
Django Facebook Documentation

Release 6.0.6

Thierry Schellenbach

May 05, 2017

Contents

1	Status	1
2	News	3
3	Demo & About	5
4	Features	7
5	Documentation	9
6	Contributing and Running tests	11
6.1	Basic documentation	12
6.2	Using the Open graph API	17
6.3	Advanced documentation	22
6.4	API documentation	24
6.5	Indices and tables	33
	Python Module Index	35

CHAPTER 1

Status

Django and Facebook are both rapidly changing at the moment. Meanwhile, I'm caught up in a startup and don't have much spare time. The library needs a good round of testing against the latest python, django and facebook graph API. Contributions are strongly appreciated. Seriously, give github a try, fork and get started :)

CHAPTER 2

News

- django-facebook will be dropping support for django < 1.8 since [django only supports](#) versions 1.8 and above.

CHAPTER 3

Demo & About

Django Facebook enables your users to easily register using the Facebook API. It converts the Facebook user data and creates regular User and Profile objects. This makes it easy to integrate with your existing Django application.

After registration Django Facebook gives you access to user's graph. Allowing for applications such as:

- Open graph/ Timeline functionality
- Seamless personalization
- Inviting friends
- Finding friends
- Posting to a users profile

Updates and tutorials can be found on my blog [mellowmorning](#)

- **Access the Facebook API, from:**
 - Your website (Using javascript OAuth)
 - Facebook canvas pages (For building facebook applications)
 - Mobile (Or any other flow giving you a valid access token)
- Django User Registration (Convert Facebook user data into a user model)
- Store likes, friends and user data locally.
- Facebook FQL access
- OAuth 2.0 compliant
- Automated reauthentication (For expired tokens)
- Includes Open Facebook (stable and tested Python client to the graph API)

Basics

- Installation
- Customizing
- Settings
- Registration backends & Redirects

Open Facebook API

- Getting an OpenFacebook object
- Making calls

Advanced

- Mobile
- Celery
- Signals
- Canvas

Contributing and Running tests

Tests are run from within the example project. You can run them yourself as follows:

install from git

```
facebook_example/manage.py test django_facebook
```

Vagrant

A vagrant development setup is included in the GIT repo. Assuming you have vagrant installed, simply type the following in your shell:

```
# First get a fresh Django-Facebook checkout
git clone git@github.com:tschellenbach/Django-facebook.git django-facebook

# Go to the directory:
cd django-facebook

# Time to start Vagrant (grab a cup of coffee after this command, it'll take a while)
↪ :)
vagrant up; vagrant provision

# Finally done?
vagrant ssh
python manage.py runserver 0:8000
```

To have a working Django Facebook example up and running at 192.168.50.42:8000/facebook/example/. For the facebook login to work simply map that ip to vagrant.mellowmorning.com (Since Facebook checks the domain)

You can run the test suite by typing:

```
python manage.py test django_facebook
```

Basic documentation

Installation

0.) Create a Facebook App

You need a facebook app to use the open graph API and make the login process work. If you don't have a facebook app, now is the time to create one. You can create a facebook app at [this url](#).

Facebook authentication only works if the domain you are working on matches your app domain. Be sure to configure the right app domain in your facebook application settings.

An example:

Your site is `www.fashiolista.com`, your app domain is set to `fashiolista.com` and you do your development at `local.fashiolista.com`. If you try to authenticate with Facebook from a different domain you will get an authentication error.

1.) Pip install

```
pip install django_facebook
```

2.) Settings

Define the following settings in your settings.py file:

```
FACEBOOK_APP_ID=str('<your app id>')
FACEBOOK_APP_SECRET = str('your app secret')
```

Context processor

add django facebook to your installed apps:

```
'django_facebook',
```

Add this line to EACH of your templates' context processors (`TEMPLATES[*]['OPTIONS']['context_processors']` setting):

```
'django_facebook.context_processors.facebook',
# and add request if you didn't do so already
'django.core.context_processors.request',
```

The full setting on a new django 1.11 app looks like this

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                'django_facebook.context_processors.facebook',
            ],
        },
    ],
],
```

```
    },
]
```

Auth backend

Add these to your `AUTHENTICATION_BACKENDS` setting:

```
'django_facebook.auth_backends.FacebookBackend',
'django.contrib.auth.backends.ModelBackend',
```

The full setting on a new django 1.11 app looks like this:

```
AUTHENTICATION_BACKENDS = (
    'django_facebook.auth_backends.FacebookBackend',
    'django.contrib.auth.backends.ModelBackend',
)
```

3.) Urls Now, add this line to your url config:

```
url(r'^facebook/', include('django_facebook.urls')),
url(r'^accounts/', include('django_facebook.auth_urls')), #Don't add this line if you_
↳use django registration or userena for registration and auth.
```

4.) Update your models

The following step depends on your version of Django. Django versions before 1.5 need to use a custom profile model. Whereas Django 1.5 and up can use a custom user model.

A. Custom user model

If you don't already have a custom user model, simply uses the provided model by setting your `AUTH_USER_MODEL` to `FacebookCustomUser`:

```
AUTH_USER_MODEL = 'django_facebook.FacebookCustomUser'
```

Alternatively use the abstract model provided in `django_facebook.models.FacebookProfileModel`

Note: Please note that Django Facebook does not support custom user models with `USERNAME_FIELD` different than `username`.

B. Profile model

If you don't already have a custom Profile model, simply uses the provided model by setting your `AUTH_PROFILE_MODULE` to `FacebookProfile`:

```
AUTH_PROFILE_MODULE = 'django_facebook.FacebookProfile'
```

Be sure to sync the database via `python manage.py migrate --run-syncdb` after setting this up.

Otherwise Django Facebook provides an abstract model which you can inherit like this.

