
qbittorrent-api

Release 2024.9.67

Russell Martin

Sep 30, 2024

CONTENTS

1	Introduction	1
	Python Module Index	77
	Index	79

INTRODUCTION

Python client implementation for qBittorrent Web API.

Currently supports qBittorrent [v5.0.0](#) (Web API v2.11.2) released on Sept 29, 2024.

1.1 Features

- The entire qBittorrent [Web API](#) is implemented.
- qBittorrent version checking for an endpoint's existence/features is automatically handled.
- 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](#):

```
python -m pip install qbittorrent-api
```

- Install a specific release (e.g. [v2024.3.60](#)):

```
python -m pip install qbittorrent-api==2024.3.60
```

- Install direct from main:

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

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

1.3 Getting Started

```
import qbittorrentapi

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

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

# if the Client will not be long-lived or many Clients may be created
# in a relatively short amount of time, be sure to log out:
qbt_client.auth_log_out()

# or use a context manager:
with qbittorrentapi.Client(**conn_info) as qbt_client:
    if qbt_client.torrents_add(urls="...") != "Ok.":
        raise Exception("Failed to add torrent.")

# 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})")

# stop all torrents
qbt_client.torrents.stop.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.stop.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/')

```

(continues on next page)

(continued from previous page)

```
torrent.reannounce()  
torrent.upload_limit = -1
```

1.4.1 Introduction

Python client implementation for qBittorrent Web API.

Currently supports qBittorrent [v5.0.0](#) (Web API v2.11.2) released on Sept 29, 2024.

Features

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

Installation

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

```
python -m pip install qbittorrent-api
```

- Install a specific release (e.g. [v2024.3.60](#)):

```
python -m pip install qbittorrent-api==2024.3.60
```

- Install direct from main:

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

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

Getting Started

```
import qbittorrentapi  
  
# instantiate a Client using the appropriate WebUI configuration  
conn_info = dict(  
    host="localhost",  
    port=8080,  
    username="admin",  
    password="adminadmin",  
)  
qbt_client = qbittorrentapi.Client(**conn_info)
```

(continues on next page)

(continued from previous page)

```

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

# if the Client will not be long-lived or many Clients may be created
# in a relatively short amount of time, be sure to log out:
qbt_client.auth_log_out()

# or use a context manager:
with qbittorrentapi.Client(**conn_info) as qbt_client:
    if qbt_client.torrents_add(urls="...") != "Ok.":
        raise Exception("Failed to add torrent.")

# 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})")

# stop all torrents
qbt_client.torrents.stop.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=

```

(continues on next page)

(continued from previous page)

```

↪ '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.stop.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

Host, Username and Password

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

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

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

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

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

qBittorrent Session Management

- Any time a connection is established with qBittorrent, it instantiates a session to manage authentication for all subsequent API requests.
- This client will transparently manage sessions by ensuring the client is always logged in in-line with any API request including requesting a new session upon expiration of an existing session.
- However, each new `Client` instantiation will create a new session in qBittorrent.
- Therefore, if many `Client` instances will be created be sure to call `auth_log_out` for each instance or use a context manager.
- Otherwise, qBittorrent may experience abnormally high memory usage.

```
with qbittorrentapi.Client(**conn_info) as qbt_client:
    if qbt_client.torrents_add(urls="...") != "Ok.":
        raise Exception("Failed to add torrent.")
```

Untrusted Web API Certificate

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

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

Requests Configuration

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

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

- Alternatively, parameters can be specified for individual requests:

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

- Additionally, configuration for the `HTTPAdapter` for the `Session` can be specified via the `HTTPADAPTER_ARGS` parameter for `Client`:

```
qbt_client = Client(..., HTTPADAPTER_ARGS={"pool_connections": 100, "pool_maxsize": 100})
```

Additional HTTP Headers

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

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

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

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

Unimplemented API Endpoints

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

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

qBittorrent Version Checking

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

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

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

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

Disable Logging Debug Output

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

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

1.4.3 Performance

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

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

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

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

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

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

1.4.4 Exceptions

exception `APIError`

Bases: `Exception`

Base error for all exceptions from this Client.

exception `UnsupportedQbittorrentVersion`

Bases: `APIError`

Connected qBittorrent is not fully supported by this Client.

exception `FileError`

Bases: `OSError`, `APIError`

Base class for all exceptions for file handling.

exception `TorrentFileError`

Bases: `FileError`

Base class for all exceptions for torrent files.

exception `TorrentFileNotFoundError`

Bases: `TorrentFileError`

Specified torrent file does not appear to exist.

exception `TorrentFilePermissionError`

Bases: `TorrentFileError`

Permission was denied to read the specified torrent file.

exception `APIConnectionError`(*args, **kwargs)

Bases: `RequestException`, `APIError`

Base class for all communications errors including HTTP errors.

exception `LoginFailed`(*args, **kwargs)

Bases: `APIConnectionError`

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

exception `HTTPError`(*args, **kwargs)

Bases: `HTTPError`, `APIConnectionError`

Base error for all HTTP errors.

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

http_status_code: `int`

exception `HTTP4XXError`(*args, **kwargs)

Bases: `HTTPError`

Base error for all HTTP 4XX statuses.

exception `HTTP5XXError`(*args, **kwargs)

Bases: `HTTPError`

Base error for all HTTP 5XX statuses.

exception `HTTP400Error`(*args, **kwargs)

Bases: `HTTP4XXError`

HTTP 400 Status.

http_status_code: `int = 400`

exception `HTTP401Error`(*args, **kwargs)

Bases: `HTTP4XXError`

HTTP 401 Status.

http_status_code: `int = 401`

exception `HTTP403Error`(*args, **kwargs)

Bases: `HTTP4XXError`

HTTP 403 Status.

http_status_code: `int = 403`

exception HTTP404Error(*args, **kwargs)

Bases: [HTTP4XXError](#)

HTTP 404 Status.

http_status_code: `int` = 404

exception HTTP405Error(*args, **kwargs)

Bases: [HTTP4XXError](#)

HTTP 405 Status.

http_status_code: `int` = 405

exception HTTP409Error(*args, **kwargs)

Bases: [HTTP4XXError](#)

HTTP 409 Status.

http_status_code: `int` = 409

exception HTTP415Error(*args, **kwargs)

Bases: [HTTP4XXError](#)

HTTP 415 Status.

http_status_code: `int` = 415

exception HTTP500Error(*args, **kwargs)

Bases: [HTTP5XXError](#)

HTTP 500 Status.

http_status_code: `int` = 500

exception MissingRequiredParameters400Error(*args, **kwargs)

Bases: [HTTP400Error](#)

Endpoint call is missing one or more required parameters.

exception InvalidRequest400Error(*args, **kwargs)

Bases: [HTTP400Error](#)

One or more endpoint arguments are malformed.

exception Unauthorized401Error(*args, **kwargs)

Bases: [HTTP401Error](#)

Primarily reserved for XSS and host header issues.

exception Forbidden403Error(*args, **kwargs)

Bases: [HTTP403Error](#)

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

exception NotFound404Error(*args, **kwargs)

Bases: [HTTP404Error](#)

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

exception MethodNotAllowed405Error(*args, **kwargs)

Bases: [HTTP405Error](#)

HTTP method is not supported for the API endpoint.

exception Conflict409Error(*args, **kwargs)

Bases: [HTTP409Error](#)

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

exception UnsupportedMediaType415Error(*args, **kwargs)

Bases: [HTTP415Error](#)

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

exception InternalServerError500Error(*args, **kwargs)

Bases: [HTTP500Error](#)

Returned if qBittorrent errors internally while processing the request.

1.4.5 API Reference

Application

class AppAPIMixin(host=None, port=None, username=None, password=None, EXTRA_HEADERS=None, REQUESTS_ARGS=None, HTTPADAPTER_ARGS=None, VERIFY_WEBUI_CERTIFICATE=True, FORCE_SCHEME_FROM_HOST=False, RAISE_NOTIMPLEMENTEDERROR_FOR_UNIMPLEMENTED_API_ENDPOINTS=False, RAISE_ERROR_FOR_UNSUPPORTED_QBITTORRENT_VERSIONS=False, VERBOSE_RESPONSE_LOGGING=False, SIMPLE_RESPONSES=False, DISABLE_LOGGING_DEBUG_OUTPUT=False) → None

Bases: [AuthAPIMixin](#)

Implementation of all Application API methods.

Usage

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

app_build_info(**kwargs) → [BuildInfoDictionary](#)

qBittorrent build info.

This method was introduced with qBittorrent v4.2.0 (Web API v2.3).

Return type

[BuildInfoDictionary](#)

app_default_save_path(**kwargs) → str

The default path where torrents are saved.

Return type

str

app_get_directory_content(*directory_path=None*) → *DirectoryContentList*

The contents of a directory file path.

Raises

NotFound404Error – file path not found or not a directory

Parameters

directory_path (*str* | *PathLike[AnyStr]* | *None*) – file system path to directory

Return type

DirectoryContentList

app_network_interface_address_list(*interface_name=""*, ***kwargs*) → *NetworkInterfaceAddressList*

The addresses for a network interface; omit name for all addresses.

This method was introduced with qBittorrent v4.2.0 (Web API v2.3).

Parameters

interface_name (*str*) – Name of interface to retrieve addresses for

Return type

NetworkInterfaceAddressList

app_network_interface_list(***kwargs*) → *NetworkInterfaceList*

The list of network interfaces on the host.

This method was introduced with qBittorrent v4.2.0 (Web API v2.3).

Return type

NetworkInterfaceList

app_preferences(***kwargs*) → *ApplicationPreferencesDictionary*

Retrieve qBittorrent application preferences.

Return type

ApplicationPreferencesDictionary

app_send_test_email() → *None*

Sends a test email using the configured email address.

Return type

None

app_set_preferences(*prefs=None*, ***kwargs*) → *None*

Set one or more preferences in qBittorrent application.

Parameters

prefs (*Mapping[str, Any]* | *None*) – dictionary of preferences to set

Return type

None

app_shutdown(***kwargs*) → *None*

Shutdown qBittorrent.

Return type

None

app_version(***kwargs*) → *str*

qBittorrent application version.

Return type

str

app_web_api_version(**kwargs) → str

qBittorrent Web API version.

Return type

str

class Application(*args, client, **kwargs)

Allows interaction with Application API endpoints.

Usage

```
>>> from qbittorrentapi import Client
>>> client = Client(host="localhost:8080", username="admin", password=
↳ "adminadmin")
>>> # these are all the same attributes that are available as named in
↳ the
>>> # endpoints or the more pythonic names in Client (with or without
↳ 'app_' prepended)
>>> webapiVersion = client.application.webapiVersion
>>> web_api_version = client.application.web_api_version
>>> app_web_api_version = client.application.web_api_version
>>> # access and set preferences as attributes
>>> is_dht_enabled = client.application.preferences.dht
>>> # supports sending a just subset of preferences to update
>>> client.application.preferences = dict(dht=(not is_dht_enabled))
>>> prefs = client.application.preferences
>>> prefs["web_ui_clickjacking_protection_enabled"] = True
>>> client.app.preferences = prefs
>>>
>>> client.application.shutdown()
```

property build_info: *BuildInfoDictionary*

Implements *app_build_info()*.

property default_save_path: str

Implements *app_default_save_path()*.

get_directory_content(directory_path=None) → *DirectoryContentList*

Implements *app_get_directory_content()*.

Return type

DirectoryContentList

network_interface_address_list(interface_name="", **kwargs) → *NetworkInterfaceAddressList*

Implements *app_network_interface_address_list()*.

Return type

NetworkInterfaceAddressList

property network_interface_list: *NetworkInterfaceList*

Implements *app_network_interface_list()*.

property preferences: *ApplicationPreferencesDictionary*

Implements *app_preferences()*.

send_test_email() → None

Implements *app_send_test_email()*.

Return type

None

set_preferences(*prefs=None*, ***kwargs*) → NoneImplements [app_set_preferences\(\)](#).**Return type**

None

shutdown(***kwargs*) → NoneImplements [app_shutdown\(\)](#).**Return type**

None

property version: **str**Implements [app_version\(\)](#).**property web_api_version:** **str**Implements [app_web_api_version\(\)](#).**class ApplicationPreferencesDictionary**(*data=None*, ***kwargs*)Bases: [Dictionary](#)[None | int | str | bool | Sequence[JsonValueT] | Mapping[str, JsonValueT]]Response for [app_preferences\(\)](#)Definition: [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#user-content-get-application-preferences)
#user-content-get-application-preferences**class BuildInfoDictionary**(*data=None*, ***kwargs*)Bases: [Dictionary](#)[str | int]Response for [app_build_info\(\)](#)Definition: [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#user-content-get-build-info)
#user-content-get-build-info**class DirectoryContentList**(*list_entries*, *client=None*)Bases: [List](#)[str]Response for [app_get_directory_content\(\)](#)**class NetworkInterfaceList**(*list_entries*, *client=None*)Bases: [List](#)[[NetworkInterface](#)]Response for [app_network_interface_list\(\)](#)**class NetworkInterface**(*data=None*, ***kwargs*)Bases: [ListEntry](#)Item in [NetworkInterfaceList](#)**class NetworkInterfaceAddressList**(*list_entries*, *client=None*)Bases: [List](#)[str]Response for [app_network_interface_address_list\(\)](#)

AttrDict (internal)

Copyright (c) 2013 Brendan Curran-Johnson

class `AttrDict(*args, **kwargs) → None`

Bases: `Dict[str, V]`, `MutableAttr[V]`

A dict that implements `MutableAttr`.

class `MutableAttr`

Bases: `Attr[str, V]`, `MutableMapping[str, V]`, `ABC`

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

class `Attr`

Bases: `Mapping[K, V]`, `ABC`

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

A key may be used as an attribute if:

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

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

NOTE:

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

Authentication

class `AuthAPIMixin(host=None, port=None, username=None, password=None, EXTRA_HEADERS=None, REQUESTS_ARGS=None, HTTPADAPTER_ARGS=None, VERIFY_WEBUI_CERTIFICATE=True, FORCE_SCHEME_FROM_HOST=False, RAISE_NOTIMPLEMENTEDERROR_FOR_UNIMPLEMENTED_API_ENDPOINTS=False, RAISE_ERROR_FOR_UNSUPPORTED_QBITTORRENT_VERSIONS=False, VERBOSE_RESPONSE_LOGGING=False, SIMPLE_RESPONSES=False, DISABLE_LOGGING_DEBUG_OUTPUT=False) → None`

Bases: `Request`

Implementation of all Authorization API methods.

Usage

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

auth_log_in(*username=None, password=None, **kwargs*) → *None*

Log in to qBittorrent host.

Raises

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

Parameters

- **username** (*str* | *None*) – username for qBittorrent client
- **password** (*str* | *None*) – password for qBittorrent client

Return type

None

auth_log_out(***kwargs*) → *None*

End session with qBittorrent.

Return type

None

property is_logged_in: *bool*

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

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

Returns

True/False if current authorization cookie is accepted by qBittorrent

class Authorization(**args, client, **kwargs*)

Allows interaction with the Authorization API endpoints.

Usage

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

property is_logged_in: *bool*

Implements *is_logged_in()*.

log_in(*username=None, password=None, **kwargs*) → *None*

Implements *auth_log_in()*.

Return type

None

log_out(***kwargs*) → *None*

Implements *auth_log_out()*.

Return type

None

Client

