qbittorrent-api

Feb 07, 2021

Contents

CHAPTER 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.

1.1 Features

- The entire qBittorent 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 many 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 qBittorent 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 many 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_UNIMPLEMENT`

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 qBittorent 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

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

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

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

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

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 emtpy for status of all jobs

Returns *SearchStatusesList* - 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

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

class qbittorrentapi.sync.**Sync**(*client*)
Alows 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
- 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_UPLOAD = 'forcedUP'

FORCE_DOWNLOAD = 'forceDL'

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*, *content_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

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

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 'l')

Returns *TorrentInfoList* - 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

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

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

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*. Alows 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

Alows 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='/hone/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

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*)
Alows 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()*

Python Module Index

q

Index

D

E

F

H

I

L

M

N

P

Q

R

S

T

U

V

W