```
from django.db import models
from django.dispatch.dispatcher import receiver
from django_facebook.models import FacebookModel
from django.db.models.signals import post_save
from django_facebook.utils import get_user_model, get_profile_model
from your_project import settings
```

```
class MyCustomProfile(FacebookModel):
    user = models.OneToOneField(settings.AUTH_USER_MODEL)

    @receiver(post_save)
    def create_profile(sender, instance, created, **kwargs):
        """Create a matching profile whenever a user object is created."""
        if sender == get_user_model():
            user = instance
            profile_model = get_profile_model()
            if profile_model == MyCustomProfile and created:
                profile, new = MyCustomProfile.objects.get_or_create(user=instance)`
```

Remember to update `AUTH_PROFILE_MODULE` in settings to your new profile. Don't forget to update your database using `syncdb` or `south` after this step.

Note: You need a profile model attached to every user model. For new accounts this will get created automatically, but you will need to migrate older accounts.

Congratulations

Right now you should have a working registration/connect/login in flow available at `/facebook/example/!` (settings.DEBUG needs to be set to True) Test if everything is working and ensure you didn't miss a step somewhere. If you encounter any difficulties please open an issue.

Of course you now want to customize things like the login button, the page after registration etc. This is explained in the integration section.

Customizing

Now it's time to customize things a little. For a full example you can look at `connect.html` in the templates directory.

Login flow

1.) First load the css and javascript:

```
<link href="{% STATIC_URL %}django_facebook/css/facebook.css" type="text/css" rel=
  ↳"stylesheet" media="all" />
{% include 'django_facebook/_facebook_js.html' %}
```

If you encounter issues here you probably don't have django static files setup correctly.

2.) Next design the form

You can control redirects using `next`, `register_next` and `error_next`.

```
<form action="{% url 'facebook_connect' %}?facebook_login=1" method="post">
  <input type="hidden" value="{% request.path %}" name="next" />
  <input type="hidden" value="{% request.path %}" name="register_next" />
  <input type="hidden" value="{% request.path %}" name="error_next" />
  {% csrf_token %}
  <input onclick="F.connect(this.parentNode); return false;" type="image" src="{%
  ↳STATIC_URL %}django_facebook/images/facebook_login.png" />
</form>
```

Connect flow

Usually you'll also want to offer your users the ability to connect their existing account to Facebook. You can control this by setting `connect_facebook=1`. The default behaviour is not to connect automatically. (As this previously caused

users to connect their accounts to Facebook by accident)

```
<form action="{% url 'facebook_connect' %}?facebook_login=1" method="post">
  <input type="hidden" value="1" name="connect" />
  {% csrf_token %}
  <a onclick="F.connect(this.parentNode); return false;" href="javascript:void(0);">
↪Connect</a>
</form>
```

Settings

Security settings

FACEBOOK_APP_ID

Your facebook app id

FACEBOOK_APP_SECRET

Your facebook app secret

FACEBOOK_DEFAULT_SCOPE

The default scope we should use, note that registration will break without email Defaults to ['email', 'user_about_me', 'user_birthday', 'user_website']

Customizing registration

FACEBOOK_REGISTRATION_BACKEND

Allows you to overwrite the registration backend class Specify a full path to a class (defaults to `django_facebook.registration_backends.FacebookRegistrationBackend`)

Likes and Friends

FACEBOOK_STORE_LIKES

If we should store likes

FACEBOOK_STORE_FRIENDS

If we should store friends

FACEBOOK_CELERY_STORE

If celery should be used to retrieve friends and likes

FACEBOOK_CELERY_TOKEN_EXTEND

Use celery for updating tokens, recommended since it's quite slow

Redirects

For most applications you can simply use the `next`, `register_next` and `error_next` parameters to control the post registration flow. The “next” parameter provides the default next page for login, connect, error or register actions. “register_next” and “error_next” allow you to customize the next page for those specific scenarios. This is useful when you for instance want to show an introduction page to new users.

Flows

- Login (login_next, next, default)
- Connect (connect_next, next, default)
- Register (register_next, next, default)
- Error (error_next, next, default)

The default redirect is specified by the FACEBOOK_LOGIN_DEFAULT_REDIRECT setting.

If the default customizability isn't adequate for your needs you can also subclass the registration backend.

```
class CustomBackend(FacebookRegistrationBackend):
    def post_connect(action):
        # go as crazy as you want, just be sure to return a response
        response = HttpResponseRedirect('/something/')
        if action is CONNECT_ACTIONS.LOGIN:
            response = HttpResponseRedirect('/')
        return response
```

Registration Backends

Registration Backends

By default Django Facebook ships with its own registration system. It provides a basic manual registration flow and the option to connect with Facebook.

If you are looking for something more complicated it's possible to integrate with Userena or Django Registration. To add support for these systems we use the FACEBOOK_REGISTRATION_BACKEND setting.

Django Registration support Create a registration backend which subclasses both Django Facebook and Django Registration's registration backend. An example is included in facebook_example/registration_backends.py

```
# in registration_backends.py
class DjangoRegistrationDefaultBackend(DefaultBackend, NooptRegistrationBackend):
    """
    The redirect behaviour will still be controlled by the
    post_error
    post_connect
    functions
    the form and other settings will be taken from the default backend
    """
    pass

# in your settings file
FACEBOOK_REGISTRATION_BACKEND = 'registration.backends.default.DefaultBackend'
```

Django Userena support

Django Userena is easier to work with than Django Registration. It is however hard to setup unittesting with Userena, so the integration between Django Facebook and Userena might not work. Please report any bugs you run into.

```
FACEBOOK_REGISTRATION_BACKEND = 'django_facebook.registration_backends.UserenaBackend'
```

Also have a look at the userena settings file in the facebook example project. It provides a clear example of how to configure Userena and Django Facebook to work together.

Other registration systems

Supporting any other registration system is quite easy. Adjust the above settings to point to your own code. Note that the form's save method needs to return the new user object.

Also have a look at the API docs for *FacebookRegistrationBackend*

Using the Open graph API

Getting a graph object

Now that you have Django Facebook up and running you'll want to make API calls to Facebook. The first step is getting an *OpenFacebook* object setup.

User object

For users which registered through Django Facebook, you'll have an access token stored in the database. Note that by default tokens expire quickly (couple of hours), Django Facebook will try to extend these to 60 days.

