mwklient Documentation

Release 0.1.0

Luigi Russo

Dec 24, 2019

Contents

mwklient is a *MIT licensed* client library to the MediaWiki API that should work well with both Wikimedia wikis and other wikis running MediaWiki 1.16 or above. It works with Python 2.7 and 3.3+.

CHAPTER 1

Installation

Installing mwklient is simple with pip, just run this in your terminal:

```
pip install mwklient
```

CHAPTER 2

Quickstart

```
>>> site = mwklient.Site('en.wikipedia.org')
>>> page = site.pages[u'Leipäjuusto']
>>> page.text()
u'{{Unreferenced|date=September 2009}}\n[[Image:Leip\xe4juusto cheese with cloudberry␣
↪jam.jpg|thumb|Leip\xe4juusto with [[cloudberry]] jam]]\n\'\'\'Leip\xe4juusto\'\'\'␣
↪(bread cheese) or \'\'juustoleip\xe4\'\', which is also known in English as \'\'\
↪'Finnish squeaky cheese\'\'\', is a fresh [[cheese]] traditionally made from cow\'s␣
↪[[beestings]], rich milk from a cow that has recently calved.'
>>> [x for x in page.categories()]
[<Category object 'Category:Finnish cheeses' for <Site object '('https', 'en.
↪wikipedia.org')/w/'>>,
 <Category object 'Category:Articles lacking sources from September 2009' for <Site␣
↪object '('https', 'en.wikipedia.org')/w/'>>,
 <Category object 'Category:Cow's-milk cheeses' for <Site object '('https', 'en.
↪wikipedia.org')/w/'>>,
 <Category object 'Category:All articles lacking sources' for <Site object '('https',
↪'en.wikipedia.org')/w/'>>]
```

CHAPTER 3

User guide

This guide is intended as an introductory overview, and explains how to make use of the most important features of mwklient.

3.1 User guide

This guide is intended as an introductory overview, and explains how to make use of the most important features of mwklient. For detailed reference documentation of the functions and classes contained in the package, see the *Reference guide*.

3.1.1 Connecting to your site

Begin by importing the Site class:

```
>>> from mwklient import Site
```

Then try to connect to a site:

```
>>> site = Site('test.wikipedia.org')
```

By default, mwklient will connect using https. If your site doesn't support https, you need to explicitly request http like so:

```
>>> site = Site('test.wikipedia.org', scheme='http')
```

The API endpoint location

The API endpoint location on a MediaWiki site depends on the configurable $wgScriptPath. mwklient defaults to the script path '/w/' used by the Wikimedia wikis. If you get a 404 Not Found or a `mwklient.errors.InvalidResponse` error upon connecting, your site might use a different script path. You can specify it using the `path` argument:

```
>>> site = Site('my-awesome-wiki.org', path='/wiki/', )
```

Specifying a user agent

If you are connecting to a Wikimedia site, you should follow the Wikimedia User-Agent policy. The user agent should contain the tool name, the tool version and a way to contact you:

```
>>> ua = 'MyCoolTool/0.2 (xyz@example.org)'
>>> site = Site('test.wikipedia.org', clients_useragent=ua)
```

It should follow the pattern `{tool_name}/{tool_version} ({contact})`. The contact info can also be your user name and the tool version may be omitted: `RudolphBot (User:Santa Claus)`.

Note that MwKlient appends its own user agent to the end of your string.

Errors and warnings

Deprecations and other warnings from the API are logged using the standard Python logging facility, so you can handle them in any way you like. To print them to stdout:

```
>>> import logging
>>> logging.basicConfig(level=logging.WARNING)
```

Errors are thrown as exceptions. All exceptions inherit `mwklient.errors.MwklientError`.

Authenticating

mwklient supports several methods for authentication described below. By default it will also protect you from editing when not authenticated by raising a `mwklient.errors.LoginError`. If you actually *do* want to edit unauthenticated, just set

```
>>> site.force_login = False
```

OAuth

On Wikimedia wikis, the recommended authentication method is to authenticate as a owner-only consumer. Once you have obtained the *consumer token* (also called *consumer key*), the *consumer secret*, the *access token* and the *access secret*, you can authenticate like so:

```
>>> site = Site('test.wikipedia.org',
                consumer_token='my_consumer_token',
                consumer_secret='my_consumer_secret',
                access_token='my_access_token',
                access_secret='my_access_secret')
```

Old-school login

To use old-school login, call the login method:

```
>>> site.login('my_username', 'my_password')
```

If login fails, a `mwklient.errors.LoginError` will be thrown. See *`mwklient.client.Site.login()`* for all options.

