
python-oauth2 Documentation

Release 2.0.0

Markus Meyer

Jul 07, 2021

1	Usage	3
2	Installation	5
3	oauth2.grant — Grant classes and helpers	7
3.1	Three-legged OAuth	7
3.2	Two-legged OAuth	7
3.3	Helpers and base classes	7
3.4	Grant classes	8
4	oauth2.store — Storing and retrieving data	11
4.1	Data Types	11
4.2	Base Classes	11
4.3	Implementations	13
5	oauth2 — Provider Class	21
6	oauth2.web — Interaction over HTTP	23
6.1	Site adapters	23
6.2	HTTP flow	26
7	oauth2.client_authenticator — Client authentication	29
8	oauth2.tokengenerator — Generate Tokens	31
8.1	Base Class	31
8.2	Implementations	32
9	oauth2.log — Logging	33
10	oauth2.error — Error classes	35
11	Unique Access Tokens	37
11.1	What are unique access tokens?	37
11.2	Preconditions	37
11.3	Enabling the feature	38
12	Migration	39
12.1	0.7.0 -> 1.0.0	39

13 Using python-oauth2 with other frameworks	43
13.1 Tornado	43
14 Indices and tables	49
Python Module Index	51
Index	53

python-oauth2 is a framework that aims at making it easy to provide authentication via [OAuth 2.0](#) within an application stack.

Example:

```
from wsgiref.simple_server import make_server
import oauth2
import oauth2.grant
import oauth2.error
import oauth2.store.memory
import oauth2.tokengenerator
import oauth2.web.wsgi

# Create a SiteAdapter to interact with the user.
# This can be used to display confirmation dialogs and the like.
class ExampleSiteAdapter(oauth2.web.AuthorizationCodeGrantSiteAdapter,
                        oauth2.web.ImplicitGrantSiteAdapter):
    def authenticate(self, request, environ, scopes, client):
        # Check if the user has granted access
        if request.post_param("confirm") == "confirm":
            return {}

        raise oauth2.error.UserNotAuthenticated

    def render_auth_page(self, request, response, environ, scopes, client):
        response.body = '''
<html>
  <body>
    <form method="POST" name="confirmation_form">
      <input type="submit" name="confirm" value="confirm" />
      <input type="submit" name="deny" value="deny" />
    </form>
  </body>
</html>'''
        return response
```

(continues on next page)

```
def user_has_denied_access(self, request):
    # Check if the user has denied access
    if request.post_param("deny") == "deny":
        return True
    return False

# Create an in-memory storage to store your client apps.
client_store = oauth2.store.memory.ClientStore()
# Add a client
client_store.add_client(client_id="abc", client_secret="xyz",
                       redirect_uris=["http://localhost/callback"])

site_adapter = ExampleSiteAdapter()

# Create an in-memory storage to store issued tokens.
# LocalTokenStore can store access and auth tokens
token_store = oauth2.store.memory.TokenStore()

# Create the controller.
provider = oauth2.Provider(
    access_token_store=token_store,
    auth_code_store=token_store,
    client_store=client_store,
    token_generator=oauth2.tokengenerator.Uuid4()
)

# Add Grants you want to support
provider.add_grant(oauth2.grant.AuthorizationCodeGrant(site_adapter=site_adapter))
provider.add_grant(oauth2.grant.ImplicitGrant(site_adapter=site_adapter))

# Add refresh token capability and set expiration time of access tokens
# to 30 days
provider.add_grant(oauth2.grant.RefreshToken(expires_in=2592000))

# Wrap the controller with the Wsgi adapter
app = oauth2.web.wsgi.Application(provider=provider)

if __name__ == "__main__":
    httpd = make_server('', 8080, app)
    httpd.serve_forever()
```


CHAPTER 2

Installation

python-oauth2 is available on [PyPI](#):

```
pip install python-oauth2
```

Contents:

Grants are the heart of OAuth 2.0. Each Grant defines one way for a client to retrieve an authorization. They are defined in [Section 4](#) of the OAuth 2.0 spec.

OAuth 2.0 comes in two flavours of how an access token is issued: two-legged and three-legged auth. To avoid confusion they are explained in short here.

3.1 Three-legged OAuth

The “three” symbolizes the parties that are involved:

- The client that wants to access a resource on behalf of the user.
- The user who grants access to her resources.
- The server that issues the access token if the user allows it.

3.2 Two-legged OAuth

The two-legged OAuth process differs from the three-legged process by one missing participant. The user cannot allow or deny access.

So there are two remaining parties:

- The client that wants to access a resource.
- The server that issues the access.

3.3 Helpers and base classes

class `oauth2.grant.GrantHandlerFactory`
Base class every handler factory can extend.

This class defines the basic interface of each Grant.

```
class oauth2.grant.ScopeGrant (default_scope=None, scopes=None, scope_class=<class
                                'oauth2.grant.Scope'>, **kwargs)
```

Handling of scopes in the OAuth 2.0 flow.

Inherited by all grants that need to support scopes.

Parameters

- **default_scope** – The scope identifier that is returned by default. (optional)
- **scopes** – A list of strings identifying the scopes that the grant supports.
- **scope_class** – The class that does the actual handling in a request. Default: `oauth2.grant.Scope`.

```
class oauth2.grant.Scope (available=None, default=None)
```

Handling of the “scope” parameter in a request.

If `available` and `default` are both `None`, the “scope” parameter is ignored (the default).

Parameters

- **available** – A list of strings each defining one supported scope.
- **default** – Value to fall back to in case no scope is present in a request.

```
parse (request, source)
```

Parses scope value in given request.

Expects the value of the “scope” parameter in request to be a string where each requested scope is separated by a white space:

```
# One scope requested
"profile_read"

# Multiple scopes
"profile_read profile_write"
```

Parameters

- **request** – An instance of `oauth2.web.Request`.
- **source** – Where to read the scope from. Pass “body” in case of a application/x-www-form-urlencoded body and “query” in case the scope is supplied as a query parameter in the URL of a request.

```
class oauth2.grant.SiteAdapterMixin (site_adapter, **kwargs)
```

Mixed in by Grant classes that require a site adapter.

A concrete class must set the class attribute `site_adapter_class` that contains a reference to the site adapter class that this class expects.

3.4 Grant classes

```
class oauth2.grant.AuthorizationCodeGrant (unique_token=False, expires_in=0, **kwargs)
```

Bases: `oauth2.grant.GrantHandlerFactory`, `oauth2.grant.ScopeGrant`, `oauth2.grant.SiteAdapterMixin`

Implementation of the Authorization Code Grant auth flow.

This is a three-legged OAuth process.

Register an instance of this class with `oauth2.AuthorizationController` like this:

```
auth_controller = AuthorizationController()
auth_controller.add_grant_type(AuthorizationCodeGrant())
```

Changed in version 1.0.0: Require parameter `site_adapter`.

class `oauth2.grant.ImplicitGrant` (*default_scope=None, scopes=None, scope_class=<class 'oauth2.grant.Scope'>, **kwargs*)
 Bases: `oauth2.grant.GrantHandlerFactory`, `oauth2.grant.ScopeGrant`, `oauth2.grant.SiteAdapterMixin`

Implementation of the Implicit Grant auth flow.

This is a three-legged OAuth process.

Register an instance of this class with `oauth2.AuthorizationController` like this:

```
auth_controller = AuthorizationController()
auth_controller.add_grant_type(ImplicitGrant())
```

Changed in version 1.0.0: Require parameter `site_adapter`.

class `oauth2.grant.ResourceOwnerGrant` (*unique_token=False, expires_in=0, **kwargs*)
 Bases: `oauth2.grant.GrantHandlerFactory`, `oauth2.grant.ScopeGrant`, `oauth2.grant.SiteAdapterMixin`

Implementation of the Resource Owner Password Credentials Grant auth flow.