```
graph = user.get_offline_graph()
```

From the request

If you've just authenticated via Facebook you can get the graph from the request as such

```
# persistent (graph stored in session)
get_persistent_graph(request)
require_persistent_graph(request)

# not persistent
get_facebook_graph(request)
require_facebook_graph(request)
```

Typically you'll use the decorators in views where you access Facebook.

Access token

For mobile apps you'll sometimes get an access token directly

```
from open_facebook import OpenFacebook
graph = OpenFacebook(access_token)
```

Open Facebook API

Open Facebook allows you to use Facebook's open graph API with simple python code

Features

- Supported and maintained
- Tested so people can contribute
- Facebook exceptions are mapped
- Logging

Basic examples:

```
facebook = OpenFacebook(access_token)

# Getting info about me
facebook.get('me')

# Learning some more about fashiolista
facebook.get('fashiolista')

# Writing your first comment
facebook.set('fashiolista/comments', message='I love Fashiolista!')

# Posting to a users wall
facebook.set('me/feed', message='check out fashiolista',
            url='http://www.fashiolista.com')

# Liking a page
facebook.set('fashiolista/likes')

# Getting who likes cocacola
facebook.set('cocacola/likes')

# Use fql to retrieve your name
facebook.fql('SELECT name FROM user WHERE uid = me()')

# Executing fql in batch
facebook.batch_fql([
    'SELECT uid, name, pic_square FROM user WHERE uid = me()',
    'SELECT uid, rsvp_status FROM event_member WHERE eid=12345678',
])

# Uploading pictures
photo_urls = [
    'http://e.fashiocdn.com/images/entities/0/7/B/I/9/0.365x365.jpg',
    'http://e.fashiocdn.com/images/entities/0/5/e/e/r/0.365x365.jpg',
]
for photo in photo_urls:
    print facebook.set('me/feed', message='Check out Fashiolista',
                    picture=photo, url='http://www.fashiolista.com')
```

Getting an access token

Once you get your access token, Open Facebook gives you access to the Facebook API There are 3 ways of getting a facebook access_token and these are currently implemented by Django Facebook.

1. **code is passed as request parameter and traded for an access_token** using the api
2. code is passed through a signed cookie and traded for an access_token
3. **access_token is passed directly (retrieved through javascript, which would be bad security, or through one of the mobile flows.)**

If you are looking to develop your own flow for a different framework have a look at Facebook's documentation: <http://developers.facebook.com/docs/authentication/>

Also have a look at the *FacebookRequired* decorator and `get_persistent_graph()` function to understand the required functionality

Api docs:

class `open_facebook.api.OpenFacebook` (*access_token=None, prefetched_data=None, expires=None, current_user_id=None, version=None*)

The main api class, initialize using

Example:

```
graph = OpenFacebook(access_token)
print(graph.get('me'))
```

batch_fql (*queries_dict*)

queries_dict a dict with the required queries returns the query results in:

Example:

```
response = facebook.batch_fql({
    name: 'SELECT uid, name, pic_square FROM user WHERE uid = me()',
    rsvp: 'SELECT uid, rsvp_status FROM event_member WHERE eid=12345678',
})

# accessing the results
response['fql_results']['name']
response['fql_results']['rsvp']
```

Parameters *queries_dict* – A dictionary of queries to execute

Returns dict

delete (*path, *args, **kwargs*)

Delete the given bit of data

Example:

```
graph.delete(12345)
```

Parameters *path* – the id of the element to remove

fql (*query, **kwargs*)

Runs the specified query against the Facebook FQL API.

Example:

```
open_facebook.fql('SELECT name FROM user WHERE uid = me()')
```

Parameters

- **query** – The query to execute
- **kwargs** – Extra options to send to facebook

Returns dict

get (*path, version=None, **kwargs*)

Make a Facebook API call

Example:

```
open_facebook.get('me')
open_facebook.get('me', fields='id,name')
```

Parameters `path` – The path to use for making the API call

Returns dict

get_many (**ids, **kwargs*)

Make a batched Facebook API call For multiple ids

Example:

```
open_facebook.get_many('me', 'starbucks')
open_facebook.get_many('me', 'starbucks', fields='id,name')
```

Parameters `path` – The path to use for making the API call

Returns dict

get_request_url (*path='', old_api=False, version=None, **params*)

Gets the url for the request.

has_permissions (*required_permissions*)

Validate if all the required_permissions are currently given by the user

Example:

```
open_facebook.has_permissions(['publish_actions', 'read_stream'])
```

Parameters `required_permissions` – A list of required permissions

Returns bool

is_authenticated ()

Ask facebook if we have access to the users data

Returns bool

me ()

Cached method of requesting information about me

my_image_url (*size='large'*)

Returns the image url from your profile Shortcut for me/picture

Parameters `size` – the type of the image to request, see facebook for available formats

Returns string

permissions ()

Shortcut for self.get('me/permissions') with some extra parsing to turn it into a dictionary of booleans

Returns dict

set (*path, params=None, version=None, **post_data*)

Write data to facebook

Example:

```
open_facebook.set('me/feed', message='testing open facebook')
```

Parameters

- **path** – The path to use for making the API call
- **params** – A dictionary of get params

- **post_data** – The kwargs for posting to facebook

Returns dict

class `open_facebook.api.FacebookAuthorization`

Methods for getting us an access token

There are several flows we must support * js authentication flow (signed cookie) * facebook app authentication flow (signed cookie) * facebook oauth redirect (code param in url) These 3 options need to be converted to an access token

Also handles several testing scenarios * get app access token * create test user * `get_or_create_test_user`

classmethod `convert_code` (*code*, *redirect_uri*=`'http://local.mellowning.com:8000/facebook/connect/'`)

Turns a code into an access token

Example:

```
FacebookAuthorization.convert_code(code)
```

Parameters

- **code** – The code to convert
- **redirect_uri** – The redirect uri with which the code was requested

Returns dict

classmethod `create_test_user` (*app_access_token*, *permissions*=`None`, *name*=`None`)

Creates a test user with the given permissions and name

Parameters

- **app_access_token** – The application's access token
- **permissions** – The list of permissions to request for the test user
- **name** – Optionally specify the name

classmethod `extend_access_token` (*access_token*)

<https://developers.facebook.com/roadmap/offline-access-removal/> We can extend the token only once per day Normal short lived tokens last 1-2 hours Long lived tokens (given by extending) last 60 days

Example:

```
FacebookAuthorization.extend_access_token(access_token)
```

Parameters **access_token** – The access_token to extend

Returns dict

classmethod `get_app_access_token` ()

Get the access_token for the app that can be used for insights and creating test users application_id = retrieved from the developer page application_secret = retrieved from the developer page returns the application access_token

classmethod `get_or_create_test_user` (*app_access_token*, *name*=`None`, *permissions*=`None`, *force_create*=`False`)

There is no supported way of get or creating a test user However - creating a test user takes around 5s - you can only create 500 test users So this slows your testing flow quite a bit.

This method checks your test users Queries their names (stores the permissions in the name)

