oauth2-stateless Documentation

Release 1.0.0

Andrew Grytsenko

May 21, 2018

Contents

oauth2-stateless is a framework that aims at making it easy to provide authentication via OAuth 2.0 within an application stack.

CHAPTER 1

Usage

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)

(continued from previous 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.Uuid4TokenGenerator()
)

# 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

oauth2-stateless is available on PyPI:

```
pip install oauth2-stateless
```

Contents:

CHAPTER 3

oauth2.grant — Grant classes and helpers

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())
```

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())
```

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())
```

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.

CHAPTER 4

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

Parameters **code** – The authorization code.

delete_refresh_token(*refresh_token*)
Deletes a refresh token after use

Parameters **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.dynamodb — Dynamodb store adapters

class oauth2.store.dynamodb.**TokenStore**(*connect*)
Dynamodb Access Token Store

4.3.6 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*.

Returns *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.

4.3.7 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 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,
```

(continues on next page)

(continued from previous page)

```
  `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.',
  `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`))
```

(continues on next page)

(continued from previous page)

```
ENGINE = InnoDB;
```

class oauth2.store.dbapi.mysql.**MysqlAccessTokenStore**(*connection*)

class oauth2.store.dbapi.mysql.**MysqlAuthCodeStore**(*connection*)

class oauth2.store.dbapi.mysql.**MysqlClientStore**(*connection*)

CHAPTER 5

oauth2 — Provider Class

class oauth2.**Provider**(*access_token_store*, *auth_code_store*, *client_store*, *token_generator*, *client_authentication_source=<function request_body>*, *response_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*.

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*.

CHAPTER 6

oauth2.web — Interaction over HTTP

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 oauth2-stateless returning a "acess_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 oauth2-stateless returning a "acess_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 oauth2-stateless returning a "acess_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.

CHAPTER 7

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 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

CHAPTER 8

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*, *scopes*, *grant_type*, *user_id*, *client_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`
- **client_id** (*str*) – Identifier of the current client.

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

generate(*grant_type=None*, *data=None*, *scopes=None*, *user_id=None*, *client_id=None*)
Implemented by generators extending this base class.

Parameters

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

- **scopes** (*dict*) – scopes for oauth session
- **user_id** (*int*) – Identifier of the current user as returned by the `authenticate()` method of a `SiteAdapter`
- **client_id** (*str*) – Identifier of the current client.

Raises **NotImplementedError** –

refresh_generate(*grant_type=None*, *data=None*, *scopes=None*, *user_id=None*, *client_id=None*)
Implemented by refresh generators extending this base class.

Parameters

- **grant_type** (*str*) – Identifier token grant_type
- **data** (*dict*) – Arbitrary data as returned by the `authenticate()` method of a `SiteAdapter`.
- **scopes** (*dict*) – scopes for oauth session
- **user_id** (*int*) – Identifier of the current user as returned by the `authenticate()` method of a `SiteAdapter`
- **client_id** (*str*) – Identifier of the current client.

Raises **NotImplementedError** –

8.2 Implementations

class oauth2.tokengenerator.**StatelessTokenGenerator**(*secret_key*)
Bases: *oauth2.tokengenerator.TokenGenerator*

Generate a token using JSON Web Tokens tokens.

generate(*grant_type=None*, *data=None*, *scopes=None*, *user_id=None*, *client_id=None*)

Returns A new token

Return type str

refresh_generate(*grant_type=None*, *data=None*, *scopes=None*, *user_id=None*, *client_id=None*)

Returns A new refresh token

Return type str

class oauth2.tokengenerator.**URandomTokenGenerator**(*length=40*)
Bases: *oauth2.tokengenerator.TokenGenerator*

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

generate(*grant_type=None*, *data=None*, *scopes=None*, *user_id=None*, *client_id=None*)

Returns A new token

Return type str

refresh_generate(*grant_type=None*, *data=None*, *scopes=None*, *user_id=None*, *client_id=None*)

Returns A new token

Return type str

class oauth2.tokengenerator.**Uuid4TokenGenerator**

Bases: *oauth2.tokengenerator.TokenGenerator*

Generate a token using uuid4.

generate(*grant_type=None*, *data=None*, *scopes=None*, *user_id=None*, *client_id=None*)

Returns A new token

Return type str

refresh_generate(*grant_type=None*, *data=None*, *scopes=None*, *user_id=None*, *client_id=None*)

Returns A new token

Return type str

CHAPTER 9

oauth2.log — Logging

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()
```

CHAPTER 10

oauth2.error — Error classes

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 exists.

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.

CHAPTER 11

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.

CHAPTER 12

Using `oauth2-stateless` with other frameworks

12.1 aiohttp

12.2 Flask

Classes for handling flask HTTP request/response flow.