```
class Client(host="", port=None, username=None, password=None, *, EXTRA_HEADERS=None,
            REQUESTS_ARGS=None, HTTPADAPTER_ARGS=None,
            VERIFY_WEBUI_CERTIFICATE=True, FORCE_SCHEME_FROM_HOST=False,
            RAISE_NOTIMPLEMENTEDERROR_FOR_UNIMPLEMENTED_API_ENDPOINTS=False,
            RAISE_ERROR_FOR_UNSUPPORTED_QBITTORRENT_VERSIONS=False,
            VERBOSE_RESPONSE_LOGGING=False, SIMPLE_RESPONSES=False,
            DISABLE_LOGGING_DEBUG_OUTPUT=False)
```

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

Initialize API for qBittorrent client.

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

Usage

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

Parameters

- **host** ([str](#)) – hostname for qBittorrent Web API, [http[s]://]hostname[:port][/
path]
- **port** ([str](#) | [int](#) | [None](#)) – port number for qBittorrent Web API (ignored if host contains a
port)
- **username** ([str](#) | [None](#)) – username for qBittorrent Web API
- **password** ([str](#) | [None](#)) – password for qBittorrent Web API
- **SIMPLE_RESPONSES** ([bool](#)) – By default, complex objects are returned from some end-
points. These objects will allow for accessing responses' items as attributes and in-
clude 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(torrent_hash='...
' , SIMPLE_RESPONSES=True)`
- **VERIFY_WEBUI_CERTIFICATE** ([bool](#)) – 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.
- **EXTRA_HEADERS** ([Mapping](#)[[str](#), [str](#)] | [None](#)) – Dictionary of HTTP Headers to include in
all requests made to qBittorrent.
- **REQUESTS_ARGS** ([Mapping](#)[[str](#), [Any](#)] | [None](#)) – Dictionary of configuration for each HTTP
request made by [requests.request](#).
- **HTTPADAPTER_ARGS** ([Mapping](#)[[str](#), [Any](#)] | [None](#)) – Dictionary of configuration for
[HTTPAdapter](#).
- **FORCE_SCHEME_FROM_HOST** ([bool](#)) – If a scheme (i.e. http or https) is specified in host,
it will be used regardless of whether qBittorrent is configured for HTTP or HTTPS commu-

nication. Normally, this client will attempt to determine which scheme qBittorrent is actually listening on... but this can cause problems in rare cases. Defaults False.

- **RAISE_NOTIMPLEMENTEDERROR_FOR_UNIMPLEMENTED_API_ENDPOINTS** (*bool*) – Some endpoints may not be implemented in older versions of qBittorrent. Setting this to True will raise a *NotImplementedError* instead of just returning None.
- **RAISE_ERROR_FOR_UNSUPPORTED_QBITTORRENT_VERSIONS** (*bool*) – Raise *UnsupportedQbittorrentVersion* if the connected version of qBittorrent is not fully supported by this client. Defaults False.
- **DISABLE_LOGGING_DEBUG_OUTPUT** (*bool*) – Turn off debug output from logging for this package as well as requests & urllib3.

Definitions

APIKwargsT

Type Any for kwargs parameters for API methods.

class APINames(*value, names=<not given>, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: *str, 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'

TorrentCreator = 'torrentcreator'

Torrents = 'torrents'

Transfer = 'transfer'

class ClientCache(**args, client, **kwargs*)

Bases: *Generic[ClientT]*

Caches the client.

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

class ClientT

Type for this API Client.

alias of `TypeVar('ClientT', bound=Request)`

class Dictionary(data=None, **kwargs)

Bases: `AttrDict[V]`

Base definition of dictionary-like objects returned from qBittorrent.

FilesToSendT

Type for Files input to API method.

alias of `Mapping[str, bytes | Tuple[str, bytes]]`

JsonValueT

Type to define JSON.

alias of `None | int | str | bool | Sequence[JsonValueT] | Mapping[str, JsonValueT]`

class List(list_entries=None, entry_class=None, **kwargs)

Bases: `UserList[ListEntryT]`

Base definition for list-like objects returned from qBittorrent.

class ListEntry(data=None, **kwargs)

Bases: `Dictionary[None | int | str | bool | Sequence[JsonValueT] | Mapping[str, JsonValueT]]`

Base definition for objects within a list returned from qBittorrent.

class ListEntryT

Type for entry in List from API.

alias of `TypeVar('ListEntryT', bound=ListEntry)`

ListInputT

Type for List input to API method.

alias of `Iterable[Mapping[str, None | int | str | bool | Sequence[JsonValueT] | Mapping[str, JsonValueT]]]`

class TorrentState(value, names=<not given>, *values, module=None, qualname=None, type=None, start=1, boundary=None)

Bases: `str, Enum`

Torrent States as defined by qBittorrent.

Note: In qBittorrent v5.0.0:

- PAUSED_UPLOAD was renamed to STOPPED_UPLOAD
- PAUSED_DOWNLOAD was renamed to STOPPED_DOWNLOAD

Definitions:

- wiki: [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#user-content-get-torrent-list](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#user-content-get-torrent-list)
- code: <https://github.com/qbittorrent/qBittorrent/blob/8e6515be2c8cc2b335002ab8913e9dcdd7873204/src/base/bittorrent/torrent.h#L79>

Usage

```
>>> from qbittorrentapi import Client, TorrentState
>>> client = Client()
>>> # print torrent hashes for torrents that are downloading
>>> for torrent in client.torrents_info():
```

(continues on next page)

(continued from previous page)

```

>>> # 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 = TorrentState(torrent.state)
>>> print(f'{torrent.hash}: {state_enum.value}')

```

ALLOCATING = 'allocating'

CHECKING_DOWNLOAD = 'checkingDL'

CHECKING_RESUME_DATA = 'checkingResumeData'

CHECKING_UPLOAD = 'checkingUP'

DOWNLOADING = 'downloading'

ERROR = 'error'

FORCED_DOWNLOAD = 'forcedDL'

FORCED_METADATA_DOWNLOAD = 'forcedMetaDL'

FORCED_UPLOAD = 'forcedUP'

METADATA_DOWNLOAD = 'metaDL'

MISSING_FILES = 'missingFiles'

MOVING = 'moving'

PAUSED_DOWNLOAD = 'pausedDL'

pausedDL was renamed to stoppedDL in Web API v2.11.0

PAUSED_UPLOAD = 'pausedUP'

pausedUP was renamed to stoppedUP in Web API v2.11.0

QUEUED_DOWNLOAD = 'queuedDL'

QUEUED_UPLOAD = 'queuedUP'

STALLED_DOWNLOAD = 'stalledDL'

STALLED_UPLOAD = 'stalledUP'

STOPPED_DOWNLOAD = 'stoppedDL'

STOPPED_UPLOAD = 'stoppedUP'

UNKNOWN = 'unknown'

UPLOADING = 'uploading'

property is_checking: bool

Returns True if the State is categorized as Checking.

property is_complete: bool

Returns True if the State is categorized as Complete.

property is_downloading: `bool`

Returns True if the State is categorized as Downloading.

property is_errored: `bool`

Returns True if the State is categorized as Errored.

property is_paused: `bool`

Alias of `TorrentState.is_stopped`

property is_stopped: `bool`

Returns True if the State is categorized as Stopped.

property is_uploading: `bool`

Returns True if the State is categorized as Uploading.

class TrackerStatus(*value, names=<not given>, *values, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `int, Enum`

Tracker Statuses as defined by qBittorrent.

Definitions:

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

Usage

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

DISABLED = 0

NOT_CONTACTED = 1

NOT_WORKING = 4

UPDATING = 3

WORKING = 2

property display: `str`

Returns a descriptive display value for status.

Log

```
class LogAPIMixin(host=None, port=None, username=None, password=None, EXTRA_HEADERS=None,
REQUESTS_ARGS=None, HTTPADAPTER_ARGS=None,
VERIFY_WEBUI_CERTIFICATE=True, FORCE_SCHEME_FROM_HOST=False,
RAISE_NOTIMPLEMENTEDERROR_FOR_UNIMPLEMENTED_API_ENDPOINTS=False,
RAISE_ERROR_FOR_UNSUPPORTED_QBITTORRENT_VERSIONS=False,
VERBOSE_RESPONSE_LOGGING=False, SIMPLE_RESPONSES=False,
DISABLE_LOGGING_DEBUG_OUTPUT=False) → None
```

Bases: [AppAPIMixin](#)

Implementation of all Log API methods.

Usage

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

```
log_main(normal=None, info=None, warning=None, critical=None, last_known_id=None, **kwargs) → LogMainList
```

Retrieve the qBittorrent log entries. Iterate over returned object.

Parameters

- **normal** (*bool* | *None*) – False to exclude normal entries
- **info** (*bool* | *None*) – False to exclude info entries
- **warning** (*bool* | *None*) – False to exclude warning entries
- **critical** (*bool* | *None*) – False to exclude critical entries
- **last_known_id** (*str* | *int* | *None*) – only entries with an ID greater than this value will be returned

Return type

[LogMainList](#)

```
log_peers(last_known_id=None, **kwargs) → LogPeersList
```

Retrieve qBittorrent peer log.

Parameters

last_known_id (*str* | *int* | *None*) – only entries with an ID greater than this value will be returned

Return type

[LogPeersList](#)

```
class Log(client)
```

Allows interaction with Log API endpoints.

Usage

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

(continues on next page)

(continued from previous page)

```
>>> # 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(last_known_id="...")
>>> # can also filter log down with additional attributes
>>> log_info = client.log.main.info(last_known_id=1)
>>> log_warning = client.log.main.warning(last_known_id=1)
```

class `Main(*args, client, **kwargs)`

`__call__`(*normal=True, info=True, warning=True, critical=True, last_known_id=None, **kwargs*) → *LogMainList*

Implements *log_main()*.

Return type

LogMainList

critical(*last_known_id=None, **kwargs*) → *LogMainList*

Implements *log_main()* with *info=False, normal=False, and warning=False*.

Return type

LogMainList

info(*last_known_id=None, **kwargs*) → *LogMainList*

Implements *log_main()*.

Return type

LogMainList

normal(*last_known_id=None, **kwargs*) → *LogMainList*

Implements *log_main()* with *info=False*.

Return type

LogMainList

warning(*last_known_id=None, **kwargs*) → *LogMainList*

Implements *log_main()* with *info=False and normal=False*.

Return type

LogMainList

property `main`: *Main*

Implements *log_main()*.

peers(*last_known_id=None, **kwargs*) → *LogPeersList*

Implements *log_peers()*.

Return type

LogPeersList

class `LogPeersList(list_entries, client=None)`

Bases: *List[LogPeer]*

Response for *log_peers()*

class `LogPeer(data=None, **kwargs)`

Bases: *ListEntry*

Item in *LogPeersList*

class LogMainList(*list_entries*, *client=None*)

Bases: [List\[LogEntry\]](#)

Response to [log_main\(\)](#)

class LogEntry(*data=None*, ***kwargs*)

Bases: [ListEntry](#)

Item in [LogMainList](#)

Request (internal)

class QbittorrentSession

Bases: [Session](#)

Wrapper to augment Requests Session.

Requests doesn't allow Session to default certain configuration globally. This gets around that by setting defaults for each request.

request(*method*, *url*, ***kwargs*) → [Response](#)

Constructs a [Request](#), prepares it and sends it. Returns Response object.

Parameters

- **method** ([str](#)) – method for the new [Request](#) object.
- **url** ([str](#)) – URL for the new [Request](#) object.
- **params** – (optional) Dictionary or bytes to be sent in the query string for the [Request](#).
- **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the [Request](#).
- **json** – (optional) json to send in the body of the [Request](#).
- **headers** – (optional) Dictionary of HTTP Headers to send with the [Request](#).
- **cookies** – (optional) Dict or CookieJar object to send with the [Request](#).
- **files** – (optional) Dictionary of 'filename': file-like-objects for multipart encoding upload.
- **auth** – (optional) Auth tuple or callable to enable Basic/Digest/Custom HTTP Auth.
- **timeout** ([float](#) or [tuple](#)) – (optional) How long to wait for the server to send data before giving up, as a float, or a ([connect timeout](#), [read timeout](#)) tuple.
- **allow_redirects** ([bool](#)) – (optional) Set to True by default.
- **proxies** – (optional) Dictionary mapping protocol or protocol and hostname to the URL of the proxy.
- **hooks** – (optional) Dictionary mapping hook name to one event or list of events, event must be callable.
- **stream** – (optional) whether to immediately download the response content. Defaults to False.
- **verify** – (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use. Defaults to True. When set to False, requests will accept any TLS certificate presented by the server, and will ignore hostname mismatches and/or expired certificates, which will

make your application vulnerable to man-in-the-middle (MitM) attacks. Setting `verify` to `False` may be useful during local development or testing.

- **cert** – (optional) if String, path to ssl client cert file (.pem). If Tuple, ('cert', 'key') pair.

Return type

Response

class `QbittorrentURL`(*client*)

Bases: `object`

Management for the qBittorrent Web API URL.

build(*api_namespace*, *api_method*, *headers=None*, *requests_kwargs=None*, *base_path='api/v2'*) → `str`

Create a fully qualified URL for the API endpoint.

Parameters

- **api_namespace** (*APINames* | `str`) – the namespace for the API endpoint (e.g. `torrents`)
- **api_method** (`str`) – the specific method for the API endpoint (e.g. `info`)
- **base_path** (`str`) – base path for URL (e.g. `api/v2`)
- **headers** (`Mapping[str, str]` | `None`) – HTTP headers for request
- **requests_kwargs** (`Mapping[str, Any]` | `None`) – kwargs for any calls to Requests

Return type

`str`

Returns

fully qualified URL string for endpoint

build_base_url(*headers*, *requests_kwargs*) → `str`

Determine the Base URL for the Web API endpoints.

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

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

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

Parameters

- **headers** (`Mapping[str, str]`) – HTTP headers for request
- **requests_kwargs** (`Mapping[str, Any]`) – arguments from user for HTTP HEAD request

Return type

`str`

Returns

base URL as a string for Web API endpoint

detect_scheme(*base_url*, *default_scheme*, *alt_scheme*, *headers*, *requests_kwargs*) → `str`

Determine if the URL endpoint is using HTTP or HTTPS.

Parameters

- **base_url** (`ParseResult`) – urllib `ParseResult` URL object

- **default_scheme** (`str`) – default scheme to use for URL
- **alt_scheme** (`str`) – alternative scheme to use for URL if default doesn't work
- **headers** (`Mapping[str, str]`) – HTTP headers for request
- **requests_kwargs** (`Mapping[str, Any]`) – kwargs for calls to Requests

Return type`str`**Returns**

scheme (i.e. HTTP or HTTPS)

class Request(*host=None, port=None, username=None, password=None, EXTRA_HEADERS=None, REQUESTS_ARGS=None, HTTPADAPTER_ARGS=None, VERIFY_WEBUI_CERTIFICATE=True, FORCE_SCHEME_FROM_HOST=False, RAISE_NOTIMPLEMENTEDERROR_FOR_UNIMPLEMENTED_API_ENDPOINTS=False, RAISE_ERROR_FOR_UNSUPPORTED_QBITTORRENT_VERSIONS=False, VERBOSE_RESPONSE_LOGGING=False, SIMPLE_RESPONSES=False, DISABLE_LOGGING_DEBUG_OUTPUT=False*) → `None`

Bases: `object`

Facilitates HTTP requests to qBittorrent's Web API.

_auth_request(*http_method, api_namespace, api_method, _retry_backoff_factor=0.3, requests_args=None, requests_params=None, headers=None, params=None, data=None, files=None, response_class=<class 'requests.models.Response'>, version_introduced="", version_removed="", **kwargs*) → `Any`

Wraps API call with re-authorization if first attempt is not authorized.

Return type`Any`

_cast(*response, response_class, **response_kwargs*) → `Any`

Returns the API response casted to the requested class.

Parameters

- **response** (`Response`) – requests Response from API
- **response_class** (`type`) – class to return response as; if none, response is returned
- **response_kwargs** (`Any`) – request-specific configuration for response

Return type`Any`

static _format_payload(*http_method, params=None, data=None, files=None, **kwargs*) → `tuple[dict[str, Any], dict[str, Any], Mapping[str, bytes | Tuple[str, bytes]]]`

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

Parameters

- **http_method** (`str`) – get or post
- **params** (`Mapping[str, Any] | None`) – key value pairs to send with GET calls
- **data** (`Mapping[str, Any] | None`) – key value pairs to send with POST calls
- **files** (`Mapping[str, bytes | Tuple[str, bytes]] | None`) – dictionary of files to send with request

Return type

`tuple[dict[str, Any], dict[str, Any], Mapping[str, bytes | Tuple[str, bytes]]]`

`_get(_name, _method, requests_args=None, requests_params=None, headers=None, params=None, data=None, files=None, response_class=<class 'requests.models.Response'>, version_introduced="", version_removed="", **kwargs) → Any`

Send GET request.

Parameters

- **api_namespace** – the namespace for the API endpoint (e.g. [APINames](#) or `torrents`)
- **api_method** – the name for the API endpoint (e.g. `add`)
- **kwargs** (*Any*) – see `_request()`

Return type

Any

Returns

Requests [Response](#)

`_get_cast(_name, _method, response_class, requests_args=None, requests_params=None, headers=None, params=None, data=None, files=None, version_introduced="", version_removed="", **kwargs) → ResponseT`

Send GET request with casted response.

Parameters

- **api_namespace** – the namespace for the API endpoint (e.g. [APINames](#) or `torrents`)
- **api_method** – the name for the API endpoint (e.g. `add`)
- **kwargs** (*Any*) – see `_request()`

Return type

`TypeVar(ResponseT)`

`static _handle_error_responses(data, params, response) → None`

Raise proper exception if qBittorrent returns Error HTTP Status.

Return type

None

`_initialize_context() → None`

Initialize and reset communications context with qBittorrent.

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

Return type

None

`_initialize_settings(EXTRA_HEADERS=None, REQUESTS_ARGS=None, HTTPADAPTER_ARGS=None, VERIFY_WEBUI_CERTIFICATE=True, FORCE_SCHEME_FROM_HOST=False, RAISE_NOTIMPLEMENTEDERROR_FOR_UNIMPLEMENTED_API_ENDPOINTS=False, RAISE_ERROR_FOR_UNSUPPORTED_QBITTORRENT_VERSIONS=False, VERBOSE_RESPONSE_LOGGING=False, SIMPLE_RESPONSES=False, DISABLE_LOGGING_DEBUG_OUTPUT=False) → None`

Initialize lesser used configuration.

Return type`None`**`_is_endpoint_supported_for_version(endpoint, version_introduced, version_removed)`** → `bool`

Prevent using an API methods that doesn't exist in this version of qBittorrent.

Parameters

- **endpoint** (`str`) – name of the removed endpoint, e.g. torrents/ban_peers
- **version_introduced** (`str`) – the Web API version the endpoint was introduced
- **version_removed** (`str`) – the Web API version the endpoint was removed

Return type`bool`**`classmethod _list2string(input_list, delimiter='')`** → `str` | `T`

Convert entries in a list to a concatenated string.

Parameters

- **input_list** (`TypeVar(T)`) – list to convert
- **delimiter** (`str`) – delimiter for concatenation

Return type`str` | `TypeVar(T)`**`_post(_name, _method, requests_args=None, requests_params=None, headers=None, params=None, data=None, files=None, response_class=<class 'requests.models.Response'>, version_introduced="", version_removed="", **kwargs)`** → `Any`

Send POST request.

Parameters

- **api_namespace** – the namespace for the API endpoint (e.g. [APINames](#) or torrents)
- **api_method** – the name for the API endpoint (e.g. add)
- **kwargs** (`Any`) – see [_request\(\)](#)

Return type`Any`**`_post_cast(_name, _method, response_class, requests_args=None, requests_params=None, headers=None, params=None, data=None, files=None, version_introduced="", version_removed="", **kwargs)`** → `ResponseT`

Send POST request with casted response.

Parameters

- **api_namespace** – the namespace for the API endpoint (e.g. [APINames](#) or torrents)
- **api_method** – the name for the API endpoint (e.g. add)
- **kwargs** (`Any`) – see [_request\(\)](#)

Return type`TypeVar(ResponseT)`**`_request(http_method, api_namespace, api_method, requests_args=None, requests_params=None, headers=None, params=None, data=None, files=None, response_class=<class 'requests.models.Response'>, **kwargs)`** → `Any`

Meat and potatoes of sending requests to qBittorrent.

Parameters

- **http_method** (`str`) – get or post
- **api_namespace** (`APINames` | `str`) – the namespace for the API endpoint (e.g. `APINames` or `torrents`)
- **api_method** (`str`) – the name for the API endpoint (e.g. `add`)
- **requests_args** (`Mapping[str, Any]` | `None`) – default location for Requests kwargs
- **requests_params** (`Mapping[str, Any]` | `None`) – alternative location for Requests kwargs
- **headers** (`Mapping[str, str]` | `None`) – HTTP headers to send with the request
- **params** (`Mapping[str, Any]` | `None`) – key/value pairs to send with a GET request
- **data** (`Mapping[str, Any]` | `None`) – key/value pairs to send with a POST request
- **files** (`Mapping[str, bytes | Tuple[str, bytes]]` | `None`) – files to be sent with the request
- **response_class** (`type`) – class to use to cast the API response
- **kwargs** (`Any`) – arbitrary keyword arguments to send with the request

Return type`Any`

_request_manager (`http_method`, `api_namespace`, `api_method`, `_retries=1`, `_retry_backoff_factor=0.3`, `requests_args=None`, `requests_params=None`, `headers=None`, `params=None`, `data=None`, `files=None`, `response_class=<class 'requests.models.Response'>`, `version_introduced=""`, `version_removed=""`, `**kwargs`) → `Any`

Wrapper to manage request retries and severe exceptions.

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

Return type`Any`

property _session: `QbittorrentSession`

Create or return existing HTTP session.

_trigger_session_initialization() → `None`

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

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

Return type`None`

_verbose_logging (`url`, `data`, `params`, `requests_kwargs`, `response`) → `None`

Log verbose information about request; can be useful during development.

Return type`None`

RSS

