

---

# qbittorrent-api

Apr 09, 2021



---

## Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
	<b>Python Module Index</b>	<b>41</b>
	<b>Index</b>	<b>43</b>



Python client implementation for qBittorrent Web API.

Currently supports up to qBittorrent [v4.3.3](#) (Web API v2.7) released on January 18, 2021.

The full qBittorrent Web API documentation is available on their [wiki](#).

## 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 specific release:**

```
– pip install git+https://github.com/rmartin16/qbittorrent-api.git@v2020.6.4#egg=qbittorrent-api
```

- **Install direct from master:**

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

- Ensure urllib3, requests, and attrdict are installed. (These are installed automatically using the methods above.)
- 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: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 up to qBittorrent v4.3.3 (Web API v2.7) released on January 18, 2021.

The full qBittorrent Web API documentation is available on their [wiki](#).

#### 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 specific release:**

- `pip install git+https://github.com/rmartin16/qbittorrent-api.git@v2020.6.4#egg=qbittorrent-api`

- **Install direct from master:**

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

- Ensure urllib3, requests, and attrdict are installed. (These are installed automatically using the methods above.)

- 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: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()
```

### 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 `PYTHON_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.

### Host, Username and Password

- These can be provided when instantiating `Client` or calling `qbt_client.auth_log_in(username='...', password='...')`.
- Alternatively, set environment variables `PYTHON_QBITTORRENTAPI_HOST`, `PYTHON_QBITTORRENTAPI_USERNAME` and `PYTHON_QBITTORRENTAPI_PASSWORD`.

### Custom HTTP Headers

- **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'})
```

- These headers will be merged with other headers otherwise configured to be sent.

### 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 `RAISE_NOTIMPLEMENTEDERROR_FOR_UNIMPLEMENTED`

### 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.FileError`

Bases: `OSError`, `qbittorrentapi.exceptions.APIError`

Base class for all exceptions for file handling.

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

Bases: `qbittorrentapi.exceptions.FileError`

Base class for all exceptions for torrent files.

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

Bases: `qbittorrentapi.exceptions.TorrentFileError`

Specified torrent file does not appear to exist.

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

Bases: `qbittorrentapi.exceptions.TorrentFileError`

Permission was denied to read the specified torrent file.

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

Bases: `requests.exceptions.RequestException`, `qbittorrentapi.exceptions.APIError`

Base class for all communications errors including HTTP errors.

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

Bases: `qbittorrentapi.exceptions.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: `requests.exceptions.HTTPError`, `qbittorrentapi.exceptions.APIConnectionError`

Base error for all HTTP errors. All errors following a successful connection to qBittorrent are returned as HTTP statuses.

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

Bases: *qbittorrentapi.exceptions.HTTPError*

Base error for all HTTP 4XX statuses.

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

Bases: *qbittorrentapi.exceptions.HTTPError*

Base error for all HTTP 5XX statuses.

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

Bases: *qbittorrentapi.exceptions.HTTP4XXError*

HTTP 400 Status

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

Bases: *qbittorrentapi.exceptions.HTTP4XXError*

HTTP 401 Status

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

Bases: *qbittorrentapi.exceptions.HTTP4XXError*

HTTP 403 Status

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

Bases: *qbittorrentapi.exceptions.HTTP4XXError*

HTTP 404 Status

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

Bases: *qbittorrentapi.exceptions.HTTP4XXError*

HTTP 409 Status

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

Bases: *qbittorrentapi.exceptions.HTTP4XXError*

HTTP 415 Status

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

Bases: *qbittorrentapi.exceptions.HTTP5XXError*

HTTP 500 Status

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

Bases: *qbittorrentapi.exceptions.HTTP400Error*

Endpoint call is missing one or more required parameters.

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

Bases: *qbittorrentapi.exceptions.HTTP400Error*

One or more endpoint arguments are malformed.

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

Bases: *qbittorrentapi.exceptions.HTTP401Error*

Primarily reserved for XSS and host header issues.

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

Bases: *qbittorrentapi.exceptions.HTTP403Error*

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

**exception** `qbittorrentapi.exceptions.NotFound404Error(*args, **kwargs)`  
 Bases: `qbittorrentapi.exceptions.HTTP404Error`

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

**exception** `qbittorrentapi.exceptions.Conflict409Error(*args, **kwargs)`  
 Bases: `qbittorrentapi.exceptions.HTTP409Error`

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

**exception** `qbittorrentapi.exceptions.UnsupportedMediaType415Error(*args, **kwargs)`  
 Bases: `qbittorrentapi.exceptions.HTTP415Error`

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

**exception** `qbittorrentapi.exceptions.InternalServerError500Error(*args, **kwargs)`  
 Bases: `qbittorrentapi.exceptions.HTTP500Error`

Returned if qBittorrent craps on itself while processing the request...

## 1.4.5 API Reference

### Application

**class** `qbittorrentapi.app.AppAPIMixin(host="", port=None, username=None, password=None, **kwargs)`  
 Bases: `qbittorrentapi.request.Request`

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. (alias: `app_buildInfo`)

**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. (alias: `app_defaultSavePath`)

**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. (alias: `app_setPreferences`)

**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. (alias: app\_webapiVersion)

**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')
>>> # this 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.app_web_api_version
>>> # 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()
```

#### build\_info

Implements `app_build_info()`

#### default\_save\_path

Implements `app_default_save_path()`

#### preferences

Implements `app_preferences()` and `app_set_preferences()`

#### set\_preferences

 (*prefs=None, \*\*kwargs*)

Implements `app_set_preferences()`

#### shutdown

 ()

Implements `app_shutdown()`

#### version

Implements `app_version()`

#### web\_api\_version

Implements `app_web_api_version()`

```

class qbittorrentapi.app.ApplicationPreferencesDictionary (data=None,
                                                         client=None)
    Bases: qbittorrentapi.definitions.Dictionary
    Response for app_preferences ()

class qbittorrentapi.app.BuildInfoDictionary (data=None, client=None)
    Bases: qbittorrentapi.definitions.Dictionary
    Response for app_build_info ()

```

## Authentication

```

class qbittorrentapi.auth.AuthAPIMixin (host="", port=None, username=None, password=None, **kwargs)
    Bases: qbittorrentapi.request.Request
    Implementation of all Authorization API methods.

```

### Usage

```

>>> from qbittorrentapi import Client
>>> client = Client(host='localhost:8080', username='admin', password=
↳ 'adminadmin')
>>> client.auth_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 user is banned... or not logged in

### Parameters

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

### Returns

None

```
auth_log_out (**kwargs)
```

End session with qBittorrent.

```
is_logged_in
```

Returns True/False for whether a log-in attempt was ever successfully completed.

