

---

# Flask-Blogging Documentation

*Release 0.1.0*

**Gouthaman Balaraman**

June 22, 2015



<b>1</b>	<b>Quick Start Example</b>	<b>3</b>
<b>2</b>	<b>Configuring your Application</b>	<b>5</b>
<b>3</b>	<b>Blog Views</b>	<b>7</b>
<b>4</b>	<b>Screenshots</b>	<b>9</b>
4.1	Blog Page . . . . .	10
4.2	Blog Editor . . . . .	10
<b>5</b>	<b>API Documentation</b>	<b>11</b>
5.1	Module contents . . . . .	11
5.2	Submodules . . . . .	11
5.3	flask_blogging.engine module . . . . .	11
5.4	flask_blogging.processor module . . . . .	12
5.5	flask_blogging.sqlastorage module . . . . .	12
5.6	flask_blogging.storage module . . . . .	14
5.7	flask_blogging.views module . . . . .	15
5.8	flask_blogging.forms module . . . . .	15
	<b>Python Module Index</b>	<b>17</b>



Flask-Blogging is a Flask extension for adding blog support to your site. It provides a flexible mechanism to store the data in the database of your choice. It is meant to work with the authentication provided by packages such as [Flask-Login](#) or [Flask-Security](#).

The philosophy behind this extension is to provide a lean app based on markdown to provide blog support to your existing web application. This is contrary to some other packages such as that are just blogs. If you already have a web app and you need to have a blog to communicate with your user or to promote your site through content based marketing.

Out of the box Flask-Blogging has support for the following:

- Bootstrap based site
- Markdown based blog editor
- Models to store blog
- Authentication of User's choice
- Sitemap, ATOM support
- Disqus support for comments
- Google analytics for usage tracking
- Well documented, tested, and extensible design

- *Quick Start Example*
- *Configuring your Application*
- *Blog Views*
- *Screenshots*
  - *Blog Page*
  - *Blog Editor*
- *API Documentation*
  - *Module contents*
  - *Submodules*
  - *flask\_blogging.engine module*
  - *flask\_blogging.processor module*
  - *flask\_blogging.sqlastorage module*
  - *flask\_blogging.storage module*
  - *flask\_blogging.views module*
  - *flask\_blogging.forms module*



---

## Quick Start Example

---

```
from flask import Flask, render_template_string, redirect
from sqlalchemy import create_engine
from flask.ext.login import UserMixin, LoginManager, \
    login_user, logout_user
from flask_blogging import SQLAlchemyStorage, BloggingEngine

app = Flask(__name__)
app.config["SECRET_KEY"] = "secret"  # for WTF-forms and login

# extensions
engine = create_engine('sqlite:///tmp/blog.db')
sql_storage = SQLAlchemyStorage(engine)
blog_engine = BloggingEngine(app, sql_storage, url_prefix="/blog")
login_manager = LoginManager(app)

# user class for providing authentication
class User(UserMixin):
    def __init__(self, user_id):
        self.id = user_id

    def get_name(self):
        return "Paul Dirac"  # typically the user's name

@login_manager.user_loader
@blog_engine.user_loader
def load_user(user_id):
    return User(user_id)

index_template = """
<!DOCTYPE html>
<html>
  <head> </head>
  <body>
    {% if current_user.is_authenticated() %}
      <a href="/logout/">Logout</a>
    {% else %}
      <a href="/login/">Login</a>
    {% endif %}
    &nbsp;&nbsp;<a href="/blog/">Blog</a>
    &nbsp;&nbsp;<a href="/blog/sitemap.xml">Sitemap</a>
    &nbsp;&nbsp;<a href="/blog/feeds/all.atom.xml">ATOM</a>
  </body>
</html>
```

```
"""

@app.route("/")
def index():
    return render_template_string(index_template)

@app.route("/login/")
def login():
    user = User("testuser")
    login_user(user)
    return redirect("/blog")

@app.route("/logout/")
def logout():
    logout_user()
    return redirect("/")

if __name__ == "__main__":
    app.run(debug=True, port=8000, use_reloader=True)
```

The key components required to get the blog hooked is explained below.