In this Grant a user provides a user name and a password. An access token is issued if the auth server was able to verify the user by her credentials.

Register an instance of this class with `oauth2.AuthorizationController` like this:

```
auth_controller = AuthorizationController()
auth_controller.add_grant_type(ResourceOwnerGrant())
```

Changed in version 1.0.0: Require parameter `site_adapter`.

class `oauth2.grant.RefreshToken` (*expires_in, reissue_refresh_tokens=False, **kwargs*)
 Bases: `oauth2.grant.GrantHandlerFactory`, `oauth2.grant.ScopeGrant`

Handles requests for refresh tokens as defined in <http://tools.ietf.org/html/rfc6749#section-6>.

Adding a Refresh Token to the `oauth2.AuthorizationController` like this:

```
auth_controller = AuthorizationController()
auth_controller.add_grant_type(ResourceOwnerGrant(tokens_expire=600))
auth_controller.add_grant_type(RefreshToken(tokens_expire=1200))
```

will cause `oauth2.grant.AuthorizationCodeGrant` and `oauth2.grant.ResourceOwnerGrant` to include a refresh token and expiration in the response. If `tokens_expire == 0`, the tokens will never expire.

oauth2.store — Storing and retrieving data

Store adapters to persist and retrieve data during the OAuth 2.0 process or for later use. This module provides base classes that can be extended to implement your own solution specific to your needs. It also includes implementations for popular storage systems like memcache.

4.1 Data Types

class `oauth2.datatype.AccessToken` (*client_id, grant_type, token, data={}, expires_at=None, refresh_token=None, refresh_expires_at=None, scopes=[], user_id=None*)

An access token and associated data.

class `oauth2.datatype.AuthorizationCode` (*client_id, code, expires_at, redirect_uri, scopes, data=None, user_id=None*)

Holds an authorization code and additional information.

class `oauth2.datatype.Client` (*identifier, secret, authorized_grants=None, authorized_response_types=None, redirect_uris=None*)

Representation of a client application.

4.2 Base Classes

class `oauth2.store.AccessTokenStore`

Base class for persisting an access token after it has been generated.

Used by two-legged and three-legged authentication flows.

delete_refresh_token (*refresh_token*)

Deletes an access token from the store using its refresh token to identify it. This invalidates both the access token and the refresh token.

Parameters `refresh_token` – A string containing the refresh token.

Returns `None`.

Raises `oauth2.error.AccessTokenNotFound` if no data could be retrieved for given `refresh_token`.

fetch_by_refresh_token (*refresh_token*)

Fetches an access token from the store using its refresh token to identify it.

Parameters `refresh_token` – A string containing the refresh token.

Returns An instance of `oauth2.datatype.AccessToken`.

Raises `oauth2.error.AccessTokenNotFound` if no data could be retrieved for given `refresh_token`.

fetch_existing_token_of_user (*client_id, grant_type, user_id*)

Fetches an access token identified by its client id, type of grant and user id.

This method must be implemented to make use of unique access tokens.

Parameters

- `client_id` – Identifier of the client a token belongs to.
- `grant_type` – The type of the grant that created the token
- `user_id` – Identifier of the user a token belongs to.

Returns An instance of `oauth2.datatype.AccessToken`.

Raises `oauth2.error.AccessTokenNotFound` if no data could be retrieved.

save_token (*access_token*)

Stores an access token and additional data.

Parameters `access_token` – An instance of `oauth2.datatype.AccessToken`.

class `oauth2.store.AuthCodeStore`

Base class for persisting and retrieving an auth token during the Authorization Code Grant flow.

delete_code (*code*)

Deletes an authorization code after it's use per section 4.1.2.

<http://tools.ietf.org/html/rfc6749#section-4.1.2>

Parameters `code` – The authorization code.

fetch_by_code (*code*)

Returns an AuthorizationCode fetched from a storage.

Parameters `code` – The authorization code.

Returns An instance of `oauth2.datatype.AuthorizationCode`.

Raises `oauth2.error.AuthCodeNotFound` if no data could be retrieved for given code.

save_code (*authorization_code*)

Stores the data belonging to an authorization code token.

Parameters `authorization_code` – An instance of `oauth2.datatype.AuthorizationCode`.

class `oauth2.store.ClientStore`

Base class for handling OAuth2 clients.

fetch_by_client_id (*client_id*)

Retrieve a client by its identifier.

Parameters `client_id` – Identifier of a client app.

Returns An instance of `oauth2.datatype.Client`.

Raises `oauth2.error.ClientNotFoundError` if no data could be retrieved for given `client_id`.

4.3 Implementations

4.3.1 `oauth2.store.memcache` — Memcache store adapters

class `oauth2.store.memcache.TokenStore` (*mc=None, prefix='oauth2', *args, **kwargs*)

Uses memcache to store access tokens and auth tokens.

This Store supports `python-memcached`. Arguments are passed to the underlying client implementation.

Initialization by passing an object:

```
# This example uses python-memcached
import memcache

# Somewhere in your application
mc = memcache.Client(servers=['127.0.0.1:11211'], debug=0)
# ...
token_store = TokenStore(mc=mc)
```

Initialization using `python-memcached`:

```
token_store = TokenStore(servers=['127.0.0.1:11211'], debug=0)
```

4.3.2 `oauth2.store.memory` — In-memory store adapters

Read or write data from or to local memory.

Though not very valuable in a production setup, these store adapters are great for testing purposes.

class `oauth2.store.memory.ClientStore`

Stores clients in memory.

add_client (*client_id, client_secret, redirect_uris, authorized_grants=None, authorized_response_types=None*)

Add a client app.

Parameters

- **client_id** – Identifier of the client app.
- **client_secret** – Secret the client app uses for authentication against the OAuth 2.0 provider.
- **redirect_uris** – A list of URIs to redirect to.

fetch_by_client_id (*client_id*)

Retrieve a client by its identifier.

Parameters **client_id** – Identifier of a client app.

Returns An instance of `oauth2.Client`.

Raises `ClientNotFoundError`

class `oauth2.store.memory.TokenStore`

Stores tokens in memory.

Useful for testing purposes or APIs with a very limited set of clients. Use memcache or redis as storage to be able to scale.

delete_code (*code*)

Deletes an authorization code after use :param code: The authorization code.

delete_refresh_token (*refresh_token*)

Deletes a refresh token after use :param refresh_token: The refresh_token.

fetch_by_code (*code*)

Returns an AuthorizationCode.

Parameters **code** – The authorization code.

Returns An instance of `oauth2.datatype.AuthorizationCode`.

Raises `AuthCodeNotFound` if no data could be retrieved for given code.

fetch_by_refresh_token (*refresh_token*)

Find an access token by its refresh token.

Parameters **refresh_token** – The refresh token that was assigned to an `AccessToken`.

Returns The `oauth2.datatype.AccessToken`.

Raises `oauth2.error.AccessTokenNotFound`

fetch_by_token (*token*)

Returns data associated with an access token or `None` if no data was found.

Useful for cases like validation where the access token needs to be read again.

Parameters **token** – A access token code.

Returns An instance of `oauth2.datatype.AccessToken`.

fetch_existing_token_of_user (*client_id, grant_type, user_id*)

Fetches an access token identified by its client id, type of grant and user id.

This method must be implemented to make use of unique access tokens.

Parameters

- **client_id** – Identifier of the client a token belongs to.
- **grant_type** – The type of the grant that created the token
- **user_id** – Identifier of the user a token belongs to.

Returns An instance of `oauth2.datatype.AccessToken`.

Raises `oauth2.error.AccessTokenNotFound` if no data could be retrieved.

save_code (*authorization_code*)

Stores the data belonging to an authorization code token.

Parameters **authorization_code** – An instance of `oauth2.datatype.AuthorizationCode`.

save_token (*access_token*)

Stores an access token and additional data in memory.