It isn't possible to know if qBittorrent will accept whatever SID is locally cached... however, any request that is rejected because of the SID will be automatically retried after a new SID is requested.

**Returns** True/False for whether a log-in attempt was previously completed

```
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')
>>> client.auth.is_logged_in
>>> client.auth.log_in(username='admin', password='adminadmin')
>>> client.auth.log_out()
```

**is\_logged\_in**

Implements `is_logged_in()`

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

Implements `auth_log_in()`

**log\_out** ()

Implements `auth_log_out()`

## Client

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

Bases: `qbittorrentapi.app.AppAPIMixin`, `qbittorrentapi.auth.AuthAPIMixin`,  
`qbittorrentapi.log.LogAPIMixin`, `qbittorrentapi.sync.SyncAPIMixin`,  
`qbittorrentapi.transfer.TransferAPIMixin`, `qbittorrentapi.torrents.`  
`TorrentsAPIMixin`, `qbittorrentapi.rss.RSSAPIMixin`, `qbittorrentapi.search.`  
`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** – user name 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 `APIConnectionError` exception to be raised.
- **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.
- **DISABLE\_LOGGING\_DEBUG\_OUTPUT** – Turn off debug output from logging for this package as well as Requests & urllib3.

## Definitions

**class** qbittorrentapi.definitions.**APINames**

Bases: `enum.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.**Dictionary** (*data=None, client=None*)

Bases: `qbittorrentapi.definitions.ClientCache`, `qbittorrentapi._attrdict.AttrDict`

Base definition of dictionary-like objects returned from qBittorrent.

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

Bases: `qbittorrentapi.definitions.ClientCache`, `collections.UserList`

Base definition for list-like objects returned from qBittorrent.

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

Bases: `qbittorrentapi.definitions.Dictionary`

Base definition for objects within a list returned from qBittorrent.

## Log

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

Bases: `qbittorrentapi.request.Request`

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=None, client=None*)

Bases: *qbittorrentapi.definitions.List*

Response for *log\_peers()*

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

Bases: *qbittorrentapi.definitions.ListEntry*

Item in *LogPeersList*

```
class qbittorrentapi.log.LogMainList (list_entries=None, client=None)
```

Bases: `qbittorrentapi.definitions.List`

Response to `log_main()`

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

Bases: `qbittorrentapi.definitions.ListEntry`

Item in `LogMainList`

## RSS

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

Bases: `qbittorrentapi.request.Request`

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. (alias: `rss_addFeed`)

Raises `Conflict409Error` –

### Parameters

- **url** – URL of RSS feed (e.g. <http://thepiratebay.org/rss/top100/200>)
- **item\_path** – Name and/or path for new feed (e.g. `FolderSubfolderFeedName`)

Returns `None`

```
rss_add_folder (folder_path=None, **kwargs)
```

Add a RSS folder. Any intermediate folders in path must already exist. (alias: `rss_addFolder`)

Raises `Conflict409Error` –

Parameters **folder\_path** – path to new folder (e.g. `LinuxISOs`)

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 provider, the entire feed is marked as read. (alias: `rss_markAsRead`)

Raises `NotFound404Error` –

### Parameters

- **item\_path** – path to item to be refreshed (e.g. `FolderSubfolderItemName`)

- **article\_id** – article ID from `rss_items()`

**Returns** None

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

Fetch all articles matching a rule. (alias: `rss_matchingArticles`)

**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 a RSS item (folder, feed, etc). (alias: `rss_moveItem`)

**Raises** *Conflict409Error* –

**Parameters**

- **orig\_item\_path** – path to item to be removed (e.g. `FolderSubfolderItemName`)
- **new\_item\_path** – path to item to be removed (e.g. `FolderSubfolderItemName`)

**Returns** None

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

Trigger a refresh for a RSS item (alias: `rss_refreshItem`)

**Parameters** **item\_path** – path to item to be refreshed (e.g. `FolderSubfolderItemName`)

**Returns** None

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

Remove a RSS item (folder, feed, etc). (alias: `rss_removeItem`)

NOTE: Removing a folder also removes everything in it.

**Raises** *Conflict409Error* –

**Parameters** **item\_path** – path to item to be removed (e.g. `FolderSubfolderItemName`)

**Returns** None

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

Delete a RSS auto-downloading rule. (alias: `rss_removeRule`)

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

**Returns** None

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

Rename a RSS auto-download rule. (alias: `rss_renameRule`) 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\_set\_rule** (*rule\_name=None, rule\_def=None, \*\*kwargs*)

Create a new RSS auto-downloading rule. (alias: `rss_setRule`)

### 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 = 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()`

**rules**

Implements `rss_rules()`

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

Implements `rss_set_rule()`

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

Bases: `qbittorrentapi.definitions.Dictionary`

Response for `rss_items()`

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

Bases: `qbittorrentapi.definitions.Dictionary`

Response for `rss_rules()`

## Search

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

Bases: `qbittorrentapi.request.Request`

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). (alias: `search_enablePlugin`)

**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. (alias: `search_installPlugin`)

**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)

**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)

**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. (alias: *search\_uninstallPlugin*)

**Parameters** **names** - names of plugins to uninstall

**Returns** None

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

Auto update search plugins. (alias: search\_updatePlugins)

**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
>>> 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()`

**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: qbittorrentapi.definitions.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: qbittorrentapi.definitions.Dictionary

    Response for search_results()

class qbittorrentapi.search.SearchStatusesList (list_entries=None, client=None)
    Bases: qbittorrentapi.definitions.List

