urls** – tracker urls to removed from torrent

**Returns**

None

**torrents\_rename**(*torrent\_hash=None, new\_torrent\_name=None, \*\*kwargs*)

Rename a torrent.

**Raises**

- [\*NotFound404Error\*](#) –

**Parameters**

- **torrent\_hash** – hash for torrent
- **new\_torrent\_name** – new name for torrent

**Returns**

None

**torrents\_rename\_file**(*torrent\_hash=None, file\_id=None, new\_file\_name=None, old\_path=None, new\_path=None, \*\*kwargs*)

Rename a torrent file.

**Raises**

- [\*MissingRequiredParameters400Error\*](#) –
- [\*NotFound404Error\*](#) –
- [\*Conflict409Error\*](#) –

**Parameters**

- **torrent\_hash** – hash for torrent
- **file\_id** – id for file (removed in Web API 2.7)
- **new\_file\_name** – new name for file (removed in Web API 2.7)
- **old\_path** – path of file to rename (added in Web API 2.7)
- **new\_path** – new path of file to rename (added in Web API 2.7)

**Returns**

None

**torrents\_rename\_folder**(*torrent\_hash=None, old\_path=None, new\_path=None, \*\*kwargs*)

Rename a torrent folder.

**Raises**

- *MissingRequiredParameters400Error* –
- *NotFound404Error* –
- *Conflict409Error* –

#### Parameters

- **torrent\_hash** – hash for torrent
- **old\_path** – path of file to rename (added in Web API 2.7)
- **new\_path** – new path of file to rename (added in Web API 2.7)

#### Returns

None

**torrents\_resume**(*torrent\_hashes=None, \*\*kwargs*)

Resume one or more torrents in qBittorrent.

#### Parameters

**torrent\_hashes** – single torrent hash or list of torrent hashes. Or **all** for all torrents.

#### Returns

None

**torrents\_setDownloadPath**(*download\_path=None, torrent\_hashes=None, \*\*kwargs*)

Set the Download Path for one or more torrents.

#### Raises

- *Forbidden403Error* – cannot write to directory
- *Conflict409Error* – directory cannot be created

#### Parameters

- **download\_path** – file path to save torrent contents before torrent finishes downloading
- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or **all** for all torrents.

**torrents\_setSavePath**(*save\_path=None, torrent\_hashes=None, \*\*kwargs*)

Set the Save Path for one or more torrents.

#### Raises

- *Forbidden403Error* – cannot write to directory
- *Conflict409Error* – directory cannot be created

#### Parameters

- **save\_path** – file path to save torrent contents
- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or **all** for all torrents.

**torrents\_set\_auto\_management**(*enable=None, torrent\_hashes=None, \*\*kwargs*)

Enable or disable automatic torrent management for one or more torrents.

#### Parameters

- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or **all** for all torrents.
- **enable** – True or False

#### Returns

None

**torrents\_set\_category**(*category=None, torrent\_hashes=None, \*\*kwargs*)

Set a category for one or more torrents.

**Raises**

**Conflict409Error** – for bad category

**Parameters**

- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or `all` for all torrents.
- **category** – category to assign to torrent

**Returns**

None

**torrents\_set\_download\_limit**(*limit=None, torrent\_hashes=None, \*\*kwargs*)

Set the download limit for one or more torrents.

**Parameters**

- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or `all` for all torrents.
- **limit** – bytes/second (-1 sets the limit to infinity)

**Returns**

None

**torrents\_set\_download\_path**(*download\_path=None, torrent\_hashes=None, \*\*kwargs*)

Set the Download Path for one or more torrents.

**Raises**

- **Forbidden403Error** – cannot write to directory
- **Conflict409Error** – directory cannot be created

**Parameters**

- **download\_path** – file path to save torrent contents before torrent finishes downloading
- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or `all` for all torrents.

**torrents\_set\_force\_start**(*enable=None, torrent\_hashes=None, \*\*kwargs*)

Force start one or more torrents.

**Parameters**

- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or `all` for all torrents.
- **enable** – True or False (False makes this equivalent to `torrents_resume()`)

**Returns**

None

**torrents\_set\_location**(*location=None, torrent\_hashes=None, \*\*kwargs*)

Set location for torrents' files.

**Raises**

- **Forbidden403Error** – if the user doesn't have permissions to write to the location (only before v4.5.2 - write check was removed.)
- **Conflict409Error** – if the directory cannot be created at the location

**Parameters**

- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or `all` for all torrents.



- **location** – disk location to move torrent’s files

**Returns**

None

**torrents\_set\_save\_path**(*save\_path=None, torrent\_hashes=None, \*\*kwargs*)

Set the Save Path for one or more torrents.

**Raises**

- **Forbidden403Error** – cannot write to directory
- **Conflict409Error** – directory cannot be created

**Parameters**

- **save\_path** – file path to save torrent contents
- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or **all** for all torrents.

**torrents\_set\_share\_limits**(*ratio\_limit=None, seeding\_time\_limit=None, torrent\_hashes=None, \*\*kwargs*)

Set share limits for one or more torrents.

**Parameters**

- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or **all** for all torrents.
- **ratio\_limit** – max ratio to seed a torrent. (-2 means use the global value and -1 is no limit)
- **seeding\_time\_limit** – minutes (-2 means use the global value and -1 is no limit)

**Returns**

None

**torrents\_set\_super\_seeding**(*enable=None, torrent\_hashes=None, \*\*kwargs*)

Set one or more torrents as super seeding.

**Parameters**

- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or **all** for all torrents.
- **enable** – True or False

**Returns****torrents\_set\_upload\_limit**(*limit=None, torrent\_hashes=None, \*\*kwargs*)

Set the upload limit for one or more torrents.

**Parameters**

- **torrent\_hashes** – single torrent hash or list of torrent hashes. Or **all** for all torrents.
- **limit** – bytes/second (-1 sets the limit to infinity)

**Returns**

None

**torrents\_tags**(*\*\*kwargs*)

Retrieve all tag definitions.

**Returns***TagList*

**torrents\_toggle\_first\_last\_piece\_priority**(*torrent\_hashes=None, \*\*kwargs*)

Toggle priority of first/last piece downloading.

**Parameters**

**torrent\_hashes** – single torrent hash or list of torrent hashes. Or `all` for all torrents.

**Returns**

None

**torrents\_toggle\_sequential\_download**(*torrent\_hashes=None, \*\*kwargs*)

Toggle sequential download for one or more torrents.

**Parameters**

**torrent\_hashes** – single torrent hash or list of torrent hashes. Or `all` for all torrents.

**Returns**

None

**torrents\_top\_priority**(*torrent\_hashes=None, \*\*kwargs*)

Set torrent as highest priority. Torrent Queuing must be enabled.

**Raises**

[\*Conflict409Error\*](#) –

**Parameters**

**torrent\_hashes** – single torrent hash or list of torrent hashes. Or `all` for all torrents.

**Returns**

None

**torrents\_trackers**(*torrent\_hash=None, \*\*kwargs*)

Retrieve individual torrent's trackers.