```
classmethod parse_signed_data (signed_request, secret='0aceba27823a9dfefa955f76949fa4b4')
```

Thanks to <http://stackoverflow.com/questions/3302946/how-to-base64-url-decode-in-python> and <http://sunilarora.org/parsing-signedrequest-parameter-in-python-bas>

```
class open_facebook.api.FacebookConnection
```

Shared utility class implementing the parsing of Facebook API responses

```
classmethod is_server_error (e, response)
```

Checks an HTTPError to see if Facebook is down or we are using the API in the wrong way Facebook doesn't clearly distinguish between the two, so this is a bit of a hack

```
classmethod match_error_code (error_code)
```

Return the right exception class for the error code

```
classmethod raise_error (error_type, message, error_code=None)
```

Lookup the best error class for the error and raise it

Example:

```
FacebookConnection.raise_error(10, 'OAuthException')
```

Parameters

- **error_type** – the error type from the facebook api call
- **message** – the error message from the facebook api call
- **error_code** – optionally the error code which facebook send

```
classmethod request (path='', post_data=None, old_api=False, **params)
```

Main function for sending the request to facebook

Example:: FacebookConnection.request('me')

Parameters

- **path** – The path to request, examples: /me/friends/, /me/likes/
- **post_data** – A dictionary of data to post
- **parms** – The get params to include

Advanced documentation

Mobile usage

You can get an access token by using the native Facebook SDK. Subsequently send this token to your Django based API. In the view you can use the token to get a user.

```
from django_facebook.connect import connect_user
access_token = request.POST['access_token']
action, user = connect_user(request, access_token)
```

Celery, Performance and Optimization

Facebook APIs can take quite some time to respond. It's very common that you will wait between 1-3 seconds for a single API call. If you need multiple calls, pages can quickly become very sluggish.

The recommended solution is to use Celery. Celery is a task queueing system which allows you to run the API requests outside of the request, response cycle.

Step 1 - Install Celery

Step 2 - Enable Tasks

```
# use celery for storing friends and likes
FACEBOOK_CELERY_STORE = True
# use celery for extending tokens
FACEBOOK_CELERY_TOKEN_EXTEND = True
```

When writing your own Facebook functionality you will see a big speedup by using `@facebook_required_lazy` instead of `@facebook_required`

Signals

Django-facebook ships with a few signals that you can use to easily accommodate Facebook related activities with your project.

`facebook_user_registered` signal is sent whenever a new user is registered by Django-facebook, for example:

```
from django_facebook.utils import get_user_model
from django_facebook import signals

def fb_user_registered_handler(sender, user, facebook_data, **kwargs):
    # Do something involving user here

signals.facebook_user_registered.connect(user_registered, sender=get_user_model())
```

`facebook_pre_update` signal is sent just before Django-facebook updates the profile model with Facebook data. If you want to manipulate Facebook or profile information before it gets saved, this is where you should do it. For example:

```
from django_facebook import signals
from django_facebook.utils import get_user_model

def pre_facebook_update(sender, user, profile, facebook_data, **kwargs):
    profile.facebook_information_updated = datetime.datetime.now()
    # Manipulate facebook_data here

signals.facebook_pre_update.connect(pre_facebook_update, sender=get_user_model())
```

`facebook_post_update` signal is sent after Django-facebook finishes updating the profile model with Facebook data. You can perform other Facebook connect or registration related processing here.

```
from django_facebook import signals
from django_facebook.utils import get_user_model

def post_facebook_update(sender, user, profile, facebook_data, **kwargs):
    # Do other stuff

signals.facebook_post_update.connect(post_facebook_update, sender=get_user_model())
```

`facebook_post_store_friends` signal is sent after Django-facebook finishes storing the user's friends.

```
from django_facebook import signals
from django_facebook.utils import get_user_model
```

```
def post_friends(sender, user, friends, current_friends, inserted_friends, **kwargs):
    # Do other stuff

facebook_post_store_friends.connect(post_friends, sender=get_user_model())
```

facebook_post_store_likes signal is sent after Django-facebook finishes storing the user's likes. This is usefull if you want to customize what topics etc to follow.

```
from django_facebook import signals
from django_facebook.utils import get_user_model

def post_likes(sender, user, likes, current_likes, inserted_likes, **kwargs):
    # Do other stuff

facebook_post_store_likes.connect(post_likes, sender=get_user_model())
```

Canvas Application

In order to use build a facebook canvas application, you should add this to your MIDDLEWARE_CLASSES setting:

