Social Feed Manager Documentation

Release m5$_0 01$

George Washington University Libraries

October 02, 2014

Contents

Social Feed Manager is a Django application developed by George Washington University Libraries to collect social media data from Twitter. It connects to Twitter's approved API to collect data in bulk and makes it possible for scholars, students, and librarians to identify, select, collect, and preserve Twitter data for research purposes.

The application code is open source and available on github.

Contents:

CHAPTER 1

Introduction

1.1 Overview

Social Feed Manager is open source software for locally capturing public data from Twitter. It makes it possible for librarians, archivists, scholars, and students to:

- identify, select, collect, and preserve "at risk" social media data
- gather datasets of tweets in bulk for analysis in other software packages
- fill gaps in special collections
- archive the social media activity of their library or institution.

The software connects to Twitter's approved public APIs to collect tweets by specific users, search current tweets by keyword, and filter by geolocation. We hope to add other social media platforms in the future.

1.2 Features

- Collects tweets account by account
- Queries streaming APIs by keyword, user, and geolocation
- Captures Twitter's sample stream (currently ~0.5-1% of tweets)
- Manages multiple streams reliably
- Respectful of Twitter rate limits
- Groups tweets into sets for easier management
- CSV export, which can be uploaded into analysis software of the researcher's choice
- Web-based interfaces for researchers / data users and administrators
- Command-line updates for application administrator
- Streaming data output to compressed rolling files in date/time hierarchy

1.3 Current uses at George Washington University

- The University Archives is gathering tweets by university offices and student organizations, capturing an aspect of student life whose main online presence is on social media

- Faculty in the School of Media and Public Affairs are studying how journalists, activist organizations, and members of Congress tweet
- Students in digital journalism are learning how to analyze tweets to inform reporting
- Computer science faculty are using tweet datasets to train machine learning algorithms

1.4 Technical and staffing considerations

Social Feed Manager is locally hosted and requires a system administrator to set up and manage the application in a Linux (Ubuntu 12.04) environment. Storage requirements vary depending on usage of the application: collecting data account-by-account requires less storage than connecting to the streaming APIs, which accumulates large files.

Archivists, librarians and other service administrators, as determined by the library, use a web-based interface to add new Twitter users, specify keyword queries, and create sets of accounts. Researchers may directly download user timeline data from the web interface after signing into the site from an institutional IP address and using their own Twitter credentials. Currently, all captured user timeline data is available in the researcher interface and is not separated by researcher account. Accessing files generated from the streaming APIs requires mediation by a system administrator.

1.5 Development and community

Social Feed Manager was developed at The George Washington University Libraries in 2012 as a prototype application and is now being supported by multiple developers and an *IMLS Sparks! Innovation Grant <http://www.imls.gov/applicants/detail.aspx?GrantId=19>* [1]. Several libraries and archives have installed it and are providing feedback to help prioritize development of new features.

The software is available for use, study, copying, and modification under a free and open source software license (MIT license). We welcome others to become involved in the project and contribute to the code.

1.5.1 Contact us

- sfm-dev Google Group
- Developers responsible for the app include Dan Chudnov (@dchud), Dan Kerchner (@dankerchner), Ankushi Sharma (@ankushis), and Laura Wrubel (@liblaura).

1.5.2 Resources

Social Feed Manager on Github

Google Group (updates about new releases and discussion of features)

[1] Institute of Museum and Library Services Grant LG-46-13-0257

CHAPTER 2

Installation and Configuration

2.1 Background

Social Feed Manager is not a simple "click to run" application. It is best if you or somebody you can work with has some experience as a unix/linux systems administrator when attempting to install this application. It's not the world's most complicated app, but you will need to install system software and configure it, set up a database, get application credentials from Twitter, and then use all of that to configure and run SFM, which is a python/django application that plugs into a web server. If these are new tasks for you, you might want to work with another person with a little more experience.

Another key consideration is platform. At GW Libraries we develop, test, and run SFM strictly on Ubuntu LTS servers (virtual machines, actually), so these docs reflect that. If you want to install SFM into another environment you will be on your own to some degree. But if you stick with Ubuntu LTS (currently 12.04) these instructions should work for you if you follow them precisely.

We develop and run SFM inside a virtualenv, which is a commonly used sandbox / isolation technique for Python applications. This allows a Python application and its third-party library dependencies to be installed into one independent virtual environment for the app side-by-side with other applications (and perhaps their own virtualenvs) on the same system, or even alongside multiple versions of SFM itself. There are many benefits to using virtualenv for this purpose. We strongly recommend that you do the same, and these instructions will guide you.

2.2 Dependencies

These instructions assume you have a brand new Ubuntu 12.04 LTS server.

SFM is developed and managed using Ubuntu 12.04, Python 2.7, PostgreSQL 9.1+, Apache 2+, and other dependencies we'll run into in a moment. If you want to use something else, you're on your own, but let us know and please feel welcome to submit a pull request with your own suggestions.

First, install these system-level packages:

```
% sudo apt-get install git apache2 python-dev python-virtualenv postgresql libxml2-dev libxslt1-dev l
```

(Optionally, change to the directory where you wish to install SFM. Your user will need write permissions here.)

