launchpadlib Documentation

Release 1.11.0

LAZR Developers <lazr-developers@lists.launchpad.net>

Jun 30, 2023

CONTENTS

See https://help.launchpad.net/API/launchpadlib .

CHAPTER

ONE

LAUNCHPADLIB

launchpadlib is the standalone Python language bindings to Launchpad's web services API. It is officially supported by Canonical, although third party packages may be available to provide bindings to other programming languages.

1.1 Set up

launchpadlib writes to $HOME, so isolate ourselves.

```
>>> from fixtures import (
...     EnvironmentVariable,
...     TempDir,
...     )
>>> tempdir_fixture = TempDir()
>>> tempdir_fixture.setUp()
>>> home_fixture = EnvironmentVariable('HOME', tempdir_fixture.path)
>>> home_fixture.setUp()
```

1.2 OAuth authentication

The Launchpad API requires user authentication via OAuth, and launchpadlib provides a high level interface to OAuth for the most common use cases. Several pieces of information are necessary to complete the OAuth request:

- A consumer key, which is unique to the application using the API
- An access token, which represents the user to the web service
- An access token secret, essentially a password for the token

Consumer keys are hard-baked into the application. They are generated by the application developer and registered with Launchpad independently of the use of the application. Since consumer keys are arbitrary, a registered consumer key can be paired with a secret, but most open source applications will forgo this since it's not really a secret anyway.

The access token cannot be provided directly. Instead, the application generates an unauthenticated request token, exchanging this for an access token and a secret after obtaining approval to do so from the user. This permission is typically gained by redirecting the user through their trusted web browser, then back to the application.

This entire exchange is managed by launchpadlib's credentials classes. Credentials can be stored in a file, though the security of this depends on the implementation of the file object. In the simplest case, the application will request a new access token every time.

```
>>> from launchpadlib.credentials import Consumer
>>> consumer = Consumer('launchpad-library')
>>> consumer.key
'launchpad-library'
>>> consumer.secret
''
```

Salgado has full access to the Launchpad API. Out of band, the application itself obtains Salgado's approval to access the Launchpad API on his behalf. How the application does this is up to the application, provided it conforms to the OAuth protocol. Once this happens, we have Salgado's credentials for accessing Launchpad.

```
>>> from launchpadlib.credentials import AccessToken
>>> access_token = AccessToken('salgado-change-anything', 'test')
```

And now these credentials are used to access the root service on Salgado's behalf.

```
>>> from launchpadlib.credentials import Credentials
>>> credentials = Credentials(
...     consumer_name=consumer.key, consumer_secret=consumer.secret,
...     access_token=access_token)
```

```
>>> from launchpadlib.testing.helpers import (
...     TestableLaunchpad as Launchpad)
>>> launchpad = Launchpad(credentials=credentials)
>>> list(launchpad.people)
[...]
>>> list(launchpad.bugs)
[...]
```

If available, the Gnome keyring or KDE wallet will be used to store access tokens. If a keyring/wallet is not available, the application can store the credentials on the file system, so that the next time Salgado interacts with the application, he won't have to go through the whole OAuth request dance.

```
>>> import os
>>> import tempfile
>>> fd, path = tempfile.mkstemp('.credentials')
>>> os.close(fd)
```

Once Salgado's credentials are obtained for the first time, just set the appropriate instance variables and use the save() method.

```
>>> credentials.consumer = consumer
>>> credentials.access_token = access_token
>>> credentials_file = open(path, 'w')
>>> credentials.save(credentials_file)
>>> credentials_file.close()
```

And the credentials are perfectly valid for accessing Launchpad.

```
>>> launchpad = Launchpad(credentials=credentials)
>>> list(launchpad.people)
[...]
>>> list(launchpad.bugs)
[...]
```

The credentials can also be retrieved from the file, so that the OAuth request dance can be avoided.

```
>>> credentials = Credentials()
>>> credentials_file = open(path)
>>> credentials.load(credentials_file)
>>> credentials_file.close()
>>> credentials.consumer.key
'launchpad-library'
>>> credentials.consumer.secret
''
>>> credentials.access_token.key
'salgado-change-anything'
>>> credentials.access_token.secret
'test'
```

These credentials too, are perfectly usable to access Launchpad.

```
>>> launchpad = Launchpad(credentials=credentials)
>>> list(launchpad.people)
[...]
>>> list(launchpad.bugs)
[...]
```