    Response for search_status()

class qbittorrentapi.search.SearchStatus (data=None, client=None)
    Bases: qbittorrentapi.definitions.ListEntry

    Item in SearchStatusesList

class qbittorrentapi.search.SearchCategoriesList (list_entries=None, client=None)
    Bases: qbittorrentapi.definitions.List

    Response for search_categories()

class qbittorrentapi.search.SearchCategory (data=None, client=None)
    Bases: qbittorrentapi.definitions.ListEntry

    Item in SearchCategoriesList

class qbittorrentapi.search.SearchPluginsList (list_entries=None, client=None)
    Bases: qbittorrentapi.definitions.List

    Response for search_plugins()

class qbittorrentapi.search.SearchPlugin (data=None, client=None)
    Bases: qbittorrentapi.definitions.ListEntry

    Item in SearchPluginsList

```

## Sync

```

class qbittorrentapi.sync.SyncAPIMixin (host="", port=None, username=None, password=None, **kwargs)
    Bases: qbittorrentapi.request.Request

    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. (alias: sync\_torrentPeers)

**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')
>>> # this 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: *qbittorrentapi.definitions.Dictionary*

Response for *sync\_maindata()*

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

Bases: *qbittorrentapi.definitions.Dictionary*

Response for *sync\_torrent\_peers()*

## Torrent States

**class** qbittorrentapi.definitions.TorrentStates

Bases: `enum.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/master/src/base/bittorrent/torrenthandle.h#L52>

### Usage

```
>>> from qbittorrentapi import Client
>>> from qbittorrentapi import 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_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'
```

**is\_checking**

Returns True if the State is categorized as Checking.

**is\_complete**

Returns True if the State is categorized as Complete.

**is\_downloading**

Returns True if the State is categorized as Downloading.

**is\_errored**

Returns True if the State is categorized as Errored.

**is\_paused**

Returns True if the State is categorized as Paused.

**is\_uploading**

Returns True if the State is categorized as Uploading.

## Torrents

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

Bases: qbittorrentapi.request.Request

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, con-
             tent_layout=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: 1) single instance of bytes: useful if torrent file already read from disk or downloaded from internet. 2) single instance of file handle to torrent file: use `open(<filepath>, 'rb')` to open the torrent file. 3) single instance of a filepath to torrent file: e.g. `'/home/user/torrent_filename.torrent'` 4) an iterable of the single instances above to send more than one torrent file 5) 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 v2.6.2)
- **content\_layout** – Original, Subfolder, or NoSubfolder to control filesystem structure for content (added in Web API v2.7)

**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. (alias: `torrents_addPeers`)

**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. (alias: `torrents_addTags`) 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. (alias: `torrents_addTrackers`)

**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 highest priority. Torrent Queuing must be enabled. (alias: `torrents_bottomPrio`)

**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 :return: *TorrentCategoriesDictionary*

**torrents\_create\_category** (*name=None, save\_path=None, \*\*kwargs*)

Create a new torrent category. (alias: `torrents_createCategory`)

Note: `save_path` is not available until web API version 2.1.0

**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

**Returns** None

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

Create one or more tags. (alias: `torrents_createTags`)

**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. (alias: `torrents_decreasePrio`)

**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. (alias: `torrents_deleteTags`)

**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. (alias: `torrents_downloadLimit`)

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

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

Edit an existing category. (alias: `torrents_editCategory`)

Note: `torrents/editCategory` not available until web API version 2.1.0

**Raises** *Conflict409Error* -

**Parameters**

- **name** – category to edit
- **save\_path** – new location to save files for this category

**Returns** None

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

Replace a torrent's tracker with a different one. (alias: `torrents_editTrackers`)

**Raises**

- *InvalidRequest400* -
- *NotFound404Error* -
- *Conflict409Error* -

**Parameters**

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

**Returns** None

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

Set priority for one or more files. (alias: `torrents_filePrio`)

**Raises**

- *InvalidRequest400* – 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. (alias: *torrents\_increasePrio*)

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, \*\*kwargs*)

Retrieves list of info for torrents. Note: hashes is available starting web API version 2.0.1

Parameters

- **status\_filter** – Filter list by all, downloading, completed, paused, active, inactive, resumed stalled, stalled\_uploading and stalled\_downloading added in Web API v2.4.1
- **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 ‘|’)

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. (alias: *torrents\_pieceHashes*)

Raises **NotFound404Error** –

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

Returns *TorrentPieceInfoList*

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

Retrieve individual torrent’s pieces’ states. (alias: *torrents\_pieceStates*)

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 not available web API version 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. (alias: *torrents\_removeCategories*)

**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. (alias: *torrents\_removeTags*)

**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. (alias: *torrents\_removeTrackers*)

**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 v2.7)
- **new\_file\_name** – new name for file (removed in Web API v2.7)
- **old\_path** – path of file to rename (added in Web API v2.7)
- **new\_path** – new path of file to rename (added in Web API v2.7)

**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 v2.7)
- **new\_path** – new path of file to rename (added in Web API v2.7)

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

Enable or disable automatic torrent management for one or more torrents. (alias: `torrents_setAutoManagement`)

**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. (alias: `torrents_setCategory`)

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. (alias: `torrents_setDownloadLimit`)

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

Force start one or more torrents. (alias: `torrents_setForceStart`)

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’s files. (alias: `torrents_setLocation`)

Raises

- **Forbidden403Error** – if the user doesn’t have permissions to write to the location
- **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\_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. (alias: `torrents_setSuperSeeding`)

**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. (alias: `torrents_setUploadLimit`)

**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. (alias: `torrents_toggleFirstLastPiecePrio`)

**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. (alias: `torrents_toggleSequentialDownload`)

**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. (alias: `torrents_topPrio`)

**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. (alias: `torrents_uploadLimit`)

**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')
>>> # this 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, **kwargs`)

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

Bases: `qbittorrentapi.definitions.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')
>>> # this 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]
```

(continues on next page)

(continued from previous page)

