oauth2client Documentation

Release 1.5.2

Google, Inc.

February 20, 2016

Contents

making OAuth2 just a little less painful

`oauth2client` makes it easy to interact with OAuth2-protected resources, especially those related to Google APIs. You can also start with general information about using OAuth2 with Google APIs.

CHAPTER 1

Getting started

We recommend installing via `pip`:

```
$ pip install --upgrade oauth2client
```

You can also install from source:

```
$ git clone https://github.com/google/oauth2client
$ cd oauth2client
$ python setup.py install
```

CHAPTER 2

Using pypy

- In order to use crypto libraries (e.g. for service accounts) you will need to install one of `pycrypto` or `pyOpenSSL`.
- Using `pycrypto` with `pypy` will be in general problematic. If `libgmp` is installed on your machine, the `pycrypto` install will attempt to build `_fastmath.c`. However, this file uses CPython implementation details and hence can't be built in `pypy` (as of `pypy` 2.6 and `pycrypto` 2.6.1). In order to install

```
with_gmp=no pip install --upgrade pycrypto
```

  See discussions on the pypy issue tracker and the pycrypto issue tracker.

- Using `pyOpenSSL` with versions of `pypy` before 2.6 may be in general problematic since `pyOpenSSL` depends on the `cryptography` library. For versions of `cryptography` before 1.0, importing `pyOpenSSL` with it caused massive startup costs. In order to address this slow startup, `cryptography` 1.0 made some changes in how it used `cffi` when means it can't be used on versions of `pypy` before 2.6.

  The default version of `pypy` you get when installed

```
apt-get install pypy pypy-dev
```

  on Ubuntu 14.04 is 2.2.1. In order to upgrade, you'll need to use the pypy/ppa PPA:

```
apt-get purge pypy pypy-dev
add-apt-repository ppa:pypy/ppa
apt-get update
apt-get install pypy pypy-dev
```

2.1 Downloads

- Most recent release tarball
- Most recent release zipfile
- Complete release list

CHAPTER 3

Library Documentation

- Complete library index: genindex
- Index of all modules: modindex
- Search all documentation: search

CHAPTER 4

Contributing

Please see the contributing page for more information. In particular, we love pull requests – but please make sure to sign the contributor license agreement.

4.1 oauth2client package

4.1.1 Submodules

oauth2client.appengine module

oauth2client.client module

An OAuth 2.0 client.

Tools for interacting with OAuth 2.0 protected resources.

class oauth2client.client.**AccessTokenCredentials**(*access_token*, *user_agent*, *revoke_uri=None*)

Bases: *oauth2client.client.OAuth2Credentials*

Credentials object for OAuth 2.0.

Credentials can be applied to an httplib2.Http object using the authorize() method, which then signs each request from that object with the OAuth 2.0 access token. This set of credentials is for the use case where you have acquired an OAuth 2.0 access_token from another place such as a JavaScript client or another web application, and wish to use it from Python. Because only the access_token is present it can not be refreshed and will in time expire.

AccessTokenCredentials objects may be safely pickled and unpickled.

Usage:

```
credentials = AccessTokenCredentials('<an access token>',
    'my-user-agent/1.0')
http = httplib2.Http()
http = credentials.authorize(http)
```

Raises AccessTokenCredentialsExpired – raised when the access_token expires or is revoked.

classmethod from_json(*s*)

exception oauth2client.client.**AccessTokenCredentialsError**
Bases: *oauth2client.client.Error*

Having only the access_token means no refresh is possible.

class oauth2client.client.**AccessTokenInfo**(*access_token*, *expires_in*)
Bases: tuple

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

__getstate__()
Exclude the OrderedDict from pickling

__repr__()
Return a nicely formatted representation string

access_token
Alias for field number 0

expires_in
Alias for field number 1

exception oauth2client.client.**AccessTokenRefreshError**
Bases: *oauth2client.client.Error*

Error trying to refresh an expired access token.

exception oauth2client.client.**ApplicationDefaultCredentialsError**
Bases: *oauth2client.client.Error*

Error retrieving the Application Default Credentials.

class oauth2client.client.**AssertionCredentials**(**args*, ***kwargs*)
Bases: *oauth2client.client.GoogleCredentials*

Abstract Credentials object used for OAuth 2.0 assertion grants.

This credential does not require a flow to instantiate because it represents a two legged flow, and therefore has all of the required information to generate and refresh its own access tokens. It must be subclassed to generate the appropriate assertion string.

AssertionCredentials objects may be safely pickled and unpickled.

class oauth2client.client.**Credentials**
Bases: object

Base class for all Credentials objects.

Subclasses must define an authorize() method that applies the credentials to an HTTP transport.

Subclasses must also specify a classmethod named 'from_json' that takes a JSON string as input and returns an instantiated Credentials object.

NON_SERIALIZED_MEMBERS = ['store']

apply(*headers*)
Add the authorization to the headers.

Parameters **headers** – dict, the headers to add the Authorization header to.

authorize(*http*)
Take an httplib2.Http instance (or equivalent) and authorizes it.

Authorizes it for the set of credentials, usually by replacing http.request() with a method that adds in the appropriate headers and then delegates to the original Http.request() method.

Parameters **http** – httplib2.Http, an http object to be used to make the refresh request.

classmethod from_json(*unused_data*)
Instantiate a Credentials object from a JSON description of it.

The JSON should have been produced by calling .to_json() on the object.

Parameters **unused_data** – dict, A deserialized JSON object.

Returns An instance of a Credentials subclass.

classmethod new_from_json(*s*)
Utility class method to instantiate a Credentials subclass from JSON.

Expects the JSON string to have been produced by to_json().

Parameters **s** – string or bytes, JSON from to_json().

Returns An instance of the subclass of Credentials that was serialized with to_json().

refresh(*http*)
Forces a refresh of the access_token.

Parameters **http** – httplib2.Http, an http object to be used to make the refresh request.

revoke(*http*)
Revokes a refresh_token and makes the credentials void.