```
import logging
import os
import signal
import sys
from wsgiref.simple_server import WSGIRequestHandler, make_server

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

from flask import Flask
from oauth2 import Provider
from oauth2.compatibility import json, parse_qs, urlencode
from oauth2.error import UserNotAuthenticated
from oauth2.grant import AuthorizationCodeGrant
from oauth2.store.memory import ClientStore, TokenStore
from oauth2.tokengenerator import Uuid4TokenGenerator
from oauth2.web import AuthorizationCodeGrantSiteAdapter
from oauth2.web.flask import oauth_request_hook
from flask import render_template_string

if sys.version_info >= (3, 0):
    from multiprocessing import Process
    from urllib.request import urlopen
else:
    from multiprocessing.process import Process
    from urllib2 import urlopen
```

(continues on next page)

(continued from previous page)

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


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

    def render_auth_page(self, request, response, environ, scopes, client):
        confirmation_template = """
            <p><a href="{{ url }}&confirm=1">confirm</a></p>
            <p><a href="{{ url }}&confirm=0">deny</a></p>
        """
        page_url = request.path + "?" + request.query_string
        main_login_body = render_template_string(confirmation_template, url=page_url)
        response.body = main_login_body
        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
        return False

    def token(self):
        pass


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

    def __init__(self):
        self.access_token_result = None
```

(continues on next page)

(continued from previous page)

```
        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.items()])
        return [body.encode('utf-8')]

    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"

        token_result = urlopen(token_endpoint, urlencode(post_params).encode('utf-8'))
        result = json.loads(token_result.read().decode('utf-8'))

        self.access_token_result = result
        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"}

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

        query_params = 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...")
```

(continues on next page)

(continued from previous page)

```
        auth_endpoint = self.api_server_url + "/authorize"
        query = 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 = 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_result is None:
            if self.auth_token is None:
                return self._request_auth_token()
            return self._request_access_token()
        confirmation = "Current access token '%s' of type '%s'" % (self.access_token,␣
↪self.token_type)
        return "200 OK", confirmation, {}


def run_app_server():
    app = ClientApplication()

    try:
        httpd = make_server('', 8080, app, handler_class=ClientRequestHandler)

        print("Starting Client app on http://localhost:8080/...")
        httpd.serve_forever()
    except KeyboardInterrupt:
        httpd.server_close()


def run_auth_server():
    client_store = ClientStore()
    client_store.add_client(client_id="abc", client_secret="xyz", redirect_uris=[
↪"http://localhost:8080/callback"])

    token_store = TokenStore()
    site_adapter = TestSiteAdapter()

    provider = Provider(access_token_store=token_store,
                        auth_code_store=token_store, client_store=client_store,
                        token_generator=Uuid4TokenGenerator())
    provider.add_grant(AuthorizationCodeGrant(site_adapter=site_adapter))

    app = Flask(__name__)

    flask_hook = oauth_request_hook(provider)
    app.add_url_rule('/authorize', 'authorize', view_func=flask_hook(site_adapter.
↪authenticate),
                     methods=['GET', 'POST'])
    app.add_url_rule('/token', 'token', view_func=flask_hook(site_adapter.token),␣
↪methods=['POST'])

    app.run(host='0.0.0.0', port=8081)
    print("Starting OAuth2 server on http://localhost:8081/...")
```

(continues on next page)

(continued from previous page)

```
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:8080/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.flask.**Request**(*request*)
Contains data of the current HTTP request.

oauth2.web.flask.**oauth_request_hook**(*provider*)
Initialise Oauth2 interface bewtween flask and oauth2 server

12.3 Tornado

Warning: Tornado support is currently experimental.

Use Tornado to serve token requests:

```
import logging
import os
import signal
import sys
from wsgiref.simple_server import WSGIRequestHandler, make_server

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

from oauth2 import Provider
from oauth2.compatibility import json, parse_qs, urlencode
from oauth2.error import UserNotAuthenticated
from oauth2.grant import AuthorizationCodeGrant
from oauth2.store.memory import ClientStore, TokenStore
from oauth2.tokengenerator import Uuid4TokenGenerator
from oauth2.web import AuthorizationCodeGrantSiteAdapter
from oauth2.web.tornado import OAuth2Handler
from tornado.ioloop import IOLoop
from tornado.web import Application, url

if sys.version_info >= (3, 0):
```

(continues on next page)

(continued from previous page)

```
    from multiprocessing import Process
    from urllib.request import urlopen
else:
    from multiprocessing.process import Process
    from urllib2 import urlopen

logging.basicConfig(level=logging.DEBUG)


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 oauth2-stateless application.
    """
    def address_string(self):
        return "oauth2-stateless"


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):
        page_url = request.path + "?" + request.query_string
        response.body = self.CONFIRMATION_TEMPLATE.format(url=page_url)

        return response

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

(continues on next page)

(continued from previous page)

```
                return
        raise UserNotAuthenticated

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


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

    def __init__(self):
        self.access_token_result = None
        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.items()])
        return [body.encode('utf-8')]

    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"

        token_result = urlopen(token_endpoint, urlencode(post_params).encode('utf-8'))
        result = json.loads(token_result.read().decode('utf-8'))

        self.access_token_result = result
        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)
```

(continues on next page)

(continued from previous page)

```
        print(confirmation)
        return "302 Found", "", {"Location": "/app"}

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

        query_params = 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 = 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 = 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_result is None:
            if self.auth_token is None:
                return self._request_auth_token()
            return self._request_access_token()
        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('', 8080, app, handler_class=ClientRequestHandler)

        print("Starting Client app on http://localhost:8080/...")
        httpd.serve_forever()
    except KeyboardInterrupt:
        httpd.server_close()


def run_auth_server():
    client_store = ClientStore()
    client_store.add_client(client_id="abc", client_secret="xyz", redirect_uris=[
↪"http://localhost:8080/callback"])
```

(continues on next page)

(continued from previous page)

```
    token_store = TokenStore()

    provider = Provider(access_token_store=token_store,
                        auth_code_store=token_store, client_store=client_store,
                        token_generator=Uuid4TokenGenerator())
    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(8081)
        print("Starting OAuth2 server on http://localhost:8081/...")
        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:8080/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 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

P

R

S

T

U