Next, get this code using git:

```
% git clone https://github.com/gwu-libraries/social-feed-manager.git
% cd social-feed-manager
```

Create and activate a virtualenv:

```
% virtualenv --no-site-packages ENV
% source ENV/bin/activate
```

Note that the first command creates a virtualenv, and the second command activates it. For nearly all of the following instructions, we assume you are active in the virtualenv you just created.

Prep PostgreSQL, the database SFM uses. First, update `/etc/postgresql/9.1/main/pg_hba.conf` to enable local database connections or otherwise as you prefer. Note that there's more than one way to do this, so if this is new to you, read this background information, or ask a friendly sysadmin for help.

```
http://www.postgresql.org/docs/9.1/static/auth-pg-hba-conf.html
```

For example, you could add a line like this (you will probably need to use `sudo` to edit the file):

```
local   all             all                     md5
```

When you've edited `pg_hba.conf`, save it, then restart postgresql.

```
% sudo service postgresql restart
```

When that succeeds, su to the `postgres` account to create a postgresql user and database. Substitute your own preferred values for the all caps values below, but do use the single quotes around your password when you create it in the third line.

```
% sudo su - postgres
(postgres)% psql
postgres=# create user YOURSFMDBUSERNAME with createdb password 'YOURSFMDBPASSWORD';
CREATE ROLE
postgres=# \q
(postgres)% createdb -O YOURSFMDBUSERNAME sfm -W
Password: YOURSFMDBPASSWORD
(postgres)% (ctrl-d to log out of postgres acct)
```

Now install the python library requirements in your virtualenv using `pip`. This might take a few minutes. This step requires that you successfully installed all of the system-level packages above. Note that these python packages are being installed into your virtualenv, not system-wide, which is what we want. They will only be available while you are in this activated virtualenv.

```
% pip install -r requirements.txt
```

That should be everything you have to install. Now it's on to configuring the SFM app itself. Progress!

2.3 Configuration

Now we'll configure the SFM app itself. Before we do, though, did you go through all of the steps above? If not, or if you're using a different platform, you might have different results. So now's a good time to check that each of these tasks is done:

- installed system-level dependencies using apt-get install
- cloned the social-feed-manager repo using git
- activated the virtualenv sandbox for your sfm setup
- configured postgresql, restarted it, created a db and a user
- installed app-level dependencies in the sfm virtualenv using pip

If you haven't done all of these, please go back and be sure you do.

Next we configure the SFM app, which takes a few steps:

- set configuration parameters for SFM itself
- obtain Twitter API credentials and specify them in the SFM config
- set up the database

If you aren't already there, cd into the social-feed-manager/sfm directory first:

```
% cd sfm
```

Django uses a `settings.py` file for most configurations; SFM also uses a second `local_settings.py` file for installation details like database name and user and Twitter API authentication information. We include a template version of that file in the `social-feed-manager/sfm/sfm` directory to make it easy to get started. You'll copy that to your own `local_settings.py` file and edit that to specify your configuration.

Copy the template to your own local settings file:

```
% cp sfm/local_settings.py.template sfm/local_settings.py
```

Edit this file and set appropriate values for just these parameters at first, we'll go back later and get the rest:

- ADMINS (specify your name and email address in the format provided)
- DATABASES (NAME, USER, PASSWORD as you defined for postgres above; HOST should be 'localhost' assuming your database and application are on the same server, as per these instructions.)
- DATA_DIR (create a directory to hold data files, then specify it here; use a new directory that is not inside the social-feed-manager directory)
- TWITTER_DEFAULT_USER (the name of the twitter account you'll use to connect to the API; we'll specify the other TWITTER_* settings in a bit)

Next, do the same for the `wsgi.py` file, copy its template to a new file specific to your installation:

```
% cp sfm/wsgi.py.template sfm/wsgi.py
```

In this new file `wsgi.py`, uncomment just the three lines below the one that starts with "if using a virtualenv...", then specify the location of your virtualenv in the second of these lines. When you're done, it should look something like this:

```
import site
ENV = '/home/dchud/social-feed-manager/ENV'
site.addsitedir(ENV + '/lib/python2.7/site-packages')
```

WSGI is a specification for connecting applications like SFM to web servers; this file tells a web server where to look for the SFM app and its dependencies on your system. We'll configure the web server later.

Our next step is critical - register your SFM instance with Twitter's "Application Management" page. Log in to Twitter using the account you specified as `TWITTER_DEFAULT_USER`, then visit this page:

```
https://dev.twitter.com/apps/new
```

Here, create an app for your instance of SFM. In addition to the required values, set the application type to "read only", and give it a callback URL. The callback URL can be the same as your website URL, but you have to provide a value or the authorization loop between twitter/oauth and django-social-auth/ sfm will not work correctly.

Did you give it a callback URL? Good. It's required. Really.