Parameters **http** – httplib2.Http, an http object to be used to make the revoke request.

to_json()
Creating a JSON representation of an instance of Credentials.

Returns string, a JSON representation of this instance, suitable to pass to from_json().

exception oauth2client.client.**CryptoUnavailableError**
Bases: *oauth2client.client.Error*, exceptions.NotImplementedError

Raised when a crypto library is required, but none is available.

class oauth2client.client.**DeviceFlowInfo**
Bases: *oauth2client.client.DeviceFlowInfo*

Intermediate information the OAuth2 for devices flow.

classmethod FromResponse(*response*)
Create a DeviceFlowInfo from a server response.

The response should be a dict containing entries as described here:

http://tools.ietf.org/html/draft-ietf-oauth-v2-05#section-3.7.1

exception oauth2client.client.**Error**
Bases: exceptions.Exception

Base error for this module.

class oauth2client.client.**Flow**
Bases: object

Base class for all Flow objects.

exception oauth2client.client.**FlowExchangeError**
Bases: *oauth2client.client.Error*

Error trying to exchange an authorization grant for an access token.

class oauth2client.client.**GoogleCredentials**(*access_token*, *client_id*, *client_secret*, *refresh_token*, *token_expiry*, *token_uri*, *user_agent*, *revoke_uri='https://accounts.google.com/o/oauth2/revoke'*)

Bases: *oauth2client.client.OAuth2Credentials*

Application Default Credentials for use in calling Google APIs.

The Application Default Credentials are being constructed as a function of the environment where the code is being run. More details can be found on this page: https://developers.google.com/accounts/docs/application-default-credentials

Here is an example of how to use the Application Default Credentials for a service that requires authentication:

```
from googleapiclient.discovery import build
from oauth2client.client import GoogleCredentials

credentials = GoogleCredentials.get_application_default()
service = build('compute', 'v1', credentials=credentials)

PROJECT = 'bamboo-machine-422'
ZONE = 'us-central1-a'
request = service.instances().list(project=PROJECT, zone=ZONE)
response = request.execute()

print(response)
```

create_scoped(*scopes*)

Create a Credentials object for the given scopes.

The Credentials type is preserved.

create_scoped_required()

Whether this Credentials object is scopeless.

create_scoped(scopes) method needs to be called in order to create a Credentials object for API calls.

static from_stream(*credential_filename*)

Create a Credentials object by reading information from a file.

It returns an object of type GoogleCredentials.

Parameters **credential_filename** – the path to the file from where the credentials are to be read

Raises *ApplicationDefaultCredentialsError* – raised when the credentials fail to be retrieved.

static get_application_default()

Get the Application Default Credentials for the current environment.

Raises *ApplicationDefaultCredentialsError* – raised when the credentials fail to be retrieved.

serialization_data

Get the fields and values identifying the current credentials.

exception oauth2client.client.**HttpAccessTokenRefreshError**(**args*, ***kwargs*)

Bases: *oauth2client.client.AccessTokenRefreshError*

Error (with HTTP status) trying to refresh an expired access token.

class oauth2client.client.**MemoryCache**

Bases: object

httplib2 Cache implementation which only caches locally.

delete(*key*)

get(*key*)

set(*key*, *value*)

exception oauth2client.client.**NonAsciiHeaderError**
Bases: *oauth2client.client.Error*

Header names and values must be ASCII strings.

class oauth2client.client.**OAuth2Credentials**(**args*, ***kwargs*)
Bases: *oauth2client.client.Credentials*

Credentials object for OAuth 2.0.

Credentials can be applied to an httplib2.Http object using the authorize() method, which then adds the OAuth 2.0 access token to each request.

OAuth2Credentials objects may be safely pickled and unpickled.

__getstate__()
Trim the state down to something that can be pickled.

__setstate__(*state*)
Reconstitute the state of the object from being pickled.

access_token_expired
True if the credential is expired or invalid.

If the token_expiry isn't set, we assume the token doesn't expire.

apply(*headers*)
Add the authorization to the headers.

Parameters **headers** – dict, the headers to add the Authorization header to.

authorize(*http*)
Authorize an httplib2.Http instance with these credentials.

The modified http.request method will add authentication headers to each request and will refresh access_tokens when a 401 is received on a request. In addition the http.request method has a credentials property, http.request.credentials, which is the Credentials object that authorized it.

Parameters **http** – An instance of httplib2.Http or something that acts like it.

Returns A modified instance of http that was passed in.

Example:

```
h = httplib2.Http()
h = credentials.authorize(h)
```

You can't create a new OAuth subclass of httplib2.Authentication because it never gets passed the absolute URI, which is needed for signing. So instead we have to overload 'request' with a closure that adds in the Authorization header and then calls the original version of 'request()'.

classmethod **from_json**(*s*)
Instantiate a Credentials object from a JSON description of it.

The JSON should have been produced by calling .to_json() on the object.

Parameters **data** – dict, A deserialized JSON object.

Returns An instance of a Credentials subclass.

get_access_token(*http=None*)
Return the access token and its expiration information.

If the token does not exist, get one. If the token expired, refresh it.

has_scopes(*scopes*)
Verify that the credentials are authorized for the given scopes.

Returns True if the credentials authorized scopes contain all of the scopes given.

Parameters **scopes** – list or string, the scopes to check.

Notes

There are cases where the credentials are unaware of which scopes are authorized. Notably, credentials obtained and stored before this code was added will not have scopes, AccessTokenCredentials do not have scopes. In both cases, you can use refresh_scopes() to obtain the canonical set of scopes.

refresh(*http*)
Forces a refresh of the access_token.

Parameters **http** – httplib2.Http, an http object to be used to make the refresh request.

retrieve_scopes(*http*)
Retrieves the canonical list of scopes for this access token.

Gets the scopes from the OAuth2 provider.

Parameters **http** – httplib2.Http, an http object to be used to make the refresh request.