Parameters **access_token** – An instance of `oauth2.datatype.AccessToken`.

4.3.3 `oauth2.store.mongodb` — MongoDB store adapters

Store adapters to read/write data to from/to mongodb using pymongo.

class `oauth2.store.mongodb.MongoDbStore` (*collection*)
Base class extended by all concrete store adapters.

class `oauth2.store.mongodb.AccessTokenStore` (*collection*)
Create a new instance like this:

```
from pymongo import MongoClient

client = MongoClient('localhost', 27017)

db = client.test_database

access_token_store = AccessTokenStore(collection=db["access_tokens"])
```

class `oauth2.store.mongodb.AuthCodeStore` (*collection*)
Create a new instance like this:

```
from pymongo import MongoClient

client = MongoClient('localhost', 27017)

db = client.test_database

access_token_store = AuthCodeStore(collection=db["auth_codes"])
```

class `oauth2.store.mongodb.ClientStore` (*collection*)
Create a new instance like this:

```
from pymongo import MongoClient

client = MongoClient('localhost', 27017)

db = client.test_database

access_token_store = ClientStore(collection=db["clients"])
```

4.3.4 `oauth2.store.redisdb` — Redis store adapters

class `oauth2.store.redisdb.TokenStore` (*rs=None, prefix='oauth2', *args, **kwargs*)

class `oauth2.store.redisdb.ClientStore` (*rs=None, prefix='oauth2', *args, **kwargs*)

4.3.5 `oauth2.store.dbapi` — PEP249 compatible stores

DBApi 2.0 (PEP249) compatible implementation of data stores.

class `oauth2.store.dbapi.DatabaseStore` (*connection*)
Base class providing functionality used by a variety of store classes.

class `oauth2.store.dbapi.DbApiAccessTokenStore` (*connection*)
Base class of a DBApi 2.0 compatible `oauth2.store.AccessTokenStore`.

A concrete implementation extends this class and defines all or a subset of the `*_query` class attributes.

create_access_token_query = None

Insert an access token.

create_data_query = None

Insert one entry of additional data associated with an access token.

create_scope_query = None

Insert one scope associated with an access token.

delete_refresh_token (*refresh_token*)

Deletes an access token by its refresh token.

Parameters **refresh_token** – The refresh token of an access token as a *str*.

delete_refresh_token_query = None

Delete an access token by its refresh token.

fetch_by_refresh_token (*refresh_token*)

Retrieves an access token by its refresh token.

Parameters **refresh_token** – The refresh token of an access token as a *str*.

Returns An instance of *oauth2.datatype.AccessToken*.

Raises *oauth2.error.AccessTokenNotFound* if not access token could be retrieved.

fetch_by_refresh_token_query = None

Retrieve an access token by its refresh token.

fetch_data_by_access_token_query = None

Retrieve all data associated with an access token.

fetch_existing_token_of_user (*client_id, grant_type, user_id*)

Retrieve an access token issued to a client and user for a specific grant.

Parameters

- **client_id** – The identifier of a client as a *str*.
- **grant_type** – The type of grant.
- **user_id** – The identifier of the user the access token has been issued to.

Returns An instance of *oauth2.datatype.AccessToken*.

Raises *oauth2.error.AccessTokenNotFound* if not access token could be retrieved.

fetch_existing_token_of_user_query = None

Retrieve an access token issued to a client and user for a specific grant.

fetch_scopes_by_access_token_query = None

Retrieve all scopes associated with an access token.

save_token (*access_token*)

Creates a new entry for an access token in the database.

Parameters **access_token** –

An instance of *oauth2.datatype.AccessToken*.

return *True*.

class *oauth2.store.dbapi.DbApiAuthCodeStore* (*connection*)

Base class of a DBApi 2.0 compatible *oauth2.store.AuthCodeStore*.

A concrete implementation extends this class and defines all or a subset of the **_query* class attributes.

create_auth_code_query = None

Insert an auth code.

create_data_query = None

Insert one entry of additional data associated with an auth code.

create_scope_query = None

Insert one scope associated with an auth code.

delete_code (*code*)

Delete an auth code identified by its code.

Parameters *code* – The code of an auth code.

delete_code_query = None

Delete an auth code.

fetch_by_code (*code*)

Retrieves an auth code by its code.

Parameters *code* – The code of an auth code.

Returns An instance of *oauth2.datatype.AuthorizationCode*.

Raises *oauth2.error.AuthCodeNotFound* if no auth code could be retrieved.

fetch_code_query = None

Retrieve an auth code by its code.

fetch_data_query = None

Retrieve all data associated with an auth code.

fetch_scopes_query = None

Retrieve all scopes associated with an auth code.

save_code (*authorization_code*)

Creates a new entry of an auth code in the database.

Parameters *authorization_code* – An instance of *oauth2.datatype.AuthorizationCode*.

Returns *True* if everything went fine.

class *oauth2.store.dbapi.DbApiClientStore* (*connection*)

Base class of a DBApi 2.0 compatible *oauth2.store.ClientStore*.

A concrete implementation extends this class and defines all or a subset of the *_query class attributes.

fetch_by_client_id (*client_id*)

Retrieves a client by its identifier.

Parameters *client_id* – The identifier of a client.

Returns An instance of *oauth2.datatype.Client*.

Raises *oauth2.error.ClientError* if no client could be retrieved.

fetch_client_query = None

Retrieve a client by its identifier.

fetch_grants_query = None

Retrieve all grants that a client is allowed to use.

fetch_redirect_uris_query = None

Retrieve all redirect URIs of a client.

`fetch_response_types_query = None`
 Retrieve all response types that a client supports.

4.3.6 `oauth2.store.dbapi.mysql` — Mysql store adapters

Adapters to use mysql as the storage backend.

This module uses the API defined in `oauth2.store.dbapi`. Therefore no logic is defined here. Instead all classes define the queries required by `oauth2.store.dbapi`.

The queries have been created for the following SQL tables in mind:

```
CREATE TABLE IF NOT EXISTS `testdb`.`access_tokens` (
  `id` INT NOT NULL AUTO_INCREMENT COMMENT 'Unique identifier',
  `client_id` VARCHAR(32) NOT NULL COMMENT 'The identifier of a client. Assuming it
  ↪is an arbitrary text which is a maximum of 32 characters long.',
  `grant_type` ENUM('authorization_code', 'implicit', 'password', 'client_credentials
  ↪', 'refresh_token') NOT NULL COMMENT 'The type of a grant for which a token has
  ↪been issued.',
  `token` CHAR(36) NOT NULL COMMENT 'The access token.',
  `expires_at` TIMESTAMP NULL COMMENT 'The timestamp at which the token expires.',
  `refresh_token` CHAR(36) NULL COMMENT 'The refresh token.',
  `refresh_expires_at` TIMESTAMP NULL COMMENT 'The timestamp at which the refresh
  ↪token expires.',
  `user_id` INT NOT NULL COMMENT 'The identifier of the user this token belongs to.',
  PRIMARY KEY (`id`),
  INDEX `fetch_by_refresh_token` (`refresh_token` ASC),
  INDEX `fetch_existing_token_of_user` (`client_id` ASC, `grant_type` ASC, `user_id`
  ↪ASC))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `testdb`.`access_token_scopes` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(32) NOT NULL COMMENT 'The name of scope.',
  `access_token_id` INT NOT NULL COMMENT 'The unique identifier of the access token
  ↪this scope belongs to.',
  PRIMARY KEY (`id`))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `testdb`.`access_token_data` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `key` VARCHAR(32) NOT NULL COMMENT 'The key of an entry converted to the key in a
  ↪Python dict.',
  `value` VARCHAR(32) NOT NULL COMMENT 'The value of an entry converted to the value
  ↪in a Python dict.',
  `access_token_id` INT NOT NULL COMMENT 'The unique identifier of the access token a
  ↪row belongs to.',
  PRIMARY KEY (`id`))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `testdb`.`auth_codes` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `client_id` VARCHAR(32) NOT NULL COMMENT 'The identifier of a client. Assuming it
  ↪is an arbitrary text which is a maximum of 32 characters long.',
  `code` CHAR(36) NOT NULL COMMENT 'The authorisation code.',
  `expires_at` TIMESTAMP NOT NULL COMMENT 'The timestamp at which the token expires.',
  `redirect_uri` VARCHAR(128) NULL COMMENT 'The redirect URI send by the client
  ↪during the request of an authorisation code.',
```

