django-feed-reader

Release 2.0.0

Gareth Simpson

May 11, 2024

CONTENTS:

1 Django Feed Reader **1**

1.1 Features . . . 1
1.2 Installation . . . 1
1.3 Basic Models . . . 2
1.4 Refreshing feeds . . . 2
1.5 Polling with cron . . . 3
1.6 Polling with celery . . . 3
1.7 Tracking read/unread state of feeds . . . 3
1.8 Cloudflare Busting . . . 4

2 Commands **5**

3 Models **7**

4 Utils **13**

5 Indices and tables **15**

Index **17**

CHAPTER

ONE

DJANGO FEED READER

This is a simple Django module to allow you subscribe to RSS (and other) feeds.

This app has no UI, it just reads and stores the feeds for you to use as you see fit.

This app builds on top of the FeedParser library to provide feed management, storage, scheduling etc.

1.1 Features

- Consumes RSS, Atom and JSONFeed feeds.
- Parses feeds liberally to try and accomodate simple errors.
- Will attempt to bypass Cloudflare protection of feeds
- Supports enclosure (podcast) discovery
- Automatic feed scheduling based on frequency of updates

1.2 Installation

`django-feed-reader` is written in Python 3 and supports Django 2.2+

- `pip install django-feed-reader`
- Add `feeds` to your `INSTALLED_APPS`
- **Setup some values in `settings.py` so that your feed reader politely announces itself to servers**
  - Set `FEEDS_USER_AGENT` to the name and (optionally version) of your service e.g. `"ExampleFeeder/1.2"`
  - Set `FEEDS_SERVER` to preferred web address of your service so that feed hosts can locate you if required e.g. `https://example.com`
- Setup a mechanism to periodically refresh the feeds (see below)

1.2.1 Optional Settings

- **`FEEDS_VERIFY_HTTPS` (Default True)**
  - Older versions of this library did not verify https connections when fetching feeds. Set this to `False` to revert to the old behaviour.
- **`KEEP_OLD_ENCLOSURES` (Default False)**
  - Some feeds (particularly podcasts with Dynamic Ad Insertion) will change their enclosure urls between reads. By default, old enclosures are deleted and replaced with new ones. Set this to true, to retain old enclosures - they will have their `is_current` flag set to `False`
- **`SAVE_JSON` (Default False)**
  - If set, Sources and Posts will store a JSON representation of the all the data retrieved from the feed so that uncommon or custom attributes can be retrieved. Caution - this will dramatically increase tha amount of space used in your database.
- **`DRIPFEED_KEY` (Default None)**
  - If set to a valid Dripfeed API Key, then feeds that are blocked by Cloudflare will be automatically polled via Dripfeed instead.

1.3 Basic Models

A feed is represented by a `Source` object which has (among other things) a `feed_url`.

To start reading a feed, simply create a new `Source` with the desired `feed_url`

`Source` objects have `Post` children which contain the content.

A `Post` may have `Enclosure` (or more) which is what podcasts use to send their audio. The app does not download enclosures, if you want to do that you will need to do that in your project using the url provided.

1.4 Refreshing feeds

To conserve resources with large feed lists, the module will adjust how often it polls feeds based on how often they are updated. The fastest it will poll a feed is every hour. The slowest it will poll is every 24 hours.

Sources that don't get updated are polled progressively more slowly until the 24 hour limit is reached. When a feed changes, its polling frequency increases.

You will need to decided how and when to run the poller. When the poller runs, it checks all feeds that are currently due. The ideal frequency to run it is every 5 - 10 minutes.

1.5 Polling with cron

Set up a job that calls `python manage.py refreshfeeds` on your desired schedule.

Be careful to ensure you're running out of the correct directory and with the correct python environment.

1.6 Polling with celery

Create a new celery task and schedule in your app (see the celery documentation for details). Your `tasks.py` should look something like this:

```
from celery import shared_task
from feeds.utils import update_feeds

@shared_task
def get_those_feeds():

  # the number is the max number of feeds to poll in one go
  update_feeds(30)
```

1.7 Tracking read/unread state of feeds

There are two ways to track the read/unread state of feeds depending on your needs.

1.7.1 Single User Installations

If your usage is just for a single user, then there are helper methods on a Source to track your read state.

All posts come in unread. You can get the current number of unread posts from `Source.unread_count`.

To get a ResultSet of all the unread posts from a feed call `Source.get_unread_posts`

To mark all posts on a fed as read call `Source.mark_read`

To get all of the posts in a feed regardless of read status, a page at a time call `Source.get_paginated_posts` which returns a tuple of (Posts, Paginator)

1.7.2 Multi-User Installations

To allow multiple users to follow the same feed with individual read/unread status, create a new `Subscription` for that Source and User.

Subscription has the same helper methods for retrieving posts and marking read as Source.

You can also arrange feeds into a folder-like hierarchy using Subscriptions. Every Subscription has an optional `parent`. Subscriptions with a `None` parent are considered at the root level. By convention, Subscriptions that are acting as parent folders should have a `None source`

Subscriptions have a `name` field which by convention should be a display name if it is a folder or the name of the Source it is tracking. However this can be set to any value if you want to give a personally-meaningful name to a feed who's name is cryptic.