```
'django_facebook.middleware.FacebookCanvasMiddleWare'
```

This middleware will check for the signed_request parameter in the url and take the appropriate action:

- redirect to app authorization dialog if user has not authorized the app, some permission is missing or any other error.
- login the current facebook user in django's system and store the access token.

API documentation

Django Facebook

API

class django_facebook.api.**FacebookUserConverter** (*open_facebook*)

This conversion class helps you to convert Facebook users to Django users

Helps with - extracting and prepopulating full profile data - invite flows - importing and storing likes

facebook_profile_data ()

Returns the facebook profile data, together with the image locations

facebook_registration_data (*username=True*)

Gets all registration data and ensures its correct input for a django registration

get_and_store_friends (*user*)

Gets and stores your facebook friends to DB Both the get and the store run in a async task when FACEBOOK_CELERY_STORE = True

get_and_store_likes (*user*)

Gets and stores your facebook likes to DB Both the get and the store run in a async task when FACEBOOK_CELERY_STORE = True

get_friends (*limit=5000*)

Connects to the facebook api and gets the users friends

get_likes (*limit=5000*)

Parses the facebook response and returns the likes

registered_friends (*user*)

Returns all profile models which are already registered on your site and a list of friends which are not on your site

store_friends (*user, friends*)

Stores the given friends locally for this user Quite slow, better do this using celery on a secondary db

store_likes (*user, likes*)

Given a user and likes store these in the db Note this can be a heavy operation, best to do it in the background using celery

`django_facebook.api.get_facebook_graph` (*request=None, access_token=None, redirect_uri=None, raise_=False*)

given a request from one of these - js authentication flow (signed cookie) - facebook app authentication flow (signed cookie) - facebook oauth redirect (code param in url) - mobile authentication flow (direct access_token) - offline access token stored in user profile

returns a graph object

redirect path is the path from which you requested the token for some reason facebook needs exactly this uri when converting the code to a token falls back to the current page without code in the request params specify `redirect_uri` if you are not posting and receiving the code on the same page

`django_facebook.api.get_persistent_graph` (*request, *args, **kwargs*)

Wraps itself around get facebook graph But stores the graph in the session, allowing usage across multiple pageviews. Note that Facebook session's expire at some point, you can't store this for permanent usage Atleast not without asking for the `offline_access` permission

`django_facebook.api.require_facebook_graph` (*request, *args, **kwargs*)

Just like `get_facebook_graph`, but instead of returning `None` raise an `OpenFacebookException` if we can't access facebook

`django_facebook.api.require_persistent_graph` (*request, *args, **kwargs*)

Just like `get_persistent_graph`, but instead of returning `None` raise an `OpenFacebookException` if we can't access facebook

AUTH BACKENDS

Connect

`django_facebook.connect.connect_user` (*request, access_token=None, facebook_graph=None, connect_facebook=False*)

Given a request either

- (if authenticated) connect the user
- login
- register

`django_facebook.connect.update_connection` (*request, graph*)

A special purpose view for updating the connection with an existing user - updates the access token (already done in `get_graph`) - sets the `facebook_id` if nothing is specified - stores friends and likes if possible

Decorators

class `django_facebook.decorators.FacebookRequired` (*fn*, *scope=None*, *canvas=False*, *page_tab=False*, *extra_params=None*)

Decorator which makes the view require the given Facebook perms, redirecting to the log-in page if necessary.

Note we don't actually query the permissions, we just try in the view and upon a permission error redirect to `login_url` Querying the permissions would slow down things

authenticate (*fn*, *request*, **args*, ***kwargs*)
Authenticate the user

There are three options a.) We have permissions, proceed with the view b.) We tried getting permissions and failed, abort... c.) We are about to ask for permissions

authentication_failed (*fn*, *request*, **args*, ***kwargs*)
Execute the view but don't pass the graph to indicate we couldn't get the right permissions

get_redirect_uri (*request*)
return the redirect uri to use for oauth authorization this needs to be the same for requesting and accepting the token

oauth_redirect (*oauth_url*, *redirect_uri*, *e=None*)
Redirect to Facebook's oAuth dialog

class `django_facebook.decorators.FacebookRequiredLazy` (*fn*, *scope=None*, *canvas=False*, *page_tab=False*, *extra_params=None*)

Decorator which makes the view require the given Facebook perms, redirecting to the log-in page if necessary.

Based on exceptions instead of a permission check Faster, but more prone to bugs

Use this in combination with `require_persistent_graph`

`django_facebook.decorators.facebook_required` (*fn=None*, **decorator_args*, ***decorator_kwargs*)

Decorator which makes the view require the given Facebook perms, redirecting to the log-in page if necessary.

Note we don't actually query the permissions, we just try in the view and upon a permission error redirect to `login_url` Querying the permissions would slow down things

`django_facebook.decorators.facebook_required_lazy` (*fn=None*, **decorator_args*, ***decorator_kwargs*)

Decorator which makes the view require the given Facebook perms, redirecting to the log-in page if necessary.

Based on exceptions instead of a permission check Faster, but more prone to bugs

Use this in combination with `require_persistent_graph`

Exceptions

exception `django_facebook.exceptions.AlreadyConnectedError` (*users*)

Raised when another user account is already connected to your Facebook id

exception `django_facebook.exceptions.AlreadyRegistered`

Raised if you try to register when there's already an account with the given email or facebook id

exception `django_facebook.exceptions.FacebookException`

Base class for Facebook related exceptions

exception `django_facebook.exceptions.IncompleteProfileError`

Raised when we get insufficient data to create a profile for a user. One example is a Facebook token, without permissions to see the email.

exception `django_facebook.exceptions.MissingPermissionsError`

Raised if we lack permissions

Forms

Forms and validation code for user registration.

```
class django_facebook.forms.FacebookRegistrationFormUniqueEmail (data=None,
                                                                    files=None,
                                                                    auto_id=u'id_%s',
                                                                    prefix=None,
                                                                    initial=None, error_class=<class
                                                                    'django.forms.utils.ErrorList'>,
                                                                    label=
                                                                    label_suffix=None,
                                                                    empty_permitted=False,
                                                                    field_order=None,
                                                                    use_required_attribute=None,
                                                                    renderer=None)
```

Some basic validation, adapted from django registration

clean ()

Verify that the values entered into the two password fields match. Note that an error here will end up in `non_field_errors ()` because it doesn't apply to a single field.

clean_email ()

Validate that the supplied email address is unique for the site.

clean_username ()

Validate that the username is alphanumeric and is not already in use.

Models

Registration backends

class `django_facebook.registration_backends.FacebookRegistrationBackend`

A backend compatible with Django Registration It is extremely simple and doesn't handle things like redirects etc (These are already handled by Django Facebook)

register (*request*, *form=None*, ***kwargs*)

Create and immediately log in a new user.

class `django_facebook.registration_backends.NooptRegistrationBackend`

Noopt backends forms the basis of support for backends which handle the actual registration in the registration form

get_form_class (*request*)

Returns the form class to use for registration

Parameters *request* – the request object

get_registration_template ()

Returns the template to use for registration

- post_connect** (*request, user, action*)
Handles the redirect after connecting
- post_error** (*request, additional_params=None*)
Handles the redirect after connecting
- register** (*request, form=None, **kwargs*)
Implement your registration logic in this method

Parameters

- **request** – the request object
- **form** – the form with the users data
- **kwargs** – additional data

Tasks

Utils

class `django_facebook.utils.ScriptRedirect` (*redirect_to, show_body=True*)
Redirect for Facebook Canvas pages

`django_facebook.utils.cleanup_oauth_url` (*redirect_uri*)
We have to maintain order with respect to the queryparams which is a bit of a pain TODO: Very hacky will subclass QueryDict to SortedQueryDict at some point And use a decent sort function

`django_facebook.utils.clear_persistent_graph_cache` (*request*)
Clears the caches for the graph cache

`django_facebook.utils.error_next_redirect` (*request, default='/', additional_params=None, next_key=None, redirect_url=None, canvas=False*)
Short cut for an error next redirect

`django_facebook.utils.get_class_for` (*purpose*)
Usage: `conversion_class = get_class_for('user_conversion')`

`django_facebook.utils.get_class_from_string` (*path, default=None*)
Return the class specified by the string.

IE: `django.contrib.auth.models.User` Will return the user class or cause an ImportError

`django_facebook.utils.get_django_registration_version` ()
Returns new, old or None depending on the version of django registration Old works with forms New works with backends

`django_facebook.utils.get_form_class` (*backend, request*)
Will use registration form in the following order: 1. User configured RegistrationForm 2. backend.get_form_class(request) from django-registration 0.8 3. RegistrationFormUniqueEmail from django-registration < 0.8

`django_facebook.utils.get_instance_for` (*purpose, *args, **kwargs*)
Usage:

```
conversion_instance = get_instance_for(
    'facebook_user_conversion', user=user)
```

`django_facebook.utils.get_migration_data` ()
Support for Django custom user models See this blog post for inspiration

<http://kevindias.com/writing/django-custom-user-models-south-and-reusable-apps/> https://github.com/stephenmcd/mezzanine/blob/master/mezzanine/core/migrations/0005_auto__chg_field_sitepermission_user__del_unique_sitepermission_user.py

`django_facebook.utils.get_oauth_url` (*scope*, *redirect_uri*, *extra_params=None*)
Returns the OAuth URL for the given scope and redirect_uri

`django_facebook.utils.get_profile` (*user*)
Get profile

`django_facebook.utils.get_profile_from_user` (*user*)
Tries to get the profile according to the class configured on AUTH_PROFILE_MODULE

`django_facebook.utils.get_profile_model` ()
Get the profile model if present otherwise return None

`django_facebook.utils.get_registration_backend` ()
Ensures compatibility with the new and old version of django registration

`django_facebook.utils.get_url_field` ()
This should be compatible with both django 1.3, 1.4 and 1.5 In 1.5 the verify_exists argument is removed and always False

`django_facebook.utils.get_user_model` ()
For Django < 1.5 backward compatibility

`django_facebook.utils.mass_get_or_create` (**args*, ***kwargs*)
Updates the data by inserting all not found records Doesn't delete records if not in the new data

```
example usage >>> model_class = ListItem #the class for which you are doing the insert >>> base_query_set
= ListItem.objects.filter(user=request.user, list=1) #query for retrieving currently stored items >>> id_field
= 'user_id' #the id field on which to check >>> default_dict = {'12': dict(comment='my_new_item'),
'13': dict(comment='super')} #list of default values for inserts >>> global_defaults = dict(user=request.user,
list_id=1) #global defaults
```

`django_facebook.utils.parse_scope` (*scope*)
Turns 'email,user_about_me' or ('email','user_about_me') into a nice consistent ['email','user_about_me']

`django_facebook.utils.parse_signed_request` (*signed_request_string*)
Just here for your convenience, actual logic is in the FacebookAuthorization class

`django_facebook.utils.queryset_iterator` (*queryset*, *chunksizes=1000*, *getfunc=<built-in function getattr>*)

“ Iterate over a Django Queryset ordered by the primary key

This method loads a maximum of chunksize (default: 1000) rows in it's memory at the same time while django normally would load all rows in it's memory. Using the iterator() method only causes it to not preload all the classes.

Note that the implementation of the iterator does not support ordered query sets.

`django_facebook.utils.replication_safe` (*f*)
Usually views which do a POST will require the next page to be read from the master database. (To prevent issues with replication lag).

However certain views like login do not have this issue. They do a post, but don't modify data which you'll show on subsequent pages.

This decorator marks these views as safe. This ensures requests on the next page are allowed to use the slave db

`django_facebook.utils.response_redirect` (*redirect_url*, *script_redirect=False*)
Abstract away canvas redirects

`django_facebook.utils.simplify_class_decorator` (*class_decorator*)

Makes the decorator syntax uniform Regardless if you call the decorator like

Decorator examples:: @decorator or @decorator() or @decorator(staff=True)

Complexity, Python's class based decorators are weird to say the least: <http://www.artima.com/weblogs/viewpost.jsp?thread=240845>

This function makes sure that your decorator class always gets called with

Methods called:

```
__init__(fn, *option_args, *option_kwargs)
__call__()
    return a function which accepts the *args and *kwargs intended
    for fn
```

`django_facebook.utils.to_bool` (*input, default=False*)

Take a request value and turn it into a bool Never raises errors

`django_facebook.utils.to_int` (*input, default=0, exception=(<type 'exceptions.ValueError'>, <type 'exceptions.TypeError'>), regexp=None*)

Convert the given input to an integer or return default

When trying to convert the exceptions given in the exception parameter are automatically caught and the default will be returned.

The regexp parameter allows for a regular expression to find the digits in a string. When True it will automatically match any digit in the string. When a (regexp) object (has a search method) is given, that will be used. When a string is given, re.compile will be run over it first

The last group of the regexp will be used as value

`django_facebook.utils.update_user_attributes` (*user, profile, attributes_dict, save=False*)

Write the attributes either to the user or profile instance

Views

`django_facebook.views.connect` (**args, **kwargs*)

Exception and validation functionality around the _connect view Separated this out from _connect to preserve readability Don't bother reading this code, skip to _connect for the bit you're interested in :)

`django_facebook.views.disconnect` (*request*)

Removes Facebook from the users profile And redirects to the specified next page

Open Facebook

Exceptions

Facebook error classes also see http://fbdevwiki.com/wiki/Error_codes#User_Permission_Errors

exception `open_facebook.exceptions.AliasException`

When you send a request to a non existant url facebook gives this error instead of a 404....

exception `open_facebook.exceptions.FacebookUnreachable`

Timeouts, 500s, SSL errors etc

exception `open_facebook.exceptions.FeedActionLimit`

When you posted too many times from one user account

exception `open_facebook.exceptions.OpenFacebookException`

BaseClass for all open facebook errors

classmethod `codes_list()`

Returns the codes as a list of instructions

classmethod `range()`

Returns for how many codes this Exception, matches with the eventual goal of matching an error to the most specific error class

exception `open_facebook.exceptions.OpenGraphException`

Raised when we get error 3502, representing a problem with facebook open graph data on the page

exception `open_facebook.exceptions.ParameterException`

100-189 190 and up are oauth errors

exception `open_facebook.exceptions.ParseException`

Anything preventing us from parsing the Facebook response

exception `open_facebook.exceptions.PermissionException`

200-300

exception `open_facebook.exceptions.UnknownException`

Raised when facebook themselves don't know what went wrong

`open_facebook.exceptions.convert_unreachable_exception(e, error_format=u'Facebook is unreachable %s')`

Converts an SSLError, HTTPError or URLError into something subclassing FacebookUnreachable allowing code to easily try except this

`open_facebook.exceptions.map_unreachable_exception(e)`

We always raise the original and new subclass to

- preserve backwards compatibility

Utils

`open_facebook.utils.base64_url_decode_php_style(inp)`

PHP follows a slightly different protocol for base64 url decode. For a full explanation see: <http://stackoverflow.com/questions/3302946/how-to-base64-url-decode-in-python> and <http://sunilarora.org/parsing-signedrequest-parameter-in-python-bas>

`open_facebook.utils.camel_to_underscore(name)`

Convert camelcase style naming to underscore style naming

e.g. SpamEggs -> spam_eggs

`open_facebook.utils.encode_params(params_dict)`

Take the dictionary of params and encode keys and values to ascii if it's unicode

`open_facebook.utils.import_statsd()`

Import only the statd by wolph not the mozilla statsd TODO: Move to mozilla statsd which is more widely used

`open_facebook.utils.is_json(content)`

Unfortunately facebook returns 500s which mean they are down Or 500s with a nice error message because you use open graph wrong

So we have to figure out which is which :)

`class open_facebook.utils.memoized` (*func*)

Decorator. Caches a function's return value each time it is called. If called later with the same arguments, the cached value is returned (not reevaluated).

`open_facebook.utils.merge_urls` (*generated_url, human_url*)

merge the `generated_url` with the `human_url` following this rules: params introduced by `generated_url` are kept final params order comes from `generated_url` there's an hack to support things like this `http://url?param¶m=value`