---

## Configuring your Application

---

The *BloggngEngine* class is the gateway to configure blogging support to your web app. You should create the *BloggngEngine* instance like this:

```
blogging_engine = BloggngEngine()
```

You also need to pick the storage for blog. That can be done as:

```
from sqlalchemy import create_engine

engine = create_engine("sqlite:///tmp/sqlite.db")
storage = SQLAStorage(engine)
```

Once you have created the blogging engine and the storage, you can connect with your app using the *init\_app* method as shown below:

```
blogging_engine.init_app(app, storage)
```

*Flask-Blogging* lets the developer pick the authentication that is suitable, and hence requires her to provide a way to load user information. You will need to provide a *BloggngEngine.user\_loader* callback. This callback is used to load the user from the *user\_id* that is stored for each blog post. Just as in Flask-Login, it should take the *unicode user\_id* of a user, and return the corresponding user object. For example:

```
@blogging_engine.user_loader
def load_user(userid):
    return User.get(userid)
```

For the blog to have a readable display name, the *User* class must implement either the *get\_name* method or the *\_\_str\_\_* method.

The *BloggngEngine* accepts an optional *config dict* argument which is passed to all the views. The keys that are currently supported include:

SITENAME	The name of the blog to be used as the brand name (default “Flask-Blogging”)
SITEURL	The url of the site.
RENDER_TEXT	Boolean value to specify if the raw text should be rendered or not. (default True)
DISQUS_SITENAME	Disqus sitename for comments (default None)
GOOGLE_ANALYTICS	Google analytics code for usage tracking (default None)

The *BloggngEngine* accepts an optional *extensions* argument. This is a list of Markdown extensions objects to be used during the markdown processing step.



---

## Blog Views

---

There are various views that are exposed through Flask-Blogging. If the `url_prefix` argument in the `BloggingEngine` is `/blog`, then the URL for the various views are:

- `/blog/` (GET): The index blog posts with the first page of articles.
- `/blog/page/<post_id>/<optional slug>/` (GET): The blog post corresponding to the `post_id` is retrieved.
- `/blog/tag/<tag_name>/` (GET): The list of blog posts corresponding to `tag_name` is returned.
- `/blog/author/<user_id>/` (GET): The list of blog posts written by the author `user_id` is returned.
- `/blog/editor/` (GET, POST): The blog editor is shown. This view needs authentication.
- `/blog/delete/<post_id>/` (POST): The blog post given by `post_id` is deleted. This view needs authentication.
- `/blog/sitemap.xml` (GET): The sitemap with a link to all the posts is returned.

The view can be easily customised by the user by overriding with their own templates. The template pages that need to be customized are:




- `blog/index.html`: The blog index page used to serve index of posts, posts by tag, and posts by author
- `blog/editor.html`: The blog editor page.
- `blog/page.html`: The page that shows the given article.
- `blog/sitemap.xml`: The sitemap for the blog posts.





## Screenshots

## 4.1 Blog Page

 Delete
  Edit
  New


## Dirac Equation

Posted by [Paul Dirac](#) on 03 Jun, 2015


In particle physics, the Dirac equation is a relativistic wave equation derived by British physicist Paul Dirac in 1928. In its free form, or including electromagnetic interactions, it describes all spin-1/2 massive particles, for which parity is a symmetry, such as electrons and quarks, and is consistent with both the principles of quantum mechanics and the theory of special relativity,[1] and was the first theory to account fully for special relativity in the context of quantum mechanics.



Dirac's Equation is given as:


$$(\beta mc^2 + c(\alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3)) \psi(x, t) = i\hbar \frac{\partial \psi(x, t)}{\partial t}$$

 PHYSICS

---

0 Comments
 test
  Gouthaman Balar...

 Recommend
  Share
 Sort by Best



Be the first to comment.

## 4.2 Blog Editor

Title

Dirac Equation

---

## API Documentation

---

### 5.1 Module contents

Flask-Blogging is a Flask extension to add blog support to your web application. This extension uses Markdown to store and then render the webpage.