There are two helper methods in the `utils` module to help manage subscriptions as folders. `get_subscription_list_for_user` will return all Subscriptions for a User where the parent is None. `get_unread_subscription_list_for_user` will do the same but only returns Subscriptions that are unread or that have unread children if they are a folder.

1.8 Cloudflare Busting

django-feed-reader has Dripfeed support built in. If a feed becomes blocked by Cloudflare it can be polled via Dripfeed instead. This requires a Dripfeed account and API key.

CHAPTER TWO

COMMANDS

Commands that django-feed-reader adds to Django

CHAPTER

THREE

MODELS

`class feeds.models.Source(*args, **kwargs)`

This class represents a Feed to be read.

It really should have been called Feed, but what can you do?

`name`

str The name of the Feed (automatically populated)

`site_url`

str url of the website associated with the feed (automatically populated)

`feed_url`

str The URL that will be fetched to read the feed

`image_url`

str The url of an image representing the feed (automatically populated)

`description`

str The site description: may be HTML, be careful (automatically populated)

`last_polled`

datetime The last time the Feed was fetched

`due_poll`

datetime When the Feed is next due to be fetched

`last_result`

str The result the last fetch

`interval`

int How often the Feed will be fetched in minutes

`last_success`

datetime When the Feed was last read successfully

`last_change`

datetime When the Feed last changed

`live`

bool Is the Feed being actively fetched

`json`

dict Raw information about the Feed in JSON format (will not be collected unless **FEEDS_SAVE_JSON** is set to **True** in settings) `is_cloudflare`

bool Is this feed being hindered bt Cloudflare?

`property subscriber_count: int`

int he number of subscribers this feed has

`property unread_count: int`

int In a single user system how many unread articles are there?

If you need more than one user, or want to arrange feeds into folders, use a Subscription

`property best_link: str`

str The best user facing link to this feed.

Will be the **site_url** if it's present, otherwise **feed_url**

`property display_name: str`

str The best user-facing name for this feed.

Will be the the feed's **name** as described in the feed if there is one. Otherwise it will be the **best_link**

`get_unread_posts`(*newest_first=True*)

List[Post] In a single user system get all unread posts

If you need more than one user, or want to arrange feeds into folders, use a Subscription

Parameters
`newest_first` (*bool*)

Return type
list

`get_paginated_posts`(*page*, *newest_first=True*, *posts_per_page=20*)

Get a posts from the feed a page at a time

Parameters

- **page** (*int*) – The page to fetch.
- **oldest_first** (*bool*) – Get the posts in reverse chronological order (default True)
- **posts_per_page** (*int*) – The number of posts per page (default 20)
- **newest_first** (*bool*)

Returns
A tuple containting the page of posts and the paginator

Return type
Tuple[List[*Post*], Paginator]

`mark_read()`

In a single user system, mark this feed as read

`exception DoesNotExist`

`exception MultipleObjectsReturned`

`class feeds.models.Post`(**args*, ***kwargs*)

An entry in a feed

`source`

Source The source feed that this post belongs to

`title`
: **str** The post title

`body`
: **str** The main content of the feed in html or plain text

`link`
: **str** Link to this post on the web

`found`
: **datetime** When this post was first discovered

`created`
: **datetime** The created date for this post as reported in the feed

`guid`
: **str** The unique ID for this post

`author`
: **str** Name of the author of this post as reported by the feed

`index`
: **int** The number of this post in the feed for the purposes of tracking read/unread state

`image_url`
: **str** The URL of an image that represents this post

`json`
: **dict** Raw information about the Post in JSON format (will not be collected unless **FEEDS_SAVE_JSON** is set to **True** in settings)

`property current_enclosures`
: **ResultSet[Enclosure]** Returns all the current enclosures for this post

`property old_enclosures`
: **ResultSet[Enclosure]** Returns all the previous enclosures for this post

Some feeds change the URL of enclosures between reads. By default old enclosures are deleted and new ones added each time the feed is polled. To keep references to old enclosures set **FEEDS_KEEP_OLD_ENCLOSURES** to **True** in settings.

`save`(**args*, ***kwargs*)
: Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

`exception DoesNotExist`

`exception MultipleObjectsReturned`

`class feeds.models.Enclosure`(**args*, ***kwargs*)
: An enclosure on a post

`post`
: **Post** The Post that this Enclosure belongs to

`exception DoesNotExist`

exception MultipleObjectsReturned

length

int Size in bytes of the the related file

href

url The url of the enclosure

type

str The type of the enclosure

medium

str The type of the enclosure. Almost certainly one of image/video/audio

description

str A description of the enclosure - e.g. Alt text on an image

is_current

bool Is this enclosure current (if we are saving old enclosures - see above).

property is_image

bool Is the enclosure an image?

property is_audio

bool Is the enclosure audio?

property is_video

bool Is the enclosure video?

class feeds.models.**Subscription**(**args*, ***kwargs*)