**Raises**

[\*NotFound404Error\*](#) –

**Parameters**

**torrent\_hash** – hash for torrent

**Returns**

[\*TrackersList\*](#) - [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#get-torrent-trackers](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-torrent-trackers)

**torrents\_upload\_limit**(*torrent\_hashes=None, \*\*kwargs*)

Retrieve the upload limit for one or more torrents.

**Parameters**

**torrent\_hashes** – single torrent hash or list of torrent hashes. Or `all` for all torrents.

**Returns**

[\*TorrentLimitsDictionary\*](#)

**torrents\_webseeds**(*torrent\_hash=None, \*\*kwargs*)

Retrieve individual torrent's web seeds.

**Raises**

[\*NotFound404Error\*](#) –

**Parameters**

**torrent\_hash** – hash for torrent

**Returns**

*WebSeedsList* - [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#get-torrent-web-seeds](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-torrent-web-seeds)

**class** qbittorrentapi.torrents.**Torrents**(*client*)

Allows interaction with the Torrents API endpoints.

**Usage**

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
    ↪ 'adminadmin')
>>> # these are all the same attributes that are available as named in
    ↪ the
>>> # endpoints or the more pythonic names in Client (with or without
    ↪ 'torrents_' prepended)
>>> torrent_list = client.torrents.info()
>>> torrent_list_active = client.torrents.info.active()
>>> torrent_list_active_partial = client.torrents.info.active(limit=100,
    ↪ offset=200)
>>> torrent_list_downloading = client.torrents.info.downloading()
>>> # torrent looping
>>> for torrent in client.torrents.info.completed()
>>> # all torrents endpoints with a 'hashes' parameters support all
    ↪ method to apply action to all torrents
>>> client.torrents.pause.all()
>>> client.torrents.resume.all()
>>> # or specify the individual hashes
>>> client.torrents.downloadLimit(torrent_hashes=['...', '...'])
```

**add**(*urls=None, torrent\_files=None, save\_path=None, cookie=None, category=None, is\_skip\_checking=None, is\_paused=None, is\_root\_folder=None, rename=None, upload\_limit=None, download\_limit=None, use\_auto\_torrent\_management=None, is\_sequential\_download=None, is\_first\_last\_piece\_priority=None, tags=None, content\_layout=None, ratio\_limit=None, seeding\_time\_limit=None, download\_path=None, use\_download\_path=None, stop\_condition=None, \*\*kwargs*)

**class** qbittorrentapi.torrents.**TorrentDictionary**(*data, client*)

Bases: *Dictionary*

Item in *TorrentInfoList*. Allows interaction with individual torrents via the Torrents API endpoints.

**Usage**

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
    ↪ 'adminadmin')
>>> # these are all the same attributes that are available as named in
    ↪ the
>>> # endpoints or the more pythonic names in Client (with or without
    ↪ 'transfer_' prepended)
>>> torrent = client.torrents.info()[0]
>>> torrent_hash = torrent.info.hash
>>> # Attributes without inputs and a return value are properties
>>> properties = torrent.properties
```

(continues on next page)

(continued from previous page)

```
>>> trackers = torrent.trackers
>>> files = torrent.files
>>> # Action methods
>>> torrent.edit_tracker(original_url="...", new_url="...")
>>> torrent.remove_trackers(urls='http://127.0.0.2/')
>>> torrent.rename(new_torrent_name="...")
>>> torrent.resume()
>>> torrent.pause()
>>> torrent.recheck()
>>> torrent.torrents_top_priority()
>>> torrent.setLocation(location='/home/user/torrents/')
>>> torrent.setCategory(category='video')
```

**add\_tags**(tags=None, \*\*kwargs)

Implements `torrents_add_tags()`

**add\_trackers**(urls=None, \*\*kwargs)

Implements `torrents_add_trackers()`

**bottom\_priority**(\*\*kwargs)

Implements `torrents_bottom_priority()`

**decrease\_priority**(\*\*kwargs)

Implements `torrents_decrease_priority()`

**delete**(delete\_files=None, \*\*kwargs)

Implements `torrents_delete()`

**property download\_limit**

Implements `torrents_set_download_limit()`

**edit\_tracker**(orig\_url=None, new\_url=None, \*\*kwargs)

Implements `torrents_edit_tracker()`

**export**(\*\*kwargs)

Implements `torrents_export()`

**file\_priority**(file\_ids=None, priority=None, \*\*kwargs)

Implements `torrents_file_priority()`

**property files**

Implements `torrents_files()`

**increase\_priority**(\*\*kwargs)

Implements `torrents_increase_priority()`

**property info**

Implements `torrents_info()`

**pause**(\*\*kwargs)

Implements `torrents_pause()`

**property piece\_hashes**

Implements `torrents_piece_hashes()`

**property piece\_states**Implements `torrents_piece_states()`**property properties**Implements `torrents_properties()`**reannounce(\*\*kwargs)**Implements `torrents_reannounce()`**recheck(\*\*kwargs)**Implements `torrents_recheck()`**remove\_tags(tags=None, \*\*kwargs)**Implements `torrents_remove_tags()`**remove\_trackers(urls=None, \*\*kwargs)**Implements `torrents_remove_trackers()`**rename(new\_name=None, \*\*kwargs)**Implements `torrents_rename()`**rename\_file(file\_id=None, new\_file\_name=None, old\_path=None, new\_path=None, \*\*kwargs)**Implements `torrents_rename_file()`**rename\_folder(old\_path=None, new\_path=None, \*\*kwargs)**Implements `torrents_rename_folder()`**resume(\*\*kwargs)**Implements `torrents_resume()`**setDownloadPath(download\_path=None, \*\*kwargs)**Implements `torrents_set_download_path()`**setSavePath(save\_path=None, \*\*kwargs)**Implements `torrents_set_save_path()`**set\_auto\_management(enable=None, \*\*kwargs)**Implements `torrents_set_auto_management()`**set\_category(category=None, \*\*kwargs)**Implements `torrents_set_category()`**set\_download\_limit(limit=None, \*\*kwargs)**Implements `torrents_set_download_limit()`**set\_download\_path(download\_path=None, \*\*kwargs)**Implements `torrents_set_download_path()`**set\_force\_start(enable=None, \*\*kwargs)**Implements `torrents_set_force_start()`**set\_location(location=None, \*\*kwargs)**Implements `torrents_set_location()`**set\_save\_path(save\_path=None, \*\*kwargs)**Implements `torrents_set_save_path()`

**set\_share\_limits**(ratio\_limit=None, seeding\_time\_limit=None, \*\*kwargs)

Implements `torrents_set_share_limits()`

**set\_super\_seeding**(enable=None, \*\*kwargs)

Implements `torrents_set_super_seeding()`

**set\_upload\_limit**(limit=None, \*\*kwargs)

Implements `torrents_set_upload_limit()`

**property state\_enum**

Returns the formalized Enumeration for Torrent State instead of the raw string.

**sync\_local()**

Update local cache of torrent info.