HTTP authentication

If your server is configured to use HTTP authentication, you can authenticate using the `httpauth` parameter. For Basic HTTP authentication:

```
>>> site = Site('awesome.site', httpauth=('my_username', 'my_password'))
```

You can also pass in any other authentication mechanism based on the `requests.auth.AuthBase`, such as Digest authentication:

```
>>> from requests.auth import HTTPDigestAuth
>>> site = Site('awesome.site', httpauth=HTTPDigestAuth('my_username', 'my_password'))
```

SSL client certificate authentication

If your server requires a SSL client certificate to authenticate, you can pass the `client_certificate` parameter:

```
>>> site = Site('awesome.site', client_certificate='/path/to/client-and-key.pem')
```

This parameter being a proxy to `requests`' cert parameter, you can also specify a tuple (certificate, key) like:

```
>>> site = Site('awesome.site', client_certificate=('client.pem', 'key.pem'))
```

Please note that the private key must not be encrypted.

Logging out

There is no logout method because merely exiting the script deletes all cookies, achieving the same effect.

3.1.2 Page operations

Start by *connecting* to your site:

```
>>> from mwklient import Site
>>> site = Site('en.wikipedia.org')
```

For information about authenticating, please see *the section on authenticating*.

Editing or creating a page

To get the content of a specific page:

```
>>> page = site.pages['Greater guinea pig']
>>> text = page.text()
```

If a page doesn't exist, `Page.text()` just returns an empty string. If you need to test the existence of the page, use *page.exists*:

```
>>> page.exists
True
```

Edit the text as you like before saving it back to the wiki:

```
>>> page.edit(text, 'Edit summary')
```

If the page didn't exist, this operation will create it.

Listing page revisions

`Page.revisions()` returns a List object that you can iterate over using a for loop. Continuation is handled under the hood so you don't have to worry about it.

Example: Let's find out which users did the most number of edits to a page:

```
>>> users = [rev['user'] for rev in page.revisions()]
>>> unique_users = set(users)
>>> user_revisions = [{'user': user, 'count': users.count(user)} for user in unique_
↪users]
>>> sorted(user_revisions, key=lambda x: x['count'], reverse=True)[:5]
[{'count': 6, 'user': u'Wolf12345'},
 {'count': 4, 'user': u'Test-bot'},
 {'count': 4, 'user': u'Mirxaeth'},
 {'count': 3, 'user': u'192.251.192.201'},
 {'count': 3, 'user': u'78.50.51.180'}]
```

Tip: If you want to retrieve a specific number of revisions, the `itertools.islice` method can come in handy:

```
>>> from datetime import datetime
>>> from time import mktime
>>> from itertools import islice
>>> for revision in islice(page.revisions(), 5):
...     dt = datetime.fromtimestamp(mktime(revision['timestamp']))
...     print '{}'.format(dt.strftime('%F %T'))
```

Patrolling

To revert / undo the edit of a specific page:

```
>>> page.undo(rev_id, summary='reverted this edit')
```

Categories

Categories can be retrieved in the same way as pages, but you can also use `Site.categories()` and skip the namespace prefix. The returned *`Category`* object supports the same methods as the *`Page`* object, but also provides an extra function, `members()`, to list all members of a category.

The Category object can also be used itself as an iterator to yield all its members:

```
>>> category = site.categories['Python']
>>> for page in category:
>>>     print(page.name)
```

Other page operations

There are many other page operations like `backlinks()`, `embeddedin()`, etc. See the *API reference* for more.

3.1.3 Working with files

Assuming you have *connected* to your site.