Author: Gouthaman Balaraman

Date: June 1, 2015

### 5.2 Submodules

### 5.3 flask\_blogging.engine module

The BloggingEngine module.

```
class flask_blogging.engine.BloggingEngine(app=None, storage=None, url_prefix=None,  
post_processor=None, config=None, extensions=None)
```

Bases: object

The BloggingEngine is the class for initializing the blog support for your web app. Here is an example usage:

```
from flask import Flask
from flask.ext.blogging import BloggingEngine, SQLAlchemyStorage
from sqlalchemy import create_engine

app = Flask(__name__)
db_engine = create_engine("sqlite:///tmp/sqlite.db")
storage = SQLAlchemyStorage(db_engine)
blog_engine = BloggingEngine(app, storage)
```

```
__init__(app=None, storage=None, url_prefix=None, post_processor=None, config=None, extensions=None)
```

#### Parameters

- **app** (*object*) – Optional app to use
- **storage** (*object*) – The blog storage instance that implements the `Storage` class interface.
- **url\_prefix** (*str*) – (optional) The prefix for the URL of blog posts (default `None`)

- **post\_processor** (*object*) – (optional) The post processor object. If none provided, the default is used.
- **config** (*dict*) – (optional) A dictionary of config values. See docs for the keys that can be specified.
- **extensions** (*list*) – A list of markdown extensions to add to post processing step.

#### Returns

**init\_app** (*app*, *storage*)  
Initialize the engine.

#### Parameters

- **app** – The app to use
- **storage** – The blog storage instance that implements the `Storage` class interface.

**user\_loader** (*callback*)  
The decorator for loading the user.

**Parameters** **callback** – The callback function that can load a user given a unicode `user_id`.

**Returns** The callback function

## 5.4 flask\_blogging.processor module

**class** `flask_blogging.processor.PostProcessor`

Bases: `object`

**classmethod** `all_extensions()`

**classmethod** `construct_url(post)`

**static** `create_slug(title)`

**classmethod** `custom_process(post)`

Override this method to add additional processes. The result is that the `post` dict is modified or enhanced with newer key value pairs.

**Parameters** **post** (*dict*) – The post data with values for keys such as title, text, tags etc.

**classmethod** `process(post, render=True)`

This method takes the post data and renders it :param post: :param render: :return:

**classmethod** `render_text(post)`

**classmethod** `set_custom_extensions(extensions)`

## 5.5 flask\_blogging.sqlastorage module

**class** `flask_blogging.sqlastorage.SQLAStorage(engine, table_prefix='')`

Bases: `flask_blogging.storage.Storage`

The `SQLAStorage` implements the interface specified by the `Storage` class. This class uses `SQLAlchemy` to implement storage and retrieval of data from any of the databases supported by `SQLAlchemy`. This

**\_\_init\_\_** (*engine*, *table\_prefix*='')

The constructor for the `SQLAStorage` class.



### Parameters

- **engine** (*object*) – The SQLAlchemy engine instance created by calling `create_engine`. One can also use Flask-SQLAlchemy, and pass the engine property.
- **table\_prefix** (*str*) – (Optional) Prefix to use for the tables created (default "").

**count\_posts** (*tag=None, user\_id=None, include\_draft=False*)

Returns the total number of posts for the give filter

### Parameters

- **tag** (*str*) – Filter by a specific tag
- **user\_id** (*str*) – Filter by a specific user
- **include\_draft** (*bool*) – Whether to include posts marked as draft or not

**Returns** The number of posts for the given filter.

**delete\_post** (*post\_id*)

Delete the post defined by `post_id`

**Parameters** **post\_id** (*int*) – The identifier corresponding to a post

**Returns** Returns True if the post was successfully deleted and False otherwise.

**get\_post\_by\_id** (*post\_id*)

Fetch the blog post given by `post_id`

**Parameters** **post\_id** (*int*) – The post identifier for the blog post

**Returns** If the `post_id` is valid, the post data is retrieved, else returns None.

**get\_posts** (*count=10, offset=0, recent=True, tag=None, user\_id=None, include\_draft=False*)

Get posts given by filter criteria

### Parameters

- **count** (*int*) – The number of posts to retrieve (default 10)
- **offset** (*int*) – The number of posts to offset (default 0)
- **recent** (*bool*) – Order by recent posts or not
- **tag** (*str*) – Filter by a specific tag
- **user\_id** (*str*) – Filter by a specific user
- **include\_draft** (*bool*) – Whether to include posts marked as draft or not

**Returns** A list of posts, with each element a dict containing values for the following keys: (title, text, draft, post\_date, last\_modified\_date). If count is None, then all the posts are returned.

