Readability API Python Library Documentation

Release 1.0.0

Readability, LLC

Nov 08, 2017

Contents

Version 1.0.0

The official Python client library for the Readability and APIs.

Development of the readability-api package is hosted on . The package itself is hosted on and can easily be installed using pip.

CHAPTER 1

Version 1.0.0 Notice

Version 1.0 and up have fundamentally changed the objects returned by calls to the API. The underlying requests.Response objects are returned which greatly increases transparency and ease of development.

This is a departure from the 0.x releases which provided wrapped objects and hid the http request mechanics. These releases also did not use the Requests library. Version 1.0 also transitions to using for oAuth support.

In addition, 1.x introduces python3 support (woohoo!)

CHAPTER 2

Installation

```
pip install readability-api
```

CHAPTER 3

Examples

Getting a user's favorite bookmarks is easy.

```
from readability import ReaderClient

# If no client credentials are passed to ReaderClient's constructor, they
# will be looked for in your environment variables
client = ReaderClient(token_key="a user's key", token_secret"a user's secret")
bookmarks_response = client.get_bookmarks(favorite=True)

print(bookmarks_response.json())
>>> {'bookmarks': [{'user_id': 9999, 'read_percent': u'0.00', ... }
```

See *readability.ReaderClient* docs for a complete list of available functionality.

```
from readability import ParserClient

parser_client = ParserClient('your_parser_token')
parser_response = parser_client.get_article('http://paulgraham.com/altair.html')
article = parser_response.json()

print(article['title'])
>>> "What Microsoft Is this the Altair Basic of?"

print(article['content'])
>>> "<div><p>February 2015<p>One of the most valuable exercises you can try if you ...
 ↪"
```

See *readability.ParserClient* docs for a complete list of available functionality.

3.1 Authentication

Authentication can be accomplised through either a or via xAuth where a username and password are exchanged directly for a user token and secret.

That token and secret is then used to sign requests on behalf of the user. A user's credentials should never be stored and are not needed. You should favor a three legged auth flow if your application can support it. For testing purposes, or for applications where a redirect flow is prohibitive, you can use the xauth class to generate the token pair needed to sign Reader API requests.

3.1.1 Client Documentation

class `readability.auth.`**`xauth`**

Returns an OAuth token tuple that can be used with clients.ReaderClient.

Parameters

- **`base_url_template`** – Template for generating Readability API urls.
- **`consumer_key`** – Readability consumer key, otherwise read from READABILITY_CONSUMER_KEY.
- **`consumer_secret`** – Readability consumer secret, otherwise read from READABILITY_CONSUMER_SECRET.
- **`username`** – A username, otherwise read from READABILITY_USERNAME.
- **`password`** – A password, otherwise read from READABILITY_PASSWORD.

3.2 Reader API Client

The client requires four pieces of credential data. A consumer key and consumer secret can be obtained from . In addition to client credentials, a user's token key and token secret must also be used for authentication. For more information regarding auth, visit the Authentication section of the docs.

Your client key and secret can be passed to the constructor directly or set via environment variables:

Under the hood, the *ReaderClient* use the popular requests library. The objects returned by the `ReaderClient` are instances of requests.Response.

3.2.1 Client Documentation

class `readability.`**`ReaderClient`**(*token_key*, *token_secret*, *base_url_template='https://www.readability.com/api/rest/v1/{}'*, ***xargs*)

Client for interacting with the Readability Reader API.

Docs can be found at *http://www.readability.com/developers/api/reader*.

`add_bookmark`(*url*, *favorite=False*, *archive=False*, *allow_duplicates=True*)

Adds given bookmark to the authenticated user.

Parameters

- **`url`** – URL of the article to bookmark
- **`favorite`** – whether or not the bookmark should be favorited
- **`archive`** – whether or not the bookmark should be archived
- **`allow_duplicates`** – whether or not to allow duplicate bookmarks to be created for a given url

add_tags_to_bookmark(*bookmark_id*, *tags*)
Add tags to to a bookmark.

The identified bookmark must belong to the current user.

Parameters

- **bookmark_id** – ID of the bookmark to delete.
- **tags** – Comma separated tags to be applied.

archive_bookmark(*bookmark_id*)
Archives given bookmark. The requested bookmark must belong to the current user.

Parameters **bookmark_id** – ID of the bookmark to archive.

delete(*url*)
Make a HTTP DELETE request to the Readability API.

Parameters **url** – The url to which to send a DELETE request.

delete_bookmark(*bookmark_id*)
Delete a single bookmark represented by *bookmark_id*.

The requested bookmark must belong to the current user.

Parameters **bookmark_id** – ID of the bookmark to delete.

delete_tag_from_bookmark(*bookmark_id*, *tag_id*)
Remove a single tag from a bookmark.

The identified bookmark must belong to the current user.

Parameters **bookmark_id** – ID of the bookmark to delete.

favorite_bookmark(*bookmark_id*)
Favorites given bookmark. The requested bookmark must belong to the current user.

Parameters **bookmark_id** – ID of the bookmark to favorite.

get(*url*)
Make a HTTP GET request to the Reader API.

Parameters **url** – url to which to make a GET request.

get_article(*article_id*)
Get a single article represented by *article_id*.

Parameters **article_id** – ID of the article to retrieve.

get_bookmark(*bookmark_id*)
Get a single bookmark represented by *bookmark_id*.

The requested bookmark must belong to the current user.