**toggle\_first\_last\_piece\_priority**(\*\*kwargs)

Implements `torrents_toggle_first_last_piece_priority()`

**toggle\_sequential\_download**(\*\*kwargs)

Implements `torrents_toggle_sequential_download()`

**top\_priority**(\*\*kwargs)

Implements `torrents_top_priority()`

**property trackers**

Implements `torrents_trackers()`

**property upload\_limit**

Implements `torrents_upload_limit()`

**property webseeds**

Implements `torrents_webseeds()`

**class** qbittorrentapi.torrents.TorrentCategories(\*args, \*\*kwargs)

Bases: `ClientCache`

Allows interaction with torrent categories within the Torrents API endpoints.

**Usage**

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> # these are all the same attributes that are available as named in,
↳ the
>>> # endpoints or the more pythonic names in Client (with or without
↳ 'torrents_' prepended)
>>> categories = client.torrent_categories.categories
>>> # create or edit categories
>>> client.torrent_categories.create_category(name='Video', save_path='/
↳ home/user/torrents/Video')
>>> client.torrent_categories.edit_category(name='Video', save_path='/
↳ data/torrents/Video')
>>> # edit or create new by assignment
>>> client.torrent_categories.categories = dict(name='Video', save_path=
↳ '/home/user/')
>>> # delete categories
```

(continues on next page)

(continued from previous page)

```
>>> client.torrent_categories.removeCategories(categories='Video')
>>> client.torrent_categories.removeCategories(categories=['Audio',
↳ "ISOs"])
```

**property categories**Implements `torrents_categories()`

**create\_category**(*name=None, save\_path=None, download\_path=None, enable\_download\_path=None, \*\*kwargs*)

Implements `torrents_create_category()`

**edit\_category**(*name=None, save\_path=None, download\_path=None, enable\_download\_path=None, \*\*kwargs*)

Implements `torrents_edit_category()`

**remove\_categories**(*categories=None, \*\*kwargs*)

Implements `torrents_remove_categories()`

**class** qbittorrentapi.torrents.TorrentTags(\*args, \*\*kwargs)

Bases: `ClientCache`

Allows interaction with torrent tags within the “Torrent” API endpoints.

**Usage:**

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> tags = client.torrent_tags.tags
>>> client.torrent_tags.tags = 'tv show' # create category
>>> client.torrent_tags.create_tags(tags=['tv show', 'linux distro'])
>>> client.torrent_tags.delete_tags(tags='tv show')
```

**add\_tags**(*tags=None, torrent\_hashes=None, \*\*kwargs*)

Implements `torrents_add_tags()`

**create\_tags**(*tags=None, \*\*kwargs*)

Implements `torrents_create_tags()`

**delete\_tags**(*tags=None, \*\*kwargs*)

Implements `torrents_delete_tags()`

**remove\_tags**(*tags=None, torrent\_hashes=None, \*\*kwargs*)

Implements `torrents_remove_tags()`**property tags**Implements `torrents_tags()`

**class** qbittorrentapi.torrents.TorrentPropertiesDictionary(*data=None, client=None*)

Bases: `Dictionary`Response to `torrents_properties()`

**class** qbittorrentapi.torrents.TorrentLimitsDictionary(*data=None, client=None*)

Bases: `Dictionary`Response to `torrents_download_limit()`

```
class qbittorrentapi.torrents.TorrentCategoriesDictionary(data=None, client=None)
```

Bases: *Dictionary*

Response to *torrents\_categories()*

```
class qbittorrentapi.torrents.TorrentsAddPeersDictionary(data=None, client=None)
```

Bases: *Dictionary*

Response to *torrents\_add\_peers()*

```
class qbittorrentapi.torrents.TorrentFilesList(list_entries, client)
```

Bases: *List*

Response to *torrents\_files()*

```
class qbittorrentapi.torrents.TorrentFile(data=None, client=None)
```

Bases: *ListEntry*

Item in *TorrentFilesList*

```
class qbittorrentapi.torrents.WebSeedsList(list_entries, client)
```

Bases: *List*

Response to *torrents\_webseeds()*

```
class qbittorrentapi.torrents.WebSeed(data=None, client=None)
```

Bases: *ListEntry*

Item in *WebSeedsList*

```
class qbittorrentapi.torrents.TrackersList(list_entries, client)
```

Bases: *List*

Response to *torrents\_trackers()*

```
class qbittorrentapi.torrents.Tracker(data=None, client=None)
```

Bases: *ListEntry*

Item in *TrackersList*

```
class qbittorrentapi.torrents.TorrentInfoList(list_entries, client)
```

Bases: *List*

Response to *torrents\_info()*

```
class qbittorrentapi.torrents.TorrentPieceInfoList(list_entries, client)
```

Bases: *List*

Response to *torrents\_piece\_states()* and *torrents\_piece\_hashes()*

```
class qbittorrentapi.torrents.TorrentPieceData(data=None, client=None)
```

Bases: *ListEntry*

Item in *TorrentPieceInfoList*

```
class qbittorrentapi.torrents.TagList(list_entries, client)
```

Bases: *List*

Response to *torrents\_tags()*

```
class qbittorrentapi.torrents.Tag(data=None, client=None)
```

Bases: *ListEntry*

Item in *TagList*



## Transfer

**class** qbittorrentapi.transfer.**TransferAPIMixin**(*host="", port=None, username=None, password=None, \*\*kwargs*)

Bases: [AppAPIMixin](#)

Implementation of all Transfer API methods.

### Usage

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> transfer_info = client.transfer_info()
>>> client.transfer_set_download_limit(limit=1024000)
```

**transfer\_ban\_peers**(*peers=None, \*\*kwargs*)

Ban one or more peers.

#### Parameters

**peers** – one or more peers to ban. each peer should take the form ‘host:port’

#### Returns

None

**transfer\_download\_limit**(*\*\*kwargs*)

Retrieves download limit. 0 is unlimited.

#### Returns

integer

**transfer\_info**(*\*\*kwargs*)

Retrieves the global transfer info found in qBittorrent status bar.

#### Returns

[TransferInfoDictionary](#) - [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#get-global-transfer-info](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#get-global-transfer-info)

**transfer\_setSpeedLimitsMode**(*intended\_state=None, \*\*kwargs*)

Sets whether alternative speed limits are enabled.

#### Parameters

**intended\_state** – True to enable alt speed and False to disable. Leaving None will toggle the current state.

#### Returns

None

**transfer\_set\_download\_limit**(*limit=None, \*\*kwargs*)

Set the global download limit in bytes/second.

#### Parameters

**limit** – download limit in bytes/second (0 or -1 for no limit)

#### Returns

None

**transfer\_set\_speed\_limits\_mode**(*intended\_state=None, \*\*kwargs*)

Sets whether alternative speed limits are enabled.

**Parameters**

**intended\_state** – True to enable alt speed and False to disable. Leaving None will toggle the current state.

**Returns**

None

**transfer\_set\_upload\_limit**(*limit=None, \*\*kwargs*)

Set the global download limit in bytes/second.

**Parameters**

**limit** – upload limit in bytes/second (0 or -1 for no limit)

**Returns**

None

**transfer\_speed\_limits\_mode**(*\*\*kwargs*)

Retrieves whether alternative speed limits are enabled.

**Returns**

1 if alternative speed limits are currently enabled, 0 otherwise

**transfer\_toggle\_speed\_limits\_mode**(*intended\_state=None, \*\*kwargs*)

Sets whether alternative speed limits are enabled.

**Parameters**

**intended\_state** – True to enable alt speed and False to disable. Leaving None will toggle the current state.

**Returns**

None

**transfer\_upload\_limit**(*\*\*kwargs*)

Retrieves upload limit. 0 is unlimited.

**Returns**

integer

**class** qbittorrentapi.transfer.**Transfer**(\*args, \*\*kwargs)

Allows interaction with the Transfer API endpoints.

**Usage**