**save\_post** (*title, text, user\_id, tags, draft=False, post\_id=None*)

Persist the blog post data. If `post_id` is None or `post_id` is invalid, the post must be inserted into the storage. If `post_id` is a valid id, then the data must be updated.

### Parameters

- **title** (*str*) – The title of the blog post
- **text** (*str*) – The text of the blog post
- **user\_id** (*str*) – The user identifier
- **tags** (*list*) – A list of tags

- **draft** (*bool*) – (Optional) If the post is a draft or if needs to be published. (default `False`)
- **post\_id** (*int*) – (Optional) The post identifier. This should be `None` for an insert call, and a valid value for update. (default `None`)

**Returns** The `post_id` value, in case of a successful insert or update. Return `None` if there were errors.

## 5.6 flask\_blogging.storage module

**class** flask\_blogging.storage.Storage

Bases: object

**count\_posts** (*tag=None, user\_id=None, include\_draft=False*)

Returns the total number of posts for the give filter

### Parameters

- **tag** (*str*) – Filter by a specific tag
- **user\_id** (*str*) – Filter by a specific user
- **include\_draft** (*bool*) – Whether to include posts marked as draft or not

**Returns** The number of posts for the given filter.

**delete\_post** (*post\_id*)

Delete the post defined by `post_id`

**Parameters** **post\_id** (*int*) – The identifier corresponding to a post

**Returns** Returns `True` if the post was successfully deleted and `False` otherwise.

**get\_post\_by\_id** (*post\_id*)

Fetch the blog post given by `post_id`

**Parameters** **post\_id** (*int*) – The post identifier for the blog post

**Returns** If the `post_id` is valid, the post data is retrieved, else returns `None`.

**get\_posts** (*count=10, offset=0, recent=True, tag=None, user\_id=None, include\_draft=False*)

Get posts given by filter criteria

### Parameters

- **count** (*int*) – The number of posts to retrieve (default 10). If count is `None`, all posts are returned.
- **offset** (*int*) – The number of posts to offset (default 0)
- **recent** (*bool*) – Order by recent posts or not
- **tag** (*str*) – Filter by a specific tag
- **user\_id** (*str*) – Filter by a specific user
- **include\_draft** (*bool*) – Whether to include posts marked as draft or not

**Returns** A list of posts, with each element a dict containing values for the following keys: (title, text, draft, post\_date, last\_modified\_date). If count is `None`, then all the posts are returned.

**static normalize\_tags** (*tags*)

**save\_post** (*title, text, user\_id, tags, draft=False, post\_id=None*)

Persist the blog post data. If `post_id` is `None` or `post_id` is invalid, the post must be inserted into the storage. If `post_id` is a valid id, then the data must be updated.

#### Parameters

- **title** (*str*) – The title of the blog post
- **text** (*str*) – The text of the blog post
- **user\_id** (*str*) – The user identifier
- **tags** (*list*) – A list of tags
- **draft** (*bool*) – If the post is a draft or if needs to be published.
- **post\_id** (*int*) – The post identifier. This should be `None` for an insert call, and a valid value for update.

**Returns** The `post_id` value, in case of a successful insert or update. Return `None` if there were errors.

## 5.7 flask\_blogging.views module

`flask_blogging.views.delete(*args, **kwargs)`

`flask_blogging.views.editor(*args, **kwargs)`

`flask_blogging.views.index(count, page)`

Serves the page with a list of blog posts

#### Parameters

- **count** –
- **offset** –

#### Returns

`flask_blogging.views.page_by_id(post_id, slug)`

`flask_blogging.views.posts_by_author(user_id, count, page)`

`flask_blogging.views.posts_by_tag(tag, count, page)`

`flask_blogging.views.recent_feed()`

`flask_blogging.views.sitemap()`

## 5.8 flask\_blogging.forms module

```
class flask_blogging.forms.BlogEditor(formdata=<class flask_wtf.form._Auto>, obj=None,
                                     prefix='', csrf_context=None, secret_key=None,
                                     csrf_enabled=None, *args, **kwargs)
```

```
    draft = <UnboundField(BooleanField, ('draft'), {'default': False})>
```

```
    submit = <UnboundField(SubmitField, ('submit'), {})>
```

```
    tags = <UnboundField(StringField, ('tags'), {'validators': [<wtforms.validators.DataRequired object at 0x7fe37fd5ac90>]})>
```

```
    text = <UnboundField(TextAreaField, ('text'), {'validators': [<wtforms.validators.DataRequired object at 0x7fe37fd5ac90>]})>
```

```
title = <UnboundField(StringField, ('title'), {'validators': [<wtforms.validators.DataRequired object at 0x7fe37fd5ab9
```