The security of the stored credentials is left up to the file-like object. Here, the application decides to use a dubious encryption algorithm to hide Salgado's credentials.

```
>>> import io
>>> from codecs import EncodedFile
>>> encrypted_file = io.BytesIO()
>>> stream = EncodedFile(encrypted_file, 'rot_13', 'ascii')
>>> credentials.save(stream)
>>> _ = stream.seek(0, 0)
>>> print(''.join(sorted([line.decode() for line in encrypted_file])))
[1]


npprff_frperg = grfg
npprff_gbxra = fnytnqb-punatr-nalguvat
pbafhzre_frperg =
pbafhzre_xrl = ynhapucnq-yvoenel
```

```
>>> _ = stream.seek(0)
>>> credentials = Credentials()
>>> credentials.load(stream)
>>> credentials.consumer.key
'launchpad-library'
>>> credentials.consumer.secret
''
>>> credentials.access_token.key
'salgado-change-anything'
>>> credentials.access_token.secret
'test'
```

1.3 Anonymous access

An anonymous access token doesn't authenticate any particular user. Using it will give a client read-only access to the public parts of the Launchpad dataset.

```
>>> from launchpadlib.credentials import AnonymousAccessToken
>>> anonymous_token = AnonymousAccessToken()
```

```
>>> from launchpadlib.credentials import Credentials
>>> credentials = Credentials(
...     consumer_name="a consumer", access_token=anonymous_token)
>>> launchpad = Launchpad(credentials=credentials)
```

```
>>> salgado = launchpad.people['salgado']
>>> print(salgado.display_name)
Guilherme Salgado
```

An anonymous client can't modify the dataset, or read any data that's permission-controlled or scoped to a particular user.

```
>>> launchpad.me
Traceback (most recent call last):
...
lazr.restfulclient.errors.Unauthorized: HTTP Error 401: Unauthorized
...
```

```
>>> salgado.display_name = "This won't work."
>>> salgado.lp_save()
Traceback (most recent call last):
...
lazr.restfulclient.errors.Unauthorized: HTTP Error 401: Unauthorized
...
```

1.4 Convenience

When you want anonymous access, a convenience method is available for setting up a web service connection in one function call. All you have to provide is the consumer name.

```
>>> launchpad = Launchpad.login_anonymously(
...     'launchpad-library', service_root="test_dev")
>>> list(launchpad.people)
[...]
```

```
>>> launchpad.me
Traceback (most recent call last):
...
lazr.restfulclient.errors.Unauthorized: HTTP Error 401: Unauthorized
...
```

Otherwise, the application should obtain authorization from the user and get a new set of credentials directly from Launchpad.

Unfortunately, we can't test this entire process because it requires opening up a web browser, but we can test the first step, which is to get a request token.

```
>>> import launchpadlib.credentials
>>> credentials = Credentials('consumer')
```

```
>>> authorization_url = credentials.get_request_token(
...     context='firefox', web_root='test_dev')
>>> print(authorization_url)
http://launchpad.test:8085/+authorize-token?oauth_token=...&lp.context=firefox
```

We use 'test_dev' as a shorthand for the root URL of the Launchpad installation. It's defined in the 'uris' module as 'http://launchpad.test:8085/', and the launchpadlib code knows how to dereference it before using it as a URL.

Information about the request token is kept in the _request_token attribute of the Credentials object.

```
>>> credentials._request_token.key is not None
True
>>> credentials._request_token.secret is not None
True
>>> print(credentials._request_token.context)
firefox
```

Now the user must authorize that token, and this is the part we can't test–it requires opening a web browser. Once the token is authorized on the server side, we can call exchange_request_token_for_access_token() on our Credentials object, which will then be ready to use.

1.5 The dictionary request token

By default, get_request_token returns the URL that the user needs to use when granting access to the token. But you can specify a different token_format and get a dictionary instead.

```
>>> credentials = Credentials('consumer')
>>> dictionary = credentials.get_request_token(
...     context='firefox', web_root='test_dev',
...     token_format=Credentials.DICT_TOKEN_FORMAT)
```

The dictionary has useful information about the token and about the levels of authentication Launchpad offers.

```
>>> for param in sorted(dictionary.keys()):
...     print(param)
access_levels
lp.context
oauth_token
oauth_token_consumer
oauth_token_secret
```

The _request_token attribute of the Credentials object has the same fields set as if you had asked for the default URI token format.

```
>>> credentials._request_token.key is not None
True
>>> credentials._request_token.secret is not None
```

(continues on next page)

(continued from previous page)

```
True
>>> print(credentials._request_token.context)
firefox
```

1.6 Credentials file errors

If the credentials file is empty, loading it raises an exception.

```
>>> credentials = Credentials()
>>> credentials.load(io.StringIO())
Traceback (most recent call last):
...
lazr.restfulclient.errors.CredentialsFileError: No configuration for
version 1
```

It is an error to save a credentials file when no consumer or access token is available.

```
>>> credentials.consumer = None
>>> credentials.save(io.StringIO())
Traceback (most recent call last):
...
lazr.restfulclient.errors.CredentialsFileError: No consumer
```

```
>>> credentials.consumer = consumer
>>> credentials.access_token = None
>>> credentials.save(io.StringIO())
Traceback (most recent call last):
...
lazr.restfulclient.errors.CredentialsFileError: No access token
```