Getting info about a file

To get information about a file:

```
>>> file = site.images['Example.jpg']
```

where `file` is now an instance of *Image* that you can query for various properties:

```
>>> file.imageinfo
{'comment': 'Reverted to version as of 17:58, 12 March 2010',
 'descriptionshorturl': 'https://commons.wikimedia.org/w/index.php?curid=6428847',
 'descriptionurl': 'https://commons.wikimedia.org/wiki/File:Example.jpg',
 'height': 178,
 'metadata': [{'name': 'MEDIAWIKI_EXIF_VERSION', 'value': 1}],
 'sha1': 'd01b79a6781c72ac9bfff93e5e2cfbeef4efc840',
 'size': 9022,
 'timestamp': '2010-03-14T17:20:20Z',
 'url': 'https://upload.wikimedia.org/wikipedia/commons/a/a9/Example.jpg',
 'user': 'SomeUser',
 'width': 172}
```

You also have easy access to file usage:

```
>>> for page in image.imageusage():
>>>     print('Page:', page.name, '; namespace:', page.namespace)
```

See the *API reference* for more options.

> **Caution:** Note that `Image.exists` refers to whether a file exists *locally*. If a file does not exist locally, but in a shared repo like Wikimedia Commons, it will return `False`.
>
> To check if a file exists locally *or* in a shared repo, you could test if `image.imageinfo != {}`.

Downloading a file

The *Image.download()* method can be used to download the full size file. Pass it a file object and it will stream the image to it, avoiding the need for keeping the whole file in memory:

```
>>> file = site.images['Example.jpg']
>>> with open('Example.jpg', 'wb') as fd:
...     image.download(fd)
```

Uploading a file

```
>>> site.upload(open('file.jpg'), 'destination.jpg', 'Image description')
```

3.1.4 Implementation notes

Most properties and generators accept the same parameters as the API, without their two-letter prefix. Some notable exceptions:

- `Image.imageinfo` is the imageinfo of the latest image. Earlier versions can be fetched using `imagehistory()`.
- `Site.all*`: parameter `(ap)from` renamed to `start`
- `categorymembers` is implemented as `Category.members`
- `deletedrevs` is `deletedrevisions`
- `usercontribs` is `usercontributions`
- First parameters of `search` and `usercontributions` are `search` and `user`, respectively

Properties and generators are implemented as Python generators. Their limit parameter is only an indication of the number of items in one chunk. It is not the total limit. Doing `list(generator(limit = limit))` will return ALL items of generator, and not be limited by the limit value. Use `list(generator(max_items = max_items))` to limit the amount of items returned. Default chunk size is generally the maximum chunk size.

Page objects

The base Page object is called `Page` and from that derive `Category` and `Image`. When the page is retrieved via `Site.pages` or a generator, it will check automatically which of those three specific types should be returned.

For convenience, `Site.images` and `Site.categories` are also provided. These do exactly the same as *Site.Pages*, except that they require the page name without its namespace prefixed.

```
>>> page = site.Pages['Template:Stub']    # a Page object
>>> image = site.Pages['Image:Wiki.png'] # an Image object
>>> image = site.Images['Wiki.png']       # the same Image object
>>> cat = site.Pages['Category:Python']  # a Category object
>>> cat = site.Images['Python']           # the same Category object
```

CHAPTER 4

Reference guide

If you are looking for information on a specific function, class or method, this part of the documentation is for you. It's autogenerated from the source code.

4.1 Reference guide

This is the mwklient API reference, autogenerated from the source code.

4.1.1 Site

class mwklient.client.**Site**(*host*, *path='/w/'*, *ext='.php'*, *pool=None*, *retry_timeout=30*, *max_retries=25*, *wait_callback=<function Site.<lambda>>*, *clients_useragent=None*, *max_lag=3*, *compress=True*, *force_login=True*, *do_init=True*, *httpauth=None*, *reqs=None*, *consumer_token=None*, *consumer_secret=None*, *access_token=None*, *access_secret=None*, *client_certificate=None*, *custom_headers=None*, *scheme='https'*)

A MediaWiki site identified by its hostname.

```
>>> import mwklient
>>> site = mwklient.Site('en.wikipedia.org')
```

Do not include the leading "http://".

mwklient assumes that the script path (where index.php and api.php are located) is '/w/'. If the site uses a different script path, you must specify this (path must end in a '/').