```
class RSSAPIMixin(host=None, port=None, username=None, password=None, EXTRA_HEADERS=None,
    REQUESTS_ARGS=None, HTTPADAPTER_ARGS=None,
    VERIFY_WEBUI_CERTIFICATE=True, FORCE_SCHEME_FROM_HOST=False,
    RAISE_NOTIMPLEMENTEDERROR_FOR_UNIMPLEMENTED_API_ENDPOINTS=False,
    RAISE_ERROR_FOR_UNSUPPORTED_QBITTORRENT_VERSIONS=False,
    VERBOSE_RESPONSE_LOGGING=False, SIMPLE_RESPONSES=False,
    DISABLE_LOGGING_DEBUG_OUTPUT=False) → None
```

Bases: [AppAPIMixin](#)

Implementation of all RSS API methods.

Usage

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

rss_add_feed(url=None, item_path="", **kwargs) → None

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

Raises

[Conflict409Error](#) –

Parameters

- **url** ([str](#) | None) – URL of RSS feed (e.g. <https://distrowatch.com/news/torrents.xml>)
- **item_path** ([str](#)) – Name and/or path for new feed; defaults to the URL. (e.g. Folder\Subfolder\FeedName)

Return type

None

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

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

Raises

[Conflict409Error](#) –

Parameters

folder_path ([str](#) | None) – path to new folder (e.g. Linux\ISOs)

Return type

None

rss_items(include_feed_data=None, **kwargs) → [RSSItemsDictionary](#)

Retrieve RSS items and optionally feed data.

Parameters

include_feed_data ([bool](#) | None) – True or false to include feed data

Return type

[RSSItemsDictionary](#)

rss_mark_as_read(*item_path=None, article_id=None, **kwargs*) → *None*

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

This method was introduced with qBittorrent v4.2.5 (Web API v2.5.1).

Raises

NotFound404Error –

Parameters

- **item_path** (*str* | *None*) – path to item to be refreshed (e.g. Folder\Subfolder\ItemName)
- **article_id** (*str* | *int* | *None*) – article ID from *rss_items()*

Return type

None

rss_matching_articles(*rule_name=None, **kwargs*) → *RSSItemsDictionary*

Fetch all articles matching a rule.

This method was introduced with qBittorrent v4.2.5 (Web API v2.5.1).

Parameters

rule_name (*str* | *None*) – Name of rule to return matching articles

Return type

RSSItemsDictionary

rss_move_item(*orig_item_path=None, new_item_path=None, **kwargs*) → *None*

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

Raises

Conflict409Error –

Parameters

- **orig_item_path** (*str* | *None*) – path to item to be removed (e.g. Folder\Subfolder\ItemName)
- **new_item_path** (*str* | *None*) – path to item to be removed (e.g. Folder\Subfolder\ItemName)

Return type

None

rss_refresh_item(*item_path=None, **kwargs*) → *None*

Trigger a refresh for an RSS item.

Note: qBittorrent v4.1.5 through v4.1.8 all use Web API v2.2 but this endpoint was introduced with v4.1.8; so, behavior may be undefined for these versions.

Parameters

item_path (*str* | *None*) – path to item to be refreshed (e.g. Folder\Subfolder\ItemName)

Return type

None

rss_remove_item(*item_path=None, **kwargs*) → *None*

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

NOTE: Removing a folder also removes everything in it.

Raises*Conflict409Error* –**Parameters****item_path** (*str* | *None*) – path to item to be removed (e.g. Folder\Subfolder\ItemName)**Return type***None***rss_remove_rule**(*rule_name=None, **kwargs*) → *None*

Delete a RSS auto-downloading rule.

Parameters**rule_name** (*str* | *None*) – Name of rule to delete**Return type***None***rss_rename_rule**(*orig_rule_name=None, new_rule_name=None, **kwargs*) → *None*

Rename an RSS auto-download rule.

This method did not work properly until qBittorrent v4.3.0 (Web API v2.6).

Parameters

- **orig_rule_name** (*str* | *None*) – current name of rule
- **new_rule_name** (*str* | *None*) – new name for rule

Return type*None***rss_rules**(***kwargs*) → *RSSRulesDictionary*

Retrieve RSS auto-download rule definitions.

Return type*RSSRulesDictionary***rss_set_feed_url**(*url=None, item_path=None, **kwargs*) → *None*

Update the URL for an existing RSS feed.

This method was introduced with qBittorrent v4.6.0 (Web API v2.9.1).

Raises*Conflict409Error* –**Parameters**

- **url** (*str* | *None*) – URL of RSS feed (e.g. <https://distrowatch.com/news/torrents.xml>)
- **item_path** (*str* | *None*) – Name and/or path for feed (e.g. Folder\Subfolder\FeedName)

Return type*None***rss_set_rule**(*rule_name=None, rule_def=None, **kwargs*) → *None*

Create a new RSS auto-downloading rule.

Parameters

- **rule_name** (*str* | *None*) – name for new rule

- **rule_def** (`Mapping[str, qbittorrentapi.definitions.JsonValueT] | None`) – dictionary with rule fields - [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#user-content-set-auto-downloading-rule](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#user-content-set-auto-downloading-rule)

Return type`None`**class RSS(client)**

Allows interaction with RSS API endpoints.

Usage

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

class Items(*args, client, **kwargs)

__call__ (include_feed_data=None, **kwargs) → *RSSItemsDictionary*

Implements *rss_items()*.

Return type*RSSItemsDictionary*

property with_data: *RSSItemsDictionary*

Implements *rss_items()* with include_feed_data=True.

property without_data: *RSSItemsDictionary*

Implements *rss_items()* with include_feed_data=False.

add_feed(url=None, item_path="", **kwargs) → `None`

Implements *rss_add_feed()*.

Return type`None`

add_folder(folder_path=None, **kwargs) → `None`

Implements *rss_add_folder()*.

Return type`None`

property items: *Items*

Implements *rss_items()*.

mark_as_read(item_path=None, article_id=None, **kwargs) → `None`

Implements *rss_mark_as_read()*.

Return type`None`

matching_articles(*rule_name=None, **kwargs*) → *RSSItemsDictionary*

Implements *rss_matching_articles()*.

Return type

RSSItemsDictionary

move_item(*orig_item_path=None, new_item_path=None, **kwargs*) → *None*

Implements *rss_move_item()*.

Return type

None

refresh_item(*item_path=None*) → *None*

Implements *rss_refresh_item()*.

Return type

None

remove_item(*item_path=None, **kwargs*) → *None*

Implements *rss_remove_item()*.

Return type

None

remove_rule(*rule_name=None, **kwargs*) → *None*

Implements *rss_remove_rule()*.

Return type

None

rename_rule(*orig_rule_name=None, new_rule_name=None, **kwargs*) → *None*

Implements *rss_rename_rule()*.

Return type

None

property rules: *RSSRulesDictionary*

Implements *rss_rules()*.

set_feed_url(*url=None, item_path=None, **kwargs*) → *None*

Implements *rss_set_feed_url()*.

Return type

None

set_rule(*rule_name=None, rule_def=None, **kwargs*) → *None*

Implements *rss_set_rule()*.

Return type

None

class RSSItemsDictionary(*data=None, **kwargs*)

Bases: *Dictionary*[*None* | *int* | *str* | *bool* | *Sequence*[*JsonValueT*] | *Mapping*[*str*, *JsonValueT*]]

Response for *rss_items()*

class RSSRulesDictionary(*data=None, **kwargs*)

Bases: *Dictionary*[*None* | *int* | *str* | *bool* | *Sequence*[*JsonValueT*] | *Mapping*[*str*, *JsonValueT*]]

Response for *rss_rules()*

Search