```

>>> torrent_hash = torrent.info.hash
>>> # Attributes without inputs and a return value are properties
>>> properties = torrent.properties
>>> 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()

download_limit
    Implements set_download_limit()

edit_tracker (orig_url=None, new_url=None, **kwargs)
    Implements torrents_edit_tracker()

file_priority (file_ids=None, priority=None, **kwargs)
    Implements torrents_file_priority()

files
    Implements torrents_files()

increase_priority (**kwargs)
    Implements torrents_increase_priority()

info
    Implements torrents_info()

pause (**kwargs)
    Implements torrents_pause()

piece_hashes
    Implements torrents_piece_hashes()

piece_states
    Implements torrents_piece_states()

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()

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_force_start (enable=None, **kwargs)
    Implements torrents_set_force_start()

set_location (location=None, **kwargs)
    Implements torrents_set_location()

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()

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()

```

**trackers**Implements `torrents_trackers()`**upload\_limit**Implements `torrents_upload_limit()`**webseeds**Implements `torrents_webseeds()`**class** `qbittorrentapi.torrents.TorrentCategories(*args, **kwargs)`Bases: `qbittorrentapi.definitions.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')
>>> # this 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
>>> client.torrent_categories.removeCategories(categories='Video')
>>> client.torrent_categories.removeCategories(categories=['Audio',
↳ "ISOs"])

```

**categories**Implements `torrents_categories()`**create\_category** (`name=None, save_path=None, **kwargs`)Implements `torrents_create_category()`**edit\_category** (`name=None, save_path=None, **kwargs`)Implements `torrents_edit_category()`**remove\_categories** (`categories=None, **kwargs`)Implements `torrents_remove_categories()`**class** `qbittorrentapi.torrents.TorrentTags(*args, **kwargs)`Bases: `qbittorrentapi.definitions.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

```

(continues on next page)



(continued from previous page)

```
>>> 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()*

**tags**  
Implements *torrents\_tags()*

**class** qbittorrentapi.torrents.**TorrentPropertiesDictionary** (*data=None, client=None*)  
Bases: *qbittorrentapi.definitions.Dictionary*  
Response to *torrents\_properties()*

**class** qbittorrentapi.torrents.**TorrentLimitsDictionary** (*data=None, client=None*)  
Bases: *qbittorrentapi.definitions.Dictionary*  
Response to *torrents\_download\_limit()*

**class** qbittorrentapi.torrents.**TorrentCategoriesDictionary** (*data=None, client=None*)  
Bases: *qbittorrentapi.definitions.Dictionary*  
Response to *torrents\_categories()*

**class** qbittorrentapi.torrents.**TorrentsAddPeersDictionary** (*data=None, client=None*)  
Bases: *qbittorrentapi.definitions.Dictionary*  
Response to *torrents\_add\_peers()*

**class** qbittorrentapi.torrents.**TorrentFilesList** (*list\_entries=None, client=None*)  
Bases: *qbittorrentapi.definitions.List*  
Response to *torrents\_files()*

**class** qbittorrentapi.torrents.**TorrentFile** (*data=None, client=None*)  
Bases: *qbittorrentapi.definitions.ListEntry*  
Item in *TorrentFilesList*

**class** qbittorrentapi.torrents.**WebSeedsList** (*list\_entries=None, client=None*)  
Bases: *qbittorrentapi.definitions.List*  
Response to *torrents\_webseeds()*

**class** qbittorrentapi.torrents.**WebSeed** (*data=None, client=None*)  
Bases: *qbittorrentapi.definitions.ListEntry*  
Item in *WebSeedsList*

**class** qbittorrentapi.torrents.**TrackersList** (*list\_entries=None, client=None*)  
Bases: *qbittorrentapi.definitions.List*  
Response to *torrents\_trackers()*

```
class qbittorrentapi.torrents.Tracker (data=None, client=None)
    Bases: qbittorrentapi.definitions.ListEntry

    Item in TrackersList

class qbittorrentapi.torrents.TorrentInfoList (list_entries=None, client=None)
    Bases: qbittorrentapi.definitions.List

    Response to torrents_info()

class qbittorrentapi.torrents.TorrentPieceInfoList (list_entries=None, client=None)
    Bases: qbittorrentapi.definitions.List

    Response to torrents_piece_states() and torrents_piece_hashes()

class qbittorrentapi.torrents.TorrentPieceData (data=None, client=None)
    Bases: qbittorrentapi.definitions.ListEntry

    Item in TorrentPieceInfoList

class qbittorrentapi.torrents.TagList (list_entries=None, client=None)
    Bases: qbittorrentapi.definitions.List

    Response to torrents_tags()

class qbittorrentapi.torrents.Tag (data=None, client=None)
    Bases: qbittorrentapi.definitions.ListEntry

    Item in TagList
```

## Transfer

```
class qbittorrentapi.transfer.TransferAPIMixin (host="", port=None, username=None,
                                                password=None, **kwargs)
    Bases: qbittorrentapi.request.Request

    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. (alias: transfer_banPeers)
```

**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. (alias: transfer_downloadLimit)
```

**Returns** integer

```
transfer_info (**kwargs)
    Retrieves the global transfer info usually 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\_set\_download\_limit** (*limit=None, \*\*kwargs*)

Set the global download limit in bytes/second. (alias: `transfer_setDownloadLimit`)

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

**Returns** None

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

Set the global download limit in bytes/second. (alias: `transfer_setUploadLimit`)

**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. (alias: `transfer_speedLimitMode`)

**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. (alias: `transfer_toggleSpeedLimitsMode`)

**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. (alias: `transfer_uploadLimit`)

**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')
>>> # this 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()`

**download\_limit**

Implements `transfer_download_limit()`

**info**

Implements `transfer_info()`

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

Implements `transfer_set_download_limit()`

```
set_upload_limit (limit=None, **kwargs)  
    Implements transfer_set_upload_limit()  
  
speed_limits_mode  
    Implements transfer_speed_limits_mode()  
  
toggle_speed_limits_mode (intended_state=None, **kwargs)  
    Implements transfer_toggle_speed_limits_mode()  
  
upload_limit  
    Implements transfer_upload_limit()  
  
class qbittorrentapi.transfer.TransferInfoDictionary (data=None, client=None)  
    Bases: qbittorrentapi.definitions.Dictionary  
    Response to transfer_info()
```