Examples:

```
>>> site = Site('vim.wikia.com', path='/')
>>> site = Site('sourceforge.net', path='/apps/mediawiki/mwknt')
```

allcategories(*start=None*, *prefix=None*, *direc='ascending'*, *limit=None*, *generator=True*, *end=None*)
Retrieve all categories on the wiki as a generator.

allimages(*start=None*, *prefix=None*, *minsize=None*, *maxsize=None*, *limit=None*, *direc='ascending'*, *sha1=None*, *sha1base36=None*, *generator=True*, *end=None*)
Retrieve all images on the wiki as a generator.

alllinks(*start=None*, *prefix=None*, *unique=False*, *prop='title'*, *namespace='0'*, *limit=None*, *generator=True*, *end=None*)
Retrieve a list of all links on the wiki as a generator.

allpages(*start=None*, *prefix=None*, *namespace='0'*, *filterredir='all'*, *minsize=None*, *maxsize=None*, *prtype=None*, *prlevel=None*, *limit=None*, *direc='ascending'*, *filterlanglinks='all'*, *generator=True*, *end=None*)
Retrieve all pages on the wiki as a generator.

allusers(*start=None*, *prefix=None*, *group=None*, *prop=None*, *limit=None*, *witheditsonly=False*, *activeusers=False*, *rights=None*, *end=None*)
Retrieve all users on the wiki as a generator.

api(*action*, *http_method='POST'*, **args*, ***kwargs*)
Perform a generic API call and handle errors.

All arguments will be passed on.

Example

To get coordinates from the GeoData MediaWiki extension at English Wikipedia:

```
>>> site = Site('en.wikipedia.org')
>>> result = site.api('query', prop='coordinates',
titles='Oslo|Copenhagen')
>>> for page in result['query']['pages'].values():
...     if 'coordinates' in page:
...         print '{} {} {}'.format(page['title'],
...             page['coordinates'][0]['lat'],
...             page['coordinates'][0]['lon'])
Oslo 59.95 10.75
Copenhagen 55.6761 12.5683
```

Returns The raw response from the API call, as a dictionary.

ask(*query*, *title=None*)
Ask a query against Semantic MediaWiki.

API doc: https://semantic-mediawiki.org/wiki/Ask_API

Returns Generator for retrieving all search results, with each answer as a dictionary. If the query is invalid, an APIError is raised. A valid query with zero results will not raise any error.

Examples

```
>>> query = "[[Category:mycat]]|[[Has name::a name]]|?Has property"
>>> for answer in site.ask(query):
>>>     for title, data in answer.items()
>>>         print(title)
>>>         print(data)
```

blocks(*start=None*, *end=None*, *direc='older'*, *ids=None*, *users=None*, *limit=None*, *prop='id|user|by|timestamp|expiry|reason|flags'*)
Retrieve blocks as a generator.

Returns

Generator yielding dicts, each dict containing:

- user: The username or IP address of the user
- id: The ID of the block
- timestamp: When the block was added
- expiry: When the block runs out (infinity for indefinite

blocks) - reason: The reason they are blocked - allowusertalk: Key is present (empty string) if the user is allowed to edit their user talk page - by: the administrator who blocked the user - nocreate: key is present (empty string) if the user's ability to create accounts has been disabled.

Return type mwklient.listings.List

checkuserlog(*user=None*, *target=None*, *limit=10*, *direc='older'*, *start=None*, *end=None*)
Retrieve checkuserlog items as a generator.

chunk_upload(*file*, *filename*, *ignorewarnings*, *comment*, *text*)
Upload a file to the site in chunks.

This method is called by *Site.upload* if you are connecting to a newer MediaWiki installation, so it's normally not necessary to call this method directly.

Parameters

- **file** (*file-like object*) – File object or stream to upload.
- **params** (*dict*) – Dict containing upload parameters.

email(*user*, *text*, *subject*, *cc=False*)
Send email to a specified user on the wiki.

```
>>> try:
...     site.email('SomeUser', 'Some message', 'Some subject')
... except mwklient.errors.NoSpecifiedEmailError as err:
...     print 'The user does not accept email, or has not specified
an email address.'
```

Parameters

- **user** (*str*) – User name of the recipient
- **text** (*str*) – Body of the email
- **subject** (*str*) – Subject of the email
- **cc** (*bool*) – True to send a copy of the email to yourself
- **is False)** (*(default)* –

Returns Dictionary of the JSON response

Raises

- NoSpecifiedEmailError (mwklient.errors.NoSpecifiedEmailError)
- if recipient does not accept email

- *EmailError (mwklient.errors.EmailError)* – on other errors

expandtemplates (*text*, *title=None*, *generatexml=False*)
Takes wikitext (text) and expands templates.