```
>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> # these are all the same attributes that are available as named in
↳ the
>>> # endpoints or the more pythonic names in Client (with or without
↳ 'transfer_' prepended)
>>> transfer_info = client.transfer.info
>>> # access and set download/upload limits as attributes
>>> dl_limit = client.transfer.download_limit
>>> # this updates qBittorrent in real-time
>>> client.transfer.download_limit = 1024000
>>> # update speed limits mode to alternate or not
>>> client.transfer.speedLimitsMode = True
```

```

ban_peers(peers=None, **kwargs)
    Implements transfer\_ban\_peers\(\)

property download_limit
    Implements transfer\_download\_limit\(\)

property info
    Implements transfer\_info\(\)

setSpeedLimitsMode(intended_state=None, **kwargs)
    Implements transfer\_set\_speed\_limits\_mode\(\)

set_download_limit(limit=None, **kwargs)
    Implements transfer\_set\_download\_limit\(\)

set_speed_limits_mode(intended_state=None, **kwargs)
    Implements transfer\_set\_speed\_limits\_mode\(\)

set_upload_limit(limit=None, **kwargs)
    Implements transfer\_set\_upload\_limit\(\)

property speed_limits_mode
    Implements transfer\_speed\_limits\_mode\(\)

toggle_speed_limits_mode(intended_state=None, **kwargs)
    Implements transfer\_set\_speed\_limits\_mode\(\)

property upload_limit
    Implements transfer\_upload\_limit\(\)