(continues on next page)

(continued from previous page)

```

`user_id` INT NULL COMMENT 'The identifier of the user this authorisation code_
↳belongs to.',
PRIMARY KEY (`id`),
INDEX `fetch_code` (`code` ASC))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `testdb`.`auth_code_data` (
`id` INT NOT NULL AUTO_INCREMENT,
`key` VARCHAR(32) NOT NULL COMMENT 'The key of an entry converted to the key in a_
↳Python dict.',
`value` VARCHAR(32) NOT NULL COMMENT 'The value of an entry converted to the value_
↳in a Python dict.',
`auth_code_id` INT NOT NULL COMMENT 'The identifier of the authorisation code that_
↳this row belongs to.',
PRIMARY KEY (`id`))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `testdb`.`auth_code_scopes` (
`id` INT NOT NULL AUTO_INCREMENT,
`name` VARCHAR(32) NOT NULL,
`auth_code_id` INT NOT NULL,
PRIMARY KEY (`id`))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `testdb`.`clients` (
`id` INT NOT NULL AUTO_INCREMENT,
`identifier` VARCHAR(32) NOT NULL COMMENT 'The identifier of a client.',
`secret` VARCHAR(32) NOT NULL COMMENT 'The secret of a client.',
PRIMARY KEY (`id`))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `testdb`.`client_grants` (
`id` INT NOT NULL AUTO_INCREMENT,
`name` VARCHAR(32) NOT NULL,
`client_id` INT NOT NULL COMMENT 'The id of the client a row belongs to.',
PRIMARY KEY (`id`))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `testdb`.`client_redirect_uris` (
`id` INT NOT NULL AUTO_INCREMENT,
`redirect_uri` VARCHAR(128) NOT NULL COMMENT 'A URI of a client.',
`client_id` INT NOT NULL COMMENT 'The id of the client a row belongs to.',
PRIMARY KEY (`id`))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `testdb`.`client_response_types` (
`id` INT NOT NULL AUTO_INCREMENT,
`response_type` VARCHAR(32) NOT NULL COMMENT 'The response type that a client can_
↳use.',
`client_id` INT NOT NULL COMMENT 'The id of the client a row belongs to.',
PRIMARY KEY (`id`))
ENGINE = InnoDB;

```

```
class oauth2.store.dbapi.mysql.MySQLAccessTokenStore (connection)
```

```
class oauth2.store.dbapi.mysql.MySQLAuthCodeStore (connection)
```

```
class oauth2.store.dbapi.mysql.MySQLClientStore (connection)
```

```
class oauth2.Provider(access_token_store, auth_code_store, client_store, token_generator,
                    client_authentication_source=<function request_body>, re-
                    sponse_class=<class 'oauth2.web.Response'>)
```

Endpoint of requests to the OAuth 2.0 provider.

Parameters

- **access_token_store** (`oauth2.store.AccessTokenStore`) – An object that implements methods defined by `oauth2.store.AccessTokenStore`.
- **auth_code_store** (`oauth2.store.AuthCodeStore`) – An object that implements methods defined by `oauth2.store.AuthCodeStore`.
- **client_store** (`oauth2.store.ClientStore`) – An object that implements methods defined by `oauth2.store.ClientStore`.
- **token_generator** (`oauth2.tokengenerator.TokenGenerator`) – Object to generate unique tokens.
- **client_authentication_source** (*callable*) – A callable which when executed, authenticates a client. See `oauth2.client_authenticator`.
- **response_class** (`oauth2.web.Response`) – Class of the response object. Defaults to `oauth2.web.Response`.

Changed in version 1.0.0: Removed parameter `site_adapter`.

add_grant (*grant*)

Adds a Grant that the provider should support.

Parameters **grant** (`oauth2.grant.GrantHandlerFactory`) – An instance of a class that extends `oauth2.grant.GrantHandlerFactory`

dispatch (*request*, *environ*)

Checks which Grant supports the current request and dispatches to it.

Parameters

- **request** (`oauth2.web.Request`) – The incoming request.

- **environ** (*dict*) – Dict containing variables of the environment.

Returns An instance of `oauth2.web.Response`.

enable_unique_tokens ()

Enable the use of unique access tokens on all grant types that support this option.

scope_separator

Sets the separator of values in the scope query parameter. Defaults to " " (whitespace).

The following code makes the Provider use ";" instead of " ":

```
provider = Provider()
provider.scope_separator = ";"
```

Now the scope parameter in the request of a client can look like this: `scope=foo,bar`.

6.1 Site adapters

class `oauth2.web.UserFacingSiteAdapter`

Extended by site adapters that need to interact with the user.

Display HTML or redirect the user agent to another page of your website where she can do something before being returned to the OAuth 2.0 server.

render_auth_page (*request, response, environ, scopes, client*)

Defines how to display a confirmation page to the user.

Parameters

- **request** (`oauth2.web.Request`) – Incoming request data.
- **response** (`oauth2.web.Response`) – Response to return to a client.
- **environ** (*dict*) – Environment variables of the request.
- **scopes** (*list*) – A list of strings with each string being one requested scope.
- **client** (`oauth2.datatype.Client`) – The client that initiated the authorization process

Returns The response passed in as a parameter. It can contain HTML or issue a redirect.

Return type `oauth2.web.Response`

user_has_denied_access (*request*)

Checks if the user has denied access. This will lead to python-oauth2 returning a “access_denied” response to the requesting client app.

Parameters **request** (`oauth2.web.Request`) – Incoming request data.

Returns Return `True` if the user has denied access.

Return type `bool`

class `oauth2.web.AuthenticatingSiteAdapter`

Extended by site adapters that need to authenticate the user.

authenticate (*request, environ, scopes, client*)

Authenticates a user and checks if she has authorized access.

Parameters

- **request** (`oauth2.web.Request`) – Incoming request data.
- **environ** (`dict`) – Environment variables of the request.
- **scopes** (`list`) – A list of strings with each string being one requested scope.
- **client** (`oauth2.datatype.Client`) – The client that initiated the authorization process

Returns A `dict` containing arbitrary data that will be passed to the current storage adapter and saved with auth code and access token. Return a tuple in the form (*additional_data, user_id*) if you want to use *Unique Access Tokens*.

Return type `dict`

Raises `oauth2.error.UserNotAuthenticated` – If the user could not be authenticated.

class `oauth2.web.AuthorizationCodeGrantSiteAdapter`

Bases: `oauth2.web.UserFacingSiteAdapter`, `oauth2.web.AuthenticatingSiteAdapter`

Definition of a site adapter as required by `oauth2.grant.AuthorizationCodeGrant`.

authenticate (*request, environ, scopes, client*)

Authenticates a user and checks if she has authorized access.

Parameters

- **request** (`oauth2.web.Request`) – Incoming request data.
- **environ** (`dict`) – Environment variables of the request.
- **scopes** (`list`) – A list of strings with each string being one requested scope.
- **client** (`oauth2.datatype.Client`) – The client that initiated the authorization process

Returns A `dict` containing arbitrary data that will be passed to the current storage adapter and saved with auth code and access token. Return a tuple in the form (*additional_data, user_id*) if you want to use *Unique Access Tokens*.