```
>>> gen = "http://mysite.com?p1=a&p2=b&p3=c&p4=d"
>>> hum = "http://mysite.com?p4=D&p3=C&p2=B"
>>> merge_urls(gen, hum)
u'http://mysite.com?p1=a&p2=B&p3=C&p4=D'
```

```
>>> gen = "http://mysite.com?id=a&id_s=b&p_id=d"
>>> hum = "http://mysite.com?id=A&id_s=B&p_id=D"
>>> merge_urls(gen, hum)
u'http://mysite.com?id=A&id_s=B&p_id=D'
```

```
>>> gen = "http://mysite.com?p1=a&p2=b&p3=c&p4=d"
>>> hum = "http://mysite.com"
>>> merge_urls(gen, hum)
u'http://mysite.com'
```

```
>>> gen = "http://ad.zanox.com/ppc/?18595160C2000463397T&zpar4=scrapbook&
↳zpar0=e2494344_c4385641&zpar1=not_authenticated&zpar2=unknown_campaign&
↳zpar3=unknown_ref&ULP=http://www.asos.com/ASOS/ASOS-MARS-Loafer-Shoes/Prod/
↳pgeproduct.aspx?iid=1703516&cid=4172&sh=0&pge=2&pgesize=20&sort=-1&clr=Black&
↳affId=2441"
>>> hum = "http://ad.zanox.com/ppc/?18595160C2000463397T&zpar3=scrapbook&
↳ULP=http://www.asos.com/ASOS/ASOS-MARS-Loafer-Shoes/Prod/pgeproduct.aspx?
↳iid=1703516&cid=4172&sh=0&pge=2&pgesize=20&sort=-1&clr=Black&affId=2441"
>>> merge_urls(gen, hum)
u'http://ad.zanox.com/ppc/?18595160C2000463397T&zpar4=scrapbook&zpar0=e2494344_
↳c4385641&zpar1=not_authenticated&zpar2=unknown_campaign&zpar3=scrapbook&
↳ULP=http://www.asos.com/ASOS/ASOS-MARS-Loafer-Shoes/Prod/pgeproduct.aspx?
↳iid=1703516&cid=4172&sh=0&pge=2&pgesize=20&sort=-1&clr=Black&affId=2441'
```