class qbittorrentapi.transfer.TransferInfoDictionary(data=None, client=None)
    Bases: Dictionary
    Response to transfer\_info\(\)

```

## Version

**class** qbittorrentapi.\_version\_support.**Version**

Allows introspection for whether this Client supports different versions of the qBittorrent application and its Web API.

Note that if a version is not listed as “supported” here, many (if not all) methods are likely to function properly since the Web API is largely backwards and forward compatible... albeit with some notable exceptions.

**classmethod** **is\_api\_version\_supported**(api\_version)

Returns whether a version of the qBittorrent Web API is fully supported by this API client.

### Parameters

**api\_version** – version of qBittorrent Web API version such as 2.8.4

### Returns

True or False for whether version is supported

**classmethod** **is\_app\_version\_supported**(app\_version)

Returns whether a version of the qBittorrent application is fully supported by this API client.

### Parameters

**app\_version** – version of qBittorrent application such as v4.4.0

**Returns**

True or False for whether version is supported

**classmethod latest\_supported\_api\_version()**

Returns the most recent version of qBittorrent Web API that is fully supported.

**classmethod latest\_supported\_app\_version()**

Returns the most recent version of qBittorrent application that is fully supported.

**classmethod supported\_api\_versions()**

Set of all supported qBittorrent Web API versions.

**classmethod supported\_app\_versions()**

Set of all supported qBittorrent application versions.

## PYTHON MODULE INDEX

### q

`qbittorrentapi.definitions`, [16](#)  
`qbittorrentapi.exceptions`, [9](#)  
`qbittorrentapi.request`, [21](#)



## Symbols

\_cast() (*qbittorrentapi.request.Request* method), 21  
 \_get() (*qbittorrentapi.request.Request* method), 21  
 \_get\_data() (*qbittorrentapi.request.Request* static method), 21  
 \_get\_headers() (*qbittorrentapi.request.Request* static method), 21  
 \_get\_requests\_kwargs() (*qbittorrentapi.request.Request* method), 22  
 \_get\_response\_kwargs() (*qbittorrentapi.request.Request* static method), 22  
 \_handle\_error\_responses() (*qbittorrentapi.request.Request* static method), 22  
 \_initialize\_context() (*qbittorrentapi.request.Request* method), 22  
 \_initialize\_lessor() (*qbittorrentapi.request.Request* method), 22  
 \_list2string() (*qbittorrentapi.request.Request* class method), 22  
 \_post() (*qbittorrentapi.request.Request* method), 22  
 \_request() (*qbittorrentapi.request.Request* method), 23  
 \_request\_manager() (*qbittorrentapi.request.Request* method), 23  
 \_session (*qbittorrentapi.request.Request* property), 23  
 \_trigger\_session\_initialization() (*qbittorrentapi.request.Request* method), 23  
 \_verbose\_logging() (*qbittorrentapi.request.Request* method), 23

## A

add() (*qbittorrentapi.torrents.Torrents* method), 47  
 add\_feed() (*qbittorrentapi.rss.RSS* method), 28  
 add\_folder() (*qbittorrentapi.rss.RSS* method), 28  
 add\_tags() (*qbittorrentapi.torrents.TorrentDictionary* method), 48  
 add\_tags() (*qbittorrentapi.torrents.TorrentTags* method), 51  
 add\_trackers() (*qbittorrentapi.torrents.TorrentDictionary* method), 48  
 ALLOCATING (*qbittorrentapidefinitions.TorrentState* attribute), 17

APIConnectionError, 9  
 APIError, 9  
 APINames (*class in qbittorrentapi.definitions*), 16  
 app\_build\_info() (*qbittorrentapi.app.AppAPIMixin* method), 11  
 app\_default\_save\_path() (*qbittorrentapi.app.AppAPIMixin* method), 12  
 app\_preferences() (*qbittorrentapi.app.AppAPIMixin* method), 12  
 app\_set\_preferences() (*qbittorrentapi.app.AppAPIMixin* method), 12  
 app\_shutdown() (*qbittorrentapi.app.AppAPIMixin* method), 12  
 app\_version() (*qbittorrentapi.app.AppAPIMixin* method), 12  
 app\_web\_api\_version() (*qbittorrentapi.app.AppAPIMixin* method), 12  
 AppAPIMixin (*class in qbittorrentapi.app*), 11  
 Application (*class in qbittorrentapi.app*), 12  
 Application (*qbittorrentapi.definitions.APINames* attribute), 16  
 ApplicationPreferencesDictionary (*class in qbittorrentapi.app*), 13  
 Attr (*class in qbittorrentapi.\_attrdict*), 13  
 AttrDict (*class in qbittorrentapi.\_attrdict*), 13  
 auth\_log\_in() (*qbittorrentapi.auth.AuthAPIMixin* method), 14  
 auth\_log\_out() (*qbittorrentapi.auth.AuthAPIMixin* method), 14  
 AuthAPIMixin (*class in qbittorrentapi.auth*), 14  
 Authorization (*class in qbittorrentapi.auth*), 15  
 Authorization (*qbittorrentapi.definitions.APINames* attribute), 16

## B

ban\_peers() (*qbittorrentapi.transfer.Transfer* method), 54  
 bottom\_priority() (*qbittorrentapi.torrents.TorrentDictionary* method), 48  
 build\_base\_url() (*qbittorrentapi.request.URL* method), 24

`build_info` (*qbittorrentapi.app.Application* property), 13  
`build_url()` (*qbittorrentapi.request.URL* method), 24  
`build_url_path()` (*qbittorrentapi.request.URL* method), 24  
`BuildInfoDictionary` (class in *qbittorrentapi.app*), 13

## C

`categories` (*qbittorrentapi.torrents.TorrentCategories* property), 51  
`categories()` (*qbittorrentapi.search.Search* method), 32  
`CHECKING_DOWNLOAD` (*qbittorrentapi.definitions.TorrentState* attribute), 17  
`CHECKING_RESUME_DATA` (*qbittorrentapi.definitions.TorrentState* attribute), 18  
`CHECKING_UPLOAD` (*qbittorrentapi.definitions.TorrentState* attribute), 18  
`Client` (class in *qbittorrentapi.client*), 15  
`ClientCache` (class in *qbittorrentapi.definitions*), 17  
`Conflict409Error`, 11  
`create_category()` (*qbittorrentapi.torrents.TorrentCategories* method), 51  
`create_tags()` (*qbittorrentapi.torrents.TorrentTags* method), 51

## D

`decrease_priority()` (*qbittorrentapi.torrents.TorrentDictionary* method), 48  
`default_save_path` (*qbittorrentapi.app.Application* property), 13  
`delete()` (*qbittorrentapi.search.Search* method), 32  
`delete()` (*qbittorrentapi.search.SearchJobDictionary* method), 32  
`delete()` (*qbittorrentapi.torrents.TorrentDictionary* method), 48  
`delete_tags()` (*qbittorrentapi.torrents.TorrentTags* method), 51  
`detect_scheme()` (*qbittorrentapi.request.URL* method), 24  
`Dictionary` (class in *qbittorrentapi.definitions*), 17  
`DISABLED` (*qbittorrentapi.definitions.TrackerStatus* attribute), 19  
`display` (*qbittorrentapi.definitions.TrackerStatus* property), 19  
`download_limit` (*qbittorrentapi.torrents.TorrentDictionary* property), 48

`download_limit` (*qbittorrentapi.transfer.Transfer* property), 55  
`DOWNLOADING` (*qbittorrentapi.definitions.TorrentState* attribute), 18

## E

`edit_category()` (*qbittorrentapi.torrents.TorrentCategories* method), 51  
`edit_tracker()` (*qbittorrentapi.torrents.TorrentDictionary* method), 48  
`EMPTY` (*qbittorrentapi.definitions.APINames* attribute), 16  
`enable_plugin()` (*qbittorrentapi.search.Search* method), 32  
`ERROR` (*qbittorrentapi.definitions.TorrentState* attribute), 18  
`export()` (*qbittorrentapi.torrents.TorrentDictionary* method), 48

## F

`file_priority()` (*qbittorrentapi.torrents.TorrentDictionary* method), 48  
`FileError`, 9  
`files` (*qbittorrentapi.torrents.TorrentDictionary* property), 48  
`Forbidden403Error`, 11  
`FORCED_DOWNLOAD` (*qbittorrentapi.definitions.TorrentState* attribute), 18  