Return type `dict`

Raises `oauth2.error.UserNotAuthenticated` – If the user could not be authenticated.

render_auth_page (*request, response, environ, scopes, client*)

Defines how to display a confirmation page to the user.

Parameters

- **request** (`oauth2.web.Request`) – Incoming request data.
- **response** (`oauth2.web.Response`) – Response to return to a client.
- **environ** (`dict`) – Environment variables of the request.
- **scopes** (`list`) – A list of strings with each string being one requested scope.

- **client** (`oauth2.datatype.Client`) – The client that initiated the authorization process

Returns The response passed in as a parameter. It can contain HTML or issue a redirect.

Return type `oauth2.web.Response`

user_has_denied_access (`request`)

Checks if the user has denied access. This will lead to python-oauth2 returning a “access_denied” response to the requesting client app.

Parameters **request** (`oauth2.web.Request`) – Incoming request data.

Returns Return `True` if the user has denied access.

Return type `bool`

class `oauth2.web.ImplicitGrantSiteAdapter`

Bases: `oauth2.web.UserFacingSiteAdapter`, `oauth2.web.AuthenticatingSiteAdapter`

Definition of a site adapter as required by `oauth2.grant.ImplicitGrant`.

authenticate (`request`, `environ`, `scopes`, `client`)

Authenticates a user and checks if she has authorized access.

Parameters

- **request** (`oauth2.web.Request`) – Incoming request data.
- **environ** (`dict`) – Environment variables of the request.
- **scopes** (`list`) – A list of strings with each string being one requested scope.
- **client** (`oauth2.datatype.Client`) – The client that initiated the authorization process

Returns A `dict` containing arbitrary data that will be passed to the current storage adapter and saved with auth code and access token. Return a tuple in the form (`additional_data`, `user_id`) if you want to use *Unique Access Tokens*.

Return type `dict`

Raises `oauth2.error.UserNotAuthenticated` – If the user could not be authenticated.

render_auth_page (`request`, `response`, `environ`, `scopes`, `client`)

Defines how to display a confirmation page to the user.

Parameters

- **request** (`oauth2.web.Request`) – Incoming request data.
- **response** (`oauth2.web.Response`) – Response to return to a client.
- **environ** (`dict`) – Environment variables of the request.
- **scopes** (`list`) – A list of strings with each string being one requested scope.
- **client** (`oauth2.datatype.Client`) – The client that initiated the authorization process

Returns The response passed in as a parameter. It can contain HTML or issue a redirect.

Return type `oauth2.web.Response`

user_has_denied_access (*request*)

Checks if the user has denied access. This will lead to python-oauth2 returning a “access_denied” response to the requesting client app.

Parameters **request** (`oauth2.web.Request`) – Incoming request data.

Returns Return `True` if the user has denied access.

Return type `bool`

class `oauth2.web.ResourceOwnerGrantSiteAdapter`

Bases: `oauth2.web.AuthenticatingSiteAdapter`

Definition of a site adapter as required by `oauth2.grant.ResourceOwnerGrant`.

authenticate (*request, environ, scopes, client*)

Authenticates a user and checks if she has authorized access.

Parameters

- **request** (`oauth2.web.Request`) – Incoming request data.
- **environ** (`dict`) – Environment variables of the request.
- **scopes** (`list`) – A list of strings with each string being one requested scope.
- **client** (`oauth2.datatype.Client`) – The client that initiated the authorization process

Returns A `dict` containing arbitrary data that will be passed to the current storage adapter and saved with auth code and access token. Return a tuple in the form (*additional_data, user_id*) if you want to use *Unique Access Tokens*.

Return type `dict`

Raises `oauth2.error.UserNotAuthenticated` – If the user could not be authenticated.

6.2 HTTP flow

class `oauth2.web.Request`

Base class defining the interface of a request.

get_param (*name, default=None*)

Retrieve a parameter from the query string of the request.

header (*name, default=None*)

Retrieve a header of the request.

method

Returns the HTTP method of the request.

path

Returns the current path portion of the current uri.

Used by some grants to determine which action to take.

post_param (*name, default=None*)

Retrieve a parameter from the body of the request.

class `oauth2.web.Response`

Contains data returned to the requesting user agent.

add_header (*header, value*)

Add a header to the response.

oauth2.client_authenticator — Client authentication

Every client that sends a request to obtain an access token needs to authenticate with the provider.

The authentication of confidential clients can be handled in several ways, some of which come bundled with this module.

class `oauth2.client_authenticator.ClientAuthenticator` (*client_store, source*)

Handles authentication of a client both by its identifier as well as by its identifier and secret.

Parameters

- **client_store** (`oauth2.store.ClientStore`) – The Client Store to retrieve a client from.
- **source** (*callable*) – A callable that returns a tuple (<client_id>, <client_secret>)

by_identifier (*request*)

Authenticates a client by its identifier.

Parameters **request** (`oauth2.web.Request`) – The incoming request

Returns The identified client

Return type `oauth2.datatype.Client`

Raises

class `OAuthInvalidNoRedirectError`

by_identifier_secret (*request*)

Authenticates a client by its identifier and secret (aka password).

Parameters **request** (`oauth2.web.Request`) – The incoming request

Returns The identified client

Return type `oauth2.datatype.Client`

Raises `OAuthInvalidError` – If the client could not be found, is not allowed to use the current grant or supplied invalid credentials

`oauth2.client_authenticator.http_basic_auth(request)`

Extracts the credentials of a client using HTTP Basic Auth.

Expects the `client_id` to be the username and the `client_secret` to be the password part of the Authorization header.

Parameters `request` (`oauth2.web.Request`) – The incoming request

Returns A tuple in the format of (<CLIENT ID>, <CLIENT SECRET>)

Return type tuple

`oauth2.client_authenticator.request_body(request)`

Extracts the credentials of a client from the *application/x-www-form-urlencoded* body of a request.

Expects the `client_id` to be the value of the `client_id` parameter and the `client_secret` to be the value of the `client_secret` parameter.

Parameters `request` (`oauth2.web.Request`) – The incoming request

Returns A tuple in the format of (<CLIENT ID>, <CLIENT SECRET>)

Return type tuple

oauth2.tokengenerator — Generate Tokens

Provides various implementations of algorithms to generate an Access Token or Refresh Token.

8.1 Base Class

class `oauth2.tokengenerator.TokenGenerator`

Base class of every token generator.

create_access_token_data (*data*, *grant_type*, *user_id*)

Create data needed by an access token.

Parameters

- **data** (*dict*) – Arbitrary data as returned by the `authenticate()` method of a `SiteAdapter`.
- **grant_type** (*str*) –
- **user_id** (*int*) – Identifier of the current user as returned by the `authenticate()` method of a `SiteAdapter`.

Returns A dict containing the `access_token` and the `token_type`. If the value of `TokenGenerator.expires_in` is larger than 0, a `refresh_token` will be generated too.

Return type dict

Changed in version 1.1.0: New parameters `data` and `user_id`

generate (*data=None*, *user_id=None*)

Implemented by generators extending this base class.

Parameters

- **data** (*dict*) – Arbitrary data as returned by the `authenticate()` method of a `SiteAdapter`.

- **user_id** (*int*) – Identifier of the current user as returned by the `authenticate()` method of a `SiteAdapter`.