Returns A set of strings containing the canonical list of scopes.

revoke(*http*)
Revokes a refresh_token and makes the credentials void.

Parameters **http** – httplib2.Http, an http object to be used to make the revoke request.

set_store(*store*)
Set the Storage for the credential.

Parameters **store** – Storage, an implementation of Storage object. This is needed to store the latest access_token if it has expired and been refreshed. This implementation uses locking to check for updates before updating the access_token.

to_json()

exception oauth2client.client.**OAuth2DeviceCodeError**
Bases: *oauth2client.client.Error*

Error trying to retrieve a device code.

class oauth2client.client.**OAuth2WebServerFlow**(**args*, ***kwargs*)
Bases: *oauth2client.client.Flow*

Does the Web Server Flow for OAuth 2.0.

OAuth2WebServerFlow objects may be safely pickled and unpickled.

step1_get_authorize_url(**args*, ***kwargs*)
Returns a URI to redirect to the provider.

Parameters

- **redirect_uri** – string, Either the string 'urn:ietf:wg:oauth:2.0:oob' for a non-web-based application, or a URI that handles the callback from the authorization server. This parameter is deprecated, please move to passing the redirect_uri in via the constructor.
- **state** – string, Opaque state string which is passed through the OAuth2 flow and returned to the client as a query parameter in the callback.

Returns A URI as a string to redirect the user to begin the authorization flow.

step1_get_device_and_user_codes(**args*, ***kwargs*)
Returns a user code and the verification URL where to enter it

Returns A user code as a string for the user to authorize the application An URL as a string where the user has to enter the code

step2_exchange(**args*, ***kwargs*)
Exchanges a code for OAuth2Credentials.

Parameters

- **code** – string, a dict-like object, or None. For a non-device flow, this is either the response code as a string, or a dictionary of query parameters to the redirect_uri. For a device flow, this should be None.
- **http** – httplib2.Http, optional http instance to use when fetching credentials.
- **device_flow_info** – DeviceFlowInfo, return value from step1 in the case of a device flow.

Returns An OAuth2Credentials object that can be used to authorize requests.

Raises

- *FlowExchangeError* – if a problem occurred exchanging the code for a refresh_token.
- ValueError – if code and device_flow_info are both provided or both missing.

class oauth2client.client.**SETTINGS**
Bases: object

Settings namespace for globally defined values.

env_name = None

class oauth2client.client.**SignedJwtAssertionCredentials**(**args*, ***kwargs*)
Bases: *oauth2client.client.AssertionCredentials*

Credentials object used for OAuth 2.0 Signed JWT assertion grants.

This credential does not require a flow to instantiate because it represents a two legged flow, and therefore has all of the required information to generate and refresh its own access tokens.

SignedJwtAssertionCredentials requires either PyOpenSSL, or PyCrypto 2.6 or later. For App Engine you may also consider using AppAssertionCredentials.

MAX_TOKEN_LIFETIME_SECS = 3600

classmethod from_json(*s*)

class oauth2client.client.**Storage**
Bases: object

Base class for all Storage objects.

Store and retrieve a single credential. This class supports locking such that multiple processes and threads can operate on a single store.

acquire_lock()
Acquires any lock necessary to access this Storage.

This lock is not reentrant.

delete()
Delete credential.

Frees any resources associated with storing the credential. The Storage lock must *not* be held when this is called.

Returns None

get()
Retrieve credential.

The Storage lock must *not* be held when this is called.

Returns oauth2client.client.Credentials

locked_delete()
Delete a credential.

The Storage lock must be held when this is called.

locked_get()
Retrieve credential.

The Storage lock must be held when this is called.

Returns oauth2client.client.Credentials

locked_put(*credentials*)
Write a credential.

The Storage lock must be held when this is called.

Parameters **credentials** – Credentials, the credentials to store.

put(*credentials*)
Write a credential.

The Storage lock must be held when this is called.

Parameters **credentials** – Credentials, the credentials to store.

release_lock()
Release the Storage lock.

Trying to release a lock that isn't held will result in a RuntimeError.

exception oauth2client.client.**TokenRevokeError**
Bases: *oauth2client.client.Error*

Error trying to revoke a token.

exception oauth2client.client.**UnknownClientSecretsFlowError**
Bases: *oauth2client.client.Error*

The client secrets file called for an unknown type of OAuth 2.0 flow.

exception oauth2client.client.**VerifyJwtTokenError**
Bases: *oauth2client.client.Error*

Could not retrieve certificates for validation.

oauth2client.client.**clean_headers**(*headers*)
Forces header keys and values to be strings, i.e not unicode.

The httplib module just concats the header keys and values in a way that may make the message header a unicode string, which, if it then tries to contatenate to a binary request body may result in a unicode decode error.

Parameters **headers** – dict, A dictionary of headers.

Returns The same dictionary but with all the keys converted to strings.

oauth2client.client.**credentials_from_clientsecrets_and_code**(**args*, ***kwargs*)
Returns OAuth2Credentials from a clientsecrets file and an auth code.

Will create the right kind of Flow based on the contents of the clientsecrets file or will raise InvalidClientSecretsError for unknown types of Flows.

Parameters

- **filename** – string, File name of clientsecrets.
- **scope** – string or iterable of strings, scope(s) to request.
- **code** – string, An authorization code, most likely passed down from the client
- **message** – string, A friendly string to display to the user if the clientsecrets file is missing or invalid. If message is provided then sys.exit will be called in the case of an error. If message in not provided then clientsecrets.InvalidClientSecretsError will be raised.
- **redirect_uri** – string, this is generally set to 'postmessage' to match the redirect_uri that the client specified
- **http** – httplib2.Http, optional http instance to use to do the fetch
- **cache** – An optional cache service client that implements get() and set() methods. See clientsecrets.loadfile() for details.
- **device_uri** – string, OAuth 2.0 device authorization endpoint

Returns An OAuth2Credentials object.