`FORCED_METADATA_DOWNLOAD` (*qbittorrentapi.definitions.TorrentState* attribute), 18  
`FORCED_UPLOAD` (*qbittorrentapi.definitions.TorrentState* attribute), 18

## H

`HTTP400Error`, 9  
`HTTP401Error`, 10  
`HTTP403Error`, 10  
`HTTP404Error`, 10  
`HTTP405Error`, 10  
`HTTP409Error`, 10  
`HTTP415Error`, 10  
`HTTP4XXError`, 9  
`HTTP500Error`, 10  
`HTTP5XXError`, 9  
`http_status_code` (*qbittorrentapi.exceptions.HTTP400Error* attribute), 10  
`http_status_code` (*qbittorrentapi.exceptions.HTTP401Error* attribute), 10



[http\\_status\\_code](#) (*qbittorrentapi.exceptions.HTTP403Error* attribute), 10  
[http\\_status\\_code](#) (*qbittorrentapi.exceptions.HTTP404Error* attribute), 10  
[http\\_status\\_code](#) (*qbittorrentapi.exceptions.HTTP405Error* attribute), 10  
[http\\_status\\_code](#) (*qbittorrentapi.exceptions.HTTP409Error* attribute), 10  
[http\\_status\\_code](#) (*qbittorrentapi.exceptions.HTTP415Error* attribute), 10  
[http\\_status\\_code](#) (*qbittorrentapi.exceptions.HTTP500Error* attribute), 10  
[http\\_status\\_code](#) (*qbittorrentapi.exceptions.HTTPError* attribute), 9  
[HTTPError](#), 9  
**I**  
[increase\\_priority\(\)](#) (*qbittorrentapi.torrents.TorrentDictionary* method), 48  
[info](#) (*qbittorrentapi.torrents.TorrentDictionary* property), 48  
[info](#) (*qbittorrentapi.transfer.Transfer* property), 55  
[install\\_plugin\(\)](#) (*qbittorrentapi.search.Search* method), 32  
[InternalServerError500Error](#), 11  
[InvalidRequest400Error](#), 10  
[is\\_api\\_version\\_supported\(\)](#) (*qbittorrentapi.\_version\_support.Version* class method), 55  
[is\\_app\\_version\\_supported\(\)](#) (*qbittorrentapi.\_version\_support.Version* class method), 55  
[is\\_checking](#) (*qbittorrentapi.definitions.TorrentState* property), 18  
[is\\_complete](#) (*qbittorrentapi.definitions.TorrentState* property), 18  
[is\\_downloading](#) (*qbittorrentapi.definitions.TorrentState* property), 18  
[is\\_errored](#) (*qbittorrentapi.definitions.TorrentState* property), 18  
[is\\_logged\\_in](#) (*qbittorrentapi.auth.AuthAPIMixin* property), 14  
[is\\_logged\\_in](#) (*qbittorrentapi.auth.Authorization* property), 15  
[is\\_paused](#) (*qbittorrentapi.definitions.TorrentState* property), 18  
[is\\_uploading](#) (*qbittorrentapi.definitions.TorrentState* property), 18  
**L**  
[latest\\_supported\\_api\\_version\(\)](#) (*qbittorrentapi.\_version\_support.Version* class method), 56  
[latest\\_supported\\_app\\_version\(\)](#) (*qbittorrentapi.\_version\_support.Version* class method), 56  
[List](#) (class in *qbittorrentapi.definitions*), 17  
[ListEntry](#) (class in *qbittorrentapi.definitions*), 17  
[Log](#) (class in *qbittorrentapi.log*), 20  
[Log](#) (*qbittorrentapi.definitions.APINames* attribute), 16  
[log\\_in\(\)](#) (*qbittorrentapi.auth.Authorization* method), 15  
[log\\_main\(\)](#) (*qbittorrentapi.log.LogAPIMixin* method), 19  
[log\\_out\(\)](#) (*qbittorrentapi.auth.Authorization* method), 15  
[log\\_peers\(\)](#) (*qbittorrentapi.log.LogAPIMixin* method), 20  
[LogAPIMixin](#) (class in *qbittorrentapi.log*), 19  
[LogEntry](#) (class in *qbittorrentapi.log*), 20  
[LoginFailed](#), 9  
[LogMainList](#) (class in *qbittorrentapi.log*), 20  
[LogPeer](#) (class in *qbittorrentapi.log*), 20  
[LogPeersList](#) (class in *qbittorrentapi.log*), 20  
**M**  
[mark\\_as\\_read\(\)](#) (*qbittorrentapi.rss.RSS* method), 28  
[matching\\_articles\(\)](#) (*qbittorrentapi.rss.RSS* method), 28  
[METADATA\\_DOWNLOAD](#) (*qbittorrentapi.definitions.TorrentState* attribute), 18  
[MethodNotAllowed405Error](#), 11  
[MISSING\\_FILES](#) (*qbittorrentapi.definitions.TorrentState* attribute), 18  
[MissingRequiredParameters400Error](#), 10  
**module**  
     *qbittorrentapi.definitions*, 16  
     *qbittorrentapi.exceptions*, 9  
     *qbittorrentapi.request*, 21  
[move\\_item\(\)](#) (*qbittorrentapi.rss.RSS* method), 28  
[MOVING](#) (*qbittorrentapi.definitions.TorrentState* attribute), 18  
[MutableAttr](#) (class in *qbittorrentapi.\_attrdict*), 13  
**N**  
[NOT\\_CONTACTED](#) (*qbittorrentapi.definitions.TrackerStatus* attribute), 19

NOT\_WORKING (*qbittorrentapi.definitions.TrackerStatus* attribute), 19  
NotFound404Error, 11

## P

pause() (*qbittorrentapi.torrents.TorrentDictionary* method), 48  
PAUSED\_DOWNLOAD (*qbittorrentapi.definitions.TorrentState* attribute), 18  
PAUSED\_UPLOAD (*qbittorrentapi.definitions.TorrentState* attribute), 18  
peers() (*qbittorrentapi.log.Log* method), 20  
piece\_hashes (*qbittorrentapi.torrents.TorrentDictionary* property), 48  
piece\_states (*qbittorrentapi.torrents.TorrentDictionary* property), 48  
plugins (*qbittorrentapi.search.Search* property), 32  
preferences (*qbittorrentapi.app.Application* property), 13  
properties (*qbittorrentapi.torrents.TorrentDictionary* property), 49

## Q

qbittorrentapi.definitions module, 16  
qbittorrentapi.exceptions module, 9  
qbittorrentapi.request module, 21  
QUEUED\_DOWNLOAD (*qbittorrentapi.definitions.TorrentState* attribute), 18  
QUEUED\_UPLOAD (*qbittorrentapi.definitions.TorrentState* attribute), 18

## R

reannounce() (*qbittorrentapi.torrents.TorrentDictionary* method), 49  
recheck() (*qbittorrentapi.torrents.TorrentDictionary* method), 49  
refresh\_item() (*qbittorrentapi.rss.RSS* method), 28  
remove\_categories() (*qbittorrentapi.torrents.TorrentCategories* method), 51  
remove\_item() (*qbittorrentapi.rss.RSS* method), 28  
remove\_rule() (*qbittorrentapi.rss.RSS* method), 28  
remove\_tags() (*qbittorrentapi.torrents.TorrentDictionary* method), 49

remove\_tags() (*qbittorrentapi.torrents.TorrentTags* method), 51  
remove\_trackers() (*qbittorrentapi.torrents.TorrentDictionary* method), 49  
rename() (*qbittorrentapi.torrents.TorrentDictionary* method), 49  
rename\_file() (*qbittorrentapi.torrents.TorrentDictionary* method), 49  
rename\_folder() (*qbittorrentapi.torrents.TorrentDictionary* method), 49  
rename\_rule() (*qbittorrentapi.rss.RSS* method), 28  
Request (class in *qbittorrentapi.request*), 21  
results() (*qbittorrentapi.search.Search* method), 32  
results() (*qbittorrentapi.search.SearchJobDictionary* method), 32  
resume() (*qbittorrentapi.torrents.TorrentDictionary* method), 49  
RSS (class in *qbittorrentapi.rss*), 28  
RSS (*qbittorrentapi.definitions.APINames* attribute), 16  
rss\_add\_feed() (*qbittorrentapi.rss.RSSAPIMixin* method), 25  
rss\_add\_folder() (*qbittorrentapi.rss.RSSAPIMixin* method), 25  
rss\_items() (*qbittorrentapi.rss.RSSAPIMixin* method), 25  
rss\_mark\_as\_read() (*qbittorrentapi.rss.RSSAPIMixin* method), 25  
rss\_matching\_articles() (*qbittorrentapi.rss.RSSAPIMixin* method), 26  
rss\_move\_item() (*qbittorrentapi.rss.RSSAPIMixin* method), 26  
rss\_refresh\_item() (*qbittorrentapi.rss.RSSAPIMixin* method), 26  
rss\_remove\_item() (*qbittorrentapi.rss.RSSAPIMixin* method), 26  
rss\_remove\_rule() (*qbittorrentapi.rss.RSSAPIMixin* method), 26  
rss\_rename\_rule() (*qbittorrentapi.rss.RSSAPIMixin* method), 27  
rss\_rules() (*qbittorrentapi.rss.RSSAPIMixin* method), 27  
rss\_set\_feed\_url() (*qbittorrentapi.rss.RSSAPIMixin* method), 27  
rss\_set\_rule() (*qbittorrentapi.rss.RSSAPIMixin* method), 27  
rss\_setFeedURL() (*qbittorrentapi.rss.RSSAPIMixin* method), 27  