```
class SearchAPIMixin(host=None, port=None, username=None, password=None, EXTRA_HEADERS=None,
                     REQUESTS_ARGS=None, HTTPADAPTER_ARGS=None,
                     VERIFY_WEBUI_CERTIFICATE=True, FORCE_SCHEME_FROM_HOST=False,
                     RAISE_NOTIMPLEMENTEDERROR_FOR_UNIMPLEMENTED_API_ENDPOINTS=False,
                     RAISE_ERROR_FOR_UNSUPPORTED_QBITTORRENT_VERSIONS=False,
                     VERBOSE_RESPONSE_LOGGING=False, SIMPLE_RESPONSES=False,
                     DISABLE_LOGGING_DEBUG_OUTPUT=False) → None
```

Bases: [AppAPIMixin](#)

Implementation for all Search API methods.

Usage

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

search_categories(plugin_name=None, **kwargs) → [SearchCategoriesList](#)

Retrieve categories for search.

This method was introduced with qBittorrent v4.1.4 (Web API v2.1.1) and removed with qBittorrent v4.3.0 (Web API v2.6).

Parameters

plugin_name (str | None) – Limit categories returned by plugin(s) (supports all and enabled)

Return type

[SearchCategoriesList](#)

search_delete(search_id=None, **kwargs) → None

Delete a search job.

This method was introduced with qBittorrent v4.1.4 (Web API v2.1.1).

Raises

[NotFound404Error](#) –

Parameters

search_id (str | int | None) – ID of search to delete

Return type

None

search_download_torrent(url=None, plugin=None, **kwargs) → None

Download a .torrent file or magnet for a search plugin.

This method was introduced with qBittorrent v5.0.0 (Web API v2.11).

Parameters

- **url** (str | None) – URL for .torrent file or magnet
- **plugin** (str | None) – Name of the plugin

Return type*None***search_enable_plugin**(*plugins=None, enable=None, **kwargs*) → *None*

Enable or disable search plugin(s).

This method was introduced with qBittorrent v4.1.4 (Web API v2.1.1).

Parameters

- **plugins** (*str* | *Iterable[str]* | *None*) – list of plugin names
- **enable** (*bool* | *None*) – Defaults to True if None or unset; use False to disable

Return type*None***search_install_plugin**(*sources=None, **kwargs*) → *None*

Install search plugins from either URL or file.

This method was introduced with qBittorrent v4.1.4 (Web API v2.1.1).

Parameters**sources** (*str* | *Iterable[str]* | *None*) – list of URLs or filepaths**Return type***None***search_plugins**(***kwargs*) → *SearchPluginsList*

Retrieve details of search plugins.

This method was introduced with qBittorrent v4.1.4 (Web API v2.1.1).

Return type*SearchPluginsList***search_results**(*search_id=None, limit=None, offset=None, **kwargs*) → *SearchResultsDictionary*

Retrieve the results for the search.

This method was introduced with qBittorrent v4.1.4 (Web API v2.1.1).

Raises

- *NotFound404Error* –
- *Conflict409Error* –

Parameters

- **search_id** (*str* | *int* | *None*) – ID of search job
- **limit** (*str* | *int* | *None*) – number of results to return
- **offset** (*str* | *int* | *None*) – where to start returning results

Return type*SearchResultsDictionary***search_start**(*pattern=None, plugins=None, category=None, **kwargs*) → *SearchJobDictionary*

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

This method was introduced with qBittorrent v4.1.4 (Web API v2.1.1).

Raises*Conflict409Error* –

Parameters

- **pattern** (`str` | `None`) – term to search for
- **plugins** (`str` | `Iterable[str]` | `None`) – list of plugins to use for searching (supports ‘all’ and ‘enabled’)
- **category** (`str` | `None`) – categories to limit search; dependent on plugins. (supports ‘all’)

Return type`SearchJobDictionary`**search_status**(*search_id=None*, ***kwargs*) → `SearchStatusesList`

Retrieve status of one or all searches.

This method was introduced with qBittorrent v4.1.4 (Web API v2.1.1).

Raises`NotFound404Error` –**Parameters****search_id** (`str` | `int` | `None`) – ID of search to get status; leave empty for status of all jobs**Return type**`SearchStatusesList`**search_stop**(*search_id=None*, ***kwargs*) → `None`

Stop a running search.

This method was introduced with qBittorrent v4.1.4 (Web API v2.1.1).

Raises`NotFound404Error` –**Parameters****search_id** (`str` | `int` | `None`) – ID of search job to stop**Return type**`None`**search_uninstall_plugin**(*names=None*, ***kwargs*) → `None`

Uninstall search plugins.

This method was introduced with qBittorrent v4.1.4 (Web API v2.1.1).

Parameters**names** (`str` | `Iterable[str]` | `None`) – names of plugins to uninstall**Return type**`None`**search_update_plugins**(***kwargs*) → `None`

Auto update search plugins.

This method was introduced with qBittorrent v4.1.4 (Web API v2.1.1).

Return type`None`**class Search**(**args*, *client*, ***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) → SearchCategoriesList

Implements `search_categories()`.

Return type

SearchCategoriesList

delete(search_id=None, **kwargs) → None

Implements `search_delete()`.

Return type

None

download_torrent(url=None, plugin=None, **kwargs) → None

Implements `search_download_torrent()`.

Return type

None

enable_plugin(plugins=None, enable=None, **kwargs) → None

Implements `search_enable_plugin()`.

Return type

None

install_plugin(sources=None, **kwargs) → None

Implements `search_install_plugin()`.

Return type

None

property plugins: SearchPluginsList

Implements `search_plugins()`.

results(search_id=None, limit=None, offset=None, **kwargs) → SearchResultsDictionary

Implements `search_results()`.

Return type

SearchResultsDictionary

start(*pattern=None, plugins=None, category=None, **kwargs*) → *SearchJobDictionary*

Implements *search_start()*.

Return type

SearchJobDictionary

status(*search_id=None, **kwargs*) → *SearchStatusesList*

Implements *search_status()*.

Return type

SearchStatusesList

stop(*search_id=None, **kwargs*) → *None*

Implements *search_stop()*.

Return type

None

uninstall_plugin(*sources=None, **kwargs*) → *None*

Implements *search_uninstall_plugin()*.

Return type

None

update_plugins(***kwargs*) → *None*

Implements *search_update_plugins()*.

Return type

None

class SearchJobDictionary(*data, client*)

Bases: *ClientCache[SearchAPIMixin]*, *Dictionary[None | int | str | bool | Sequence[JsonValueT] | Mapping[str, JsonValueT]]*

Response for *search_start()*

delete(***kwargs*) → *None*

Implements *search_delete()*.

Return type

None

results(*limit=None, offset=None, **kwargs*) → *SearchResultsDictionary*

Implements *search_results()*.

Return type

SearchResultsDictionary

status(***kwargs*) → *SearchStatusesList*

Implements *search_status()*.

Return type

SearchStatusesList

stop(***kwargs*) → *None*

Implements *search_stop()*.

Return type

None

class SearchResultsDictionary(data=None, **kwargs)

Bases: [Dictionary](#)[None | int | str | bool | Sequence[JsonValueT] | Mapping[str, JsonValueT]]

Response for [search_results\(\)](#)

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

class SearchStatusesList(list_entries, client=None)

Bases: [List](#)[[SearchStatus](#)]

Response for [search_status\(\)](#)

Definition: [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#user-content-get-search-status](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#user-content-get-search-status)

class SearchStatus(data=None, **kwargs)

Bases: [ListEntry](#)

Item in [SearchStatusesList](#)

class SearchCategoriesList(list_entries, client=None)

Bases: [List](#)[[SearchCategory](#)]

Response for [search_categories\(\)](#)

class SearchCategory(data=None, **kwargs)

Bases: [ListEntry](#)

Item in [SearchCategoriesList](#)

class SearchPluginsList(list_entries, client=None)

Bases: [List](#)[[SearchPlugin](#)]

Response for [search_plugins\(\)](#).

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

class SearchPlugin(data=None, **kwargs)

Bases: [ListEntry](#)

Item in [SearchPluginsList](#)

Sync

class SyncAPIMixin(host=None, port=None, username=None, password=None, EXTRA_HEADERS=None, REQUESTS_ARGS=None, HTTPADAPTER_ARGS=None, VERIFY_WEBUI_CERTIFICATE=True, FORCE_SCHEME_FROM_HOST=False, RAISE_NOTIMPLEMENTEDERROR_FOR_UNIMPLEMENTED_API_ENDPOINTS=False, RAISE_ERROR_FOR_UNSUPPORTED_QBITTORRENT_VERSIONS=False, VERBOSE_RESPONSE_LOGGING=False, SIMPLE_RESPONSES=False, DISABLE_LOGGING_DEBUG_OUTPUT=False) → None

Bases: [AppAPIMixin](#)

Implementation of all Sync API Methods.

Usage

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

sync_maindata(rid=0, **kwargs) → *SyncMainDataDictionary*

Retrieves sync data.

Parameters

rid (str | int) – response ID

Return type

SyncMainDataDictionary

sync_torrent_peers(torrent_hash=None, rid=0, **kwargs) → *SyncTorrentPeersDictionary*

Retrieves torrent sync data.

Raises

NotFound404Error –

Parameters

- **torrent_hash** (str | None) – hash for torrent
- **rid** (str | int) – response ID

Return type

SyncTorrentPeersDictionary

class Sync(client) → None

Allows interaction with the Sync API endpoints.

Usage:

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

class MainData(client) → None

__call__(rid=0, **kwargs) → *SyncMainDataDictionary*

Call self as a function.

Return type

SyncMainDataDictionary

delta(**kwargs) → *SyncMainDataDictionary*

Implements *sync_maindata()* to return updates since last call.

Return type

SyncMainDataDictionary

reset_rid() → *None*

Resets RID so the next request includes everything.

Return type

None

class TorrentPeers(client) → *None*

__call__(torrent_hash=None, rid=0, **kwargs) → *SyncTorrentPeersDictionary*

Implements *sync_torrent_peers()*.

Return type

SyncTorrentPeersDictionary

delta(torrent_hash=None, **kwargs) → *SyncTorrentPeersDictionary*

Implements *sync_torrent_peers()* to return updates since last call.

Return type

SyncTorrentPeersDictionary

reset_rid() → *None*

Resets RID so the next request includes everything.

Return type

None

property maindata: *MainData*

Implements *sync_maindata()*.

property torrentPeers: *TorrentPeers*

Implements *sync_torrent_peers()*.

property torrent_peers: *TorrentPeers*

Implements *sync_torrent_peers()*.

class SyncMainDataDictionary(data=None, **kwargs)

Bases: *Dictionary*[*None* | *int* | *str* | *bool* | *Sequence*[*JsonValueT*] | *Mapping*[*str*, *JsonValueT*]]

Response for *sync_maindata()*

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

class SyncTorrentPeersDictionary(data=None, **kwargs)

Bases: *Dictionary*[*None* | *int* | *str* | *bool* | *Sequence*[*JsonValueT*] | *Mapping*[*str*, *JsonValueT*]]

Response for *sync_torrent_peers()*

Definition: [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#user-content-get-torrent-peers-data](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#user-content-get-torrent-peers-data)

Torrent Creator

```
class TorrentCreatorAPIMixin(host=None, port=None, username=None, password=None,
                             EXTRA_HEADERS=None, REQUESTS_ARGS=None,
                             HTTPADAPTER_ARGS=None, VERIFY_WEBUI_CERTIFICATE=True,
                             FORCE_SCHEME_FROM_HOST=False,
                             RAISE_NOTIMPLEMENTEDERROR_FOR_UNIMPLEMENTED_API_ENDPOINTS=False,
                             RAISE_ERROR_FOR_UNSUPPORTED_QBITTORRENT_VERSIONS=False,
                             VERBOSE_RESPONSE_LOGGING=False, SIMPLE_RESPONSES=False,
                             DISABLE_LOGGING_DEBUG_OUTPUT=False) → None
```

Bases: [AppAPIMixin](#)

Implementation of all TorrentCreator API methods.

Usage

```
>>> from qbittorrentapi import Client
>>> client = Client(host="localhost:8080", username="admin", password=
↳ "adminadmin")
>>> task = client.torrentcreator_add_task(source_path="/path/to/data")
>>> if TaskStatus(task.status().status) == TaskStatus.FINISHED:
>>>     torrent_data = task.torrent_file()
>>> task.delete()
>>> # or
>>> client.torrentcreator_delete_task(task_id=task.taskID)
```

```
torrentcreator_add_task(source_path=None, torrent_file_path=None, format=None,
                        start_seeding=None, is_private=None, optimize_alignment=None,
                        padded_file_size_limit=None, piece_size=None, comment=None,
                        trackers=None, url_seeds=None, **kwargs) → TorrentCreatorTaskDictionary
```

Add a task to create a new torrent.

This method was introduced with qBittorrent v5.0.0 (Web API v2.10.4).

Raises

[Conflict409Error](#) – too many existing torrent creator tasks

Parameters

- **source_path** ([str](#) | [PathLike](#)[[Any](#)] | [None](#)) – source file path for torrent content
- **torrent_file_path** ([str](#) | [PathLike](#)[[Any](#)] | [None](#)) – file path to save torrent
- **format** ([Literal](#)['v1', 'v2', 'hybrid'] | [None](#)) – BitTorrent V1 or V2; defaults to “hybrid” format if [None](#)
- **start_seeding** ([bool](#) | [None](#)) – should qBittorrent start seeding this torrent?
- **is_private** ([bool](#) | [None](#)) – is the torrent private or not?
- **optimize_alignment** ([bool](#) | [None](#)) – should optimized alignment be enforced for new torrent?
- **padded_file_size_limit** ([int](#) | [None](#)) – size limit for padding files
- **piece_size** ([int](#) | [None](#)) – size of the pieces
- **comment** ([str](#) | [None](#)) – comment
- **trackers** ([str](#) | [list](#)[[str](#)] | [None](#)) – list of trackers to add

- **url_seeds** (*str* | *list[str]* | *None*) – list of URLs seeds to add

Return type*TorrentCreatorTaskDictionary***torrentcreator_delete_task**(*task_id=None*, ***kwargs*) → *None*

Delete a torrent creation task.

This method was introduced with qBittorrent v5.0.0 (Web API v2.10.4).

Raises*NotFound404Error* – task not found**Parameters****task_id** (*str* | *None*) – ID of torrent creation task**Return type***None***torrentcreator_status**(*task_id=None*, ***kwargs*) → *TorrentCreatorTaskStatusList*

Status for a torrent creation task.

This method was introduced with qBittorrent v5.0.0 (Web API v2.10.4).

Raises*NotFound404Error* – task not found**Parameters****task_id** (*str* | *None*) – ID of torrent creation task**Return type***TorrentCreatorTaskStatusList***torrentcreator_torrent_file**(*task_id=None*, ***kwargs*) → *bytes*

Retrieve torrent file for created torrent.

This method was introduced with qBittorrent v5.0.0 (Web API v2.10.4).

Raises

- *NotFound404Error* – task not found
- *Conflict409Error* – torrent creation is not finished or failed

Parameters**task_id** (*str* | *None*) – ID of torrent creation task**Return type***bytes***class TorrentCreator**(**args, client, **kwargs*)

Allows interaction with TorrentCreator 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
↳ 'torrentcreator_' prepended)
>>> task = client.torrentcreator.add_task(source_path="/path/to/data")
```

(continues on next page)

(continued from previous page)

```
>>> if TaskStatus(task.status().status) == TaskStatus.FINISHED:
>>>     torrent_data = task.torrent_file()
>>> task.delete()
>>> # or
>>> client.torrentcreator.delete_task(task_id=task.taskID)
```

add_task(*source_path=None, torrent_file_path=None, format=None, start_seeding=None, is_private=None, optimize_alignment=None, padded_file_size_limit=None, piece_size=None, comment=None, trackers=None, url_seeds=None, **kwargs*) → *TorrentCreatorTaskDictionary*

Implements *torrentcreator_add_task()*.

Return type

TorrentCreatorTaskDictionary

delete_task(*task_id=None, **kwargs*) → *None*

Implements *torrentcreator_delete_task()*.

Return type

None

status(*task_id=None, **kwargs*) → *TorrentCreatorTaskStatusList*

Implements *torrentcreator_status()*.

Return type

TorrentCreatorTaskStatusList

torrent_file(*task_id=None, **kwargs*) → *bytes*

Implements *torrentcreator_torrent_file()*.

Return type

bytes

class TorrentCreatorTaskDictionary(*data, client*)

Bases: *ClientCache[TorrentCreatorAPIMixin]*, *Dictionary[None | int | str | bool | Sequence[JsonValueT] | Mapping[str, JsonValueT]]*

Response for *torrentcreator_add_task()*

delete(***kwargs*) → *None*

Implements *torrentcreator_delete_task()*.

Return type

None

status(***kwargs*) → *TorrentCreatorTaskStatus*

Implements *torrentcreator_status()*.

Return type

TorrentCreatorTaskStatus

torrent_file(***kwargs*) → *bytes*

Implements *torrentcreator_torrent_file()*.

Return type

bytes

class TorrentCreatorTaskStatus(data=None, **kwargs)

Bases: [ListEntry](#)

Item in [TorrentCreatorTaskStatusList](#)

Definition: not documented...yet

class TaskStatus(value, names=<not given>, *values, module=None, qualname=None, type=None, start=1, boundary=None)

Bases: [Enum](#)

Enumeration of possible task statuses.

FAILED = 'Failed'

FINISHED = 'Finished'

QUEUED = 'Queued'

RUNNING = 'Running'

class TorrentCreatorTaskStatusList(list_entries, client=None)

Bases: [List](#)[[TorrentCreatorTaskStatus](#)]

Response for [torrentcreator_status\(\)](#)

Torrents

class TorrentsAPIMixin(host=None, port=None, username=None, password=None, EXTRA_HEADERS=None, REQUESTS_ARGS=None, HTTPADAPTER_ARGS=None, VERIFY_WEBUI_CERTIFICATE=True, FORCE_SCHEME_FROM_HOST=False, RAISE_NOTIMPLEMENTEDERROR_FOR_UNIMPLEMENTED_API_ENDPOINTS=False, RAISE_ERROR_FOR_UNSUPPORTED_QBITTORRENT_VERSIONS=False, VERBOSE_RESPONSE_LOGGING=False, SIMPLE_RESPONSES=False, DISABLE_LOGGING_DEBUG_OUTPUT=False) → None

Bases: [AppAPIMixin](#)

Implementation of all Torrents API methods.

Usage

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

torrents_add(urls=None, torrent_files=None, save_path=None, cookie=None, category=None, is_skip_checking=None, is_paused=None, is_root_folder=None, rename=None, upload_limit=None, download_limit=None, use_auto_torrent_management=None, is_sequential_download=None, is_first_last_piece_priority=None, tags=None, content_layout=None, ratio_limit=None, seeding_time_limit=None, download_path=None, use_download_path=None, stop_condition=None, add_to_top_of_queue=None, inactive_seeding_time_limit=None, share_limit_action=None, ssl_certificate=None, ssl_private_key=None, ssl_dh_params=None, is_stopped=None, **kwargs) → str

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

Returns `Ok.` for success and `Fails.` for failure.

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** (`str` | `Iterable[str]` | `None`) – single instance or an iterable of URLs (`http://`, `https://`, `magnet:`, `bc://bt/`)
- **torrent_files** (`TypeVar(TorrentFilesT, bytes, str, IO[bytes], Mapping[str, bytes | str | IO[bytes]], Iterable[bytes | str | IO[bytes]])` | `None`) – several options are available to send torrent files to qBittorrent:
 - single instance of bytes: useful if torrent file already read from disk or downloaded from internet.
 - single instance of file handle to torrent file: use `open(<filepath>, 'rb')` to open the torrent file.
 - single instance of a filepath to torrent file: e.g. `/home/user/torrent_filename.torrent`
 - an iterable of the single instances above to send more than one torrent file
 - dictionary with key/value pairs of torrent name and single instance of above object

Note: The torrent name in a dictionary is useful to identify which torrent file errored. qBittorrent provides back that name in the error text. If a torrent name is not provided, then the name of the file will be used. And in the case of bytes (or if filename cannot be determined), the value `'torrent__n'` will be used.