Parameters **bookmark_id** – ID of the bookmark to retrieve.

get_bookmark_tags(*bookmark_id*)
Retrieve tags that have been applied to a bookmark.

The requested bookmark must belong to the current user.

Parameters **bookmark_id** – ID of the bookmark to delete.

get_bookmarks(***filters*)
Get Bookmarks for the current user.

Filters:

Parameters

- **archive** – Filter Bookmarks returned by archived status.
- **favorite** – Filter Bookmarks returned by favorite status.
- **domain** – Filter Bookmarks returned by a domain.
- **added_since** – Filter bookmarks by date added (since this date).
- **added_until** – Filter bookmarks by date added (until this date).
- **opened_since** – Filter bookmarks by date opened (since this date).
- **opened_until** – Filter bookmarks by date opened (until this date).
- **archived_since** – Filter bookmarks by date archived (since this date.)
- **archived_until** – Filter bookmarks by date archived (until this date.)
- **updated_since** – Filter bookmarks by date updated (since this date.)
- **updated_until** – Filter bookmarks by date updated (until this date.)
- **page** – What page of results to return. Default is 1.
- **per_page** – How many results to return per page. Default is 20, max is 50.
- **only_deleted** – Return only bookmarks that this user has deleted.
- **tags** – Comma separated string of tags to filter bookmarks.

get_tag(*tag_id*)
Get a single tag represented by *tag_id*.

The requested tag must belong to the current user.

Parameters **tag_id** – ID fo the tag to retrieve.

get_tags()
Get all tags belonging to the current user.

get_user()
Retrives the current user.

post(*url*, *post_params=None*)
Make a HTTP POST request to the Reader API.

Parameters

- **url** – url to which to make a POST request.
- **post_params** – parameters to be sent in the request's body.

set_read_percent_of_bookmark(*bookmark_id*, *read_percent*)
Set the read percentage of given bookmark. The requested bookmark must belong to the current user.

Parameters

- **bookmark_id** – ID of the bookmark to update.
- **read_percent** – The read progress made in this article, where 1.0 means the bottom and 0.0 means the very top.

update_bookmark(*bookmark_id*, *favorite=None*, *archive=None*, *read_percent=None*)
Updates given bookmark. The requested bookmark must belong to the current user.

Parameters

- **bookmark_id** – ID of the bookmark to update.
- **(optional)** (*read_percent*) – Whether this article is favorited or not.
- **(optional)** – Whether this article is archived or not.
- **(optional)** – The read progress made in this article, where 1.0 means the bottom and 0.0 means the very top.

3.3 Parser API Client

The Parser API is an API for programmatically extracting content and metadata from html documents. Unlike the Reader API, the Parser API does not require oAuth authentication but rather a single *token* query parameter that must be used to sign every requests. You can find your token by visiting your Readability account settings page.

This *token* can then be passed to the constructor or can be set via environment variables.

```
export READABILITY_PARSER_TOKEN='your parser token here'
```

```
from readability import ParserClient
client = ParserClient(token='your parser token')
```

Under the hood, the *ParserClient* uses the popular requests library. The objects returned by client calls are instances of requests.Response.

3.3.1 Client Documentation

class readability.**ParserClient**(*base_url_template='https://www.readability.com/api/content/v1/{}'*, ***xargs*)

Client for interacting with the Readability Parser API.

Docs can be found at *http://www.readability.com/developers/api/parser*.

get(*url*)

Make an HTTP GET request to the Parser API.

Parameters **url** – url to which to make the request

get_article(*url=None*, *article_id=None*, *max_pages=25*)

Send a GET request to the *parser* endpoint of the parser API to get back the representation of an article.

The article can be identified by either a URL or an id that exists in Readability.

Note that either the *url* or *article_id* param should be passed.

Parameters

- **(optional)** (*article_id*) – The url of an article whose content is wanted.
- **(optional)** – The id of an article in the Readability system whose content is wanted.
- **max_pages** – The maximum number of pages to parse and combine. The default is 25.

get_article_status(*url=None*, *article_id=None*)

Send a HEAD request to the *parser* endpoint to the parser API to get the articles status.

Returned is a *requests.Response* object. The id and status for the article can be extracted from the *X-Article-Id* and *X-Article-Status* headers.

Note that either the *url* or *article_id* param should be passed.

Parameters

- **(optional)** (*article_id*) – The url of an article whose content is wanted.
- **(optional)** – The id of an article in the Readability system whose content is wanted.

get_confidence(*url=None*, *article_id=None*)
Send a GET request to the *confidence* endpoint of the Parser API.

Note that either the *url* or *article_id* param should be passed.

Parameters

- **(optional)** (*article_id*) – The url of an article whose content is wanted.
- **(optional)** – The id of an article in the Readability system whose content is wanted.

get_root()
Send a GET request to the root resource of the Parser API.

head(*url*)
Make an HTTP HEAD request to the Parser API.

Parameters **url** – url to which to make the request

post(*url*, *post_params=None*)
Make an HTTP POST request to the Parser API.

Parameters

- **url** – url to which to make the request
- **post_params** – POST data to send along. Expected to be a dict.

post_article_content(*content*, *url*, *max_pages=25*)
POST content to be parsed to the Parser API.

Note: Even when POSTing content, a url must still be provided.

Parameters

- **content** – the content to be parsed
- **url** – the url that represents the content
- **(optional)** (*max_pages*) – the maximum number of pages to parse and combine. Default is 25.

Index