The credentials file is not intended to be edited, but because it's human readable, that's of course possible. If the credentials file gets corrupted, an error is raised.

```
>>> credentials_file = io.StringIO("""\
... [1]
... #consumer_key: aardvark
... consumer_secret: badger
... access_token: caribou
... access_secret: dingo
... """)
>>> credentials.load(credentials_file)
Traceback (most recent call last):
...
configparser.NoOptionError: No option 'consumer_key' in section: '1'
```

```
>>> credentials_file = io.StringIO("""\
... [1]
... consumer_key: aardvark
... #consumer_secret: badger
... access_token: caribou
```

(continues on next page)

(continued from previous page)

```
... access_secret: dingo
... """)
>>> credentials.load(credentials_file)
Traceback (most recent call last):
...
configparser.NoOptionError: No option 'consumer_secret' in section: '1'
```

```
>>> credentials_file = io.StringIO("""\
... [1]
... consumer_key: aardvark
... consumer_secret: badger
... #access_token: caribou
... access_secret: dingo
... """)
>>> credentials.load(credentials_file)
Traceback (most recent call last):
...
configparser.NoOptionError: No option 'access_token' in section: '1'
```

```
>>> credentials_file = io.StringIO("""\
... [1]
... consumer_key: aardvark
... consumer_secret: badger
... access_token: caribou
... #access_secret: dingo
... """)
>>> credentials.load(credentials_file)
Traceback (most recent call last):
...
configparser.NoOptionError: No option 'access_secret' in section: '1'
```

1.7 Bad credentials

The application is not allowed to access Launchpad with a bad access token.

```
>>> access_token = AccessToken('bad', 'no-secret')
>>> credentials = Credentials(
...     consumer_name=consumer.key, consumer_secret=consumer.secret,
...     access_token=access_token)
>>> launchpad = Launchpad(credentials=credentials)
Traceback (most recent call last):
...
lazr.restfulclient.errors.Unauthorized: HTTP Error 401: Unauthorized
...
```

The application is not allowed to access Launchpad with a consumer name that doesn't match the credentials.

```
>>> access_token = AccessToken('salgado-change-anything', 'test')
>>> credentials = Credentials(
...     consumer_name='not-the-launchpad-library',
```

(continues on next page)

(continued from previous page)

```
...     access_token=access_token)
>>> launchpad = Launchpad(credentials=credentials)
Traceback (most recent call last):
...
lazr.restfulclient.errors.Unauthorized: HTTP Error 401: Unauthorized
...
```

The application is not allowed to access Launchpad with a bad access secret.

```
>>> access_token = AccessToken('hgm2VK35vXD6rLg5pxWw', 'bad-secret')
>>> credentials = Credentials(
...     consumer_name=consumer.key, consumer_secret=consumer.secret,
...     access_token=access_token)
>>> launchpad = Launchpad(credentials=credentials)
Traceback (most recent call last):
...
lazr.restfulclient.errors.Unauthorized: HTTP Error 401: Unauthorized
...
```

1.8 Clean up

```
>>> os.remove(path)
>>> home_fixture.cleanUp()
>>> tempdir_fixture.cleanUp()
```

CHAPTER

TWO

NAMED OPERATIONS

launchpadlib can transparently determine the size of the list even when the size is not directly provided, but is only available through a link.

```
>>> from launchpadlib.testing.helpers import salgado_with_full_permissions
>>> launchpad = salgado_with_full_permissions.login(version="devel")
```

```
>>> results = launchpad.people.find(text='s')
>>> 'total_size' in results._wadl_resource.representation.keys()
False
>>> 'total_size_link' in results._wadl_resource.representation.keys()
True
>>> len(results) > 1
True
```

Of course, launchpadlib can also determine the size when the size _is_ directly provided.

```
>>> results = launchpad.people.find(text='salgado')
>>> 'total_size' in results._wadl_resource.representation.keys()
True
>>> len(results) == 1
True
```

CHAPTER

THREE

TOP-LEVEL COLLECTIONS

The launchpad web service's top-level collections provide access to Launchpad-wide objects like projects and people.

```
>>> import httplib2
>>> httplib2.debuglevel = 1
```

```
>>> from launchpadlib.testing.helpers import salgado_with_full_permissions
>>> launchpad = salgado_with_full_permissions.login()
send: ...
...
```

It's possible to do key-based lookups on the top-level collections. The bug collection does lookups by bug ID.

```
>>> bug = launchpad.bugs[1]
send: b'GET /.../bugs/1 ...'
...
```