- **save_path** (`str` | `None`) – location to save the torrent data
- **cookie** (`str` | `None`) – cookie(s) to retrieve torrents by URL
- **category** (`str` | `None`) – category to assign to torrent(s)
- **is_skip_checking** (`bool` | `None`) – True to skip hash checking
- **is_paused** (`bool` | `None`) – Adds torrent in stopped state; alias for `is_stopped`
- **is_root_folder** (`bool` | `None`) – True or False to create root folder (superseded by `content_layout` with v4.3.2)
- **rename** (`str` | `None`) – new name for torrent(s)
- **upload_limit** (`str` | `int` | `None`) – upload limit in bytes/second
- **download_limit** (`str` | `int` | `None`) – download limit in bytes/second
- **use_auto_torrent_management** (`bool` | `None`) – True or False to use automatic torrent management
- **is_sequential_download** (`bool` | `None`) – True or False for sequential download
- **is_first_last_piece_priority** (`bool` | `None`) – True or False for first and last piece download priority
- **tags** (`str` | `Iterable[str]` | `None`) – tag(s) to assign to torrent(s) (added in Web API v2.6.2)

- **content_layout** (`Literal['Original', 'Subfolder', 'NoSubFolder'] | None`) – Original, Subfolder, or NoSubfolder to control filesystem structure for content (added in Web API v2.7)
- **ratio_limit** (`str | float | None`) – share limit as ratio of upload amt over download amt; e.g. 0.5 or 2.0 (added in Web API v2.8.1)
- **seeding_time_limit** (`str | int | None`) – number of minutes to seed torrent (added in Web API v2.8.1)
- **download_path** (`str | None`) – location to download torrent content before moving to save_path (added in Web API v2.8.4)
- **use_download_path** (`bool | None`) – True or False whether download_path should be used... defaults to True if download_path is specified (added in Web API v2.8.4)
- **stop_condition** (`Literal['MetadataReceived', 'FilesChecked'] | None`) – MetadataReceived or FilesChecked to stop the torrent when started automatically (added in Web API v2.8.15)
- **add_to_top_of_queue** (`bool | None`) – puts torrent at top to the queue (added in Web API v2.8.19)
- **inactive_seeding_time_limit** (`str | int | None`) – limit for seeding while inactive (added in Web API v2.9.2)
- **share_limit_action** (`Literal['Stop', 'Remove', 'RemoveWithContent', 'EnableSuperSeeding'] | None`) – override default action when share limit is reached (added in Web API v2.10.4)
- **ssl_certificate** (`str | None`) – peer certificate (in PEM format) (added in Web API v2.10.4)
- **ssl_private_key** (`str | None`) – peer private key (added in Web API v2.10.4)
- **ssl_dh_params** (`str | None`) – Diffie-Hellman parameters (added in Web API v2.10.4)
- **is_stopped** (`bool | None`) – Adds torrent in stopped state; alias for is_paused (added in Web API v2.11.0)

Return type`str`

torrents_add_peers(*peers=None, torrent_hashes=None, **kwargs*) → *TorrentsAddPeersDictionary*

Add one or more peers to one or more torrents.

This method was introduced with qBittorrent v4.4.0 (Web API v2.3.0).

Raises

InvalidRequest400Error – for invalid peers

Parameters

- **peers** (`str | Iterable[str] | None`) – one or more peers to add. each peer should take the form ‘host:port’
- **torrent_hashes** (`str | Iterable[str] | None`) – single torrent hash or list of torrent hashes. Or all for all torrents.

Return type*TorrentsAddPeersDictionary*

torrents_add_tags(tags=None, torrent_hashes=None, **kwargs) → None

Add one or more tags to one or more torrents.

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

This method was introduced with qBittorrent v4.2.0 (Web API v2.3.0).

Parameters

- **tags** (str | Iterable[str] | None) – tag name or list of tags
- **torrent_hashes** (str | Iterable[str] | None) – single torrent hash or list of torrent hashes. Or all for all torrents.

Return type

None

torrents_add_trackers(torrent_hash=None, urls=None, **kwargs) → None

Add trackers to a torrent.

Raises

NotFound404Error –

Parameters

- **torrent_hash** (str | None) – hash for torrent
- **urls** (str | Iterable[str] | None) – tracker URLs to add to torrent

Return type

None

torrents_add_webseeds(torrent_hash=None, urls=None, **kwargs) → None

Add webseeds to a torrent.

Raises

- **NotFound404Error** – torrent not found
- **InvalidRequest400Error** – invalid URL

Parameters

- **torrent_hash** (str | None) – hash for torrent
- **urls** (str | Iterable[str] | None) – list of webseed URLs to add to torrent

Return type

None

torrents_bottom_priority(torrent_hashes=None, **kwargs) → None

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

Raises

Conflict409Error –

Parameters

torrent_hashes (str | Iterable[str] | None) – single torrent hash or list of torrent hashes.
Or all for all torrents.

Return type

None

torrents_categories(***kwargs*) → *TorrentCategoriesDictionary*

Retrieve all category definitions.

This method was introduced with qBittorrent v4.1.4 (Web API v2.1.1).

Note: torrents/categories is not available until v2.1.0

Return type

TorrentCategoriesDictionary

torrents_count() → *int*

Retrieve count of torrents.

Return type

int

torrents_create_category(*name=None, save_path=None, download_path=None, enable_download_path=None, **kwargs*) → *None*

Create a new torrent category.

Raises

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

Parameters

- **name** (*str* | *None*) – name for new category
- **save_path** (*str* | *None*) – location to save torrents for this category (added in Web API 2.1.0)
- **download_path** (*str* | *None*) – download location for torrents with this category
- **enable_download_path** (*bool* | *None*) – True or False to enable or disable download path

Return type

None

torrents_create_tags(*tags=None, **kwargs*) → *None*

Create one or more tags.

This method was introduced with qBittorrent v4.2.0 (Web API v2.3.0).

Parameters

tags (*str* | *Iterable[str]* | *None*) – tag name or list of tags

Return type

None

torrents_decrease_priority(*torrent_hashes=None, **kwargs*) → *None*

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

Raises

Conflict409Error –

Parameters

torrent_hashes (*str* | *Iterable[str]* | *None*) – single torrent hash or list of torrent hashes.
Or all for all torrents.

Return type

None

torrents_delete(*delete_files=False, torrent_hashes=None, **kwargs*) → [None](#)

Remove a torrent from qBittorrent and optionally delete its files.

Parameters

- **torrent_hashes** ([str](#) | [Iterable\[str\]](#) | [None](#)) – single torrent hash or list of torrent hashes. Or all for all torrents.
- **delete_files** ([bool](#) | [None](#)) – True to delete the torrent's files

Return type

[None](#)

torrents_delete_tags(*tags=None, **kwargs*) → [None](#)

Delete one or more tags.

This method was introduced with qBittorrent v4.2.0 (Web API v2.3.0).

Parameters

tags ([str](#) | [Iterable\[str\]](#) | [None](#)) – tag name or list of tags

Return type

[None](#)

torrents_download_limit(*torrent_hashes=None, **kwargs*) → [TorrentLimitsDictionary](#)

Retrieve the download limit for one or more torrents.

Return type

[TorrentLimitsDictionary](#)

torrents_edit_category(*name=None, save_path=None, download_path=None, enable_download_path=None, **kwargs*) → [None](#)

Edit an existing category.

This method was introduced with qBittorrent v4.1.3 (Web API v2.1.0).

Raises

[Conflict409Error](#) – if category name is not valid or unable to create

Parameters

- **name** ([str](#) | [None](#)) – category to edit
- **save_path** ([str](#) | [None](#)) – new location to save files for this category
- **download_path** ([str](#) | [None](#)) – download location for torrents with this category
- **enable_download_path** ([bool](#) | [None](#)) – True or False to enable or disable download path

Return type

[None](#)

torrents_edit_tracker(*torrent_hash=None, original_url=None, new_url=None, **kwargs*) → [None](#)

Replace a torrent's tracker with a different one.

This method was introduced with qBittorrent v4.1.4 (Web API v2.2.0).

Raises

- [InvalidRequest400Error](#) –
- [NotFound404Error](#) –
- [Conflict409Error](#) –

Parameters

- **torrent_hash** (`str` | `None`) – hash for torrent
- **original_url** (`str` | `None`) – URL for existing tracker
- **new_url** (`str` | `None`) – new URL to replace

Return type`None`

torrents_edit_webseed(*torrent_hash=None, orig_url=None, new_url=None, **kwargs*) → `None`

Edit a webseed for a torrent.

Raises

- **NotFound404Error** – torrent not found
- **Conflict409Error** – orig_url is not a webseed for the torrent
- **InvalidRequest400Error** – invalid URL

Parameters

- **torrent_hash** (`str` | `None`) – hash for torrent
- **orig_url** (`str` | `None`) – webseed URL to be replaced
- **new_url** (`str` | `None`) – webseed URL to replace with

Return type`None`

torrents_export(*torrent_hash=None, **kwargs*) → `bytes`

Export a .torrent file for the torrent.

This method was introduced with qBittorrent v4.5.0 (Web API v2.8.14).

Raises

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

Parameters

torrent_hash (`str` | `None`) – hash for torrent

Return type`bytes`

torrents_file_priority(*torrent_hash=None, file_ids=None, priority=None, **kwargs*) → `None`

Set priority for one or more files.

Raises

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

Parameters

- **torrent_hash** (`str` | `None`) – hash for torrent
- **file_ids** (`str` | `int` | `Iterable[str | int]` | `None`) – single file ID or a list.

- **priority** (`str` | `int` | `None`) – priority for file(s) - [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#user-content-set-file-priority](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#user-content-set-file-priority)

Return type`None`**torrents_files**(*torrent_hash=None, **kwargs*) → *TorrentFilesList*

Retrieve individual torrent's files.

Raises*NotFound404Error* –**Parameters****torrent_hash** (`str` | `None`) – hash for torrent**Return type***TorrentFilesList***torrents_increase_priority**(*torrent_hashes=None, **kwargs*) → `None`

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

Raises*Conflict409Error* –**Parameters****torrent_hashes** (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes.
Or all for all torrents.**Return type**`None`**torrents_info**(*status_filter=None, category=None, sort=None, reverse=None, limit=None, offset=None, torrent_hashes=None, tag=None, private=None, **kwargs*) → *TorrentInfoList*

Retrieves list of info for torrents.

Parameters

- **status_filter** (`Literal['all', 'downloading', 'seeding', 'completed', 'paused', 'stopped', 'active', 'inactive', 'resumed', 'running', 'stalled', 'stalled_uploading', 'stalled_downloading', 'checking', 'moving', 'errored']` | `None`) – Filter list by torrent status:
 - Original options: all, downloading, seeding, completed, paused, active, inactive, resumed, errored
 - Added in Web API v2.4.1: stalled, stalled_uploading, and stalled_downloading
 - Added in Web API v2.8.4: checking
 - Added in Web API v2.8.18: moving
 - Added in Web API v2.11.0: stopped (replaced paused), running (replaced resumed)
- **category** (`str` | `None`) – Filter list by category
- **sort** (`str` | `None`) – Sort list by any property returned
- **reverse** (`bool` | `None`) – Reverse sorting
- **limit** (`str` | `int` | `None`) – Limit length of list
- **offset** (`str` | `int` | `None`) – Start of list (if < 0, offset from end of list)

- **torrent_hashes** (*str* | *Iterable[str]* | *None*) – Filter list by hash (separate multiple hashes with a '|') (added in Web API v2.0.1)
- **tag** (*str* | *None*) – Filter list by tag (empty string means “untagged”; no “tag” parameter means “any tag”; added in Web API v2.8.3)
- **private** (*bool* | *None*) – Filter list by private flag - use None to ignore; (added in Web API v2.11.1)

Return type*TorrentInfoList***torrents_pause**(*torrent_hashes=None, **kwargs*) → *None*

Stop one or more torrents in qBittorrent.

Parameters

torrent_hashes (*str* | *Iterable[str]* | *None*) – single torrent hash or list of torrent hashes.
Or all for all torrents.

Return type*None***torrents_piece_hashes**(*torrent_hash=None, **kwargs*) → *TorrentPieceInfoList*

Retrieve individual torrent’s pieces’ hashes.

Raises*NotFound404Error* –**Parameters****torrent_hash** (*str* | *None*) – hash for torrent**Return type***TorrentPieceInfoList***torrents_piece_states**(*torrent_hash=None, **kwargs*) → *TorrentPieceInfoList*

Retrieve individual torrent’s pieces’ states.

Raises*NotFound404Error* –**Parameters****torrent_hash** (*str* | *None*) – hash for torrent**Return type***TorrentPieceInfoList***torrents_properties**(*torrent_hash=None, **kwargs*) → *TorrentPropertiesDictionary*

Retrieve individual torrent’s properties.

Raises*NotFound404Error* –**Parameters****torrent_hash** (*str* | *None*) – hash for torrent**Return type***TorrentPropertiesDictionary***torrents_reannounce**(*torrent_hashes=None, **kwargs*) → *None*

Reannounce a torrent.

This method was introduced with qBittorrent v4.1.2 (Web API v2.0.2).

Parameters

torrent_hashes (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes.
Or all for all torrents.

Return type

`None`

torrents_recheck(*torrent_hashes=None*, ***kwargs*) → `None`

Recheck a torrent in qBittorrent.

Parameters

torrent_hashes (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes.
Or all for all torrents.

Return type

`None`

torrents_remove_categories(*categories=None*, ***kwargs*) → `None`

Delete one or more categories.

Parameters

categories (`str` | `Iterable[str]` | `None`) – categories to delete

Return type

`None`

torrents_remove_tags(*tags=None*, *torrent_hashes=None*, ***kwargs*) → `None`

Add one or more tags to one or more torrents.

This method was introduced with qBittorrent v4.2.0 (Web API v2.3.0).

Parameters

- **tags** (`str` | `Iterable[str]` | `None`) – tag name or list of tags
- **torrent_hashes** (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes. Or all for all torrents.

Return type

`None`

torrents_remove_trackers(*torrent_hash=None*, *urls=None*, ***kwargs*) → `None`

Remove trackers from a torrent.

This method was introduced with qBittorrent v4.1.4 (Web API v2.2.0).

Raises

- `NotFound404Error` –
- `Conflict409Error` –

Parameters

- **torrent_hash** (`str` | `None`) – hash for torrent
- **urls** (`str` | `Iterable[str]` | `None`) – tracker URLs to removed from torrent

Return type

`None`

torrents_remove_webseeds(*torrent_hash=None*, *urls=None*, ***kwargs*) → `None`

Remove webseeds from a torrent.

Raises

- **NotFound404Error** –
- **InvalidRequest400Error** – invalid URL

Parameters

- **torrent_hash** (`str` | `None`) – hash for torrent
- **urls** (`str` | `Iterable[str]` | `None`) – list of webseed URLs to add to torrent

Return type`None`

torrents_rename(*torrent_hash=None, new_torrent_name=None, **kwargs*) → `None`

Rename a torrent.

Raises

- **NotFound404Error** –

Parameters

- **torrent_hash** (`str` | `None`) – hash for torrent
- **new_torrent_name** (`str` | `None`) – new name for torrent

Return type`None`

torrents_rename_file(*torrent_hash=None, file_id=None, new_file_name=None, old_path=None, new_path=None, **kwargs*) → `None`

Rename a torrent file.

This method was introduced with qBittorrent v4.2.1 (Web API v2.4.0).

Raises

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

Parameters

- **torrent_hash** (`str` | `None`) – hash for torrent
- **file_id** (`str` | `int` | `None`) – id for file (removed in Web API v2.7)
- **new_file_name** (`str` | `None`) – new name for file (removed in Web API v2.7)
- **old_path** (`str` | `None`) – path of file to rename (added in Web API v2.7)
- **new_path** (`str` | `None`) – new path of file to rename (added in Web API v2.7)

Return type`None`

torrents_rename_folder(*torrent_hash=None, old_path=None, new_path=None, **kwargs*) → `None`

Rename a torrent folder.

This method was introduced with qBittorrent v4.3.2 (Web API v2.7).

Raises

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

- **`Conflict409Error`** –

Parameters

- **`torrent_hash`** (`str` | `None`) – hash for torrent
- **`old_path`** (`str` | `None`) – path of file to rename (added in Web API v2.7)
- **`new_path`** (`str` | `None`) – new path of file to rename (added in Web API v2.7)

Return type`None`**`torrents_resume`**(`torrent_hashes=None`, `**kwargs`) → `None`

Start one or more torrents in qBittorrent.

Parameters**`torrent_hashes`** (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes. Or `all` for all torrents.**Return type**`None`**`torrents_set_auto_management`**(`enable=None`, `torrent_hashes=None`, `**kwargs`) → `None`

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

Parameters

- **`torrent_hashes`** (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes. Or `all` for all torrents.
- **`enable`** (`bool` | `None`) – Defaults to `True` if `None` or unset; use `False` to disable

Return type`None`**`torrents_set_category`**(`category=None`, `torrent_hashes=None`, `**kwargs`) → `None`

Set a category for one or more torrents.