### q

`qbittorrentapi.definitions`, [13](#)  
`qbittorrentapi.exceptions`, [7](#)



## A

`add()` (*qbittorrentapi.torrents.Torrents method*), 33  
`add_feed()` (*qbittorrentapi.rss.RSS method*), 17  
`add_folder()` (*qbittorrentapi.rss.RSS method*), 17  
`add_tags()` (*qbittorrentapi.torrents.TorrentDictionary method*), 34  
`add_tags()` (*qbittorrentapi.torrents.TorrentTags method*), 37  
`add_trackers()` (*qbittorrentapi.torrents.TorrentDictionary method*), 34  
`ALLOCATING` (*qbittorrentapi.definitions.TorrentStates attribute*), 23  
`APIConnectionError`, 7  
`APIError`, 7  
`APINames` (*class in qbittorrentapi.definitions*), 13  
`app_build_info()` (*qbittorrentapi.app.AppAPIMixin method*), 9  
`app_default_save_path()` (*qbittorrentapi.app.AppAPIMixin method*), 9  
`app_preferences()` (*qbittorrentapi.app.AppAPIMixin method*), 9  
`app_set_preferences()` (*qbittorrentapi.app.AppAPIMixin method*), 9  
`app_shutdown()` (*qbittorrentapi.app.AppAPIMixin method*), 10  
`app_version()` (*qbittorrentapi.app.AppAPIMixin method*), 10  
`app_web_api_version()` (*qbittorrentapi.app.AppAPIMixin method*), 10  
`AppAPIMixin` (*class in qbittorrentapi.app*), 9  
`Application` (*class in qbittorrentapi.app*), 10  
`Application` (*qbittorrentapi.definitions.APINames attribute*), 13  
`ApplicationPreferencesDictionary` (*class in qbittorrentapi.app*), 10  
`auth_log_in()` (*qbittorrentapi.auth.AuthAPIMixin method*), 11

`auth_log_out()` (*qbittorrentapi.auth.AuthAPIMixin method*), 11  
`AuthAPIMixin` (*class in qbittorrentapi.auth*), 11  
`Authorization` (*class in qbittorrentapi.auth*), 11  
`Authorization` (*qbittorrentapi.definitions.APINames attribute*), 13

## B

`ban_peers()` (*qbittorrentapi.transfer.Transfer method*), 39  
`bottom_priority()` (*qbittorrentapi.torrents.TorrentDictionary method*), 34  
`build_info` (*qbittorrentapi.app.Application attribute*), 10  
`BuildInfoDictionary` (*class in qbittorrentapi.app*), 11

## C

`categories` (*qbittorrentapi.torrents.TorrentCategories attribute*), 36  
`categories()` (*qbittorrentapi.search.Search method*), 20  
`CHECKING_DOWNLOAD` (*qbittorrentapi.definitions.TorrentStates attribute*), 23  
`CHECKING_RESUME_DATA` (*qbittorrentapi.definitions.TorrentStates attribute*), 23  
`CHECKING_UPLOAD` (*qbittorrentapi.definitions.TorrentStates attribute*), 23  
`Client` (*class in qbittorrentapi.client*), 12  
`Conflict409Error`, 9  
`create_category()` (*qbittorrentapi.torrents.TorrentCategories method*), 36  
`create_tags()` (*qbittorrentapi.torrents.TorrentTags method*), 37

## D

[decrease\\_priority\(\)](#) (*qbittorrentapi.torrents.TorrentDictionary* method), 34  
[default\\_save\\_path\(\)](#) (*qbittorrentapi.app.Application* attribute), 10  
[delete\(\)](#) (*qbittorrentapi.search.Search* method), 20  
[delete\(\)](#) (*qbittorrentapi.search.SearchJobDictionary* method), 21  
[delete\(\)](#) (*qbittorrentapi.torrents.TorrentDictionary* method), 34  
[delete\\_tags\(\)](#) (*qbittorrentapi.torrents.TorrentTags* method), 37  
[Dictionary](#) (class in *qbittorrentapi.definitions*), 13  
[download\\_limit](#) (*qbittorrentapi.torrents.TorrentDictionary* attribute), 34  
[download\\_limit](#) (*qbittorrentapi.transfer.Transfer* attribute), 39  
[DOWNLOADING](#) (*qbittorrentapi.definitions.TorrentStates* attribute), 23

## E

[edit\\_category\(\)](#) (*qbittorrentapi.torrents.TorrentCategories* method), 36  
[edit\\_tracker\(\)](#) (*qbittorrentapi.torrents.TorrentDictionary* method), 34  
[EMPTY](#) (*qbittorrentapi.definitions.APINames* attribute), 13  
[enable\\_plugin\(\)](#) (*qbittorrentapi.search.Search* method), 20  
[ERROR](#) (*qbittorrentapi.definitions.TorrentStates* attribute), 23

## F

[file\\_priority\(\)](#) (*qbittorrentapi.torrents.TorrentDictionary* method), 34  
[FileError](#), 7  
[files](#) (*qbittorrentapi.torrents.TorrentDictionary* attribute), 34  
[Forbidden403Error](#), 8  
[FORCED\\_DOWNLOAD](#) (*qbittorrentapi.definitions.TorrentStates* attribute), 23  
[FORCED\\_UPLOAD](#) (*qbittorrentapi.definitions.TorrentStates* attribute), 23

## H

[HTTP400Error](#), 8  
[HTTP401Error](#), 8

[HTTP403Error](#), 8  
[HTTP404Error](#), 8  
[HTTP409Error](#), 8  
[HTTP415Error](#), 8  
[HTTP4XXError](#), 8  
[HTTP500Error](#), 8  
[HTTP5XXError](#), 8  
[HTTPError](#), 7