Raises

- *FlowExchangeError* – if the authorization code cannot be exchanged for an access token
- *UnknownClientSecretsFlowError* – if the file describes an unknown kind of Flow.
- clientsecrets.InvalidClientSecretsError – if the clientsecrets file is invalid.

oauth2client.client.**credentials_from_code**(**args*, ***kwargs*)
Exchanges an authorization code for an OAuth2Credentials object.

Parameters

- **client_id** – string, client identifier.
- **client_secret** – string, client secret.
- **scope** – string or iterable of strings, scope(s) to request.
- **code** – string, An authorization code, most likely passed down from the client
- **redirect_uri** – string, this is generally set to 'postmessage' to match the redirect_uri that the client specified
- **http** – httplib2.Http, optional http instance to use to do the fetch

- **token_uri** – string, URI for token endpoint. For convenience defaults to Google's endpoints but any OAuth 2.0 provider can be used.
- **auth_uri** – string, URI for authorization endpoint. For convenience defaults to Google's endpoints but any OAuth 2.0 provider can be used.
- **revoke_uri** – string, URI for revoke endpoint. For convenience defaults to Google's endpoints but any OAuth 2.0 provider can be used.
- **device_uri** – string, URI for device authorization endpoint. For convenience defaults to Google's endpoints but any OAuth 2.0 provider can be used.

Returns An OAuth2Credentials object.

Raises

- FlowExchangeError if the authorization code cannot be exchanged for an
- access token

oauth2client.client.**flow_from_clientsecrets**(**args*, ***kwargs*)

Create a Flow from a clientsecrets file.

Will create the right kind of Flow based on the contents of the clientsecrets file or will raise InvalidClientSecretsError for unknown types of Flows.

Parameters

- **filename** – string, File name of client secrets.
- **scope** – string or iterable of strings, scope(s) to request.
- **redirect_uri** – string, Either the string 'urn:ietf:wg:oauth:2.0:oob' for a non-web-based application, or a URI that handles the callback from the authorization server.
- **message** – string, A friendly string to display to the user if the clientsecrets file is missing or invalid. If message is provided then sys.exit will be called in the case of an error. If message in not provided then clientsecrets.InvalidClientSecretsError will be raised.
- **cache** – An optional cache service client that implements get() and set() methods. See clientsecrets.loadfile() for details.
- **login_hint** – string, Either an email address or domain. Passing this hint will either pre-fill the email box on the sign-in form or select the proper multi-login session, thereby simplifying the login flow.
- **device_uri** – string, URI for device authorization endpoint. For convenience defaults to Google's endpoints but any OAuth 2.0 provider can be used.

Returns A Flow object.

Raises

- *UnknownClientSecretsFlowError* – if the file describes an unknown kind of Flow.
- clientsecrets.InvalidClientSecretsError – if the clientsecrets file is invalid.

oauth2client.client.**save_to_well_known_file**(*credentials*, *well_known_file=None*)

Save the provided GoogleCredentials to the well known file.

Parameters

- **credentials** – the credentials to be saved to the well known file; it should be an instance of GoogleCredentials

- **well_known_file** – the name of the file where the credentials are to be saved; this parameter is supposed to be used for testing only

oauth2client.client.**verify_id_token**(**args*, ***kwargs*)

Verifies a signed JWT id_token.

This function requires PyOpenSSL and because of that it does not work on App Engine.

Parameters

- **id_token** – string, A Signed JWT.
- **audience** – string, The audience 'aud' that the token should be for.
- **http** – httplib2.Http, instance to use to make the HTTP request. Callers should supply an instance that has caching enabled.
- **cert_uri** – string, URI of the certificates in JSON format to verify the JWT against.

Returns The deserialized JSON in the JWT.

Raises

- *oauth2client.crypt.AppIdentityError* – if the JWT fails to verify.
- *CryptoUnavailableError* – if no crypto library is available.

oauth2client.clientsecrets module

Utilities for reading OAuth 2.0 client secret files.

A client_secrets.json file contains all the information needed to interact with an OAuth 2.0 protected service.

exception oauth2client.clientsecrets.**Error**

Bases: exceptions.Exception

Base error for this module.

exception oauth2client.clientsecrets.**InvalidClientSecretsError**

Bases: *oauth2client.clientsecrets.Error*

Format of ClientSecrets file is invalid.

oauth2client.clientsecrets.**load**(*fp*)

oauth2client.clientsecrets.**loadfile**(*filename*, *cache=None*)

Loading of client_secrets JSON file, optionally backed by a cache.

Typical cache storage would be App Engine memcache service, but you can pass in any other cache client that implements these methods:

- `get(key, namespace=ns)`
- `set(key, value, namespace=ns)`

Usage:

```
# without caching
client_type, client_info = loadfile('secrets.json')
# using App Engine memcache service
from google.appengine.api import memcache
client_type, client_info = loadfile('secrets.json', cache=memcache)
```

Parameters

- **filename** – string, Path to a client_secrets.json file on a filesystem.
- **cache** – An optional cache service client that implements get() and set()
- **If not specified, the file is always being loaded from** (*methods.*) – a filesystem.

Raises *InvalidClientSecretsError* – In case of a validation error or some I/O failure. Can happen only on cache miss.

Returns (client_type, client_info) tuple, as _loadfile() normally would. JSON contents is validated only during first load. Cache hits are not validated.

oauth2client.clientsecrets.**loads**(*s*)

oauth2client.crypt module

Crypto-related routines for oauth2client.

exception oauth2client.crypt.**AppIdentityError**
Bases: exceptions.Exception

Error to indicate crypto failure.

oauth2client.crypt.**make_signed_jwt**(*signer*, *payload*)
Make a signed JWT.

See http://self-issued.info/docs/draft-jones-json-web-token.html.

Parameters

- **signer** – crypt.Signer, Cryptographic signer.
- **payload** – dict, Dictionary of data to convert to JSON and then sign.