To avoid triggering an HTTP request when simply looking up an object, you can use a different syntax:

```
>>> bug = launchpad.bugs(1)
```

The HTTP request will happen when you need information that can only be obtained from the web service.

```
>>> print(bug.id)
send: b'GET /.../bugs/1 ...'
...
1
```

Let's look at some more collections. The project collection does lookups by project name.

```
>>> project = launchpad.projects('firefox')
>>> print(project.name)
send: b'GET /.../firefox ...'
...
firefox
```

The project group collection does lookups by project group name.

```
>>> group = launchpad.project_groups('gnome')
>>> print(group.name)
send: b'GET /.../gnome ...'
...
gnome
```

The distribution collection does lookups by distribution name.

```
>>> distribution = launchpad.distributions('ubuntu')
>>> print(distribution.name)
send: b'GET /.../ubuntu ...'
...
ubuntu
```

The person collection does lookups by a person's Launchpad name.

```
>>> person = launchpad.people('salgado')
>>> print(person.name)
send: b'GET /.../~salgado ...'
...
salgado
```

```
>>> team = launchpad.people('rosetta-admins')
>>> print(team.name)
send: b'GET /1.0/~rosetta-admins ...'
...
rosetta-admins
```

How does launchpadlib know that 'salgado' is a person and 'rosetta-admins' is a team?

```
>>> print(person.resource_type_link)
http://.../1.0/#person
>>> 'default_membership_period' in person.lp_attributes
False
```

```
>>> print(team.resource_type_link)
http://.../1.0/#team
>>> 'default_membership_period' in team.lp_attributes
True
```

The truth is that it doesn't know, not before making that HTTP request. Until an HTTP request is made, launchpadlib assumes everything in launchpad.people[] is a team (since a team has strictly more capabilities than a person).

```
>>> person2 = launchpad.people('salgado')
>>> 'default_membership_period' in person2.lp_attributes
True
```

But accessing any attribute of an object–even trying to see what kind of object 'salgado' is–will trigger the HTTP request that will determine that 'salgado' is actually a person.

```
>>> print(person2.resource_type_link)
send: b'GET /.../~salgado ...'
...
http://.../1.0/#person
```

```
>>> 'default_membership_period' in person2.lp_attributes
False
```

Accessing an attribute of an object that might be a team will trigger the HTTP request, and then cause an error if the object turns out not to be a team.

```
>>> person3 = launchpad.people('salgado')
>>> person3.default_membership_period
Traceback (most recent call last):
AttributeError: ...api.launchpad.../~salgado object has no attribute 'default_membership_
↪period'
```

Cleanup.

```
>>> httplib2.debuglevel = 0
```

CHAPTER

FOUR

PEOPLE AND TEAMS

The Launchpad web service, like Launchpad itself, exposes a unified interface to people and teams. In other words, people and teams occupy the same namespace. You treat people and teams as the same type of object, and need to inspect the object to know whether you're dealing with a person or a team.

4.1 People

You can access Launchpad people through the web service interface. The list of people is available from the service root.

```
>>> from launchpadlib.testing.helpers import salgado_with_full_permissions
>>> launchpad = salgado_with_full_permissions.login()
>>> people = launchpad.people
```

The list of people is not fetched until you actually use data.

```
>>> print(people._wadl_resource.representation)
None
```

```
>>> len(people)
4
```

```
>>> print(people._wadl_resource.representation)
{...}
```

The 'me' attribute is also available from the service root. It's a quick way to get a reference to your own user account.

```
>>> me = launchpad.me
>>> print(me.name)
salgado
```

You can find a person by name.

```
>>> salgado = launchpad.people['salgado']
>>> print(salgado.name)
salgado
>>> print(salgado.display_name)
Guilherme Salgado
>>> salgado.is_team
False
```

But if no person by that name is registered, you get the expected KeyError.

```
>>> launchpad.people['not-a-registered-person']
Traceback (most recent call last):
...
KeyError: 'not-a-registered-person'
```

It's not possible to slice a single person from the top-level collection of people. launchpadlib will try to use the value you pass in as a person's name, which will almost always fail.

```
>>> launchpad.people[1]
Traceback (most recent call last):
...
KeyError: 1
```

You can find a person by email.

```
>>> email = salgado.preferred_email_address.email
>>> salgado = launchpad.people.getByEmail(email=email)
>>> print(salgado.name)
salgado
```

Besides a name and a display name, a person has many other attributes that you can read.