## **f**

- `flask_bloggin`, [11](#)
- `flask_bloggin.engine`, [11](#)
- `flask_bloggin.forms`, [15](#)
- `flask_bloggin.sqlastorage`, [12](#)
- `flask_bloggin.storage`, [14](#)
- `flask_bloggin.views`, [15](#)



## Symbols

`__init__()` (flask\_blogging.engine.BloggingEngine method), 11  
`__init__()` (flask\_blogging.sqlastorage.SQLAStorage method), 12

## A

`all_extensions()` (flask\_blogging.processor.PostProcessor class method), 12

## B

`BlogEditor` (class in flask\_blogging.forms), 15  
`BloggingEngine` (class in flask\_blogging.engine), 11

## C

`construct_url()` (flask\_blogging.processor.PostProcessor class method), 12  
`count_posts()` (flask\_blogging.sqlastorage.SQLAStorage method), 13  
`count_posts()` (flask\_blogging.storage.Storage method), 14  
`create_slug()` (flask\_blogging.processor.PostProcessor static method), 12  
`custom_process()` (flask\_blogging.processor.PostProcessor class method), 12

## D

`delete()` (in module flask\_blogging.views), 15  
`delete_post()` (flask\_blogging.sqlastorage.SQLAStorage method), 13  
`delete_post()` (flask\_blogging.storage.Storage method), 14  
`draft` (flask\_blogging.forms.BlogEditor attribute), 15

## E

`editor()` (in module flask\_blogging.views), 15

## F

`flask_blogging` (module), 11  
`flask_blogging.engine` (module), 11

`flask_blogging.forms` (module), 15  
`flask_blogging.sqlastorage` (module), 12  
`flask_blogging.storage` (module), 14  
`flask_blogging.views` (module), 15

## G

`get_post_by_id()` (flask\_blogging.sqlastorage.SQLAStorage method), 13  
`get_post_by_id()` (flask\_blogging.storage.Storage method), 14  
`get_posts()` (flask\_blogging.sqlastorage.SQLAStorage method), 13  
`get_posts()` (flask\_blogging.storage.Storage method), 14

## I

`index()` (in module flask\_blogging.views), 15  
`init_app()` (flask\_blogging.engine.BloggingEngine method), 12

## N

`normalize_tags()` (flask\_blogging.storage.Storage static method), 14

## P

`page_by_id()` (in module flask\_blogging.views), 15  
`PostProcessor` (class in flask\_blogging.processor), 12  
`posts_by_author()` (in module flask\_blogging.views), 15  
`posts_by_tag()` (in module flask\_blogging.views), 15  
`process()` (flask\_blogging.processor.PostProcessor class method), 12

## R

`recent_feed()` (in module flask\_blogging.views), 15  
`render_text()` (flask\_blogging.processor.PostProcessor class method), 12

## S

`save_post()` (flask\_blogging.sqlastorage.SQLAStorage method), 13  
`save_post()` (flask\_blogging.storage.Storage method), 14

`set_custom_extensions()` (`flask_blogging.processor.PostProcessor`  
class method), [12](#)

`sitemap()` (in module `flask_blogging.views`), [15](#)

`SQLAStorage` (class in `flask_blogging.sqlastorage`), [12](#)

`Storage` (class in `flask_blogging.storage`), [14](#)

`submit` (`flask_blogging.forms.BlogEditor` attribute), [15](#)

## T

`tags` (`flask_blogging.forms.BlogEditor` attribute), [15](#)

`text` (`flask_blogging.forms.BlogEditor` attribute), [15](#)

`title` (`flask_blogging.forms.BlogEditor` attribute), [15](#)

## U

`user_loader()` (`flask_blogging.engine.BloggingEngine`  
method), [12](#)