Raises**`Conflict409Error`** – for bad category**Parameters**

- **`torrent_hashes`** (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes. Or `all` for all torrents.
- **`category`** (`str` | `None`) – category to assign to torrent

Return type`None`**`torrents_set_download_limit`**(`limit=None`, `torrent_hashes=None`, `**kwargs`) → `None`

Set the download limit for one or more torrents.

Parameters

- **`torrent_hashes`** (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes. Or `all` for all torrents.
- **`limit`** (`str` | `int` | `None`) – bytes/second (`-1` sets the limit to infinity)

Return type`None`

torrents_set_download_path(*download_path=None, torrent_hashes=None, **kwargs*) → [None](#)

Set the Download Path for one or more torrents.

This method was introduced with qBittorrent v4.4.0 (Web API v2.8.4).

Raises

- [Forbidden403Error](#) – cannot write to directory
- [Conflict409Error](#) – cannot create directory

Parameters

- **download_path** ([str](#) | [None](#)) – file path to save torrent contents before torrent finishes downloading
- **torrent_hashes** ([str](#) | [Iterable\[str\]](#) | [None](#)) – single torrent hash or list of torrent hashes. Or all for all torrents.

Return type

[None](#)

torrents_set_force_start(*enable=None, torrent_hashes=None, **kwargs*) → [None](#)

Force start one or more torrents.

Parameters

- **torrent_hashes** ([str](#) | [Iterable\[str\]](#) | [None](#)) – single torrent hash or list of torrent hashes. Or all for all torrents.
- **enable** ([bool](#) | [None](#)) – Defaults to True if None or unset; False is equivalent to [torrents_resume\(\)](#).

Return type

[None](#)

torrents_set_location(*location=None, torrent_hashes=None, **kwargs*) → [None](#)

Set location for torrents' files.

Raises

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

Parameters

- **torrent_hashes** ([str](#) | [Iterable\[str\]](#) | [None](#)) – single torrent hash or list of torrent hashes. Or all for all torrents.
- **location** ([str](#) | [None](#)) – disk location to move torrent's files

Return type

[None](#)

torrents_set_save_path(*save_path=None, torrent_hashes=None, **kwargs*) → [None](#)

Set the Save Path for one or more torrents.

This method was introduced with qBittorrent v4.4.0 (Web API v2.8.4).

Raises

- [Forbidden403Error](#) – cannot write to directory
- [Conflict409Error](#) – cannot create directory

Parameters

- **save_path** (`str` | `None`) – file path to save torrent contents
- **torrent_hashes** (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes. Or `all` for all torrents.

Return type`None`

torrents_set_share_limits(*ratio_limit=None, seeding_time_limit=None, inactive_seeding_time_limit=None, torrent_hashes=None, **kwargs*) → `None`

Set share limits for one or more torrents.

This method was introduced with qBittorrent v4.1.1 (Web API v2.0.1).

Parameters

- **torrent_hashes** (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes. Or `all` for all torrents.
- **ratio_limit** (`str` | `int` | `None`) – max ratio to seed a torrent. (-2 means use the global value and -1 is no limit)
- **seeding_time_limit** (`str` | `int` | `None`) – minutes (-2 means use the global value and -1 is no limit)
- **inactive_seeding_time_limit** (`str` | `int` | `None`) – minutes (-2 means use the global value and -1 is no limit) (added in Web API v2.9.2)

Return type`None`

torrents_set_super_seeding(*enable=None, torrent_hashes=None, **kwargs*) → `None`

Set one or more torrents as super seeding.

Parameters

- **torrent_hashes** (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes. Or `all` for all torrents.
- **enable** (`bool` | `None`) – Defaults to `True` if `None` or unset; `False` to disable

Return type`None`

torrents_set_upload_limit(*limit=None, torrent_hashes=None, **kwargs*) → `None`

Set the upload limit for one or more torrents.

Parameters

- **torrent_hashes** (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes. Or `all` for all torrents.
- **limit** (`str` | `int` | `None`) – bytes/second (-1 sets the limit to infinity)

Return type`None`

torrents_start(*torrent_hashes=None, **kwargs*) → `None`

Start one or more torrents in qBittorrent.

Parameters

torrent_hashes (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes.
Or all for all torrents.

Return type

`None`

torrents_stop(*torrent_hashes=None, **kwargs*) → `None`

Stop one or more torrents in qBittorrent.

Parameters

torrent_hashes (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes.
Or all for all torrents.

Return type

`None`

torrents_tags(***kwargs*) → `TagList`

Retrieve all tag definitions.

This method was introduced with qBittorrent v4.2.0 (Web API v2.3.0).

Return type

`TagList`

torrents_toggle_first_last_piece_priority(*torrent_hashes=None, **kwargs*) → `None`

Toggle priority of first/last piece downloading.

Parameters

torrent_hashes (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes.
Or all for all torrents.

Return type

`None`

torrents_toggle_sequential_download(*torrent_hashes=None, **kwargs*) → `None`

Toggle sequential download for one or more torrents.

Parameters

torrent_hashes (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes.
Or all for all torrents.

Return type

`None`

torrents_top_priority(*torrent_hashes=None, **kwargs*) → `None`

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

Raises

`Conflict409Error` –

Parameters

torrent_hashes (`str` | `Iterable[str]` | `None`) – single torrent hash or list of torrent hashes.
Or all for all torrents.

Return type

`None`

torrents_trackers(*torrent_hash=None, **kwargs*) → `TrackersList`

Retrieve individual torrent's trackers. Tracker status is defined in `TrackerStatus`.

Raises*NotFound404Error* –**Parameters****torrent_hash** (*str* | *None*) – hash for torrent**Return type***TrackersList***torrents_upload_limit**(*torrent_hashes=None, **kwargs*) → *TorrentLimitsDictionary*

Retrieve the upload limit for one or more torrents.

Parameters**torrent_hashes** (*str* | *Iterable[str]* | *None*) – single torrent hash or list of torrent hashes.
Or all for all torrents.**Return type***TorrentLimitsDictionary***torrents_webseeds**(*torrent_hash=None, **kwargs*) → *WebSeedsList*

Retrieve individual torrent's web seeds.

Raises*NotFound404Error* – torrent not found**Parameters****torrent_hash** (*str* | *None*) – hash for torrent**Return type***WebSeedsList***class Torrents**(*client*) → *None*

Allows interaction with the Torrents API endpoints.

Usage

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

add(*urls=None, torrent_files=None, save_path=None, cookie=None, category=None, is_skip_checking=None, is_paused=None, is_root_folder=None, rename=None, upload_limit=None, download_limit=None, use_auto_torrent_management=None, is_sequential_download=None, is_first_last_piece_priority=None, tags=None, content_layout=None, ratio_limit=None, seeding_time_limit=None, download_path=None, use_download_path=None, stop_condition=None, add_to_top_of_queue=None, inactive_seeding_time_limit=None, share_limit_action=None, ssl_certificate=None, ssl_private_key=None, ssl_dh_params=None, is_stopped=None, **kwargs*) → *str*
 Implements `torrents_add()`.

Return type
str

add_trackers(*torrent_hash=None, urls=None, **kwargs*) → *None*
 Implements `torrents_add_trackers()`.

Return type
None

add_webseeds(*torrent_hash=None, urls=None, **kwargs*) → *None*
 Implements `torrents_add_webseeds()`.

Return type
None

count() → *int*
 Implements `torrents_count()`.

Return type
int

edit_tracker(*torrent_hash=None, original_url=None, new_url=None, **kwargs*) → *None*
 Implements `torrents_edit_tracker()`.

Return type
None

edit_webseed(*torrent_hash=None, orig_url=None, new_url=None, **kwargs*) → *None*
 Implements `torrents_edit_webseed()`.

Return type
None

export(*torrent_hash=None, **kwargs*) → *bytes*
 Implements `torrents_export()`.

Return type
bytes

file_priority(*torrent_hash=None, file_ids=None, priority=None, **kwargs*) → *None*
 Implements `torrents_file_priority()`.

Return type
None

files(*torrent_hash=None, **kwargs*) → *TorrentFilesList*
 Implements `torrents_files()`.

Return type
TorrentFilesList

piece_hashes(*torrent_hash=None, **kwargs*) → *TorrentPieceInfoList*

Implements *torrents_piece_hashes()*.

Return type

TorrentPieceInfoList

piece_states(*torrent_hash=None, **kwargs*) → *TorrentPieceInfoList*

Implements *torrents_piece_states()*.

Return type

TorrentPieceInfoList

properties(*torrent_hash=None, **kwargs*) → *TorrentPropertiesDictionary*

Implements *torrents_properties()*.

Return type

TorrentPropertiesDictionary

remove_trackers(*torrent_hash=None, urls=None, **kwargs*) → *None*

Implements *torrents_remove_trackers()*.

Return type

None

remove_webseeds(*torrent_hash=None, urls=None, **kwargs*) → *None*

Implements *torrents_remove_webseeds()*.

Return type

None

rename(*torrent_hash=None, new_torrent_name=None, **kwargs*) → *None*

Implements *torrents_rename()*.

Return type

None

rename_file(*torrent_hash=None, file_id=None, new_file_name=None, old_path=None, new_path=None, **kwargs*) → *None*

Implements *torrents_rename_file()*.

Return type

None

rename_folder(*torrent_hash=None, old_path=None, new_path=None, **kwargs*) → *None*

Implements *torrents_rename_folder()*.

Return type

None

trackers(*torrent_hash=None, **kwargs*) → *TrackersList*

Implements *torrents_trackers()*.

Return type

TrackersList

webseeds(*torrent_hash=None, **kwargs*) → *WebSeedsList*

Implements *torrents_webseeds()*.

Return type

WebSeedsList

class `TorrentDictionary`(*data, client*) → `None`

Bases: `ClientCache[TorrentsAPIMixin]`, `ListEntry`

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

Usage

```
>>> from qbittorrentapi import Client
>>> client = Client(host="localhost:8080", username="admin", password=
↳ "adminadmin")
>>> # these are all the same attributes that are available as named in
↳ the
>>> # endpoints or the more pythonic names in Client (with or without
↳ 'transfer_' prepended)
>>> torrent = client.torrents.info()[0]
>>> torrent_hash = torrent.info.hash
>>> # Attributes without inputs and a return value are properties
>>> properties = torrent.properties
>>> 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.start()
>>> torrent.stop()
>>> torrent.recheck()
>>> torrent.torrents_top_priority()
>>> torrent.setLocation(location="/home/user/torrents/")
>>> torrent.setCategory(category="video")
```

add_tags(*tags=None, **kwargs*) → `None`

Implements `torrents_add_tags()`.

Return type

`None`

add_trackers(*urls=None, **kwargs*) → `None`

Implements `torrents_add_trackers()`.

Return type

`None`

add_webseeds(*urls, **kwargs*) → `None`

Implements `torrents_add_webseeds()`.

Return type

`None`

bottom_priority(***kwargs*) → `None`

Implements `torrents_bottom_priority()`.

Return type

`None`

decrease_priority(***kwargs*) → `None`

Implements `torrents_decrease_priority()`.

Return type

None

delete(*delete_files=None*, ***kwargs*) → None

Implements `torrents_delete()`.

Return type

None

property download_limit: `int`

Implements `torrents_download_limit()`.

edit_tracker(*orig_url=None*, *new_url=None*, ***kwargs*) → None

Implements `torrents_edit_tracker()`.

Return type

None

edit_webseed(*orig_url=None*, *new_url=None*, ***kwargs*) → None

Implements `torrents_edit_webseed()`.

Return type

None

export(***kwargs*) → bytes

Implements `torrents_export()`.

Return type

bytes

file_priority(*file_ids=None*, *priority=None*, ***kwargs*) → None

Implements `torrents_file_priority()`.

Return type

None

property files: `TorrentFilesList`

Implements `torrents_files()`.

increase_priority(***kwargs*) → None

Implements `torrents_increase_priority()`.

Return type

None

property info: `TorrentDictionary`

Returns data from `torrents_info()` for the torrent.

pause(***kwargs*) → None

Implements `torrents_stop()`.

Return type

None

property piece_hashes: `TorrentPieceInfoList`

Implements `torrents_piece_hashes()`.

property piece_states: `TorrentPieceInfoList`

Implements `torrents_piece_states()`.

property properties: *TorrentPropertiesDictionary*

Implements *torrents_properties()*.

reannounce(**kwargs) → None

Implements *torrents_reannounce()*.

Return type

None

recheck(**kwargs) → None

Implements *torrents_recheck()*.

Return type

None

remove_tags(tags=None, **kwargs) → None

Implements *torrents_remove_tags()*.

Return type

None

remove_trackers(urls=None, **kwargs) → None

Implements *torrents_remove_trackers()*.

Return type

None

remove_webseeds(urls=None, **kwargs) → None

Implements *torrents_remove_webseeds()*.

Return type

None

rename(new_name=None, **kwargs) → None

Implements *torrents_rename()*.

Return type

None

rename_file(file_id=None, new_file_name=None, old_path=None, new_path=None, **kwargs) → None

Implements *torrents_rename_file()*.

Return type

None

rename_folder(old_path=None, new_path=None, **kwargs) → None

Implements *torrents_rename_folder()*.

Return type

None

resume(**kwargs) → None

Implements *torrents_start()*.

Return type

None

set_auto_management(enable=None, **kwargs) → None

Implements *torrents_set_auto_management()*.

Return type

None

set_category(*category=None, **kwargs*) → None

Implements `torrents_set_category()`.

Return type

None

set_download_limit(*limit=None, **kwargs*) → None

Implements `torrents_set_download_limit()`.

Return type

None

set_download_path(*download_path=None, **kwargs*) → None

Implements `torrents_set_download_path()`.

Return type

None

set_force_start(*enable=None, **kwargs*) → None

Implements `torrents_set_force_start()`.

Return type

None

set_location(*location=None, **kwargs*) → None

Implements `torrents_set_location()`.

Return type

None

set_save_path(*save_path=None, **kwargs*) → None

Implements `torrents_set_save_path()`.

Return type

None

set_share_limits(*ratio_limit=None, seeding_time_limit=None, inactive_seeding_time_limit=None, **kwargs*) → None

Implements `torrents_set_share_limits()`.

Return type

None

set_super_seeding(*enable=None, **kwargs*) → None

Implements `torrents_set_super_seeding()`.

Return type

None

set_upload_limit(*limit=None, **kwargs*) → None

Implements `torrents_set_upload_limit()`.

Return type

None

start(***kwargs*) → None

Implements `torrents_start()`.

Return type

None

property state_enum: *TorrentState*

Torrent state enum.

stop(kwargs)** → *None*

Implements *torrents_stop()*.

Return type

None

sync_local() → *None*

Update local cache of torrent info.

Return type

None

toggle_first_last_piece_priority(kwargs)** → *None*

Implements *torrents_toggle_first_last_piece_priority()*.

Return type

None

toggle_sequential_download(kwargs)** → *None*

Implements *torrents_toggle_sequential_download()*.

Return type

None

top_priority(kwargs)** → *None*

Implements *torrents_top_priority()*.

Return type

None

property trackers: *TrackersList*

Implements *torrents_trackers()*.

property upload_limit: *int*

Implements *torrents_upload_limit()*.

property webseeds: *WebSeedsList*

Implements *torrents_webseeds()*.

class TorrentCategories(*args, client, **kwargs)

Bases: *ClientCache[TorrentsAPIMixin]*

Allows interaction with torrent categories within the Torrents API endpoints.

Usage

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

(continues on next page)

(continued from previous page)

```

↪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"])

```

property **categories**: *TorrentCategoriesDictionary*

Implements *torrents_categories()*.

create_category(name=None, save_path=None, download_path=None, enable_download_path=None, **kwargs) → None

Implements *torrents_create_category()*.

Return type

None

edit_category(name=None, save_path=None, download_path=None, enable_download_path=None, **kwargs) → None

Implements *torrents_edit_category()*.

Return type

None

remove_categories(categories=None, **kwargs) → None

Implements *torrents_remove_categories()*.

Return type

None

class TorrentTags(*args, client, **kwargs)

Bases: *ClientCache[TorrentsAPIMixin]*

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

Usage:

```

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

```

add_tags(tags=None, torrent_hashes=None, **kwargs) → None

Implements *torrents_add_tags()*.

Return type

None

create_tags(tags=None, **kwargs) → None

Implements `torrents_create_tags()`.

Return type

None

delete_tags(tags=None, **kwargs) → None

Implements `torrents_delete_tags()`.

Return type

None

remove_tags(tags=None, torrent_hashes=None, **kwargs) → None

Implements `torrents_remove_tags()`.