Raises `NotImplementedError` –

Changed in version 1.1.0: New parameters `data` and `user_id`

8.2 Implementations

class `oauth2.tokengenerator.URandomTokenGenerator` (*length=40*)

Bases: `oauth2.tokengenerator.TokenGenerator`

Create a token using `os.urandom()`.

generate (*data=None, user_id=None*)

Returns A new token

Return type `str`

class `oauth2.tokengenerator.Uuid4`

Bases: `oauth2.tokengenerator.TokenGenerator`

Generate a token using `uuid4`.

generate (*data=None, user_id=None*)

Returns A new token

Return type `str`

Logging support

There are two loggers available:

- `oauth2.application`: Logging of uncaught exceptions
- `oauth2.general`: General purpose logging of debug errors and warnings

If logging has not been configured, you will likely see this error:

```
No handlers could be found for logger "oauth2.application"
```

Make sure that logging is configured to avoid this:

```
import logging
logging.basicConfig()
```


Errors raised during the OAuth 2.0 flow.

class `oauth2.error.AccessTokenNotFound`

Error indicating that an access token could not be read from the storage backend by an instance of `oauth2.store.AccessTokenStore`.

class `oauth2.error.AuthCodeNotFound`

Error indicating that an authorization code could not be read from the storage backend by an instance of `oauth2.store.AuthCodeStore`.

class `oauth2.error.ClientNotFoundError`

Error raised by an implementation of `oauth2.store.ClientStore` if a client does not exist.

class `oauth2.error.OAuthBaseError` (*error, error_uri=None, explanation=None*)

Base class used by all OAuth 2.0 errors.

Parameters

- **error** – Identifier of the error.
- **error_uri** – Set this to delivery an URL to your documentation that describes the error. (optional)
- **explanation** – Short message that describes the error. (optional)

class `oauth2.error.OAuthInvalidError` (*error, error_uri=None, explanation=None*)

Indicates an error during validation of a request.

class `oauth2.error.UserNotAuthenticated`

Raised by a `oauth2.web.SiteAdapter` if a user could not be authenticated.

Unique Access Tokens

This page explains the concepts of unique access tokens and how to enable this feature.

11.1 What are unique access tokens?

When the use of unique access tokens is enabled the Provider will respond with an existing access token to subsequent requests of a client instead of issuing a new token on each request.

An existing access token will be returned if the following conditions are met:

- The access token has been issued for the requesting client
- The access token has been issued for the same user as in the current request
- The requested scope is the same as in the existing access token
- The requested type is the same as in the existing access token

Note: Unique access tokens are currently supported by `oauth2.grant.AuthorizationCodeGrant` and `oauth2.grant.ResourceOwnerGrant`.

11.2 Preconditions

As stated in the previous section, a unique access token is bound not only to a client but also to a user. To make this work the Provider needs some kind of identifier that is unique for each user (typically the ID of a user in the database). The identifier is stored along with all the other information of an access token. It has to be returned as the second item of a tuple by your implementation of `oauth2.web.AuthenticatingSiteAdapter.authenticate`:

```
class MySiteAdapter(SiteAdapter):
```

(continues on next page)

(continued from previous page)

```
def authenticate(self, request, environ, scopes):
    // Your logic here

    return None, user["id"]
```

11.3 Enabling the feature

Unique access tokens are turned off by default. They can be turned on for each grant individually:

```
auth_code_grant = oauth2.grant.AuthorizationCodeGrant(unique_token=True)
provider = oauth2.Provider() // Parameters omitted for readability
provider.add_grant(auth_code_grant)
```

or you can enable them for all grants that support this feature after initialization of *oauth2.Provider*:

```
provider = oauth2.Provider() // Parameters omitted for readability
provider.enable_unique_tokens()
```

Note: If you enable the feature but forgot to make *oauth2.web.AuthenticatingSiteAdapter.authenticate* return a user identifier, the Provider will respond with an error to requests for a token.

12.1 0.7.0 -> 1.0.0

12.1.1 One site adapter per grant

Starting from 1.0.0, the grants `oauth2.grant.AuthorizationCodeGrant`, `oauth2.grant.ImplicitGrant` and `oauth2.grant.ResourceOwnerGrant` expect the parameter `site_adapter` to be passed to them.

`oauth2.Provider` does not accept the parameter `site_adapter` anymore.

The base class `oauth2.web.SiteAdapter` does not exist anymore.

Code that looks like this in version 0.7.0

```
from oauth2 import Provider
from oauth2.web import SiteAdapter
from oauth2.grant import AuthorizationCodeGrant

class ExampleSiteAdapter(SiteAdapter):
    ...

provider = Provider(
    ...,
    site_adapter=ExampleSiteAdapter(),
    ...
)
provider.add_grant(AuthorizationCodeGrant())
```

has to be rewritten to look similar to the following

```
from oauth2 import Provider
from oauth2.web import AuthorizationCodeGrantSiteAdapter
from oauth2.grant import AuthorizationCodeGrant
```

(continues on next page)

(continued from previous page)

```

class ExampleSiteAdapter(AuthorizationCodeGrantSiteAdapter):
    # Override the methods defined in AuthorizationCodeGrantSiteAdapter to suite your
    ↪needs
    ...

# No site_adapter anymore
provider = Provider(...)

provider.add_grant(AuthorizationCodeGrant(site_adapter=ExampleSiteAdapter()))

```

12.1.2 WSGI adapter classes refactoring

All code to connect python-oauth2 with a WSGI compliant server has been moved to the module `oauth2.web.wsgi`.

Also the class `Wsgi` has been renamed to `Application` and now expects the parameter `provider` instead of `server`.

Before:

```

from oauth2.web import Wsgi

# Instantiating storage and provider...

app = Wsgi(server=provider)

```

After:

```

from oauth2.web.wsgi import Application

# Instantiating storage and provider...

app = Application(provider=provider)

```

12.1.3 Client passed to methods `authenticate` and `render_auth_page` of a Site Adapter

Before:

```

class ExampleSiteAdapter(AuthenticatingSiteAdapter, UserFacingSiteAdapter):
    def authenticate(self, request, environ, scopes):
        # code

    def render_auth_page(self, request, response, environ, scopes):
        # code

```

After:

```

class ExampleSiteAdapter(AuthenticatingSiteAdapter, UserFacingSiteAdapter):
    def authenticate(self, request, environ, scopes, client):
        # code

```

(continues on next page)

(continued from previous page)

```
def render_auth_page(self, request, response, environ, scopes, client):  
    # code
```

Using `python-oauth2` with other frameworks

13.1 Tornado

Warning: Tornado support is currently experimental.

Use Tornado to serve token requests:

```
import logging
import json
import os
import signal
import sys
import urllib
import urlparse

from multiprocessing.process import Process

sys.path.insert(0, os.path.abspath(os.path.realpath(__file__) + '/../../../'))

from oauth2 import Provider
from oauth2.error import UserNotAuthenticated
from oauth2.grant import AuthorizationCodeGrant
from oauth2.tokengenerator import Uuid4
from oauth2.store.memory import ClientStore, TokenStore
from oauth2.web import AuthorizationCodeGrantSiteAdapter
from oauth2.web.tornado import OAuth2Handler
from tornado.ioloop import IOLoop
from tornado.web import Application, url
from wsgiref.simple_server import make_server, WSGIRequestHandler

logging.basicConfig(level=logging.DEBUG)
```

(continues on next page)

```
class ClientRequestHandler (WSGIRequestHandler):
    """
    Request handler that enables formatting of the log messages on the console.

    This handler is used by the client application.
    """
    def address_string(self):
        return "client app"

class OAuthRequestHandler (WSGIRequestHandler):
    """
    Request handler that enables formatting of the log messages on the console.

    This handler is used by the python-oauth2 application.
    """
    def address_string(self):
        return "python-oauth2"

class TestSiteAdapter (AuthorizationCodeGrantSiteAdapter):
    """
    This adapter renders a confirmation page so the user can confirm the auth
    request.
    """

    CONFIRMATION_TEMPLATE = """
<html>
  <body>
    <p>
      <a href="{url}&confirm=1">confirm</a>
    </p>
    <p>
      <a href="{url}&confirm=0">deny</a>
    </p>
  </body>
</html>
    """

    def render_auth_page(self, request, response, environ, scopes, client):
        url = request.path + "?" + request.query_string
        response.body = self.CONFIRMATION_TEMPLATE.format(url=url)

        return response

    def authenticate(self, request, environ, scopes, client):
        if request.method == "GET":
            if request.get_param("confirm") == "1":
                return
            raise UserNotAuthenticated

    def user_has_denied_access(self, request):
        if request.method == "GET":
            if request.get_param("confirm") == "0":
                return True
```

