mwclient Documentation

Release 0.8.4

Bryan Tong Minh

May 11, 2017

Contents

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

```
pip install mwclient
```

CHAPTER 2

Quickstart

```
>>> site = mwclient.Site(('https', '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 mwclient.

User guide

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

Connecting to your site

Begin by importing the Site class:

```
>>> from mwclient import Site
```

Then try to connect to a site:

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

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

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

The API endpoint location

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

```
>>> site = mwclient.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 and identify your tool like so:

```
>>> ua = 'MyCoolTool/0.2 run by User:Xyz'
>>> site = mwclient.Site('test.wikipedia.org', clients_useragent=ua)
```

Note that Mwclient appends ' - MwClient/{version} ({url})' to your string.

Authenticating

Mwclient supports several methods for authentication described below. By default it will also protect you from editing when not authenticated by raising a `mwclient.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 = mwclient.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 `mwclient.errors.LoginError` will be thrown.

HTTP authentication

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

```
>>> site = mwclient.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 = mwclient.Site('awesome.site', httpauth=HTTPDigestAuth('my_username', 'my_
↪password'))
```

SSL client certificate authentication

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

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

Please note that the private key must not be encrypted.

Page operations

Start by *connecting* to your site:

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

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

Editing or creating a page

To get the wikitext for 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.save(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'))
```

Other page operations

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

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 != {}`.

Uploading a file

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

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.

Reference guide

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

Site

class mwclient.client.**Site**(*host*, *path='/w/'*, *ext='.php'*, *pool=None*, *retry_timeout=30*, *max_retries=25*, *wait_callback=<function <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*)

A MediaWiki site identified by its hostname.

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

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

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

allcategories(*start=None*, *prefix=None*, *dir='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*, *dir='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*, *dir='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

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

Each block is a dictionary 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.

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

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

```
>>> try:
...     site.email('SomeUser', 'Some message', 'Some subject')
... except mwclient.errors.NoSpecifiedEmailError as e:
...     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 (default is False)

Returns Dictionary of the JSON response

Raises

- *NoSpecifiedEmailError (mwclient.errors.NoSpecifiedEmailError)* – if recipient does not accept email
- *EmailError (mwclient.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.

The generator returns dictionaries containing three keys: - url: the URL linked to. - ns: namespace of the wiki page - pageid: the ID of the wiki page - title: the page title.

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=None*, *prop=None*, *start=None*, *end=None*, *dir='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.

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

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*, *dir='older'*, *namespace=None*, *prop=None*, *show=None*, *limit=None*, *type=None*, *toponly=None*)
List recent changes to the wiki, à la Special:Recentchanges.

revisions (*revids*, *prop='ids|timestamp|flags|comment|user'*, *expandtemplates=False*, *diffto='prev'*)
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.
- **expandtemplates** (*bool*) – Expand templates in *rvprop=content* output.
- **diffto** (*str*) – Revision ID to diff each revision to. Use "prev", "next" and "cur" for the previous, next and current revision respectively.

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 titles only. 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 mwclient.listings.List

upload (*file=None*, *filename=None*, *description=''*, *ignore=False*, *file_size=None*, *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.
- **file_size** (*int*) – Deprecated in mwclient 0.7
- **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*, *dir='older'*, *namespace=None*, *prop=None*, *show=None*, *limit=None*)
List the contributions made by a given user to the wiki, à la Special:Contributions.

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

static **version_tuple_from_generator** (*string*, *prefix='MediaWiki '*)
Return a version tuple from a MediaWiki Generator string.

Example

"MediaWiki 1.5.1" → (1, 5, 1)

Parameters **prefix** (*str*) – The expected prefix of the string

watchlist (*allrev=False*, *start=None*, *end=None*, *namespace=None*, *dir='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

Page

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

backlinks (*namespace=None*, *filterredir='all'*, *redirect=False*, *limit=None*, *generator=True*)
List pages that link to the current page, similar to Special:Whatlinkshere.

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

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

Example

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

categories(*generator=True*)
List categories used on the current page.

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

delete(*reason=''*, *watch=False*, *unwatch=False*, *oldimage=False*)
Delete page.

If user does not have permission to delete page, an InsufficientPermission exception is raised.

edit(**args*, ***kwargs*)
Deprecated. Use page.text() instead

embeddedin(*namespace=None*, *filterredir='all'*, *limit=None*, *generator=True*)
List pages that transclude the current page.

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

Parameters

- **namespace** (*int*) – Restricts search to a given namespace (Default: None)
- **filterredir** (*str*) – How to filter redirects, either 'all' (default), 'redirects' or 'nonredirects'.
- **limit** (*int*) – Maximum amount of pages to return per request
- **generator** (*bool*) – Use generator

Returns Page iterator

Return type mwclient.listings.List

extlinks()
List external links from the current page.

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

images(*generator=True*)
List files/images embedded in the current page.

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

iwlinks()
List interwiki links from the current page.

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

langlinks(***kwargs*)
List interlanguage links from the current page.

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

links(*namespace=None*, *generator=True*, *redirects=False*)
List links to other pages from the current page.

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

move(*new_title*, *reason=''*, *move_talk=True*, *no_redirect=False*)
Move (rename) page to new_title.

If user account is an administrator, specify no_redirect as True to not leave a redirect.

If user does not have permission to move page, an InsufficientPermission exception is raised.

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*, *dir='older'*, *user=None*, *excludeuser=None*, *limit=50*, *prop='ids|timestamp|flags|comment|user'*, *expandtemplates=False*, *section=None*, *diffto=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.
- **dir** (*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 of this section.
- **diffto** (*str*) – Revision ID to diff each revision to. Use "prev", "next" and "cur" for the previous, next and current revision respectively.

Returns Revision iterator

Return type mwclient.listings.List

save(*text*, *summary=u''*, *minor=False*, *bot=True*, *section=None*, ***kwargs*)
Update the text of a section or the whole page by performing an edit operation.

templates(*namespace=None*, *generator=True*)
List templates used on the current page.

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

text(*section=None*, *expandtemplates=False*, *cache=True*)
Get the current wikitext of the page, or of a specific section.

If the page does not exist, an empty string is returned. By default, results will be cached and if you call text() again with the same section and expandtemplates the result will come from the cache. The cache is stored on the instance, so it lives as long as the instance does.

Parameters

- **section** (*int*) – numbered section or *None* to get the whole page (default: *None*)
- **expandtemplates** (*bool*) – set to *True* to expand templates (default: *False*)
- **cache** (*bool*) – set to *False* to disable caching (default: *True*)

Image

class mwclient.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

InsufficientPermission

class mwclient.errors.**InsufficientPermission**

CHAPTER 5

Development

Looking for information on contributing to mwclient development?

Development

Mwclient development is coordinated at https://github.com/mwclient/mwclient. Patches are very welcome. There's currently no chat room or mailing list for the project, but don't hesitate to use the issue tracker at GitHub for general discussions.

Development environment

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

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

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 .
```

Running tests

To run the automated tests, install the test dependencies and run pytest:

```
$ pip install pytest pytest-pep8 responses
$ py.test
```

To run tests with different Python versions in isolated virtualenvs, you can use Tox:

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

Note that the test suite is quite limited yet. If you'd like to expand it by adding more tests, please go ahead!

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/mwclient/mwclient.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

V

W