A subscription to a Source Feed by a User

Subscriptions are also the way folder structures are set up

exception DoesNotExist

exception MultipleObjectsReturned

user

User The owner of the Subscription

source

Source The source feed of the subscription. If this is **None** then this is actually a folder

parent

Subscription The parent folder of the subscription. **None** if the subscription is at the root leve

is_river

bool Indicates if the feed/folder should be displayed in a "River of News" style

name

str The display name of the subscription - typically should be set to the name of the source where present

property unread_count: int

int The number of undread posts in teh subscription

If the subscription is acting as a folder, this will total up the unread counts of all children `get_unread_posts`(*oldest_first=True*)

Returns all the unread posts in a subscription

`get_paginated_posts`(*page*, *oldest_first=True*, *posts_per_page=20*)

Get a posts from the feed a page at a time

Parameters

- `page` (*int*) – The page to fetch.
- `posts_per_page` (*int*) – The number of posts per page (default 20)
- `oldest_first` (*bool*)

Returns

A tuple containting the page of posts and the paginator

Return type

Tuple[List[*Post*], Paginator]

`mark_read`()

Marks all the posts in the subscription as read

If the subscription is acting as a folder then it will mark all children as read as well.

CHAPTER

FOUR

UTILS

This module contains useful utility functions for manipulating your feeds.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

A

`author` (*feeds.models.Post attribute*), 9

B

`best_link` (*feeds.models.Source property*), 8
`body` (*feeds.models.Post attribute*), 9

C

`created` (*feeds.models.Post attribute*), 9
`current_enclosures` (*feeds.models.Post property*), 9

D

`description` (*feeds.models.Enclosure attribute*), 10
`description` (*feeds.models.Source attribute*), 7
`display_name` (*feeds.models.Source property*), 8
`due_poll` (*feeds.models.Source attribute*), 7

E

`Enclosure` (*class in feeds.models*), 9
`Enclosure.DoesNotExist`, 9
`Enclosure.MultipleObjectsReturned`, 9

F

`feed_url` (*feeds.models.Source attribute*), 7
`found` (*feeds.models.Post attribute*), 9

G

`get_paginated_posts()` (*feeds.models.Source method*), 8
`get_paginated_posts()` (*feeds.models.Subscription method*), 11
`get_unread_posts()` (*feeds.models.Source method*), 8
`get_unread_posts()` (*feeds.models.Subscription method*), 10
`guid` (*feeds.models.Post attribute*), 9

H

`href` (*feeds.models.Enclosure attribute*), 10

I

`image_url` (*feeds.models.Post attribute*), 9
`image_url` (*feeds.models.Source attribute*), 7
`index` (*feeds.models.Post attribute*), 9
`interval` (*feeds.models.Source attribute*), 7
`is_audio` (*feeds.models.Enclosure property*), 10
`is_cloudflare` (*feeds.models.Source attribute*), 7
`is_current` (*feeds.models.Enclosure attribute*), 10
`is_image` (*feeds.models.Enclosure property*), 10
`is_river` (*feeds.models.Subscription attribute*), 10
`is_video` (*feeds.models.Enclosure property*), 10

J

`json` (*feeds.models.Post attribute*), 9
`json` (*feeds.models.Source attribute*), 7

L

`last_change` (*feeds.models.Source attribute*), 7
`last_polled` (*feeds.models.Source attribute*), 7
`last_result` (*feeds.models.Source attribute*), 7
`last_success` (*feeds.models.Source attribute*), 7
`length` (*feeds.models.Enclosure attribute*), 10
`link` (*feeds.models.Post attribute*), 9
`live` (*feeds.models.Source attribute*), 7

M

`mark_read()` (*feeds.models.Source method*), 8
`mark_read()` (*feeds.models.Subscription method*), 11
`medium` (*feeds.models.Enclosure attribute*), 10

N

`name` (*feeds.models.Source attribute*), 7
`name` (*feeds.models.Subscription attribute*), 10

O

`old_enclosures` (*feeds.models.Post property*), 9

P

`parent` (*feeds.models.Subscription attribute*), 10
`Post` (*class in feeds.models*), 8
`post` (*feeds.models.Enclosure attribute*), 9
`Post.DoesNotExist`, 9
`Post.MultipleObjectsReturned`, 9

S

save() (*feeds.models.Post method*), 9
site_url (*feeds.models.Source attribute*), 7
Source (*class in feeds.models*), 7
source (*feeds.models.Post attribute*), 8
source (*feeds.models.Subscription attribute*), 10
Source.DoesNotExist, 8
Source.MultipleObjectsReturned, 8
subscriber_count (*feeds.models.Source property*), 8
Subscription (*class in feeds.models*), 10
Subscription.DoesNotExist, 10
Subscription.MultipleObjectsReturned, 10

T

title (*feeds.models.Post attribute*), 8
type (*feeds.models.Enclosure attribute*), 10

U

unread_count (*feeds.models.Source property*), 8
unread_count (*feeds.models.Subscription property*), 10
user (*feeds.models.Subscription attribute*), 10