Returns string, The JWT for the payload.

oauth2client.crypt.**verify_signed_jwt_with_certs**(*jwt*, *certs*, *audience=None*)
Verify a JWT against public certs.

See http://self-issued.info/docs/draft-jones-json-web-token.html.

Parameters

- **jwt** – string, A JWT.
- **certs** – dict, Dictionary where values of public keys in PEM format.
- **audience** – string, The audience, 'aud', that this JWT should contain. If None then the JWT's 'aud' parameter is not verified.

Returns dict, The deserialized JSON payload in the JWT.

Raises *AppIdentityError* – if any checks are failed.

oauth2client.devshell module

OAuth 2.0 utitilies for Google Developer Shell environment.

exception oauth2client.devshell.**CommunicationError**
Bases: *oauth2client.devshell.Error*

Errors for communication with the Developer Shell server.

class oauth2client.devshell.**CredentialInfoResponse**(*json_string*)
Bases: object

Credential information response from Developer Shell server.

The credential information response from Developer Shell socket is a PBLite-formatted JSON array with fields encoded by their index in the array:

- Index 0 - user email
- Index 1 - default project ID. None if the project context is not known.
- Index 2 - OAuth2 access token. None if there is no valid auth context.

class oauth2client.devshell.**DevshellCredentials**(*user_agent=None*)
Bases: *oauth2client.client.GoogleCredentials*

Credentials object for Google Developer Shell environment.

This object will allow a Google Developer Shell session to identify its user to Google and other OAuth 2.0 servers that can verify assertions. It can be used for the purpose of accessing data stored under the user account.

This credential does not require a flow to instantiate because it represents a two legged flow, and therefore has all of the required information to generate and refresh its own access tokens.

classmethod from_json(*json_data*)

project_id

serialization_data

user_email

exception oauth2client.devshell.**Error**
Bases: exceptions.Exception

Errors for this module.

exception oauth2client.devshell.**NoDevshellServer**
Bases: *oauth2client.devshell.Error*

Error when no Developer Shell server can be contacted.

oauth2client.django_orm module

OAuth 2.0 utilities for Django.

Utilities for using OAuth 2.0 in conjunction with the Django datastore.

class oauth2client.django_orm.**CredentialsField**(**args*, ***kwargs*)
Bases: django.db.models.fields.Field

contribute_to_class(*cls*, *name*, ***kwargs*)

get_internal_type()

get_prep_value(*value*)

to_python(*value*)

value_to_string(*obj*)
Convert the field value from the provided model to a string.

Used during model serialization.

Parameters **obj** – db.Model, model object

Returns string, the serialized field value

class oauth2client.django_orm.**FlowField**(**args*, ***kwargs*)
Bases: django.db.models.fields.Field

contribute_to_class(*cls*, *name*, ***kwargs*)

get_internal_type()

get_prep_value(*value*)

to_python(*value*)

value_to_string(*obj*)
Convert the field value from the provided model to a string.

Used during model serialization.

Parameters **obj** – db.Model, model object

Returns string, the serialized field value

class oauth2client.django_orm.**Storage**(*model_class*, *key_name*, *key_value*, *property_name*)
Bases: *oauth2client.client.Storage*

Store and retrieve a single credential to and from the Django datastore.

This Storage helper presumes the Credentials have been stored as a CredenialsField on a db model class.

locked_delete()
Delete Credentials from the datastore.

locked_get()
Retrieve stored credential.

Returns oauth2client.Credentials

locked_put(*credentials*, *overwrite=False*)
Write a Credentials to the Django datastore.

Parameters

- **credentials** – Credentials, the credentials to store.
- **overwrite** – Boolean, indicates whether you would like these credentials to overwrite any existing stored credentials.

oauth2client.file module

Utilities for OAuth.

Utilities for making it easier to work with OAuth 2.0 credentials.

exception oauth2client.file.**CredentialsFileSymbolicLinkError**
Bases: exceptions.Exception

Credentials files must not be symbolic links.

class oauth2client.file.**Storage**(*filename*)
Bases: *oauth2client.client.Storage*

Store and retrieve a single credential to and from a file.

acquire_lock()
Acquires any lock necessary to access this Storage.

This lock is not reentrant.

locked_delete()
Delete Credentials file.

Parameters **credentials** – Credentials, the credentials to store.

locked_get()
Retrieve Credential from file.

Returns oauth2client.client.Credentials

Raises CredentialsFileSymbolicLinkError if the file is a symbolic link.

locked_put(*credentials*)
Write Credentials to file.

Parameters **credentials** – Credentials, the credentials to store.

Raises CredentialsFileSymbolicLinkError if the file is a symbolic link.

release_lock()
Release the Storage lock.

Trying to release a lock that isn't held will result in a RuntimeError.

oauth2client.flask_util module

Utilities for the Flask web framework

Provides a Flask extension that makes using OAuth2 web server flow easier. The extension includes views that handle the entire auth flow and a `@required` decorator to automatically ensure that user credentials are available.

Configuration

To configure, you'll need a set of OAuth2 web application credentials from the Google Developer's Console.

```
from oauth2client.flask_util import UserOAuth2

app = Flask(__name__)

app.config['SECRET_KEY'] = 'your-secret-key'

app.config['GOOGLE_OAUTH2_CLIENT_SECRETS_JSON'] = 'client_secrets.json'

# or, specify the client id and secret separately
app.config['GOOGLE_OAUTH2_CLIENT_ID'] = 'your-client-id'
app.config['GOOGLE_OAUTH2_CLIENT_SECRET'] = 'your-client-secret'

oauth2 = UserOAuth2(app)
```

Usage

Once configured, you can use the *UserOAuth2.required()* decorator to ensure that credentials are available within a view.

```
# Note that app.route should be the outermost decorator.
@app.route('/needs_credentials')
@oauth2.required
def example():
    # http is authorized with the user's credentials and can be used
    # to make http calls.
    http = oauth2.http()

    # Or, you can access the credentials directly
    credentials = oauth2.credentials
```

If you want credentials to be optional for a view, you can leave the decorator off and use *UserOAuth2.has_credentials()* to check.

```
@app.route('/optional')
def optional():
    if oauth2.has_credentials():
        return 'Credentials found!'
    else:
        return 'No credentials!'
```

When credentials are available, you can use *UserOAuth2.email* and *UserOAuth2.user_id* to access information from the ID Token, if available.

```
@app.route('/info')
@oauth2.required
def info():
    return "Hello, {} ({})".format(oauth2.email, oauth2.user_id)
```

URLs & Trigging Authorization

The extension will add two new routes to your application:

- `"oauth2.authorize"` -> `/oauth2authorize`
- `"oauth2.callback"` -> `/oauth2callback`

When configuring your OAuth2 credentials on the Google Developer's Console, be sure to add `http[s]://[your-app-url]/oauth2callback` as an authorized callback url.