```
>>> gen = "http://mysite.com?invalidparam&p=2"
>>> hum = "http://mysite.com?p=1"
>>> merge_urls(gen, hum)
u'http://mysite.com?invalidparam&p=1'
```

`open_facebook.utils.send_warning` (*message, request=None, e=None, **extra_data*)

Uses the logging system to send a message to logging and sentry

`open_facebook.utils.smart_str` (*s, encoding='utf-8', strings_only=False, errors='strict'*)

Adapted from django, needed for urlencoding Returns a bytestring version of 's', encoded as specified in 'encoding'. If `strings_only` is True, don't convert (some) non-string-like objects.

`open_facebook.utils.start_statsd` (*path*)

Simple wrapper to save some typing

`open_facebook.utils.validate_is_instance` (*instance, classes*)

Usage `validate_is_instance(10, int)` `validate_is_instance('a', (str, unicode))`

Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

d

`django_facebook.api`, 24
`django_facebook.connect`, 25
`django_facebook.decorators`, 26
`django_facebook.exceptions`, 26
`django_facebook.forms`, 27
`django_facebook.registration_backends`,
27
`django_facebook.settings`, 15
`django_facebook.utils`, 28
`django_facebook.views`, 30

O

`open_facebook.api`, 17
`open_facebook.exceptions`, 30
`open_facebook.utils`, 31

A

AliasException, 30
 AlreadyConnectedError, 26
 AlreadyRegistered, 26
 authenticate() (django_facebook.decorators.FacebookRequired decorator method), 26
 authentication_failed() (django_facebook.decorators.FacebookRequired decorator method), 26

B

base64_url_decode_php_style() (in module open_facebook.utils), 31
 batch_fql() (open_facebook.api.OpenFacebook method), 19

C

camel_to_underscore() (in module open_facebook.utils), 31
 clean() (django_facebook.forms.FacebookRegistrationFormUniqueEmail method), 27
 clean_email() (django_facebook.forms.FacebookRegistrationFormUniqueEmail method), 27
 clean_username() (django_facebook.forms.FacebookRegistrationFormUniqueEmail method), 27
 cleanup_oauth_url() (in module django_facebook.utils), 28
 clear_persistent_graph_cache() (in module django_facebook.utils), 28
 codes_list() (open_facebook.exceptions.OpenFacebookException class method), 31
 connect() (in module django_facebook.views), 30
 connect_user() (in module django_facebook.connect), 25
 convert_code() (open_facebook.api.FacebookAuthorization class method), 21
 convert_unreachable_exception() (in module open_facebook.exceptions), 31
 create_test_user() (open_facebook.api.FacebookAuthorization class method), 21

D

delete() (open_facebook.api.OpenFacebook method), 19
 disconnect() (in module django_facebook.views), 30
 django_facebook.api (module), 24
 django_facebook.connect (module), 25
 django_facebook.decorators (module), 26
 django_facebook.exceptions (module), 26
 django_facebook.forms (module), 27
 django_facebook.registration_backends (module), 27
 django_facebook.settings (module), 15
 django_facebook.utils (module), 28
 django_facebook.views (module), 30

E

encode_params() (in module open_facebook.utils), 31
 error_next_redirect() (in module django_facebook.utils), 28
 extend_access_token() (open_facebook.api.FacebookAuthorization class method), 21

F

facebook_profile_data() (django_facebook.api.FacebookUserConverter method), 24
 facebook_registration_data() (django_facebook.api.FacebookUserConverter method), 24
 facebook_required() (in module django_facebook.decorators), 26
 facebook_required_lazy() (in module django_facebook.decorators), 26
 FacebookAuthorization (class in open_facebook.api), 21
 FacebookConnection (class in open_facebook.api), 22
 FacebookException, 26
 FacebookRegistrationBackend (class in django_facebook.registration_backends), 27
 FacebookRegistrationFormUniqueEmail (class in django_facebook.forms), 27

FacebookRequired (class in django_facebook.decorators), 26
 FacebookRequiredLazy (class in django_facebook.decorators), 26
 FacebookUnreachable, 30
 FacebookUserConverter (class in django_facebook.api), 24
 FeedActionLimit, 30
 fql() (open_facebook.api.OpenFacebook method), 19

G

get() (open_facebook.api.OpenFacebook method), 19
 get_and_store_friends() (django_facebook.api.FacebookUserConverter method), 24
 get_and_store_likes() (django_facebook.api.FacebookUserConverter method), 24
 get_app_access_token() (open_facebook.api.FacebookAuthorization class method), 21
 get_class_for() (in module django_facebook.utils), 28
 get_class_from_string() (in module django_facebook.utils), 28
 get_django_registration_version() (in module django_facebook.utils), 28
 get_facebook_graph() (in module django_facebook.api), 25
 get_form_class() (django_facebook.registration_backends.NooptRegistrationBackend class method), 27
 get_form_class() (in module django_facebook.utils), 28
 get_friends() (django_facebook.api.FacebookUserConverter method), 24
 get_instance_for() (in module django_facebook.utils), 28
 get_likes() (django_facebook.api.FacebookUserConverter method), 25
 get_many() (open_facebook.api.OpenFacebook method), 20
 get_migration_data() (in module django_facebook.utils), 28
 get_oauth_url() (in module django_facebook.utils), 29
 get_or_create_test_user() (open_facebook.api.FacebookAuthorization class method), 21
 get_persistent_graph() (in module django_facebook.api), 25
 get_profile() (in module django_facebook.utils), 29
 get_profile_from_user() (in module django_facebook.utils), 29
 get_profile_model() (in module django_facebook.utils), 29
 get_redirect_uri() (django_facebook.decorators.FacebookRequired method), 26
 get_registration_backend() (in module django_facebook.utils), 29
 get_registration_template() (django_facebook.registration_backends.NooptRegistrationBackend

method), 27
 get_request_url() (open_facebook.api.OpenFacebook method), 20
 get_url_field() (in module django_facebook.utils), 29
 get_user_model() (in module django_facebook.utils), 29

H

has_permissions() (open_facebook.api.OpenFacebook method), 20

I

import_statsd() (in module open_facebook.utils), 31
 IncompleteProfileError, 26
 is_authenticated() (open_facebook.api.OpenFacebook Converter method), 20
 is_json() (in module open_facebook.utils), 31
 is_server_error() (open_facebook.api.FacebookConnection class method), 22

M

map_unreachable_exception() (in module open_facebook.exceptions), 31
 mass_get_or_create() (in module django_facebook.utils), 29
 match_error_code() (open_facebook.api.FacebookConnection class method), 22
 me() (open_facebook.api.OpenFacebook method), 20
 memoized (class in open_facebook.utils), 31
 merge_urls() (in module open_facebook.utils), 32
 MissingPermissionsError, 27
 my_image_url() (open_facebook.api.OpenFacebook method), 20

N

NooptRegistrationBackend (class in django_facebook.registration_backends), 27

O

oauth_redirect() (django_facebook.decorators.FacebookRequired method), 26
 open_facebook.api (module), 17
 open_facebook.exceptions (module), 30
 open_facebook.utils (module), 31
 OpenFacebook (class in open_facebook.api), 18
 OpenFacebookException, 30
 OpenGraphException, 31
 OpenGraphException, 31
 ParameterException, 31
 parse_scope() (in module django_facebook.utils), 29
 parse_signed_data() (open_facebook.api.FacebookAuthorization class method), 21

parse_signed_request() (in module django_facebook.utils), 29
 ParseException, 31
 PermissionException, 31
 permissions() (open_facebook.api.OpenFacebook method), 20
 post_connect() (django_facebook.registration_backends.NooptRegistrationBackend method), 27
 post_error() (django_facebook.registration_backends.NooptRegistrationBackend method), 28

V

validate_is_instance() (in module open_facebook.utils), 32

Q

queryset_iterator() (in module django_facebook.utils), 29

R

raise_error() (open_facebook.api.FacebookConnection class method), 22
 range() (open_facebook.exceptions.OpenFacebookException class method), 31
 register() (django_facebook.registration_backends.FacebookRegistrationBackend method), 27
 register() (django_facebook.registration_backends.NooptRegistrationBackend method), 28
 registered_friends() (django_facebook.api.FacebookUserConverter method), 25
 replication_safe() (in module django_facebook.utils), 29
 request() (open_facebook.api.FacebookConnection class method), 22
 require_facebook_graph() (in module django_facebook.api), 25
 require_persistent_graph() (in module django_facebook.api), 25
 response_redirect() (in module django_facebook.utils), 29

S

ScriptRedirect (class in django_facebook.utils), 28
 send_warning() (in module open_facebook.utils), 32
 set() (open_facebook.api.OpenFacebook method), 20
 simplify_class_decorator() (in module django_facebook.utils), 29
 smart_str() (in module open_facebook.utils), 32
 start_statsd() (in module open_facebook.utils), 32
 store_friends() (django_facebook.api.FacebookUserConverter method), 25
 store_likes() (django_facebook.api.FacebookUserConverter method), 25

T

to_bool() (in module django_facebook.utils), 30
 to_int() (in module django_facebook.utils), 30

U

UnknownException, 31