API doc: https://www.mediawiki.org/wiki/API:Expandtemplates

exturlusage (*query*, *prop=None*, *protocol='http'*, *namespace=None*, *limit=None*)
Retrieve the list of pages that link to a particular domain or URL, as a generator.

This API call mirrors the Special:LinkSearch function on-wiki.

Query can be a domain like 'bbc.co.uk'. Wildcards can be used, e.g. '*.bbc.co.uk'. Alternatively, a query can contain a full domain name and some or all of a URL: e.g. '*.wikipedia.org/wiki/*'

See <https://meta.wikimedia.org/wiki/Help:Linksearch> for details.

Returns

Generator yielding dicts, each dict containing:

- url: The URL linked to.
- ns: Namespace of the wiki page
- pageid: The ID of the wiki page
- title: The page title.

Return type mwklient.listings.List

get (*action*, **args*, ***kwargs*)
Perform a generic API call using GET.

This is just a shorthand for calling api() with http_method='GET'. All arguments will be passed on.

Returns The raw response from the API call, as a dictionary.

logevents (*type_t=None*, *prop=None*, *start=None*, *end=None*, *direc='older'*, *user=None*, *title=None*, *limit=None*, *action=None*)
Retrieve logevents as a generator.

login (*username=None*, *password=None*, *cookies=None*, *domain=None*)
Login to the wiki using a username and password. The method returns nothing if the login was successful, but raises and error if it was not

Parameters

- **username** (*str*) – MediaWiki username
- **password** (*str*) – MediaWiki password
- **cookies** (*dict*) – Custom cookies to include with the log-in request.
- **domain** (*str*) – Sends domain name for authentication; used by some MediaWiki plugins like the 'LDAP Authentication' extension.

Raises

- *LoginError (mwklient.errors.LoginError)* – Login failed, the reason can be obtained from e.code and e.info (where e is the exception object) and will be one of the API:Login errors. The most common error code is "Failed", indicating a wrong username or password.
- MaximumRetriesExceeded – API call to log in failed and was retried
- until all retries were exhausted. This will not occur if the
- credentials are merely incorrect. See MaximumRetriesExceeded for

- possible reasons.
- `APIError` – An API error occurred. Rare, usually indicates an
- internal server error.

post (*action*, **args*, ***kwargs*)
Perform a generic API call using POST.

This is just a shorthand for calling api() with http_method='POST'. All arguments will be passed on.

Returns The raw response from the API call, as a dictionary.

random (*namespace*, *limit=20*)
Retrieve a generator of random pages from a particular namespace.

limit specifies the number of random articles retrieved. namespace is a namespace identifier integer.

Generator contains dictionary with namespace, page ID and title.

raw_api (*action*, *http_method='POST'*, **args*, ***kwargs*)
Send a call to the API.

raw_call (*script*, *data*, *files=None*, *retry_on_error=True*, *http_method='POST'*)
Perform a generic request and return the raw text.

In the event of a network problem, or a HTTP response with status code 5XX, we'll wait and retry the configured number of times before giving up if *retry_on_error* is True.

requests.exceptions.HTTPError is still raised directly for HTTP responses with status codes in the 4XX range, and invalid HTTP responses.

Parameters

- **script** (*str*) – Script name, usually 'api'.
- **data** (*dict*) – Post data
- **files** (*dict*) – Files to upload
- **retry_on_error** (*bool*) – Retry on connection error
- **http_method** (*str*) – The HTTP method, defaults to 'POST'

Returns The raw text response.

raw_index (*action*, *http_method='POST'*, **args*, ***kwargs*)
Sends a call to index.php rather than the API.

recentchanges (*start=None*, *end=None*, *direc='older'*, *namespace=None*, *prop=None*, *show=None*, *limit=None*, *type_t=None*, *toponly=None*)
List recent changes to the wiki, à la Special:Recentchanges.

revisions (*revids*, *prop='ids|timestamp|flags|comment|user'*)
Get data about a list of revisions.