Typically you don't not need to use these routes directly, just be sure to decorate any views that require credentials with `@oauth2.required`. If needed, you can trigger authorization at any time by redirecting the user to the URL returned by *UserOAuth2.authorize_url()*.

```
@app.route('/login')
def login():
    return oauth2.authorize_url("/")
```

Incremental Auth

This extension also supports Incremental Auth. To enable it, configure the extension with `include_granted_scopes`.

```
oauth2 = UserOAuth2(app, include_granted_scopes=True)
```

Then specify any additional scopes needed on the decorator, for example:

```
@app.route('/drive')
@oauth2.required(scopes=["https://www.googleapis.com/auth/drive"])
def requires_drive():
    ...

@app.route('/calendar')
@oauth2.required(scopes=["https://www.googleapis.com/auth/calendar"])
def requires_calendar():
    ...
```

The decorator will ensure that the the user has authorized all specified scopes before allowing them to access the view, and will also ensure that credentials do not lose any previously authorized scopes.

Storage

By default, the extension uses a Flask session-based storage solution. This means that credentials are only available for the duration of a session. It also means that with Flask's default configuration, the credentials will be visible in the session cookie. It's highly recommended to use database-backed session and to use https whenever handling user credentials.

If you need the credentials to be available longer than a user session or available outside of a request context, you will need to implement your own `oauth2client.Storage`.

class `oauth2client.flask_util.`**FlaskSessionStorage**
Bases: *oauth2client.client.Storage*

Storage implementation that uses Flask sessions.

Note that flask's default sessions are signed but not encrypted. Users can see their own credentials and non-https connections can intercept user credentials. We strongly recommend using a server-side session implementation.

locked_delete()

locked_get()

locked_put(*credentials*)

class `oauth2client.flask_util.`**UserOAuth2**(*app=None*, **args*, ***kwargs*)
Bases: `object`

Flask extension for making OAuth 2.0 easier.

Configuration values:

- `GOOGLE_OAUTH2_CLIENT_SECRETS_JSON` path to a client secrets json file, obtained from the credentials screen in the Google Developers console.
- `GOOGLE_OAUTH2_CLIENT_ID` the oauth2 credentials' client ID. This is only needed if `GOOGLE_OAUTH2_CLIENT_SECRETS_JSON` is not specified.
- `GOOGLE_OAUTH2_CLIENT_SECRET` the oauth2 credentials' client secret. This is only needed if `GOOGLE_OAUTH2_CLIENT_SECRETS_JSON` is not specified.

If app is specified, all arguments will be passed along to init_app.

If no app is specified, then you should call init_app in your application factory to finish initialization.

authorize_url(*return_url*, ***kwargs*)
Creates a URL that can be used to start the authorization flow.

When the user is directed to the URL, the authorization flow will begin. Once complete, the user will be redirected to the specified return URL.

Any kwargs are passed into the flow constructor.

authorize_view()
Flask view that starts the authorization flow.

Starts flow by redirecting the user to the OAuth2 provider.

callback_view()
Flask view that handles the user's return from OAuth2 provider.

On return, exchanges the authorization code for credentials and stores the credentials.

credentials
The credentials for the current user or None if unavailable.

email
Returns the user's email address or None if there are no credentials.

The email address is provided by the current credentials' id_token. This should not be used as unique identifier as the user can change their email. If you need a unique identifier, use user_id.

has_credentials()
Returns True if there are valid credentials for the current user.

http(**args*, ***kwargs*)
Returns an authorized http instance.

Can only be called if there are valid credentials for the user, such as inside of a view that is decorated with @required.

Parameters

- ***args** – Positional arguments passed to httplib2.Http constructor.
- ****kwargs** – Positional arguments passed to httplib2.Http constructor.

Raises ValueError if no credentials are available.

init_app(*app*, *scopes=None*, *client_secrets_file=None*, *client_id=None*, *client_secret=None*, *authorize_callback=None*, *storage=None*, ***kwargs*)
Initialize this extension for the given app.

Parameters

- **app** – A Flask application.
- **scopes** – Optional list of scopes to authorize.
- **client_secrets_file** – Path to a file containing client secrets. You can also specify the GOOGLE_OAUTH2_CLIENT_SECRETS_JSON config value.
- **client_id** – If not specifying a client secrets file, specify the OAuth2 client id. You can also specify the GOOGLE_OAUTH2_CLIENT_ID config value. You must also provide a client secret.
- **client_secret** – The OAuth2 client secret. You can also specify the GOOGLE_OAUTH2_CLIENT_SECRET config value.
- **authorize_callback** – A function that is executed after successful user authorization.
- **storage** – A oauth2client.client.Storage subclass for storing the credentials. By default, this is a Flask session based storage.

- **kwargs** – Any additional args are passed along to the Flow constructor.

required(*decorated_function=None*, *scopes=None*, ***decorator_kwargs*)
Decorator to require OAuth2 credentials for a view.

If credentials are not available for the current user, then they will be redirected to the authorization flow. Once complete, the user will be redirected back to the original page.

user_id
Returns the a unique identifier for the user

Returns None if there are no credentials.