(continues on next page)

(continued from previous page)

```

    return False

class ClientApplication(object):
    """
    Very basic application that simulates calls to the API of the
    python-oauth2 app.
    """
    callback_url = "http://localhost:8081/callback"
    client_id = "abc"
    client_secret = "xyz"
    api_server_url = "http://localhost:8080"

    def __init__(self):
        self.access_token = None
        self.auth_token = None
        self.token_type = ""

    def __call__(self, env, start_response):
        if env["PATH_INFO"] == "/app":
            status, body, headers = self._serve_application(env)
        elif env["PATH_INFO"] == "/callback":
            status, body, headers = self._read_auth_token(env)
        else:
            status = "301 Moved"
            body = ""
            headers = {"Location": "/app"}

        start_response(status,
                       [(header, val) for header, val in headers.iteritems()])
        return body

    def _request_access_token(self):
        print("Requesting access token...")

        post_params = {"client_id": self.client_id,
                       "client_secret": self.client_secret,
                       "code": self.auth_token,
                       "grant_type": "authorization_code",
                       "redirect_uri": self.callback_url}
        token_endpoint = self.api_server_url + "/token"

        result = urllib.urlopen(token_endpoint,
                                urllib.urlencode(post_params))
        content = ""
        for line in result:
            content += line

        result = json.loads(content)
        self.access_token = result["access_token"]
        self.token_type = result["token_type"]

        confirmation = "Received access token '%s' of type '%s'" % (self.access_token,
        ↪ self.token_type)
        print(confirmation)
        return "302 Found", "", {"Location": "/app"}

```

(continues on next page)

(continued from previous page)

```
def _read_auth_token(self, env):
    print("Receiving authorization token...")

    query_params = urlparse.parse_qs(env["QUERY_STRING"])

    if "error" in query_params:
        location = "/app?error=" + query_params["error"][0]
        return "302 Found", "", {"Location": location}

    self.auth_token = query_params["code"][0]

    print("Received temporary authorization token '%s'" % (self.auth_token,))

    return "302 Found", "", {"Location": "/app"}

def _request_auth_token(self):
    print("Requesting authorization token...")

    auth_endpoint = self.api_server_url + "/authorize"
    query = urllib.urlencode({"client_id": "abc",
                             "redirect_uri": self.callback_url,
                             "response_type": "code"})

    location = "%s?%s" % (auth_endpoint, query)

    return "302 Found", "", {"Location": location}

def _serve_application(self, env):
    query_params = urlparse.parse_qs(env["QUERY_STRING"])

    if ("error" in query_params
        and query_params["error"][0] == "access_denied"):
        return "200 OK", "User has denied access", {}

    if self.access_token is None:
        if self.auth_token is None:
            return self._request_auth_token()
        else:
            return self._request_access_token()
    else:
        confirmation = "Current access token '%s' of type '%s'" % (self.access_
→token, self.token_type)
        return "200 OK", str(confirmation), {}

def run_app_server():
    app = ClientApplication()

    try:
        httpd = make_server('', 8081, app, handler_class=ClientRequestHandler)

        print("Starting Client app on http://localhost:8081/...")
        httpd.serve_forever()
    except KeyboardInterrupt:
        httpd.server_close()
```

(continues on next page)

(continued from previous page)

```

def run_auth_server():
    client_store = ClientStore()
    client_store.add_client(client_id="abc", client_secret="xyz",
                           redirect_uris=["http://localhost:8081/callback"])

    token_store = TokenStore()

    provider = Provider(access_token_store=token_store,
                       auth_code_store=token_store, client_store=client_store,
                       token_generator=Uuid4())
    provider.add_grant(AuthorizationCodeGrant(site_adapter=TestSiteAdapter()))

    try:
        app = Application([
            url(provider.authorize_path, OAuth2Handler, dict(provider=provider)),
            url(provider.token_path, OAuth2Handler, dict(provider=provider)),
        ])

        app.listen(8080)
        print("Starting OAuth2 server on http://localhost:8080/...")
        IOloop.current().start()

    except KeyboardInterrupt:
        IOloop.close()

def main():
    auth_server = Process(target=run_auth_server)
    auth_server.start()
    app_server = Process(target=run_app_server)
    app_server.start()
    print("Access http://localhost:8081/app in your browser")

    def sigint_handler(signal, frame):
        print("Terminating servers...")
        auth_server.terminate()
        auth_server.join()
        app_server.terminate()
        app_server.join()

    signal.signal(signal.SIGINT, sigint_handler)

if __name__ == "__main__":
    main()

```

```

class oauth2.web.tornado.OAuth2Handler(application, request, **kwargs)

```

```

    initialize(provider)

```


CHAPTER 14

Indices and tables

- `genindex`
- `modindex`
- `search`

O

oauth2, 1
oauth2.client_authenticator, 29
oauth2.error, 35
oauth2.grant, 7
oauth2.log, 33
oauth2.store, 11
oauth2.store.dbapi, 15
oauth2.store.dbapi.mysql, 18
oauth2.store.memcache, 13
oauth2.store.memory, 13
oauth2.store.mongodb, 15
oauth2.store.redisdb, 15
oauth2.tokengenerator, 31
oauth2.web, 23
oauth2.web.tornado, 43

A

AccessToken (class in *oauth2.datatype*), 11
 AccessTokenNotFound (class in *oauth2.error*), 35
 AccessTokenStore (class in *oauth2.store*), 11
 AccessTokenStore (class in *oauth2.store.mongodb*), 15
 add_client() (*oauth2.store.memory.ClientStore* method), 13
 add_grant() (*oauth2.Provider* method), 21
 add_header() (*oauth2.web.Response* method), 26
 AuthCodeNotFound (class in *oauth2.error*), 35
 AuthCodeStore (class in *oauth2.store*), 12
 AuthCodeStore (class in *oauth2.store.mongodb*), 15
 authenticate() (*oauth2.web.AuthenticatingSiteAdapter* method), 24
 authenticate() (*oauth2.web.AuthorizationCodeGrantSiteAdapter* method), 24
 authenticate() (*oauth2.web.ImplicitGrantSiteAdapter* method), 25
 authenticate() (*oauth2.web.ResourceOwnerGrantSiteAdapter* method), 26
 AuthenticatingSiteAdapter (class in *oauth2.web*), 23
 AuthorizationCode (class in *oauth2.datatype*), 11
 AuthorizationCodeGrant (class in *oauth2.grant*), 8
 AuthorizationCodeGrantSiteAdapter (class in *oauth2.web*), 24

B

by_identifier() (*oauth2.client_authenticator.ClientAuthenticator* method), 29
 by_identifier_secret() (*oauth2.client_authenticator.ClientAuthenticator* method), 29

C

Client (class in *oauth2.datatype*), 11

ClientAuthenticator (class in *oauth2.client_authenticator*), 29
 ClientNotFoundError (class in *oauth2.error*), 35
 ClientStore (class in *oauth2.store*), 12
 ClientStore (class in *oauth2.store.memory*), 13
 ClientStore (class in *oauth2.store.mongodb*), 15
 ClientStore (class in *oauth2.store.redisdb*), 15
 create_access_token_data() (*oauth2.tokengenerator.TokenGenerator* method), 31
 create_access_token_query (*oauth2.store.dbapi.DbApiAccessTokenStore* attribute), 15
 create_auth_code_query (*oauth2.store.dbapi.DbApiAuthCodeStore* attribute), 16
 create_data_query (*oauth2.store.dbapi.DbApiAccessTokenStore* attribute), 16
 create_data_query (*oauth2.store.dbapi.DbApiAuthCodeStore* attribute), 17
 create_scope_query (*oauth2.store.dbapi.DbApiAccessTokenStore* attribute), 16
 create_scope_query (*oauth2.store.dbapi.DbApiAuthCodeStore* attribute), 17