RSSAPIMixin (class in *qbittorrentapi.rss*), 25  
RSSitemsDictionary (class in *qbittorrentapi.rss*), 29  
RSSRulesDictionary (class in *qbittorrentapi.rss*), 29  
rules (*qbittorrentapi.rss.RSS* property), 28

## S

Search (class in <i>qbittorrentapi.search</i> ), 31	
Search ( <i>qbittorrentapi.definitions.APINames</i> attribute), 17	
search_categories() ( <i>qbittorrentapi.search.SearchAPIMixin</i> method), 29	
search_delete() ( <i>qbittorrentapi.search.SearchAPIMixin</i> method), 29	
search_enable_plugin() ( <i>qbittorrentapi.search.SearchAPIMixin</i> method), 29	
search_install_plugin() ( <i>qbittorrentapi.search.SearchAPIMixin</i> method), 30	
search_plugins() ( <i>qbittorrentapi.search.SearchAPIMixin</i> method), 30	
search_results() ( <i>qbittorrentapi.search.SearchAPIMixin</i> method), 30	
search_start() ( <i>qbittorrentapi.search.SearchAPIMixin</i> method), 30	
search_status() ( <i>qbittorrentapi.search.SearchAPIMixin</i> method), 30	
search_stop() ( <i>qbittorrentapi.search.SearchAPIMixin</i> method), 31	
search_uninstall_plugin() ( <i>qbittorrentapi.search.SearchAPIMixin</i> method), 31	
search_update_plugins() ( <i>qbittorrentapi.search.SearchAPIMixin</i> method), 31	
SearchAPIMixin (class in <i>qbittorrentapi.search</i> ), 29	
SearchCategoriesList (class in <i>qbittorrentapi.search</i> ), 33	
SearchCategory (class in <i>qbittorrentapi.search</i> ), 33	
SearchJobDictionary (class in <i>qbittorrentapi.search</i> ), 32	
SearchPlugin (class in <i>qbittorrentapi.search</i> ), 33	
SearchPluginsList (class in <i>qbittorrentapi.search</i> ), 33	
SearchResultsDictionary (class in <i>qbittorrentapi.search</i> ), 32	
SearchStatus (class in <i>qbittorrentapi.search</i> ), 33	
SearchStatusesList (class in <i>qbittorrentapi.search</i> ), 32	
set_auto_management() ( <i>qbittorrentapi.torrents.TorrentDictionary</i> method), 49	
set_category() ( <i>qbittorrentapi.torrents.TorrentDictionary</i> method), 49	
set_download_limit() ( <i>qbittorrentapi.torrents.TorrentDictionary</i> method), 49	
set_download_limit() ( <i>qbittorrentapi.transfer.Transfer</i> method), 55	
set_download_path() ( <i>qbittorrentapi.torrents.TorrentDictionary</i> method), 49	
set_feed_url() ( <i>qbittorrentapi.rss.RSS</i> method), 28	
set_force_start() ( <i>qbittorrentapi.torrents.TorrentDictionary</i> method), 49	
set_location() ( <i>qbittorrentapi.torrents.TorrentDictionary</i> method), 49	
set_preferences() ( <i>qbittorrentapi.app.Application</i> method), 13	
set_rule() ( <i>qbittorrentapi.rss.RSS</i> method), 29	
set_save_path() ( <i>qbittorrentapi.torrents.TorrentDictionary</i> method), 49	
set_share_limits() ( <i>qbittorrentapi.torrents.TorrentDictionary</i> method), 49	
set_speed_limits_mode() ( <i>qbittorrentapi.transfer.Transfer</i> method), 55	
set_super_seeding() ( <i>qbittorrentapi.torrents.TorrentDictionary</i> method), 50	
set_upload_limit() ( <i>qbittorrentapi.torrents.TorrentDictionary</i> method), 50	
set_upload_limit() ( <i>qbittorrentapi.transfer.Transfer</i> method), 55	
setDownloadPath() ( <i>qbittorrentapi.torrents.TorrentDictionary</i> method), 49	
setFeedURL() ( <i>qbittorrentapi.rss.RSS</i> method), 28	
setSavePath() ( <i>qbittorrentapi.torrents.TorrentDictionary</i> method), 49	
setSpeedLimitsMode() ( <i>qbittorrentapi.transfer.Transfer</i> method), 55	
shutdown() ( <i>qbittorrentapi.app.Application</i> method), 13	
speed_limits_mode ( <i>qbittorrentapi.transfer.Transfer</i> property), 55	
STALLED_DOWNLOAD ( <i>qbittorrentapi.definitions.TorrentState</i> attribute), 18	
STALLED_UPLOAD ( <i>qbittorrentapi.definitions.TorrentState</i> attribute), 18	
start() ( <i>qbittorrentapi.search.Search</i> method), 32	

`state_enum` (*qbittorrentapi.torrents.TorrentDictionary* property), 50

`status()` (*qbittorrentapi.search.Search* method), 32

`status()` (*qbittorrentapi.search.SearchJobDictionary* method), 32

`stop()` (*qbittorrentapi.search.Search* method), 32

`stop()` (*qbittorrentapi.search.SearchJobDictionary* method), 32

`supported_api_versions()` (*qbittorrentapi.\_version\_support.Version* class method), 56

`supported_app_versions()` (*qbittorrentapi.\_version\_support.Version* class method), 56

`Sync` (class in *qbittorrentapi.sync*), 34

`Sync` (*qbittorrentapidefinitions.APINames* attribute), 17

`sync_local()` (*qbittorrentapi.torrents.TorrentDictionary* method), 50

`sync_maindata()` (*qbittorrentapi.sync.SyncAPIMixin* method), 33

`sync_torrent_peers()` (*qbittorrentapi.sync.SyncAPIMixin* method), 33

`SyncAPIMixin` (class in *qbittorrentapi.sync*), 33

`SyncMainDataDictionary` (class in *qbittorrentapi.sync*), 34

`SyncTorrentPeersDictionary` (class in *qbittorrentapi.sync*), 34

## T

`Tag` (class in *qbittorrentapi.torrents*), 52

`TagList` (class in *qbittorrentapi.torrents*), 52

`tags` (*qbittorrentapi.torrents.TorrentTags* property), 51

`toggle_first_last_piece_priority()` (*qbittorrentapi.torrents.TorrentDictionary* method), 50

`toggle_sequential_download()` (*qbittorrentapi.torrents.TorrentDictionary* method), 50

`toggle_speed_limits_mode()` (*qbittorrentapi.transfer.Transfer* method), 55

`top_priority()` (*qbittorrentapi.torrents.TorrentDictionary* method), 50

`TorrentCategories` (class in *qbittorrentapi.torrents*), 50

`TorrentCategoriesDictionary` (class in *qbittorrentapi.torrents*), 51

`TorrentDictionary` (class in *qbittorrentapi.torrents*), 47

`TorrentFile` (class in *qbittorrentapi.torrents*), 52

`TorrentFileError`, 9

`TorrentFileNotFoundError`, 9

`TorrentFilePermissionError`, 9

`TorrentFilesList` (class in *qbittorrentapi.torrents*), 52

`TorrentInfoList` (class in *qbittorrentapi.torrents*), 52

`TorrentLimitsDictionary` (class in *qbittorrentapi.torrents*), 51

`TorrentPieceData` (class in *qbittorrentapi.torrents*), 52

`TorrentPieceInfoList` (class in *qbittorrentapi.torrents*), 52

`TorrentPropertiesDictionary` (class in *qbittorrentapi.torrents*), 51

`Torrents` (class in *qbittorrentapi.torrents*), 47

`Torrents` (*qbittorrentapidefinitions.APINames* attribute), 17

`torrents_add()` (*qbittorrentapi.torrents.TorrentsAPIMixin* method), 35

`torrents_add_peers()` (*qbittorrentapi.torrents.TorrentsAPIMixin* method), 36

`torrents_add_tags()` (*qbittorrentapi.torrents.TorrentsAPIMixin* method), 36

`torrents_add_trackers()` (*qbittorrentapi.torrents.TorrentsAPIMixin* method), 36

`torrents_bottom_priority()` (*qbittorrentapi.torrents.TorrentsAPIMixin* method), 37

`torrents_categories()` (*qbittorrentapi.torrents.TorrentsAPIMixin* method), 37

`torrents_create_category()` (*qbittorrentapi.torrents.TorrentsAPIMixin* method), 37

`torrents_create_tags()` (*qbittorrentapi.torrents.TorrentsAPIMixin* method), 37

`torrents_decrease_priority()` (*qbittorrentapi.torrents.TorrentsAPIMixin* method), 37

`torrents_delete()` (*qbittorrentapi.torrents.TorrentsAPIMixin* method), 38

`torrents_delete_tags()` (*qbittorrentapi.torrents.TorrentsAPIMixin* method), 38

`torrents_download_limit()` (*qbittorrentapi.torrents.TorrentsAPIMixin* method), 38

`torrents_edit_category()` (*qbittorrentapi.torrents.TorrentsAPIMixin* method), 38

`torrents_edit_tracker()` (*qbittorrentapi.torrents.TorrentsAPIMixin* method), 38