## I

[increase\\_priority\(\)](#) (*qbittorrentapi.torrents.TorrentDictionary* method), 34  
[info](#) (*qbittorrentapi.torrents.TorrentDictionary* attribute), 34  
[info](#) (*qbittorrentapi.transfer.Transfer* attribute), 39  
[install\\_plugin\(\)](#) (*qbittorrentapi.search.Search* method), 20  
[InternalServerError500Error](#), 9  
[InvalidRequest400Error](#), 8  
[is\\_checking](#) (*qbittorrentapi.definitions.TorrentStates* attribute), 23  
[is\\_complete](#) (*qbittorrentapi.definitions.TorrentStates* attribute), 24  
[is\\_downloading](#) (*qbittorrentapi.definitions.TorrentStates* attribute), 24  
[is\\_errored](#) (*qbittorrentapi.definitions.TorrentStates* attribute), 24  
[is\\_logged\\_in](#) (*qbittorrentapi.auth.AuthAPIMixin* attribute), 11  
[is\\_logged\\_in](#) (*qbittorrentapi.auth.Authorization* attribute), 12  
[is\\_paused](#) (*qbittorrentapi.definitions.TorrentStates* attribute), 24  
[is\\_uploading](#) (*qbittorrentapi.definitions.TorrentStates* attribute), 24

## L

[List](#) (class in *qbittorrentapi.definitions*), 13  
[ListEntry](#) (class in *qbittorrentapi.definitions*), 13  
[Log](#) (class in *qbittorrentapi.log*), 14  
[Log](#) (*qbittorrentapi.definitions.APINames* attribute), 13  
[log\\_in\(\)](#) (*qbittorrentapi.auth.Authorization* method), 12  
[log\\_main\(\)](#) (*qbittorrentapi.log.LogAPIMixin* method), 14  
[log\\_out\(\)](#) (*qbittorrentapi.auth.Authorization* method), 12  
[log\\_peers\(\)](#) (*qbittorrentapi.log.LogAPIMixin* method), 14  
[LogAPIMixin](#) (class in *qbittorrentapi.log*), 13  
[LogEntry](#) (class in *qbittorrentapi.log*), 15



LoginFailed, 7  
 LogMainList (class in qbittorrentapi.log), 14  
 LogPeer (class in qbittorrentapi.log), 14  
 LogPeersList (class in qbittorrentapi.log), 14

## M

mark\_as\_read() (qbittorrentapi.rss.RSS method), 17  
 matching\_articles() (qbittorrentapi.rss.RSS method), 17  
 METADATA\_DOWNLOAD (qbittorrentapi.definitions.TorrentStates attribute), 23  
 MISSING\_FILES (qbittorrentapi.definitions.TorrentStates attribute), 23  
 MissingRequiredParameters400Error, 8  
 move\_item() (qbittorrentapi.rss.RSS method), 17  
 MOVING (qbittorrentapi.definitions.TorrentStates attribute), 23

## N

NotFound404Error, 9

## P

pause() (qbittorrentapi.torrents.TorrentDictionary method), 34  
 PAUSED\_DOWNLOAD (qbittorrentapi.definitions.TorrentStates attribute), 23  
 PAUSED\_UPLOAD (qbittorrentapi.definitions.TorrentStates attribute), 23  
 peers() (qbittorrentapi.log.Log method), 14  
 piece\_hashes (qbittorrentapi.torrents.TorrentDictionary attribute), 34  
 piece\_states (qbittorrentapi.torrents.TorrentDictionary attribute), 34  
 plugins (qbittorrentapi.search.Search attribute), 20  
 preferences (qbittorrentapi.app.Application attribute), 10  
 properties (qbittorrentapi.torrents.TorrentDictionary attribute), 34

## Q

qbittorrentapi.definitions (module), 13  
 qbittorrentapi.exceptions (module), 7  
 QUEUED\_DOWNLOAD (qbittorrentapi.definitions.TorrentStates attribute), 23

QUEUED\_UPLOAD (qbittorrentapi.definitions.TorrentStates attribute), 23

## R

reannounce() (qbittorrentapi.torrents.TorrentDictionary method), 34  
 recheck() (qbittorrentapi.torrents.TorrentDictionary method), 35  
 refresh\_item() (qbittorrentapi.rss.RSS method), 17  
 remove\_categories() (qbittorrentapi.torrents.TorrentCategories method), 36  
 remove\_item() (qbittorrentapi.rss.RSS method), 17  
 remove\_rule() (qbittorrentapi.rss.RSS method), 17  
 remove\_tags() (qbittorrentapi.torrents.TorrentDictionary method), 35  
 remove\_tags() (qbittorrentapi.torrents.TorrentTags method), 37  
 remove\_trackers() (qbittorrentapi.torrents.TorrentDictionary method), 35  
 rename() (qbittorrentapi.torrents.TorrentDictionary method), 35  
 rename\_file() (qbittorrentapi.torrents.TorrentDictionary method), 35  
 rename\_folder() (qbittorrentapi.torrents.TorrentDictionary method), 35  
 rename\_rule() (qbittorrentapi.rss.RSS method), 17  
 results() (qbittorrentapi.search.Search method), 20  
 results() (qbittorrentapi.search.SearchJobDictionary method), 21  
 resume() (qbittorrentapi.torrents.TorrentDictionary method), 35  
 RSS (class in qbittorrentapi.rss), 17  
 RSS (qbittorrentapi.definitions.APINames attribute), 13  
 rss\_add\_feed() (qbittorrentapi.rss.RSSAPIMixin method), 15  
 rss\_add\_folder() (qbittorrentapi.rss.RSSAPIMixin method), 15  
 rss\_items() (qbittorrentapi.rss.RSSAPIMixin method), 15  
 rss\_mark\_as\_read() (qbittorrentapi.rss.RSSAPIMixin method), 15  
 rss\_matching\_articles() (qbittorrentapi.rss.RSSAPIMixin method), 16  
 rss\_move\_item() (qbittorrentapi.rss.RSSAPIMixin method), 16  
 rss\_refresh\_item() (qbittorrentapi.rss.RSSAPIMixin method), 16

rss\_remove\_item() (*qbittorrentapi.rss.RSSAPIMixin* method), 16  
 rss\_remove\_rule() (*qbittorrentapi.rss.RSSAPIMixin* method), 16  
 rss\_rename\_rule() (*qbittorrentapi.rss.RSSAPIMixin* method), 16  
 rss\_rules() (*qbittorrentapi.rss.RSSAPIMixin* method), 16  
 rss\_set\_rule() (*qbittorrentapi.rss.RSSAPIMixin* method), 16  
 RSSAPIMixin (class in *qbittorrentapi.rss*), 15  
 RSSItemsDictionary (class in *qbittorrentapi.rss*), 17  
 RSSRulesDictionary (class in *qbittorrentapi.rss*), 18  
 rules (*qbittorrentapi.rss.RSS* attribute), 17