> XXX 05-Jun-2008 BarryWarsaw Some of these attributes are links to further collections and are not yet tested. Tests will be added in future branches.

```
>>> salgado.karma
0
>>> print(salgado.homepage_content)
None
>>> #salgado.mugshot
>>> #salgado.languages
>>> salgado.hide_email_addresses
False
>>> salgado.date_created
datetime.datetime(2005, 6, 6, 8, 59, 51, 596025, ...)
>>> print(salgado.time_zone)
UTC
>>> salgado.is_valid
True
>>> #salgado.wiki_names
>>> #salgado.irc_nicknames
>>> #salgado.jabber_ids
>>> #salgado.team_memberships
>>> #salgado.open_membership_invitations
>>> #salgado.teams_participated_in
>>> #salgado.teams_indirectly_participated_in
>>> #salgado.confirmed_email_addresses
>>> #salgado.preferred_email_address
>>> print(salgado.mailing_list_auto_subscribe_policy)
Ask me when I join a team
>>> print(salgado.visibility)
Public
```

4.2 Teams

You also access teams using the same interface.

```
>>> team = launchpad.people['ubuntu-team']
>>> print(team.name)
ubuntu-team
>>> print(team.display_name)
Ubuntu Team
>>> team.is_team
True
```

Regular people have team attributes, but they're not used.

```
>>> print(salgado.team_owner)
None
```

You can find out how a person has membership in a team.

> # XXX: salgado, 2008-08-01: Commented because method has been Unexported; # it should be re-enabled after the operation is exported again. # >>> path = salgado.findPathToTeam(# ... team=launchpad.people['mailing-list-experts']) # >>> [team.name for team in path] # [u'admins', u'mailing-list-experts']

You can create a new team through the web interface. The simplest case of this requires only the new team's name, owner and display name.

```
>>> launchpad.people['bassists']
Traceback (most recent call last):
...
KeyError: 'bassists'
```

```
>>> bassists = launchpad.people.newTeam(
...     name='bassists', display_name='Awesome Rock Bass Players')
>>> print(bassists.name)
bassists
>>> print(bassists.display_name)
Awesome Rock Bass Players
>>> bassists.is_team
True
```

And of course, that team is now accessible directly.

```
>>> bassists = launchpad.people['bassists']
>>> print(bassists.name)
bassists
>>> print(bassists.display_name)
Awesome Rock Bass Players
```

You cannot create the same team twice.

```
>>> launchpad.people.newTeam(name='bassists', display_name='Bass Gods')
Traceback (most recent call last):
...
```

(continues on next page)

(continued from previous page)

```
lazr.restfulclient.errors.BadRequest: HTTP Error 400: Bad Request
...
```

Actually, the exception contains other useful information.

```
>>> from launchpadlib.errors import HTTPError
>>> try:
...     launchpad.people.newTeam(
...         name='bassists', display_name='Bass Gods')
... except HTTPError as e:
...     error = e
>>> error.response['status']
'400'
>>> print(error.content.decode())
name: bassists is already in use by another person or team.
```

Besides a name and a display name, a team has many other attributes that you can read.

```
>>> bassists.karma
0
>>> print(bassists.homepage_content)
None
>>> bassists.hide_email_addresses
False
>>> bassists.date_created
datetime.datetime(...)
>>> print(bassists.time_zone)
UTC
>>> bassists.is_valid
True
>>> #bassists.team_memberships
>>> #bassists.open_membership_invitations
>>> #bassists.teams_participated_in
>>> #bassists.teams_indirectly_participated_in
>>> #bassists.confirmed_email_addresses
>>> #bassists.team_owner
>>> #bassists.preferred_email_address
>>> #bassists.members
>>> #bassists.admins
>>> #bassists.participants
>>> #bassists.deactivated_members
>>> #bassists.expired_members
>>> #bassists.invited_members
>>> #bassists.member_memberships
>>> #bassists.proposed_members
>>> print(bassists.visibility)
Public
>>> print(bassists.team_description)
None
>>> print(bassists.subscription_policy)
Moderated Team
>>> print(bassists.renewal_policy)
```

(continues on next page)

(continued from previous page)

```
invite them to apply for renewal
>>> print(bassists.default_membership_period)
None
>>> print(bassists.default_renewal_period)
None
```

CHAPTER

FIVE

HOSTED FILES

The Launchpad web service sets restrictions on what kinds of documents can be written to a particular file. This test shows what happens when you try to upload a non-image for a field that expects an image.

```
>>> from launchpadlib.testing.helpers import salgado_with_full_permissions
>>> launchpad = salgado_with_full_permissions.login()
>>> from launchpadlib.errors import HTTPError
```

```
>>> mugshot = launchpad.me.mugshot
>>> file_handle = mugshot.open("w", "image/png", "nonimage.txt")
>>> file_handle.content_type
'image/png'
>>> file_handle.filename
'nonimage.txt'
>>> file_handle.write(b"Not an image.")
>>> try:
...     file_handle.close()
... except HTTPError as e:
...     print(e.content.decode())

The file uploaded was not recognized as an image; please
check it and retry.
```