The id is provided by the current credentials' id_token.

oauth2client.gce module

Utilities for Google Compute Engine

Utilities for making it easier to use OAuth 2.0 on Google Compute Engine.

class oauth2client.gce.**AppAssertionCredentials**(**args*, ***kwargs*)
Bases: *oauth2client.client.AssertionCredentials*

Credentials object for Compute Engine Assertion Grants

This object will allow a Compute Engine instance to identify itself to Google and other OAuth 2.0 servers that can verify assertions. It can be used for the purpose of accessing data stored under an account assigned to the Compute Engine instance itself.

This credential does not require a flow to instantiate because it represents a two legged flow, and therefore has all of the required information to generate and refresh its own access tokens.

create_scoped(*scopes*)

create_scoped_required()

classmethod from_json(*json_data*)

serialization_data

oauth2client.keyring_storage module

A keyring based Storage.

A Storage for Credentials that uses the keyring module.

class oauth2client.keyring_storage.**Storage**(*service_name*, *user_name*)
Bases: *oauth2client.client.Storage*

Store and retrieve a single credential to and from the keyring.

To use this module you must have the keyring module installed. See <http://pypi.python.org/pypi/keyring/>. This is an optional module and is not installed with oauth2client by default because it does not work on all the platforms that oauth2client supports, such as Google App Engine.

The keyring module <http://pypi.python.org/pypi/keyring/> is a cross-platform library for access the keyring capabilities of the local system. The user will be prompted for their keyring password when this module is used, and the manner in which the user is prompted will vary per platform.

Usage:

```
    from oauth2client.keyring_storage import Storage

    s = Storage('name_of_application', 'user1')
    credentials = s.get()
```

acquire_lock()
Acquires any lock necessary to access this Storage.

This lock is not reentrant.

locked_delete()
Delete Credentials file.

Parameters **credentials** – Credentials, the credentials to store.

locked_get()
Retrieve Credential from file.

Returns oauth2client.client.Credentials

locked_put(*credentials*)
Write Credentials to file.

Parameters **credentials** – Credentials, the credentials to store.

release_lock()
Release the Storage lock.

Trying to release a lock that isn't held will result in a RuntimeError.

oauth2client.locked_file module

Locked file interface that should work on Unix and Windows pythons.

This module first tries to use fcntl locking to ensure serialized access to a file, then falls back on a lock file if that is unavialable.

Usage:

```
f = LockedFile('filename', 'r+b', 'rb')
f.open_and_lock()
if f.is_locked():
  print('Acquired filename with r+b mode')
  f.file_handle().write('locked data')
else:
  print('Acquired filename with rb mode')
f.unlock_and_close()
```

exception oauth2client.locked_file.**AlreadyLockedException**
Bases: exceptions.Exception

Trying to lock a file that has already been locked by the LockedFile.

exception oauth2client.locked_file.**CredentialsFileSymbolicLinkError**
Bases: exceptions.Exception

Credentials files must not be symbolic links.

class oauth2client.locked_file.**LockedFile**(**args*, ***kwargs*)
Bases: object

Represent a file that has exclusive access.

file_handle()
Return the file_handle to the opened file.

filename()
Return the filename we were constructed with.

is_locked()
Return whether we successfully locked the file.

open_and_lock(*timeout=0*, *delay=0.05*)
Open the file, trying to lock it.

Parameters

- **timeout** – float, The number of seconds to try to acquire the lock.
- **delay** – float, The number of seconds to wait between retry attempts.

Raises

- *AlreadyLockedException* – if the lock is already acquired.
- IOError – if the open fails.

unlock_and_close()
Unlock and close a file.

oauth2client.locked_file.**validate_file**(*filename*)

oauth2client.multistore_file module

Multi-credential file store with lock support.

This module implements a JSON credential store where multiple credentials can be stored in one file. That file supports locking both in a single process and across processes.

The credential themselves are keyed off of:

- client_id
- user_agent
- scope

The format of the stored data is like so:

```
{
  'file_version': 1,
  'data': [
      {
          'key': {
              'clientId': '<client id>',
              'userAgent': '<user agent>',
              'scope': '<scope>'
          },
          'credential': {
              # JSON serialized Credentials.
          }
      }
  ]
}
```

exception oauth2client.multistore_file.**Error**
Bases: exceptions.Exception

Base error for this module.

exception oauth2client.multistore_file.**NewerCredentialStoreError**
Bases: *oauth2client.multistore_file.Error*

The credential store is a newer version than supported.

oauth2client.multistore_file.**get_all_credential_keys**(**args*, ***kwargs*)
Gets all the registered credential keys in the given Multistore.

Parameters

- **filename** – The JSON file storing a set of credentials
- **warn_on_readonly** – if True, log a warning if the store is readonly

Returns A list of the credential keys present in the file. They are returned as dictionaries that can be passed into get_credential_storage_custom_key to get the actual credentials.

oauth2client.multistore_file.**get_credential_storage**(**args*, ***kwargs*)
Get a Storage instance for a credential.

Parameters

- **filename** – The JSON file storing a set of credentials
- **client_id** – The client_id for the credential
- **user_agent** – The user agent for the credential
- **scope** – string or iterable of strings, Scope(s) being requested
- **warn_on_readonly** – if True, log a warning if the store is readonly

Returns An object derived from client.Storage for getting/setting the credential.

oauth2client.multistore_file.**get_credential_storage_custom_key**(**args*, ***kwargs*)
Get a Storage instance for a credential using a dictionary as a key.

Allows you to provide a dictionary as a custom key that will be used for credential storage and retrieval.

Parameters

- **filename** – The JSON file storing a set of credentials
- **key_dict** – A dictionary to use as the key for storing this credential. There is no ordering of the keys in the dictionary. Logically equivalent dictionaries will produce equivalent storage keys.
- **warn_on_readonly** – if True, log a warning if the store is readonly

Returns An object derived from client.Storage for getting/setting the credential.

oauth2client.multistore_file.**get_credential_storage_custom_string_key**(**args*, ***kwargs*)
Get a Storage instance for a credential using a single string as a key.