See also the *Page.revisions()* method.

API doc: https://www.mediawiki.org/wiki/API:Revisions

Example: Get revision text for two revisions:

```
>>> for revision in site.revisions([689697696, 689816909],
prop='content'):
...     print revision['*']
```

Parameters

- **revids** (*list*) – A list of (max 50) revisions.
- **prop** (*str*) – Which properties to get for each revision.

Returns A list of revisions

search(*search*, *namespace='0'*, *what=None*, *redirects=False*, *limit=None*)
Perform a full text search.

API doc: https://www.mediawiki.org/wiki/API:Search

Example

```
>>> for result in site.search('prefix:Template:Citation/'):
...     print(result.get('title'))
```

Parameters

- **search** (*str*) – The query string
- **namespace** (*int*) – The namespace to search (default: 0)
- **what** (*str*) – Search scope: 'text' for fulltext, or 'title' for
- **only.** (*titles*) – Depending on the search backend, both options may not be available. For instance CirrusSearch doesn't support 'title', but instead provides an "intitle:" query string filter.
- **redirects** (*bool*) – Include redirect pages in the search (option removed in MediaWiki 1.23).

Returns Search results iterator

Return type mwklient.listings.List

upload(*file=None*, *filename=None*, *description=''*, *ignore=False*, *url=None*, *filekey=None*, *comment=None*)
Upload a file to the site.

Note that one of *file*, *filekey* and *url* must be specified, but not more than one. For normal uploads, you specify *file*.

Parameters

- **file** (*str*) – File object or stream to upload.
- **filename** (*str*) – Destination filename, don't include namespace prefix like 'File:'
- **description** (*str*) – Wikitext for the file description page.
- **ignore** (*bool*) – True to upload despite any warnings.
- **url** (*str*) – URL to fetch the file from.
- **filekey** (*str*) – Key that identifies a previous upload that was stashed temporarily.
- **comment** (*str*) – Upload comment. Also used as the initial page text for new files if *description* is not specified.

Example

```
>>> client.upload(open('somefile', 'rb'), filename='somefile.jpg',
                  description='Some description')
```

Returns JSON result from the API.

Raises

- `errors.InsufficientPermission`
- `requests.exceptions.HTTPError`

usercontributions(*user*, *start=None*, *end=None*, *direc='older'*, *namespace=None*, *prop=None*, *show=None*, *limit=None*, *uselang=None*)
List the contributions made by a given user to the wiki

API doc: https://www.mediawiki.org/wiki/API:Usercontribs

users(*users*, *prop='blockinfo|groups|editcount'*)
Get information about a list of users.

API doc: https://www.mediawiki.org/wiki/API:Users

watchlist(*allrev=False*, *start=None*, *end=None*, *namespace=None*, *direc='older'*, *prop=None*, *show=None*, *limit=None*)
List the pages on the current user's watchlist.

API doc: https://www.mediawiki.org/wiki/API:Watchlist

4.1.2 Page

class mwklient.page.**Page**(*site*, *name*, *info=None*, *extra_properties=None*)

can(*action*)
Check if the current user has the right to carry out some action with the current page.

Example

```
>>> page.can('edit')
True
```

cannot(*action*)
Check if the current user has not the right to carry out some action with the current page.

Example

```
>>> page.cannot('edit')
True
```

purge()
Purge server-side cache of page. This will re-render templates and other dynamic content.

redirects_to()
Returns the redirect target page, or None if the page is not a redirect page.

resolve_redirect()
Returns the redirect target page, or the current page if it's not a redirect page.

revisions(*startid=None*, *endid=None*, *start=None*, *end=None*, *direc='older'*, *user=None*, *excludeuser=None*, *limit=50*, *prop='ids|timestamp|flags|comment|user'*, *expandtemplates=False*, *section=None*, *diffto=None*, *slots=None*, *uselang=None*)
List revisions of the current page.