Of course, uploading an image works fine.

```
>>> import os
>>> def load_image(filename):
...     image_file = os.path.join(
...         os.path.dirname(__file__), 'files', filename)
...     with open(image_file, "rb") as f:
...         return f.read()
>>> image = load_image("mugshot.png")
>>> len(image)
2260
```

```
>>> file_handle = mugshot.open("w", "image/png", "a-mugshot.png")
>>> file_handle.write(image)
>>> file_handle.close()
```

== Error handling ==

The server may set restrictions on what kinds of documents can be written to a particular file.

```
>>> file_handle = mugshot.open("w", "image/png", "nonimage.txt")
>>> file_handle.content_type
'image/png'
>>> file_handle.filename
'nonimage.txt'
>>> file_handle.write(b"Not an image.")
>>> file_handle.close()
Traceback (most recent call last):
...
lazr.restfulclient.errors.BadRequest: HTTP Error 400: Bad Request
...
```

== Caching ==

Hosted file resources implement the normal server-side caching mechanism.

```
>>> file_handle = mugshot.open("w", "image/png", "image.png")
>>> file_handle.write(image)
>>> file_handle.close()
```

```
>>> import httplib2
>>> httplib2.debuglevel = 1
>>> launchpad = salgado_with_full_permissions.login()
send: ...
>>> mugshot = launchpad.me.mugshot
send: ...
```

The first request for a file retrieves the file from the server.

```
>>> len(mugshot.open().read())
send: ...
reply: 'HTTP/1.1 303 See Other...
reply: 'HTTP/1.1 200 OK...
2260
```

The second request retrieves the file from the cache. After receiving the 303 request with its Location header, no further HTTP requests are issued because the Librarian's Cache-Control: headers tell us we already have a fresh copy.

```
>>> len(mugshot.open().read())
send: ...
reply: 'HTTP/1.1 303 See Other...
header: Location...
2260
```

Finally, some cleanup code that deletes the mugshot.

```
>>> mugshot.delete()
send: b'DELETE...
reply: 'HTTP/1.1 200...
```

```
>>> httplib2.debuglevel = 0
```

CHAPTER

SIX

COMMAND-LINE SCRIPTS

Launchpad includes one command-line script to make Launchpad integration easier for third-party libraries that aren't written in Python.

This file tests the workflow underlying the command-line script as best it can.

6.1 RequestTokenApp

This class is called by the command-line script launchpad-request-token. It creates a request token on a given Launchpad installation, and returns a JSON description of the request token and the available access levels.

```
>>> import json
>>> from launchpadlib.apps import RequestTokenApp
```

```
>>> web_root = "http://launchpad.test:8085/"
>>> consumer_name = "consumer"
>>> token_app = RequestTokenApp(web_root, consumer_name, "context")
>>> token_json = json.loads(token_app.run())
```

```
>>> for param in sorted(token_json.keys()):
...     print(param)
access_levels
lp.context
oauth_token
oauth_token_consumer
oauth_token_secret
```

```
>>> print(token_json['lp.context'])
context
```

```
>>> print(token_json['oauth_token_consumer'])
consumer
```

CHAPTER

SEVEN

CONTRIBUTING

To run this project's tests, use tox.

To update the project's documentation you need to trigger a manual build on the project's dashboard on https://readthedocs.org.

7.1 Getting help

If you find bugs in this package, you can report them here:

> https://launchpad.net/launchpadlib

If you want to discuss this package, join the team and mailing list here:

> https://launchpad.net/~lazr-developers

or send a message to:

> lazr-developers@lists.launchpad.net

CHAPTER

EIGHT

NEWS FOR LAUNCHPADLIB

8.1 1.11.0 (2023-01-09)

- Move the `keyring` dependency to a new `keyring` extra.
- Support setting fake methods that return None on instances of `launchpadlib.testing.launchpad.FakeLaunchpad`.
- Allow setting `FakeLaunchpad` sample data with attributes that are links to other entries or collections.
- Fix handling of methods with no response representation in `FakeLaunchpad`.

8.2 1.10.18 (2022-10-28)

- Declare support for Python 3.11.

8.3 1.10.17 (2022-10-15)

- Generate coverage report.
- Use `pytest` as test runner.
- Fix doctests for Python 3.

8.4 1.10.16 (2022-01-21)

- Add `pre-commit` configuration.
- Remove some obsolete scripts from `contrib/`.
- Apply `black` code formatter.
- Publish documentation on Read the Docs.
- Remove remnants of `simplejson` in favour of `json`.
- Apply inclusive naming via the `woke` `pre-commit` hook.
- Optionally get credentials file from `LP_CREDENTIALS_FILE` environment variable. [bug=737473]

8.5 1.10.15.1 (2021-10-27)

- Re-release without stray files in sdist.