Allows you to provide a string as a custom key that will be used for credential storage and retrieval.

Parameters

- **filename** – The JSON file storing a set of credentials
- **key_string** – A string to use as the key for storing this credential.

- **warn_on_readonly** – if True, log a warning if the store is readonly

Returns An object derived from client.Storage for getting/setting the credential.

oauth2client.service_account module

A service account credentials class.

This credentials class is implemented on top of rsa library.

oauth2client.tools module

Command-line tools for authenticating via OAuth 2.0

Do the OAuth 2.0 Web Server dance for a command line application. Stores the generated credentials in a common file that is used by other example apps in the same directory.

oauth2client.tools.**run_flow**(**args*, ***kwargs*)

Core code for a command-line application.

The `run()` function is called from your application and runs through all the steps to obtain credentials. It takes a `Flow` argument and attempts to open an authorization server page in the user's default web browser. The server asks the user to grant your application access to the user's data. If the user grants access, the `run()` function returns new credentials. The new credentials are also stored in the `storage` argument, which updates the file associated with the `Storage` object.

It presumes it is run from a command-line application and supports the following flags:

--auth_host_name (string, default: `localhost`) Host name to use when running a local web server to handle redirects during OAuth authorization.

--auth_host_port (integer, default: `[8080, 8090]`) Port to use when running a local web server to handle redirects during OAuth authorization. Repeat this option to specify a list of values.

--[no]auth_local_webserver (boolean, default: `True`) Run a local web server to handle redirects during OAuth authorization.

The tools module defines an `ArgumentParser` the already contains the flag definitions that `run()` requires. You can pass that `ArgumentParser` to your `ArgumentParser` constructor:

```
parser = argparse.ArgumentParser(
    description=__doc__,
    formatter_class=argparse.RawDescriptionHelpFormatter,
    parents=[tools.argparser])
flags = parser.parse_args(argv)
```

Parameters

- **flow** – Flow, an OAuth 2.0 Flow to step through.
- **storage** – Storage, a `Storage` to store the credential in.
- **flags** – `argparse.Namespace`, The command-line flags. This is the object returned from calling `parse_args()` on `argparse.ArgumentParser` as described above.
- **http** – An instance of `httplib2.Http.request` or something that acts like it.

Returns Credentials, the obtained credential.

oauth2client.tools.**message_if_missing**(*filename*)
Helpful message to display if the CLIENT_SECRETS file is missing.

oauth2client.util module

Common utility library.

oauth2client.util.**positional**(*max_positional_args*)
A decorator to declare that only the first N arguments my be positional.

This decorator makes it easy to support Python 3 style keyword-only parameters. For example, in Python 3 it is possible to write:

```
def fn(pos1, *, kwonly1=None, kwonly1=None):
    ...
```

All named parameters after * must be a keyword:

```
fn(10, 'kw1', 'kw2')  # Raises exception.
fn(10, kwonly1='kw1')  # Ok.
```

Example

To define a function like above, do:

```
@positional(1)
def fn(pos1, kwonly1=None, kwonly2=None):
    ...
```

If no default value is provided to a keyword argument, it becomes a required keyword argument:

```
@positional(0)
def fn(required_kw):
    ...
```

This must be called with the keyword parameter:

```
fn()  # Raises exception.
fn(10)  # Raises exception.
fn(required_kw=10)  # Ok.
```

When defining instance or class methods always remember to account for `self` and `cls`:

```
class MyClass(object):

    @positional(2)
    def my_method(self, pos1, kwonly1=None):
        ...

    @classmethod
    @positional(2)
    def my_method(cls, pos1, kwonly1=None):
        ...
```

The positional decorator behavior is controlled by `util.positional_parameters_enforcement`, which may be set to `POSITIONAL_EXCEPTION`, `POSITIONAL_WARNING` or `POSITIONAL_IGNORE` to raise an exception, log a warning, or do nothing, respectively, if a declaration is violated.

Parameters **max_positional_arguments** – Maximum number of positional arguments. All parameters after the this index must be keyword only.

Returns A decorator that prevents using arguments after max_positional_args from being used as positional parameters.

Raises TypeError – if a key-word only argument is provided as a positional parameter, but only if util.positional_parameters_enforcement is set to POSITIONAL_EXCEPTION.

oauth2client.xsrfutil module

Helper methods for creating & verifying XSRF tokens.

oauth2client.xsrfutil.**generate_token**(**args*, ***kwargs*)

Generates a URL-safe token for the given user, action, time tuple.

Parameters

- **key** – secret key to use.
- **user_id** – the user ID of the authenticated user.
- **action_id** – a string identifier of the action they requested authorization for.
- **when** – the time in seconds since the epoch at which the user was authorized for this action. If not set the current time is used.

Returns A string XSRF protection token.

oauth2client.xsrfutil.**validate_token**(**args*, ***kwargs*)

Validates that the given token authorizes the user for the action.

Tokens are invalid if the time of issue is too old or if the token does not match what generateToken outputs (i.e. the token was forged).

Parameters

- **key** – secret key to use.
- **token** – a string of the token generated by generateToken.
- **user_id** – the user ID of the authenticated user.
- **action_id** – a string identifier of the action they requested authorization for.

Returns A boolean - True if the user is authorized for the action, False otherwise.

4.1.2 Module contents

Client library for using OAuth2, especially with Google APIs.

CHAPTER 5

Supported Python Versions

We support Python 2.6, 2.7, 3.3+. (Whatever this file says, the truth is always represented by our tox.ini).

We explicitly decided to support Python 3 beginning with version 3.3. Reasons for this include:

- Encouraging use of newest versions of Python 3
- Following the lead of prominent open-source projects
- Unicode literal support which allows for a cleaner codebase that works in both Python 2 and Python 3

O

Symbols

A

C

D

E

V