API doc: https://www.mediawiki.org/wiki/API:Revisions

Parameters

- **startid** (*int*) – Revision ID to start listing from.
- **endid** (*int*) – Revision ID to stop listing at.
- **start** (*str*) – Timestamp to start listing from.
- **end** (*str*) – Timestamp to end listing at.
- **direc** (*str*) – Direction to list in: 'older' (default) or 'newer'.
- **user** (*str*) – Only list revisions made by this user.
- **excludeuser** (*str*) – Exclude revisions made by this user.
- **limit** (*int*) – The maximum number of revisions to return per request.
- **prop** (*str*) – Which properties to get for each revision, default: 'ids|timestamp|flags|comment|user'
- **expandtemplates** (*bool*) – Expand templates in rvprop=content output
- **section** (*int*) – If rvprop=content is set, only retrieve the contents
- **this section.** (*of*) –
- **diffto** (*str*) – Revision ID to diff each revision to. Use "prev", "next" and "cur" for the previous, next and current revision respectively.
- **slots** (*str*) – The content slot (Mediawiki >= 1.32) to retrieve content from.
- **uselang** (*str*) – Language to use for parsed edit comments and other localized messages.

Returns Revision iterator

Return type mwklient.listings.List

class mwklient.listing.**Category**(*site*, *name*, *info=None*, *namespace=None*)

4.1.3 Image

class mwklient.image.**Image**(*site*, *name*, *info=None*)

download(*destination=None*)
Download the file. If *destination* is given, the file will be written directly to the stream. Otherwise the file content will be stored in memory and returned (with the risk of running out of memory for large files).

Recommended usage:

```
>>> with open(filename, 'wb') as fd:
...     image.download(fd)
```

Parameters **destination** (*file object*) – Destination file

duplicatefiles(*limit=None*)
List duplicates of the current file.

API doc: https://www.mediawiki.org/wiki/API:Duplicatefiles

imagehistory()
Get file revision info for the given file.

API doc: https://www.mediawiki.org/wiki/API:Imageinfo

imageusage(*namespace=None*, *filterredir='all'*, *redirect=False*, *limit=None*, *generator=True*)
List pages that use the given file.

API doc: https://www.mediawiki.org/wiki/API:Imageusage

4.1.4 InsufficientPermission

class mwklient.errors.**InsufficientPermission**

CHAPTER 5

Development

Looking for information on contributing to mwklient development?

5.1 Development

Mwklient development is coordinated at https://github.com/lrusso96/mwklient. Patches are very welcome. Development environment ——————————

If you plan to submit a pull request, you should first fork the mwklient repo on GitHub, then clone your own fork:

```
$ git clone git@github.com:MYUSERNAME/mwklient.git
$ cd mwklient
```

You can then use pip to do an "editable" install so that your edits will be immediately available for (both interactive and automated) testing:

```
$ pip install -e .
```

5.1.1 Test suite

mwklient ships with a test suite based on pytest. While it's far from complete, it can sometimes alert you if you break things.

The easiest way to run the tests is:

```
$ python setup.py test
```

This will make an in-place build and download test dependencies locally if needed. Tests will run faster, however, if you do an editable install and run pytest directly:

```
$ pip install pytest pytest-cov flake8 responses mock
$ pip install -e .
$ py.test
```

If you want to test with different Python versions in isolated virtualenvs, you can use Tox. A *tox.ini* file is included.

```
$ pip install tox
$ tox
```

If you would like to expand the test suite by adding more tests, please go ahead!

5.1.2 Updating/expanding the documenation

Documentation consists of both a manually compiled user guide (under `docs/user`) and a reference guide generated from the docstrings, using Sphinx autodoc with the napoleon extension. Documentation is built automatically on ReadTheDocs after each commit. To build the documentation locally for testing:

```
$ pip install Sphinx sphinx-rtd-theme
$ cd docs
$ make html
```

When writing docstrings, try to adhere to the Google style.

5.1.3 Making a pull request

Make sure to run tests before committing. When it comes to the commit message, there's no specific requirements for the format, but try to explain your changes in a clear and concise manner.

If it's been some time since you forked, please consider rebasing your branch on the main master branch to ease merging:

```
$ git remote add upstream https://github.com/lrusso96/mwklient.git
$ git rebase upstream master
```

Then push your code and open a pull request on GitHub.

CHAPTER 6

MIT License

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index