Return type

None

property tags: `TagList`

Implements `torrents_tags()`.

class TorrentPropertiesDictionary(data=None, **kwargs)

Bases: `Dictionary[None | int | str | bool | Sequence[JsonValueT] | Mapping[str, JsonValueT]]`

Response to `torrents_properties()`

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

class TorrentLimitsDictionary(data=None, **kwargs)

Bases: `Dictionary[None | int | str | bool | Sequence[JsonValueT] | Mapping[str, JsonValueT]]`

Response to `torrents_download_limit()`

class TorrentCategoriesDictionary(data=None, **kwargs)

Bases: `Dictionary[None | int | str | bool | Sequence[JsonValueT] | Mapping[str, JsonValueT]]`

Response to `torrents_categories()`

class TorrentsAddPeersDictionary(data=None, **kwargs)

Bases: `Dictionary[None | int | str | bool | Sequence[JsonValueT] | Mapping[str, JsonValueT]]`

Response to `torrents_add_peers()`

class TorrentFilesList(list_entries, client=None)

Bases: `List[TorrentFile]`

Response to `torrents_files()`

Definition: [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)#user-content-get-torrent-contents](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#user-content-get-torrent-contents)

class TorrentFile(data=None, **kwargs)

Bases: `ListEntry`

Item in `TorrentFilesList`

class WebSeedsList(list_entries, client=None)

Bases: `List[WebSeed]`

Response to `torrents_webseeds()`

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

class WebSeed(data=None, **kwargs)

Bases: [ListEntry](#)

Item in [WebSeedsList](#)

class TrackersList(list_entries, client=None)

Bases: [List\[Tracker\]](#)

Response to [torrents_trackers\(\)](#)

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

class Tracker(data=None, **kwargs)

Bases: [ListEntry](#)

Item in [TrackersList](#)

class TorrentInfoList(list_entries, client=None)

Bases: [List\[TorrentDictionary\]](#)

Response to [torrents_info\(\)](#)

Definition: [https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-\(qBittorrent-4.1\)](https://github.com/qbittorrent/qBittorrent/wiki/WebUI-API-(qBittorrent-4.1)#user-content-get-torrent-list)
#user-content-get-torrent-list

class TorrentPieceInfoList(list_entries, client=None)

Bases: [List\[TorrentPieceData\]](#)

Response to [torrents_piece_states\(\)](#) and [torrents_piece_hashes\(\)](#)

class TorrentPieceData(data=None, **kwargs)

Bases: [ListEntry](#)

Item in [TorrentPieceInfoList](#)

class TagList(list_entries, client=None)

Bases: [List\[Tag\]](#)

Response to [torrents_tags\(\)](#)

class Tag(data=None, **kwargs)

Bases: [ListEntry](#)

Item in [TagList](#)

Transfer

class TransferAPIMixin(host=None, port=None, username=None, password=None,
EXTRA_HEADERS=None, REQUESTS_ARGS=None, HTTPADAPTER_ARGS=None,
VERIFY_WEBUI_CERTIFICATE=True, FORCE_SCHEME_FROM_HOST=False,
RAISE_NOTIMPLEMENTEDERROR_FOR_UNIMPLEMENTED_API_ENDPOINTS=False,
RAISE_ERROR_FOR_UNSUPPORTED_QBITTORRENT_VERSIONS=False,
VERBOSE_RESPONSE_LOGGING=False, SIMPLE_RESPONSES=False,
DISABLE_LOGGING_DEBUG_OUTPUT=False) → None

Bases: [AppAPIMixin](#)

Implementation of all Transfer API methods.

Usage

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

transfer_ban_peers(*peers=None, **kwargs*) → *None*

Ban one or more peers.

This method was introduced with qBittorrent v4.2.0 (Web API v2.3.0).

Parameters

peers (*str | Iterable[str] | None*) – one or more peers to ban. each peer should take the form ‘host:port’

Return type

None

transfer_download_limit(***kwargs*) → *int*

Retrieves download limit; 0 is unlimited.

Return type

int

transfer_info(***kwargs*) → *TransferInfoDictionary*

Retrieves the global transfer info found in qBittorrent status bar.

Return type

TransferInfoDictionary

transfer_setSpeedLimitsMode(*intended_state=None, **kwargs*) → *None*

Sets whether alternative speed limits are enabled.

Parameters

intended_state (*bool | None*) – True to enable alt speed and False to disable. Leaving None will toggle the current state.

Return type

None

transfer_set_download_limit(*limit=None, **kwargs*) → *None*

Set the global download limit in bytes/second.

Parameters

limit (*str | int | None*) – download limit in bytes/second (0 or -1 for no limit)

Return type

None

transfer_set_speed_limits_mode(*intended_state=None, **kwargs*) → *None*

Sets whether alternative speed limits are enabled.

Parameters

intended_state (*bool | None*) – True to enable alt speed and False to disable. Leaving None will toggle the current state.

Return type

None

transfer_set_upload_limit(limit=None, **kwargs) → None

Set the global download limit in bytes/second.

Parameters

limit (str | int | None) – upload limit in bytes/second (0 or -1 for no limit)

Return type

None

transfer_speed_limits_mode(**kwargs) → str

Returns 1 if alternative speed limits are currently enabled, 0 otherwise.

Return type

str

transfer_toggle_speed_limits_mode(intended_state=None, **kwargs) → None

Sets whether alternative speed limits are enabled.

Parameters

intended_state (bool | None) – True to enable alt speed and False to disable. Leaving None will toggle the current state.

Return type

None

transfer_upload_limit(**kwargs) → int

Retrieves upload limit; 0 is unlimited.

Return type

int

class Transfer(*args, client, **kwargs)

Allows interaction with the Transfer API endpoints.

Usage

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

ban_peers(peers=None, **kwargs) → None

Implements [transfer_ban_peers\(\)](#).

Return type

None

property download_limit: int

Implements [transfer_download_limit\(\)](#).

property info: *TransferInfoDictionary*

Implements *transfer_info()*.

set_download_limit(*limit=None, **kwargs*) → *None*

Implements *transfer_set_download_limit()*.

Return type

None

set_speed_limits_mode(*intended_state=None, **kwargs*) → *None*

Implements *transfer_set_speed_limits_mode()*.

Return type

None

set_upload_limit(*limit=None, **kwargs*) → *None*

Implements *transfer_set_upload_limit()*.

Return type

None

property speed_limits_mode: *str*

Implements *transfer_speed_limits_mode()*.

toggle_speed_limits_mode(*intended_state=None, **kwargs*) → *None*

Implements *transfer_set_speed_limits_mode()*.

Return type

None

property upload_limit: *int*

Implements *transfer_upload_limit()*.

class TransferInfoDictionary(*data=None, **kwargs*)

Bases: *Dictionary*[*None* | *int* | *str* | *bool* | *Sequence*[*JsonValueT*] | *Mapping*[*str*, *JsonValueT*]]

Response to *transfer_info()*

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

Version

class Version

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

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

classmethod is_api_version_supported(*api_version*) → *bool*

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

Parameters

api_version (*str*) – version of qBittorrent Web API version such as 2.8.4

Return type

bool

Returns

True or False for whether version is supported

classmethod `is_app_version_supported(app_version)` → `bool`

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

Parameters

app_version (`str`) – version of qBittorrent application such as v4.4.0

Return type

`bool`

Returns

True or False for whether version is supported

classmethod `latest_supported_api_version()` → `str`

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

Return type

`str`

classmethod `latest_supported_app_version()` → `str`

Returns the most recent version of qBittorrent that is supported.

Return type

`str`

classmethod `supported_api_versions()` → `set[str]`

Set of all supported qBittorrent Web API versions.

Return type

`set[str]`

classmethod `supported_app_versions()` → `set[str]`

Set of all supported qBittorrent application versions.

Return type

`set[str]`

PYTHON MODULE INDEX

q

`qbittorrentapi.definitions`, [19](#)
`qbittorrentapi.exceptions`, [9](#)
`qbittorrentapi.request`, [25](#)

Symbols

__call__() (*Log.Main method*), 24
 __call__() (*RSS.Items method*), 34
 __call__() (*Sync.MainData method*), 42
 __call__() (*Sync.TorrentPeers method*), 43
 _auth_request() (*Request method*), 27
 _cast() (*Request method*), 27
 _format_payload() (*Request static method*), 27
 _get() (*Request method*), 28
 _get_cast() (*Request method*), 28
 _handle_error_responses() (*Request static method*), 28
 _initialize_context() (*Request method*), 28
 _initialize_settings() (*Request method*), 28
 _is_endpoint_supported_for_version() (*Request method*), 29
 _list2string() (*Request class method*), 29
 _post() (*Request method*), 29
 _post_cast() (*Request method*), 29
 _request() (*Request method*), 29
 _request_manager() (*Request method*), 30
 _session (*Request property*), 30
 _trigger_session_initialization() (*Request method*), 30
 _verbose_logging() (*Request method*), 30

A

add() (*Torrents method*), 62
 add_feed() (*RSS method*), 34
 add_folder() (*RSS method*), 34
 add_tags() (*TorrentDictionary method*), 65
 add_tags() (*TorrentTags method*), 70
 add_task() (*TorrentCreator method*), 46
 add_trackers() (*TorrentDictionary method*), 65
 add_trackers() (*Torrents method*), 63
 add_webseeds() (*TorrentDictionary method*), 65
 add_webseeds() (*Torrents method*), 63
 ALLOCATING (*TorrentState attribute*), 21
 APIConnectionError, 10
 APIError, 9
 APIKwargsT (*in module qbittorrentapi.definitions*), 19
 APINames (*class in qbittorrentapi.definitions*), 19

app_build_info() (*AppAPIMixin method*), 12
 app_default_save_path() (*AppAPIMixin method*), 12
 app_get_directory_content() (*AppAPIMixin method*), 12
 app_network_interface_address_list() (*AppAPIMixin method*), 13
 app_network_interface_list() (*AppAPIMixin method*), 13
 app_preferences() (*AppAPIMixin method*), 13
 app_send_test_email() (*AppAPIMixin method*), 13
 app_set_preferences() (*AppAPIMixin method*), 13
 app_shutdown() (*AppAPIMixin method*), 13
 app_version() (*AppAPIMixin method*), 13
 app_web_api_version() (*AppAPIMixin method*), 13
 AppAPIMixin (*class in qbittorrentapi.app*), 12
 Application (*APINames attribute*), 19
 Application (*class in qbittorrentapi.app*), 14
 ApplicationPreferencesDictionary (*class in qbittorrentapi.app*), 15
 Attr (*class in qbittorrentapi._attrdict*), 16
 AttrDict (*class in qbittorrentapi._attrdict*), 16
 auth_log_in() (*AuthAPIMixin method*), 16
 auth_log_out() (*AuthAPIMixin method*), 17
 AuthAPIMixin (*class in qbittorrentapi.auth*), 16
 Authorization (*APINames attribute*), 19
 Authorization (*class in qbittorrentapi.auth*), 17

B

ban_peers() (*Transfer method*), 74
 bottom_priority() (*TorrentDictionary method*), 65
 build() (*QbittorrentURL method*), 26
 build_base_url() (*QbittorrentURL method*), 26
 build_info (*Application property*), 14
 BuildInfoDictionary (*class in qbittorrentapi.app*), 15

C

categories (*TorrentCategories property*), 70
 categories() (*Search method*), 39
 CHECKING_DOWNLOAD (*TorrentState attribute*), 21
 CHECKING_RESUME_DATA (*TorrentState attribute*), 21
 CHECKING_UPLOAD (*TorrentState attribute*), 21

Client (class in *qbittorrentapi.client*), 18
ClientCache (class in *qbittorrentapi.definitions*), 19
ClientT (class in *qbittorrentapi.definitions*), 19
Conflict409Error, 12
count() (Torrents method), 63
create_category() (TorrentCategories method), 70
create_tags() (TorrentTags method), 70
critical() (Log.Main method), 24

D

decrease_priority() (TorrentDictionary method), 65
default_save_path (Application property), 14
delete() (Search method), 39
delete() (SearchJobDictionary method), 40
delete() (TorrentCreatorTaskDictionary method), 46
delete() (TorrentDictionary method), 66
delete_tags() (TorrentTags method), 71
delete_task() (TorrentCreator method), 46
delta() (Sync.MainData method), 42
delta() (Sync.TorrentPeers method), 43
detect_scheme() (QbittorrentURL method), 26
Dictionary (class in *qbittorrentapi.definitions*), 19
DirectoryContentList (class in *qbittorrentapi.app*), 15
DISABLED (TrackerStatus attribute), 22
display (TrackerStatus property), 22
download_limit (TorrentDictionary property), 66
download_limit (Transfer property), 74
download_torrent() (Search method), 39
DOWNLOADING (TorrentState attribute), 21

E

edit_category() (TorrentCategories method), 70
edit_tracker() (TorrentDictionary method), 66
edit_tracker() (Torrents method), 63
edit_webseed() (TorrentDictionary method), 66
edit_webseed() (Torrents method), 63
EMPTY (APINames attribute), 19
enable_plugin() (Search method), 39
ERROR (TorrentState attribute), 21
export() (TorrentDictionary method), 66
export() (Torrents method), 63

F

FAILED (TaskStatus attribute), 47
file_priority() (TorrentDictionary method), 66
file_priority() (Torrents method), 63
FileError, 9
files (TorrentDictionary property), 66
files() (Torrents method), 63
FilesToSendT (in module *qbittorrentapi.definitions*), 20
FINISHED (TaskStatus attribute), 47
Forbidden403Error, 11
FORCED_DOWNLOAD (TorrentState attribute), 21

FORCED_METADATA_DOWNLOAD (TorrentState attribute), 21
FORCED_UPLOAD (TorrentState attribute), 21

G

get_directory_content() (Application method), 14

H

HTTP400Error, 10
HTTP401Error, 10
HTTP403Error, 10
HTTP404Error, 10
HTTP405Error, 11
HTTP409Error, 11
HTTP415Error, 11
HTTP4XXError, 10
HTTP500Error, 11
HTTP5XXError, 10
http_status_code (HTTP400Error attribute), 10
http_status_code (HTTP401Error attribute), 10
http_status_code (HTTP403Error attribute), 10
http_status_code (HTTP404Error attribute), 11
http_status_code (HTTP405Error attribute), 11
http_status_code (HTTP409Error attribute), 11
http_status_code (HTTP415Error attribute), 11
http_status_code (HTTP500Error attribute), 11
http_status_code (HTTPError attribute), 10
HTTPError, 10

I

increase_priority() (TorrentDictionary method), 66
info (TorrentDictionary property), 66
info (Transfer property), 74
info() (Log.Main method), 24
install_plugin() (Search method), 39
InternalServerError500Error, 12
InvalidRequest400Error, 11
is_api_version_supported() (Version class method), 75
is_app_version_supported() (Version class method), 76
is_checking (TorrentState property), 21
is_complete (TorrentState property), 21
is_downloading (TorrentState property), 21
is_errored (TorrentState property), 22
is_logged_in (AuthAPIMixIn property), 17
is_logged_in (Authorization property), 17
is_paused (TorrentState property), 22
is_stopped (TorrentState property), 22
is_uploading (TorrentState property), 22
items (RSS property), 34

J

JsonValueT (in module *qbittorrentapi.definitions*), 20

L

latest_supported_api_version() (*Version class method*), 76
 latest_supported_app_version() (*Version class method*), 76
 List (*class in qbittorrentapi.definitions*), 20
 ListEntry (*class in qbittorrentapi.definitions*), 20
 ListEntryT (*class in qbittorrentapi.definitions*), 20
 ListInputT (*in module qbittorrentapi.definitions*), 20
 Log (*APINames attribute*), 19
 Log (*class in qbittorrentapi.log*), 23
 Log.Main (*class in qbittorrentapi.log*), 24
 log_in() (*Authorization method*), 17
 log_main() (*LogAPIMixin method*), 23
 log_out() (*Authorization method*), 17
 log_peers() (*LogAPIMixin method*), 23
 LogAPIMixin (*class in qbittorrentapi.log*), 23
 LogEntry (*class in qbittorrentapi.log*), 25
 LoginFailed, 10
 LogMainList (*class in qbittorrentapi.log*), 24
 LogPeer (*class in qbittorrentapi.log*), 24
 LogPeersList (*class in qbittorrentapi.log*), 24

M

main (*Log property*), 24
 maindata (*Sync property*), 43
 mark_as_read() (*RSS method*), 34
 matching_articles() (*RSS method*), 34
 METADATA_DOWNLOAD (*TorrentState attribute*), 21
 MethodNotAllowed405Error, 11
 MISSING_FILES (*TorrentState attribute*), 21
 MissingRequiredParameters400Error, 11
 module
 qbittorrentapi.definitions, 19
 qbittorrentapi.exceptions, 9
 qbittorrentapi.request, 25
 move_item() (*RSS method*), 35
 MOVING (*TorrentState attribute*), 21
 MutableAttr (*class in qbittorrentapi._attrdict*), 16