<code>torrents_export()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 39	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_set_auto_management()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 43	( <i>qbittorrent</i> - <i>method</i> ),
<code>torrents_file_priority()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 39	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_set_category()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 43	( <i>qbittorrent</i> - <i>method</i> ),
<code>torrents_files()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 39	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_set_download_limit()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 44	( <i>qbittorrent</i> - <i>method</i> ),
<code>torrents_increase_priority()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 39	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_set_download_path()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 44	( <i>qbittorrent</i> - <i>method</i> ),
<code>torrents_info()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 40	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_set_force_start()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 44	( <i>qbittorrent</i> - <i>method</i> ),
<code>torrents_pause()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 40	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_set_location()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 44	( <i>qbittorrent</i> - <i>method</i> ),
<code>torrents_piece_hashes()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 40	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_set_save_path()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 45	( <i>qbittorrent</i> - <i>method</i> ),
<code>torrents_piece_states()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 40	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_set_share_limits()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 45	( <i>qbittorrent</i> - <i>method</i> ),
<code>torrents_properties()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 41	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_set_super_seeding()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 45	( <i>qbittorrent</i> - <i>method</i> ),
<code>torrents_reannounce()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 41	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_set_upload_limit()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 45	( <i>qbittorrent</i> - <i>method</i> ),
<code>torrents_recheck()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 41	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_setDownloadPath()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 43	( <i>qbittorrent</i> - <i>method</i> ),
<code>torrents_remove_categories()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 41	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_setSavePath()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 43	( <i>qbittorrent</i> - <i>method</i> ),
<code>torrents_remove_tags()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 41	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_tags()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 45	( <i>qbittorrent</i> - <i>method</i> ),
<code>torrents_remove_trackers()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 42	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_toggle_first_last_piece_priority()</code> ( <i>qbittorrent</i> <i>rentapi.torrents.TorrentsAPIMixin</i> <i>method</i> ), 45	
<code>torrents_rename()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 42	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_toggle_sequential_download()</code> ( <i>qbittorrent</i> - <i>rentapi.torrents.TorrentsAPIMixin</i> <i>method</i> ), 46	
<code>torrents_rename_file()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 42	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_top_priority()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 46	( <i>qbittorrent</i> - <i>method</i> ),
<code>torrents_rename_folder()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 42	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_trackers()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 46	( <i>qbittorrent</i> - <i>method</i> ),
<code>torrents_resume()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 43	( <i>qbittorrent</i> - <i>method</i> ),	<code>torrents_upload_limit()</code> <i>rentapi.torrents.TorrentsAPIMixin</i> 46	( <i>qbittorrent</i> - <i>method</i> ),
		<code>torrents_webseeds()</code>	( <i>qbittorrent</i> -



*rentapi.torrents.TorrentsAPIMixin* (method), 46

*TorrentsAddPeersDictionary* (class in *qbittorrentapi.torrents*), 52

*TorrentsAPIMixin* (class in *qbittorrentapi.torrents*), 35

*TorrentState* (class in *qbittorrentapi.definitions*), 17

*TorrentStates* (in module *qbittorrentapi.definitions*), 34

*TorrentTags* (class in *qbittorrentapi.torrents*), 51

*Tracker* (class in *qbittorrentapi.torrents*), 52

*trackers* (*qbittorrentapi.torrents.TorrentDictionary* property), 50

*TrackersList* (class in *qbittorrentapi.torrents*), 52

*TrackerStatus* (class in *qbittorrentapi.definitions*), 18

*Transfer* (class in *qbittorrentapi.transfer*), 54

*Transfer* (*qbittorrentapi.definitions.APINames* attribute), 17

*transfer\_ban\_peers()* (*qbittorrentapi.transfer.TransferAPIMixin* method), 53

*transfer\_download\_limit()* (*qbittorrentapi.transfer.TransferAPIMixin* method), 53

*transfer\_info()* (*qbittorrentapi.transfer.TransferAPIMixin* method), 53

*transfer\_set\_download\_limit()* (*qbittorrentapi.transfer.TransferAPIMixin* method), 53

*transfer\_set\_speed\_limits\_mode()* (*qbittorrentapi.transfer.TransferAPIMixin* method), 53

*transfer\_set\_upload\_limit()* (*qbittorrentapi.transfer.TransferAPIMixin* method), 54

*transfer\_setSpeedLimitsMode()* (*qbittorrentapi.transfer.TransferAPIMixin* method), 53

*transfer\_speed\_limits\_mode()* (*qbittorrentapi.transfer.TransferAPIMixin* method), 54

*transfer\_toggle\_speed\_limits\_mode()* (*qbittorrentapi.transfer.TransferAPIMixin* method), 54

*transfer\_upload\_limit()* (*qbittorrentapi.transfer.TransferAPIMixin* method), 54

*TransferAPIMixin* (class in *qbittorrentapi.transfer*), 53

*TransferInfoDictionary* (class in *qbittorrentapi.transfer*), 55

*uninstall\_plugin()* (*qbittorrentapi.search.Search* method), 32

*UNKNOWN* (*qbittorrentapi.definitions.TorrentState* attribute), 18

*UnsupportedMediaType415Error*, 11

*UnsupportedQbittorrentVersion*, 9

*update\_plugins()* (*qbittorrentapi.search.Search* method), 32

*UPDATING* (*qbittorrentapi.definitions.TrackerStatus* attribute), 19

*upload\_limit* (*qbittorrentapi.torrents.TorrentDictionary* property), 50

*upload\_limit* (*qbittorrentapi.transfer.Transfer* property), 55

*UPLOADING* (*qbittorrentapi.definitions.TorrentState* attribute), 18

*URL* (class in *qbittorrentapi.request*), 23

## V

*Version* (class in *qbittorrentapi.\_version\_support*), 55

*version* (*qbittorrentapi.app.Application* property), 13

## W

*web\_api\_version* (*qbittorrentapi.app.Application* property), 13

*WebSeed* (class in *qbittorrentapi.torrents*), 52

*webseeds* (*qbittorrentapi.torrents.TorrentDictionary* property), 50

*WebSeedsList* (class in *qbittorrentapi.torrents*), 52

*WORKING* (*qbittorrentapi.definitions.TrackerStatus* attribute), 19

## U

*Unauthorized401Error*, 11