8.6 1.10.15 (2021-10-27)

- Move dependencies of launchpadlib.testing to a new `testing` extra. [bug=1019700]
- Stop excluding MANIFEST.in from the sdist.
- Declare support for Python 3.9 and 3.10.
- Move code hosting to git (https://code.launchpad.net/launchpadlib).

8.7 1.10.14 (2021-09-13)

- Adjust versioning strategy to avoid importing pkg_resources, which is slow in large environments.

8.8 1.10.13 (2020-04-19)

- Fix test runs under sudo.

8.9 1.10.12 (2020-04-17)

- Postpone keyring.errors import in the same way that we postpone importing keyring itself.

8.10 1.10.11 (2020-04-14)

- Don't store credentials or open a browser window when running under sudo. [bug=1825014,1862948]
- Fall back to in-memory credentials store if no keyring backend is available. [bug=1864204]

8.11 1.10.10 (2020-02-04)

- Fix AccessToken.from_string crash on Python 3.8. [bug=1861873]

8.12 1.10.9 (2019-11-28)

- Explicitly install version.txt; launchpadlib requires it.

8.13 1.10.8 (2019-11-26)

- Squash a deprecation warning on Python >= 3.7 in launchpadlib.testing.launchpad.
- Switch from buildout to tox.
- Weaken hosted-files test slightly to avoid problems with zope.publisher >= 4.2.2.

8.14 1.10.7 (2019-05-22)

- Change 'dev' URLs from launchpad.dev to launchpad.test.

8.15 1.10.6 (2018-03-08)

- Fix saving of credentials in python3 with gnome-keyring. [bug=1685962]

8.16 1.10.5 (2017-02-02)

- Fix AccessToken.from_string crash on Python 3. [bug=1471927]
- Fix fallback if authorizing a token with a browser raises webbrowser.Error.
- Stop introduction.txt doctest from writing to $HOME.

8.17 1.10.4 (2016-07-12)

- Fix _bad_oauth_token crash on Python 3. [bug=1471894]
- Time out make_end_user_authorize_token after 15 minutes.
- Ignore PendingDeprecationWarning from lazr.restfulclient. [bug=1473577]
- Ask forgiveness rather than permission when creating cache directories.
- Fix browser token authorization on OS X. [bug=1516080]

8.18 1.10.3 (2014-12-05)

- Port to Python3.
- Detect proxies from the environment by default.

8.19 1.10.2 (2012-07-05)

- Typo in the doctest fix, discovered when trying to integrate with launchpad itself. [bug=1020667]

8.20 1.10.1 (2012-07-04)

- Fix a doctest in introduction.txt so that the test suite runs with python-2.7 (note the doctests only run when running integrated with launchpad's test suite itself). [bug=1020667]

8.21 1.10.0 (2012-06-19)

- Add environment variable, LP_DISABLE_SSL_CERTIFICATE_VALIDATION, to disable SSL certificate checks. Most useful when testing against development servers.

8.22 1.9.12 (2011-12-05)

- Move keyring base64 encoding to KeyringCredential and be more defensive about decoding. [bug=900307]

8.23 1.9.11 (2011-11-21)

- 1.9.10 was a bad release due to incomplete NEWS entries.
- Add fake Launchpad web service for unit test.
- Improve HACKING documentation.
- Improve launchpadlib directory discovery on Windows.
- Added script to delete spurious bugtasks or split a bugtask from a bug.
- Properly handle Unicode passwords if returned by the keyring.
- Base 64 encode serialized credentials before putting in keyring/wallet.

8.24 1.9.10 (2011-11-21)

- Base 64 encode serialized credentials before putting in keyring/wallet.

8.25 1.9.9 (2011-07-27)

- Fix a failing test for lazr.restfulclient 0.12.0.

8.26 1.9.8 (2011-02-28)

- Detect the error Launchpad sends when it doesn't recognize an access token, and get a new token.

8.27 1.9.7 (2011-02-15)

- Slightly tweaked the behavior of EDGE_SERVICE_ROOT, and improved tests.

8.28 1.9.6 (2011-02-14)

- Added EDGE_SERVICE_ROOT and the 'edge' alias back, though they both operate on production behind the scenes. Using the 'edge' alias will cause a deprecation warning.

8.29 1.9.5 (2011-02-08)

- Fixed a bug that prevented the deprecated get_token_and_login code from working, and that required that users of get_token_and_login get a new token on every usage.

8.30 1.9.4 (2011-01-18)

- Removed references to the 'edge' service root, which is being phased out.
- Fixed a minor bug in the upload_release_tarball contrib script which was causing tarballs to be uploaded with the wrong media type.
- The XSLT stylesheet for converting the Launchpad WADL into HTML documentation has been moved back into Launchpad.