N

network_interface_address_list() (*Application method*), 14
 network_interface_list (*Application property*), 14
 NetworkInterface (*class in qbittorrentapi.app*), 15
 NetworkInterfaceAddressList (*class in qbittorrentapi.app*), 15
 NetworkInterfaceList (*class in qbittorrentapi.app*), 15
 normal() (*Log.Main method*), 24
 NOT_CONTACTED (*TrackerStatus attribute*), 22
 NOT_WORKING (*TrackerStatus attribute*), 22
 NotFound404Error, 11

P

pause() (*TorrentDictionary method*), 66
 PAUSED_DOWNLOAD (*TorrentState attribute*), 21
 PAUSED_UPLOAD (*TorrentState attribute*), 21
 peers() (*Log method*), 24
 piece_hashes (*TorrentDictionary property*), 66
 piece_hashes() (*Torrents method*), 63
 piece_states (*TorrentDictionary property*), 66
 piece_states() (*Torrents method*), 64
 plugins (*Search property*), 39
 preferences (*Application property*), 14
 properties (*TorrentDictionary property*), 66
 properties() (*Torrents method*), 64

Q

qbittorrentapi.definitions
 module, 19
 qbittorrentapi.exceptions
 module, 9
 qbittorrentapi.request
 module, 25
 QbittorrentSession (*class in qbittorrentapi.request*), 25
 QbittorrentURL (*class in qbittorrentapi.request*), 26
 QUEUED (*TaskStatus attribute*), 47
 QUEUED_DOWNLOAD (*TorrentState attribute*), 21
 QUEUED_UPLOAD (*TorrentState attribute*), 21

R

reannounce() (*TorrentDictionary method*), 67
 recheck() (*TorrentDictionary method*), 67
 refresh_item() (*RSS method*), 35
 remove_categories() (*TorrentCategories method*), 70
 remove_item() (*RSS method*), 35
 remove_rule() (*RSS method*), 35
 remove_tags() (*TorrentDictionary method*), 67
 remove_tags() (*TorrentTags method*), 71
 remove_trackers() (*TorrentDictionary method*), 67
 remove_trackers() (*Torrents method*), 64
 remove_webseeds() (*TorrentDictionary method*), 67
 remove_webseeds() (*Torrents method*), 64
 rename() (*TorrentDictionary method*), 67
 rename() (*Torrents method*), 64
 rename_file() (*TorrentDictionary method*), 67
 rename_file() (*Torrents method*), 64
 rename_folder() (*TorrentDictionary method*), 67
 rename_folder() (*Torrents method*), 64
 rename_rule() (*RSS method*), 35
 Request (*class in qbittorrentapi.request*), 27
 request() (*QbittorrentSession method*), 25
 reset rid() (*Sync.MainData method*), 43
 reset rid() (*Sync.TorrentPeers method*), 43
 results() (*Search method*), 39

results() (*SearchJobDictionary* method), 40
resume() (*TorrentDictionary* method), 67
RSS (*APINames* attribute), 19
RSS (*class in qbittorrentapi.rss*), 34
RSS.Items (*class in qbittorrentapi.rss*), 34
rss_add_feed() (*RSSAPIMixin* method), 31
rss_add_folder() (*RSSAPIMixin* method), 31
rss_items() (*RSSAPIMixin* method), 31
rss_mark_as_read() (*RSSAPIMixin* method), 31
rss_matching_articles() (*RSSAPIMixin* method), 32
rss_move_item() (*RSSAPIMixin* method), 32
rss_refresh_item() (*RSSAPIMixin* method), 32
rss_remove_item() (*RSSAPIMixin* method), 32
rss_remove_rule() (*RSSAPIMixin* method), 33
rss_rename_rule() (*RSSAPIMixin* method), 33
rss_rules() (*RSSAPIMixin* method), 33
rss_set_feed_url() (*RSSAPIMixin* method), 33
rss_set_rule() (*RSSAPIMixin* method), 33
RSSAPIMixin (*class in qbittorrentapi.rss*), 31
RSSItemsDictionary (*class in qbittorrentapi.rss*), 35
RSSRulesDictionary (*class in qbittorrentapi.rss*), 35
rules (*RSS* property), 35
RUNNING (*TaskStatus* attribute), 47

S

Search (*APINames* attribute), 19
Search (*class in qbittorrentapi.search*), 38
search_categories() (*SearchAPIMixin* method), 36
search_delete() (*SearchAPIMixin* method), 36
search_download_torrent() (*SearchAPIMixin* method), 36
search_enable_plugin() (*SearchAPIMixin* method), 37
search_install_plugin() (*SearchAPIMixin* method), 37
search_plugins() (*SearchAPIMixin* method), 37
search_results() (*SearchAPIMixin* method), 37
search_start() (*SearchAPIMixin* method), 37
search_status() (*SearchAPIMixin* method), 38
search_stop() (*SearchAPIMixin* method), 38
search_uninstall_plugin() (*SearchAPIMixin* method), 38
search_update_plugins() (*SearchAPIMixin* method), 38
SearchAPIMixin (*class in qbittorrentapi.search*), 36
SearchCategoriesList (*class in qbittorrentapi.search*), 41
SearchCategory (*class in qbittorrentapi.search*), 41
SearchJobDictionary (*class in qbittorrentapi.search*), 40
SearchPlugin (*class in qbittorrentapi.search*), 41
SearchPluginsList (*class in qbittorrentapi.search*), 41

SearchResultsDictionary (*class in qbittorrentapi.search*), 40
SearchStatus (*class in qbittorrentapi.search*), 41
SearchStatusesList (*class in qbittorrentapi.search*), 41
send_test_email() (*Application* method), 14
set_auto_management() (*TorrentDictionary* method), 67
set_category() (*TorrentDictionary* method), 68
set_download_limit() (*TorrentDictionary* method), 68
set_download_limit() (*Transfer* method), 75
set_download_path() (*TorrentDictionary* method), 68
set_feed_url() (*RSS* method), 35
set_force_start() (*TorrentDictionary* method), 68
set_location() (*TorrentDictionary* method), 68
set_preferences() (*Application* method), 15
set_rule() (*RSS* method), 35
set_save_path() (*TorrentDictionary* method), 68
set_share_limits() (*TorrentDictionary* method), 68
set_speed_limits_mode() (*Transfer* method), 75
set_super_seeding() (*TorrentDictionary* method), 68
set_upload_limit() (*TorrentDictionary* method), 68
set_upload_limit() (*Transfer* method), 75
shutdown() (*Application* method), 15
speed_limits_mode (*Transfer* property), 75
STALLED_DOWNLOAD (*TorrentState* attribute), 21
STALLED_UPLOAD (*TorrentState* attribute), 21
start() (*Search* method), 39
start() (*TorrentDictionary* method), 68
state_enum (*TorrentDictionary* property), 68
status() (*Search* method), 40
status() (*SearchJobDictionary* method), 40
status() (*TorrentCreator* method), 46
status() (*TorrentCreatorTaskDictionary* method), 46
stop() (*Search* method), 40
stop() (*SearchJobDictionary* method), 40
stop() (*TorrentDictionary* method), 69
STOPPED_DOWNLOAD (*TorrentState* attribute), 21
STOPPED_UPLOAD (*TorrentState* attribute), 21
supported_api_versions() (*Version* class method), 76
supported_app_versions() (*Version* class method), 76
Sync (*APINames* attribute), 19
Sync (*class in qbittorrentapi.sync*), 42
Sync.MainData (*class in qbittorrentapi.sync*), 42
Sync.TorrentPeers (*class in qbittorrentapi.sync*), 43
sync_local() (*TorrentDictionary* method), 69
sync_maindata() (*SyncAPIMixin* method), 42
sync_torrent_peers() (*SyncAPIMixin* method), 42
SyncAPIMixin (*class in qbittorrentapi.sync*), 41
SyncMainDataDictionary (*class in qbittorrentapi.sync*), 43

SyncTorrentPeersDictionary (class in qbittorrentapi.sync), 43

T

Tag (class in qbittorrentapi.torrents), 72

TagList (class in qbittorrentapi.torrents), 72

tags (TorrentTags property), 71

TaskStatus (class in qbittorrentapi.torrentcreator), 47

toggle_first_last_piece_priority() (TorrentDictionary method), 69

toggle_sequential_download() (TorrentDictionary method), 69

toggle_speed_limits_mode() (Transfer method), 75

top_priority() (TorrentDictionary method), 69

torrent_file() (TorrentCreator method), 46

torrent_file() (TorrentCreatorTaskDictionary method), 46

torrent_peers (Sync property), 43

TorrentCategories (class in qbittorrentapi.torrents), 69

TorrentCategoriesDictionary (class in qbittorrentapi.torrents), 71

TorrentCreator (APINames attribute), 19

TorrentCreator (class in qbittorrentapi.torrentcreator), 45

torrentcreator_add_task() (TorrentCreatorAPIMixin method), 44

torrentcreator_delete_task() (TorrentCreatorAPIMixin method), 45

torrentcreator_status() (TorrentCreatorAPIMixin method), 45

torrentcreator_torrent_file() (TorrentCreatorAPIMixin method), 45

TorrentCreatorAPIMixin (class in qbittorrentapi.torrentcreator), 44

TorrentCreatorTaskDictionary (class in qbittorrentapi.torrentcreator), 46

TorrentCreatorTaskStatus (class in qbittorrentapi.torrentcreator), 46

TorrentCreatorTaskStatusList (class in qbittorrentapi.torrentcreator), 47

TorrentDictionary (class in qbittorrentapi.torrents), 64

TorrentFile (class in qbittorrentapi.torrents), 71

TorrentFileError, 9

TorrentFileNotFoundError, 9

TorrentFilePermissionError, 10

TorrentFilesList (class in qbittorrentapi.torrents), 71

TorrentInfoList (class in qbittorrentapi.torrents), 72

TorrentLimitsDictionary (class in qbittorrentapi.torrents), 71

torrentPeers (Sync property), 43

TorrentPieceData (class in qbittorrentapi.torrents), 72

TorrentPieceInfoList (class in qbittorrentapi.torrents), 72

TorrentPropertiesDictionary (class in qbittorrentapi.torrents), 71

Torrents (APINames attribute), 19

Torrents (class in qbittorrentapi.torrents), 62

torrents_add() (TorrentsAPIMixin method), 47

torrents_add_peers() (TorrentsAPIMixin method), 49

torrents_add_tags() (TorrentsAPIMixin method), 49

torrents_add_trackers() (TorrentsAPIMixin method), 50

torrents_add_webseeds() (TorrentsAPIMixin method), 50

torrents_bottom_priority() (TorrentsAPIMixin method), 50

torrents_categories() (TorrentsAPIMixin method), 50

torrents_count() (TorrentsAPIMixin method), 51

torrents_create_category() (TorrentsAPIMixin method), 51

torrents_create_tags() (TorrentsAPIMixin method), 51

torrents_decrease_priority() (TorrentsAPIMixin method), 51

torrents_delete() (TorrentsAPIMixin method), 51

torrents_delete_tags() (TorrentsAPIMixin method), 52

torrents_download_limit() (TorrentsAPIMixin method), 52

torrents_edit_category() (TorrentsAPIMixin method), 52

torrents_edit_tracker() (TorrentsAPIMixin method), 52

torrents_edit_webseed() (TorrentsAPIMixin method), 53

torrents_export() (TorrentsAPIMixin method), 53

torrents_file_priority() (TorrentsAPIMixin method), 53

torrents_files() (TorrentsAPIMixin method), 54

torrents_increase_priority() (TorrentsAPIMixin method), 54

torrents_info() (TorrentsAPIMixin method), 54

torrents_pause() (TorrentsAPIMixin method), 55

torrents_piece_hashes() (TorrentsAPIMixin method), 55

torrents_piece_states() (TorrentsAPIMixin method), 55

torrents_properties() (TorrentsAPIMixin method), 55

torrents_reannounce() (TorrentsAPIMixin method), 55

torrents_recheck() (TorrentsAPIMixin method), 56

torrents_remove_categories() (TorrentsAPIMixin

- method), 56
- torrents_remove_tags() (TorrentsAPIMixin method), 56
- torrents_remove_trackers() (TorrentsAPIMixin method), 56
- torrents_remove_webseeds() (TorrentsAPIMixin method), 56
- torrents_rename() (TorrentsAPIMixin method), 57
- torrents_rename_file() (TorrentsAPIMixin method), 57
- torrents_rename_folder() (TorrentsAPIMixin method), 57
- torrents_resume() (TorrentsAPIMixin method), 58
- torrents_set_auto_management() (TorrentsAPIMixin method), 58
- torrents_set_category() (TorrentsAPIMixin method), 58
- torrents_set_download_limit() (TorrentsAPIMixin method), 58
- torrents_set_download_path() (TorrentsAPIMixin method), 58
- torrents_set_force_start() (TorrentsAPIMixin method), 59
- torrents_set_location() (TorrentsAPIMixin method), 59
- torrents_set_save_path() (TorrentsAPIMixin method), 59
- torrents_set_share_limits() (TorrentsAPIMixin method), 60
- torrents_set_super_seeding() (TorrentsAPIMixin method), 60
- torrents_set_upload_limit() (TorrentsAPIMixin method), 60
- torrents_start() (TorrentsAPIMixin method), 60
- torrents_stop() (TorrentsAPIMixin method), 61
- torrents_tags() (TorrentsAPIMixin method), 61
- torrents_toggle_first_last_piece_priority() (TorrentsAPIMixin method), 61
- torrents_toggle_sequential_download() (TorrentsAPIMixin method), 61
- torrents_top_priority() (TorrentsAPIMixin method), 61
- torrents_trackers() (TorrentsAPIMixin method), 61
- torrents_upload_limit() (TorrentsAPIMixin method), 62
- torrents_webseeds() (TorrentsAPIMixin method), 62
- TorrentsAddPeersDictionary (class in qbittorrentapi.torrents), 71
- TorrentsAPIMixin (class in qbittorrentapi.torrents), 47
- TorrentState (class in qbittorrentapi.definitions), 20
- TorrentTags (class in qbittorrentapi.torrents), 70
- Tracker (class in qbittorrentapi.torrents), 72
- trackers (TorrentDictionary property), 69
- trackers() (Torrents method), 64
- TrackersList (class in qbittorrentapi.torrents), 72
- TrackerStatus (class in qbittorrentapi.definitions), 22
- Transfer (APINames attribute), 19
- Transfer (class in qbittorrentapi.transfer), 74
- transfer_ban_peers() (TransferAPIMixin method), 73
- transfer_download_limit() (TransferAPIMixin method), 73
- transfer_info() (TransferAPIMixin method), 73
- transfer_set_download_limit() (TransferAPIMixin method), 73
- transfer_set_speed_limits_mode() (TransferAPIMixin method), 73
- transfer_set_upload_limit() (TransferAPIMixin method), 73
- transfer_setSpeedLimitsMode() (TransferAPIMixin method), 73
- transfer_speed_limits_mode() (TransferAPIMixin method), 74
- transfer_toggle_speed_limits_mode() (TransferAPIMixin method), 74
- transfer_upload_limit() (TransferAPIMixin method), 74
- TransferAPIMixin (class in qbittorrentapi.transfer), 72
- TransferInfoDictionary (class in qbittorrentapi.transfer), 75
- ## U
- Unauthorized401Error, 11
- uninstall_plugin() (Search method), 40
- UNKNOWN (TorrentState attribute), 21
- UnsupportedMediaType415Error, 12
- UnsupportedQbittorrentVersion, 9
- update_plugins() (Search method), 40
- UPDATING (TrackerStatus attribute), 22
- upload_limit (TorrentDictionary property), 69
- upload_limit (Transfer property), 75
- UPLOADING (TorrentState attribute), 21
- ## V
- version (Application property), 15
- Version (class in qbittorrentapi._version_support), 75
- ## W
- warning() (Log.Main method), 24
- web_api_version (Application property), 15
- WebSeed (class in qbittorrentapi.torrents), 72
- webseeds (TorrentDictionary property), 69
- webseeds() (Torrents method), 64
- WebSeedsList (class in qbittorrentapi.torrents), 71
- with_data (RSS.Items property), 34
- without_data (RSS.Items property), 34
- WORKING (TrackerStatus attribute), 22