## S

Search (class in *qbittorrentapi.search*), 20  
 Search (*qbittorrentapi.definitions.APINames* attribute), 13  
 search\_categories() (*qbittorrentapi.search.SearchAPIMixin* method), 18  
 search\_delete() (*qbittorrentapi.search.SearchAPIMixin* method), 18  
 search\_enable\_plugin() (*qbittorrentapi.search.SearchAPIMixin* method), 18  
 search\_install\_plugin() (*qbittorrentapi.search.SearchAPIMixin* method), 18  
 search\_plugins() (*qbittorrentapi.search.SearchAPIMixin* method), 18  
 search\_results() (*qbittorrentapi.search.SearchAPIMixin* method), 19  
 search\_start() (*qbittorrentapi.search.SearchAPIMixin* method), 19  
 search\_status() (*qbittorrentapi.search.SearchAPIMixin* method), 19  
 search\_stop() (*qbittorrentapi.search.SearchAPIMixin* method), 19  
 search\_uninstall\_plugin() (*qbittorrentapi.search.SearchAPIMixin* method), 19  
 search\_update\_plugins() (*qbittorrentapi.search.SearchAPIMixin* method), 19  
 SearchAPIMixin (class in *qbittorrentapi.search*), 18  
 SearchCategoriesList (class in *qbittorrentapi.search*), 21  
 SearchCategory (class in *qbittorrentapi.search*), 21  
 SearchJobDictionary (class in *qbittorrentapi.search*), 20  
 SearchPlugin (class in *qbittorrentapi.search*), 21  
 SearchPluginsList (class in *qbittorrentapi.search*), 21  
 SearchResultsDictionary (class in *qbittorrentapi.search*), 21  
 SearchStatus (class in *qbittorrentapi.search*), 21  
 SearchStatusesList (class in *qbittorrentapi.search*), 21  
 set\_auto\_management() (*qbittorrentapi.torrents.TorrentDictionary* method), 35  
 set\_category() (*qbittorrentapi.torrents.TorrentDictionary* method), 35  
 set\_download\_limit() (*qbittorrentapi.torrents.TorrentDictionary* method), 35  
 set\_download\_limit() (*qbittorrentapi.transfer.Transfer* method), 39  
 set\_force\_start() (*qbittorrentapi.torrents.TorrentDictionary* method), 35  
 set\_location() (*qbittorrentapi.torrents.TorrentDictionary* method), 35  
 set\_preferences() (*qbittorrentapi.app.Application* method), 10  
 set\_rule() (*qbittorrentapi.rss.RSS* method), 17  
 set\_share\_limits() (*qbittorrentapi.torrents.TorrentDictionary* method), 35  
 set\_super\_seeding() (*qbittorrentapi.torrents.TorrentDictionary* method), 35  
 set\_upload\_limit() (*qbittorrentapi.torrents.TorrentDictionary* method), 35  
 set\_upload\_limit() (*qbittorrentapi.transfer.Transfer* method), 39  
 shutdown() (*qbittorrentapi.app.Application* method), 10  
 speed\_limits\_mode (*qbittorrentapi.transfer.Transfer* attribute), 40  
 STALLED\_DOWNLOAD (*qbittorrentapi.definitions.TorrentStates* attribute), 23  
 STALLED\_UPLOAD (*qbittorrentapi.definitions.TorrentStates* attribute),

23  
 start() (qbittorrentapi.search.Search method), 20  
 state\_enum (qbittorrentapi.torrents.TorrentDictionary attribute), 35  
 status() (qbittorrentapi.search.Search method), 20  
 status() (qbittorrentapi.search.SearchJobDictionary method), 21  
 stop() (qbittorrentapi.search.Search method), 20  
 stop() (qbittorrentapi.search.SearchJobDictionary method), 21  
 Sync (class in qbittorrentapi.sync), 22  
 Sync (qbittorrentapidefinitions.APINames attribute), 13  
 sync\_local() (qbittorrentapi.torrents.TorrentDictionary method), 35  
 sync\_maindata() (qbittorrentapi.sync.SyncAPIMixin method), 22  
 sync\_torrent\_peers() (qbittorrentapi.sync.SyncAPIMixin method), 22  
 SyncAPIMixin (class in qbittorrentapi.sync), 21  
 SyncMainDataDictionary (class in qbittorrentapi.sync), 22  
 SyncTorrentPeersDictionary (class in qbittorrentapi.sync), 22