8.31 1.9.3 (2011-01-10)

- The keyring package import is now delayed until the keyring needs to be accessed. This reduces launchapdlib users' exposure to unintended side effects of importing keyring (KWallet authorization dialogs and the registration of a SIGCHLD handler).

8.32 1.9.2 (2011-01-07)

- Added a missing import.

8.33 1.9.1 (2011-01-06)

- Corrected a test failure.

8.34 1.9.0 (2011-01-05)

- When an authorization token expires or becomes invalid, attempt to acquire a new one, even in the middle of a session, rather than crashing.
- The HTML generated by wadl-to-refhtml.xsl now validates.
- Most of the helper login methods have been deprecated. There are now only two helper methods:
  - Launchpad.login_anonymously, for anonymous credential-free access.
  - Launchpad.login_with, for programs that need a credential.

8.35 1.8.0 (2010-11-15)

- Store authorization tokens in the Gnome keyring or KDE wallet, when available. The credentials_file parameter of Launchpad.login_with() is now ignored.
- By default, Launchpad.login_with() now asks Launchpad for desktop-wide integration. This removes the need for each individual application to get its own OAuth token.

8.36 1.7.0 (2010-09-23)

- Removed "fake Launchpad browser" code that didn't work and was misleading developers.
- Added support for http://qastaging.launchpad.net by adding astaging to the uris.

8.37 1.6.5 (2010-08-23)

- Make launchpadlib compatible with the latest lazr.restfulclient.

8.38 1.6.4 (2010-08-18)

- Test fixes.

8.39 1.6.3 (2010-08-12)

- Instead of making the end-user hit Enter after authorizing an application to access their Launchpad account, launchpadlib will automatically poll Launchpad until the user makes a decision.
- launchpadlib now raises a more helpful exception when the end-user explicitly denies access to a launchpadlib application.
- Improved the XSLT stylesheet to reflect Launchpad's more complex top-level structure. [bug=286941]
- Test fixes. [bug=488448,616055]

8.40 1.6.2 (2010-06-21)

- Extended the optimization from version 1.6.1 to apply to Launchpad's top-level collection of people.

8.41 1.6.1 (2010-06-16)

- Added an optimization that lets launchpadlib avoid making an HTTP request in some situations.

8.42 1.6.0 (2010-04-07)

- Fixed a test to work against the latest version of Launchpad.

8.43 1.5.8 (2010-03-25)

- Use version 1.0 of the Launchpad web service by default.

8.44 1.5.7 (2010-03-16)

- Send a Referer header whenever making requests to the Launchpad website (as opposed to the web service) to avoid falling afoul of new cross-site-request-forgery countermeasures.

8.45 1.5.6 (2010-03-04)

- Fixed a minor bug when using login_with() to access a version of the Launchpad web service other than the default.
- Added a check to catch old client code that would cause newer versions of launchpadlib to make nonsensical requests to https://api.launchpad.dev/beta/beta/, and raise a helpful exception telling the developer how to fix it.

8.46 1.5.5

- Added the ability to access different versions of the Launchpad web service.

8.47 1.5.4 (2009-12-17)

- Made it easy to get anonymous access to a Launchpad instance.
- Made it easy to plug in different clients that take the user's Launchpad login and password for purposes of authorizing a request token. The most secure technique is still the default: to open the user's web browser to the appropriate Launchpad page.
- Introduced a command-line script bin/launchpad-credentials-console, which takes the user's Launchpad login and password, and authorizes a request token on their behalf.
- Introduced a command-line script bin/launchpad-request-token, which creates a request token on any Launchpad installation and dumps the JSON description of that token to standard output.
- Shorthand service names like 'edge' should now be respected everywhere in launchpadlib.

8.48 1.5.3 (2009-10-22)

- Moved some more code from launchpadlib into the more generic lazr.restfulclient.

8.49 1.5.2 (2009-10-01)

- Added a number of new sample scripts from elsewhere.
- Added a reference to the production Launchpad instance.
- Made it easier to specify a Launchpad instance to run against.

8.50 1.5.1 (2009-07-16)

- Added a sample script for uploading a release tarball to Launchpad.

8.51 1.5.0 (2009-07-09)

- Most of launchpadlib's code has been moved to the generic lazr.restfulclient library. launchpadlib now contains only code specific to Launchpad. There should be no changes in functionality.
- Moved bootstrap.py into the top-level directory. Having it in a subdirectory with a top-level symlink was breaking installation on Windows.
- The notice to the end-user (that we're opening their web browser) is now better formatted.

8.52 1.0.1 (2009-05-30)

- Correct tests for new launchpad cache behavior in librarian
- Remove build dependency on setuptools_bzr because it was causing bzr to be downloaded during installation of the package, which was unnecessary and annoying.

8.53 1.0 (2009-03-24)

- Initial release on PyPI