D

DatabaseStore (class in *oauth2.store.dbapi*), 15
 DbApiAccessTokenStore (class in *oauth2.store.dbapi*), 15
 DbApiAuthCodeStore (class in *oauth2.store.dbapi*), 16
 DbApiClientStore (class in *oauth2.store.dbapi*), 17
 delete_code() (*oauth2.store.AuthCodeStore* method), 12
 delete_code() (*oauth2.store.dbapi.DbApiAuthCodeStore* method), 17

- `delete_code()` (*oauth2.store.memory.TokenStore method*), 14
`delete_code_query` (*oauth2.store.dbapi.DbApiAuthCodeStore attribute*), 17
`delete_refresh_token()` (*oauth2.store.AccessTokenStore method*), 11
`delete_refresh_token()` (*oauth2.store.dbapi.DbApiAccessTokenStore method*), 16
`delete_refresh_token()` (*oauth2.store.memory.TokenStore method*), 14
`delete_refresh_token_query` (*oauth2.store.dbapi.DbApiAccessTokenStore attribute*), 16
`dispatch()` (*oauth2.Provider method*), 21
- ## E
- `enable_unique_tokens()` (*oauth2.Provider method*), 22
- ## F
- `fetch_by_client_id()` (*oauth2.store.ClientStore method*), 12
`fetch_by_client_id()` (*oauth2.store.dbapi.DbApiClientStore method*), 17
`fetch_by_client_id()` (*oauth2.store.memory.ClientStore method*), 13
`fetch_by_code()` (*oauth2.store.AuthCodeStore method*), 12
`fetch_by_code()` (*oauth2.store.dbapi.DbApiAuthCodeStore method*), 17
`fetch_by_code()` (*oauth2.store.memory.TokenStore method*), 14
`fetch_by_refresh_token()` (*oauth2.store.AccessTokenStore method*), 12
`fetch_by_refresh_token()` (*oauth2.store.dbapi.DbApiAccessTokenStore method*), 16
`fetch_by_refresh_token()` (*oauth2.store.memory.TokenStore method*), 14
`fetch_by_refresh_token_query` (*oauth2.store.dbapi.DbApiAccessTokenStore attribute*), 16
`fetch_by_token()` (*oauth2.store.memory.TokenStore method*), 14
`fetch_client_query` (*oauth2.store.dbapi.DbApiClientStore attribute*), 17
`fetch_code_query` (*oauth2.store.dbapi.DbApiAuthCodeStore attribute*), 17
`fetch_data_by_access_token_query` (*oauth2.store.dbapi.DbApiAccessTokenStore attribute*), 16
`fetch_data_query` (*oauth2.store.dbapi.DbApiAuthCodeStore attribute*), 17
`fetch_existing_token_of_user()` (*oauth2.store.AccessTokenStore method*), 12
`fetch_existing_token_of_user()` (*oauth2.store.dbapi.DbApiAccessTokenStore method*), 16
`fetch_existing_token_of_user()` (*oauth2.store.memory.TokenStore method*), 14
`fetch_existing_token_of_user_query` (*oauth2.store.dbapi.DbApiAccessTokenStore attribute*), 16
`fetch_grants_query` (*oauth2.store.dbapi.DbApiClientStore attribute*), 17
`fetch_redirect_uris_query` (*oauth2.store.dbapi.DbApiClientStore attribute*), 17
`fetch_response_types_query` (*oauth2.store.dbapi.DbApiClientStore attribute*), 17
`fetch_scopes_by_access_token_query` (*oauth2.store.dbapi.DbApiAccessTokenStore attribute*), 16
`fetch_scopes_query` (*oauth2.store.dbapi.DbApiAuthCodeStore attribute*), 17
- ## G
- `generate()` (*oauth2.tokengenerator.TokenGenerator method*), 31
`generate()` (*oauth2.tokengenerator.URandomTokenGenerator method*), 32
`generate()` (*oauth2.tokengenerator.Uuid4 method*), 32
`get_param()` (*oauth2.web.Request method*), 26
`GrantHandlerFactory` (*class in oauth2.grant*), 7
- ## H
- `header()` (*oauth2.web.Request method*), 26
`http_basic_auth()` (*in oauth2.client_authenticator*), 29
- ## I
- `ImplicitGrant` (*class in oauth2.grant*), 9

ImplicitGrantSiteAdapter (class in request_body () (in module oauth2.web), 25
 initialize () (oauth2.web.tornado.OAuth2Handler method), 47

M

method (oauth2.web.Request attribute), 26
 MongoDBStore (class in oauth2.store.mongodb), 15
 MySQLAccessTokenStore (class in oauth2.store.dbapi.mysql), 19
 MySQLAuthCodeStore (class in oauth2.store.dbapi.mysql), 19
 MySQLClientStore (class in oauth2.store.dbapi.mysql), 19

O

oauth2 (module), 1
 oauth2.client_authenticator (module), 29
 oauth2.error (module), 35
 oauth2.grant (module), 7
 oauth2.log (module), 33
 oauth2.store (module), 11
 oauth2.store.dbapi (module), 15
 oauth2.store.dbapi.mysql (module), 18
 oauth2.store.memcache (module), 13
 oauth2.store.memory (module), 13
 oauth2.store.mongodb (module), 15
 oauth2.store.redisdb (module), 15
 oauth2.tokengenerator (module), 31
 oauth2.web (module), 23
 oauth2.web.tornado (module), 43
 OAuth2Handler (class in oauth2.web.tornado), 47
 OAuthBaseError (class in oauth2.error), 35
 OAuthInvalidError (class in oauth2.error), 35

P

parse () (oauth2.grant.Scope method), 8
 path (oauth2.web.Request attribute), 26
 post_param () (oauth2.web.Request method), 26
 Provider (class in oauth2), 21

R

RefreshToken (class in oauth2.grant), 9
 render_auth_page () (oauth2.web.AuthorizationCodeGrantSiteAdapter method), 24
 render_auth_page () (oauth2.web.ImplicitGrantSiteAdapter method), 25
 render_auth_page () (oauth2.web.UserFacingSiteAdapter method), 23
 Request (class in oauth2.web), 26

S

save_code () (oauth2.store.AuthCodeStore method), 12
 save_code () (oauth2.store.dbapi.DbApiAuthCodeStore method), 17
 save_code () (oauth2.store.memory.TokenStore method), 14
 save_token () (oauth2.store.AccessTokenStore method), 12
 save_token () (oauth2.store.dbapi.DbApiAccessTokenStore method), 16
 save_token () (oauth2.store.memory.TokenStore method), 14
 Scope (class in oauth2.grant), 8
 scope_separator (oauth2.Provider attribute), 22
 ScopeGrant (class in oauth2.grant), 8
 SiteAdapterMixin (class in oauth2.grant), 8

T

TokenGenerator (class in oauth2.tokengenerator), 31
 TokenStore (class in oauth2.store.memcache), 13
 TokenStore (class in oauth2.store.memory), 13
 TokenStore (class in oauth2.store.redisdb), 15

U

URandomTokenGenerator (class in oauth2.tokengenerator), 32
 user_has_denied_access () (oauth2.web.AuthorizationCodeGrantSiteAdapter method), 25
 user_has_denied_access () (oauth2.web.ImplicitGrantSiteAdapter method), 25
 user_has_denied_access () (oauth2.web.UserFacingSiteAdapter method), 23
 UserFacingSiteAdapter (class in oauth2.web), 23
 UserNotAuthenticated (class in oauth2.error), 35
 Uuid4 (class in oauth2.tokengenerator), 32