## T

Tag (class in qbittorrentapi.torrents), 38  
 TagList (class in qbittorrentapi.torrents), 38  
 tags (qbittorrentapi.torrents.TorrentTags attribute), 37  
 toggle\_first\_last\_piece\_priority() (qbittorrentapi.torrents.TorrentDictionary method), 35  
 toggle\_sequential\_download() (qbittorrentapi.torrents.TorrentDictionary method), 35  
 toggle\_speed\_limits\_mode() (qbittorrentapi.transfer.Transfer method), 40  
 top\_priority() (qbittorrentapi.torrents.TorrentDictionary method), 35  
 TorrentCategories (class in qbittorrentapi.torrents), 36  
 TorrentCategoriesDictionary (class in qbittorrentapi.torrents), 37  
 TorrentDictionary (class in qbittorrentapi.torrents), 33  
 TorrentFile (class in qbittorrentapi.torrents), 37  
 TorrentFileError, 7  
 TorrentFileNotFoundError, 7  
 TorrentFilePermissionError, 7  
 TorrentFilesList (class in qbittorrentapi.torrents), 37  
 TorrentInfoList (class in qbittorrentapi.torrents), 38  
 TorrentLimitsDictionary (class in qbittorrentapi.torrents), 37  
 TorrentPieceData (class in qbittorrentapi.torrents), 38  
 TorrentPieceInfoList (class in qbittorrentapi.torrents), 38  
 TorrentPropertiesDictionary (class in qbittorrentapi.torrents), 37  
 Torrents (class in qbittorrentapi.torrents), 33  
 Torrents (qbittorrentapidefinitions.APINames attribute), 13  
 torrents\_add() (qbittorrentapi.torrents.TorrentsAPIMixin method), 24  
 torrents\_add\_peers() (qbittorrentapi.torrents.TorrentsAPIMixin method), 25  
 torrents\_add\_tags() (qbittorrentapi.torrents.TorrentsAPIMixin method), 25  
 torrents\_add\_trackers() (qbittorrentapi.torrents.TorrentsAPIMixin method), 25  
 torrents\_bottom\_priority() (qbittorrentapi.torrents.TorrentsAPIMixin method), 26  
 torrents\_categories() (qbittorrentapi.torrents.TorrentsAPIMixin method), 26  
 torrents\_create\_category() (qbittorrentapi.torrents.TorrentsAPIMixin method), 26  
 torrents\_create\_tags() (qbittorrentapi.torrents.TorrentsAPIMixin method), 26  
 torrents\_decrease\_priority() (qbittorrentapi.torrents.TorrentsAPIMixin method), 26  
 torrents\_delete() (qbittorrentapi.torrents.TorrentsAPIMixin method), 26  
 torrents\_delete\_tags() (qbittorrentapi.torrents.TorrentsAPIMixin method), 26  
 torrents\_download\_limit() (qbittorrentapi.torrents.TorrentsAPIMixin method), 27  
 torrents\_edit\_category() (qbittorrentapi.torrents.TorrentsAPIMixin method), 27  
 torrents\_edit\_tracker() (qbittorrentapi.torrents.TorrentsAPIMixin method),

27	<code>torrents_file_priority()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),	30	<code>torrents_set_category()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),
27	<code>torrents_files()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),	31	<code>torrents_set_download_limit()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),
27	<code>torrents_increase_priority()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),	31	<code>torrents_set_force_start()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),
28	<code>torrents_info()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),	31	<code>torrents_set_location()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),
28	<code>torrents_pause()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),	31	<code>torrents_set_share_limits()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),
28	<code>torrents_piece_hashes()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),	31	<code>torrents_set_super_seeding()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),
28	<code>torrents_piece_states()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),	32	<code>torrents_set_upload_limit()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),
28	<code>torrents_properties()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),	32	<code>torrents_tags()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),
29	<code>torrents_reannounce()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),		<code>torrents_toggle_first_last_piece_priority()</code> ( <i>qbittorrentapi.torrents.TorrentsAPIMixin</i> <i>method</i> ), 32	
29	<code>torrents_recheck()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),		<code>torrents_toggle_sequential_download()</code> ( <i>qbittorrentapi.torrents.TorrentsAPIMixin</i> <i>method</i> ), 32	
29	<code>torrents_remove_categories()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),	32	<code>torrents_top_priority()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),
29	<code>torrents_remove_tags()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),	32	<code>torrents_trackers()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),
29	<code>torrents_remove_trackers()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),	32	<code>torrents_upload_limit()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),
29	<code>torrents_rename()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),	33	<code>torrents_webseeds()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),
30	<code>torrents_rename_file()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),		<code>TorrentsAddPeersDictionary</code> (class in <i>qbittorrent</i> - <i>rentapi.torrents</i> ), 37	
30	<code>torrents_rename_folder()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),	24	<code>TorrentsAPIMixin</code> (class in <i>qbittorrentapi.torrents</i> ),	
30	<code>torrents_resume()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),	23	<code>TorrentStates</code> (class in <i>qbittorrentapi.definitions</i> ),	
	<code>torrents_set_auto_management()</code> <i>rentapi.torrents.TorrentsAPIMixin</i>	( <i>qbittorrent</i> - <i>method</i> ),		<code>TorrentTags</code> (class in <i>qbittorrentapi.torrents</i> ), 36	
				<code>Tracker</code> (class in <i>qbittorrentapi.torrents</i> ), 38	
				<code>trackers</code> ( <i>qbittorrentapi.torrents.TorrentDictionary</i> attribute), 35	
				<code>TrackersList</code> (class in <i>qbittorrentapi.torrents</i> ), 37	

Transfer (class in *qbittorrentapi.transfer*), 39  
 Transfer (*qbittorrentapi.definitions.APINames* attribute), 13  
 transfer\_ban\_peers() (*qbittorrentapi.transfer.TransferAPIMixin* method), 38  
 transfer\_download\_limit() (*qbittorrentapi.transfer.TransferAPIMixin* method), 38  
 transfer\_info() (*qbittorrentapi.transfer.TransferAPIMixin* method), 38  
 transfer\_set\_download\_limit() (*qbittorrentapi.transfer.TransferAPIMixin* method), 38  
 transfer\_set\_upload\_limit() (*qbittorrentapi.transfer.TransferAPIMixin* method), 39  
 transfer\_speed\_limits\_mode() (*qbittorrentapi.transfer.TransferAPIMixin* method), 39  
 transfer\_toggle\_speed\_limits\_mode() (*qbittorrentapi.transfer.TransferAPIMixin* method), 39  
 transfer\_upload\_limit() (*qbittorrentapi.transfer.TransferAPIMixin* method), 39  
 TransferAPIMixin (class in *qbittorrentapi.transfer*), 38  
 TransferInfoDictionary (class in *qbittorrentapi.transfer*), 40

## U

Unauthorized401Error, 8  
 uninstall\_plugin() (*qbittorrentapi.search.Search* method), 20  
 UNKNOWN (*qbittorrentapi.definitions.TorrentStates* attribute), 23  
 UnsupportedMediaType415Error, 9  
 update\_plugins() (*qbittorrentapi.search.Search* method), 20  
 upload\_limit (*qbittorrentapi.torrents.TorrentDictionary* attribute), 36  
 upload\_limit (*qbittorrentapi.transfer.Transfer* attribute), 40  
 UPLOADING (*qbittorrentapi.definitions.TorrentStates* attribute), 23

## V

version (*qbittorrentapi.app.Application* attribute), 10

## W

web\_api\_version (*qbittorrentapi.app.Application*