When you finish this process, you'll see a OAuth consumer key and secret for your SFM instance. At the time of this writing, they're located on the "API Keys" tab listed as the API key and the API secret. Use these as the values for these two settings in `local_settings.py`:

- TWITTER_CONSUMER_KEY
- TWITTER_CONSUMER_SECRET

These two settings along with `TWITTER_DEFAULT_USERNAME` should all be defined now with real values from your account and your SFM app's OAuth key/secret.

2.4 First time running SFM

There are several layers of "users" with SFM; the next steps are critical because if the users aren't lined up just right, SFM won't be able to use Twitter's API. It can be a little confusing, but it's important to understand what's going on here.

The first few layers of users are at the system-level. You are logged in to your machine using a system user; using that account, you installed system-level dependencies (with sudo or as root, perhaps). You also configured PostgreSQL and cloned SFM and installed SFM's dependencies with the system user. When you configured PostgreSQL you also created a user for PostgreSQL. The PostgreSQL user is what SFM uses to connect to the PostgreSQL database.

Next, there are two kinds of Twitter users we are interested in. First, you used your own Twitter account to register your SFM install with Twitter; the OAuth keys you received for that user allow SFM to connect to the Twitter API. This is separate from the accounts of Twitter users for which you want to collect tweets, which we'll also record in the system later, in the database, using SFM.

Finally, to log in and use SFM through the web, there are two kinds of SFM app-level users. You can have administrative accounts (we'll create one in a second), strictly for housekeeping purposes, and you can also have Twitter-authenticated users for day-to-day use (we'll create one of these too). The administrative accounts may be Twitter-authenticated, but they don't have to be.

This is all very confusing, yes, but it will make more sense in a few minutes.

First, we set up the database using the regular django method `syncdb`, but read the next three paragraphs first, they're important.

`syncdb` will use the settings you configured in `local_settings.py` to connect to the database and set up the tables SFM requires.

This will also ask you to create a superuser. Do this, and name it `sfmadmin`. Don't name it the same thing as your `TWITTER_DEFAULT_USER`. You will be prompted for an email address and password, fill these in and remember your password.

Did you call the superuser `sfmadmin`? Really? Good.

```
% ./manage.py syncdb
```

When that completes, we need to "migrate" the database to the most recent data model:

```
% ./manage.py migrate
```

When that completes, we're ready to run the app, finally:

```
% ./manage.py runserver
```

By default, this will run SFM using Python's built-in web server, on a high port number like 8000. If you are on a server that doesn't allow web traffic through port 8000 through the firewall, but does allow port 8080, you can specify a host and port:

```
% ./manage.py runserver sfm.example.com:8080
```

This will start the web application on sfm.example.com at port 8080.

The built-in web server is really only good for development and testing, not production, but it does provide access to everything the app does.

Next, visit the webapp in your browser: http://sfm.example.com:8080/

You should see a blue bar at the top and a request to "Please log in" and a button to "Log in with Twitter". Click that button, and now log in through Twitter using the account you specified in your `TWITTER_DEFAULT_USERNAME`. Maybe your browser is still logged in with this account because you configured your SFM instance at Twitter and got your OAuth credentials with it, in which case, great.

If this works, it should bounce you back to your sfm.example.com site and you should see an empty SFM, with no users listed, but you should be reassured to see "log out YOURNAME" in the top blue bar. If that works, you're in great shape.

Now, click "log out YOURNAME" and log out. Yes, log back out.

Next, in your browser, then, visit: http://sfm.example.com:8080/admin

You'll see a different user/pass challenge. Here, enter the SFM app-level superuser name "sfmadmin" and password you created above when you ran `syncdb`. This should drop you into the admin screen. Under "Site administration" -> "Auth", click "Users". You should see two different app users, one called "sfmadmin" and another with your `TWITTER_DEFAULT_USERNAME`. "sfmadmin" should have "Staff status" with a green checkmark; the other account does not, and has a red circle with a white minus sign. If you see all this, you are in good shape.

Next, click on "Home", then under "Social_Auth", click "User social auths". On the next screen you should see one user, with your `TWITTER_DEFAULT_USERNAME`. Click the number next to its name, and you'll see the OAuth access token for this user which allows SFM to connect to the Twitter API.

Why doesn't "sfmadmin" have a social auth? Because it only ever logged in to SFM. The sfmadmin account is only for your housekeeping needs; the other account can be used to connect and read data from the API.

What's the social auth? These are credentials that allow your SFM instance to connect to Twitter's API on behalf of your Twitter account. sfmadmin never logged in through Twitter, so it doesn't have one, and it doesn't need one.

If this is still confusing, try this: log out again, then grab a colleague and have them log in to your SFM using their own Twitter account (with the "Log in with Twitter" button on the home page). After they're done, log them out of SFM, then log back in using sfmadmin and the `/admin` URL. Under the Auth -> Users list, and in User social auths, you'll see their new sfm account. Get the difference now?

The OAuth credentials you got when you registered your SFM instance allow SFM to connect to the Twitter API to do things like let users log in to SFM themselves through Twitter. Then, when you finally do connect to the Twitter API to get data, you'll use a combination of your app-level OAuth credentials and the access token for your `TWITTER_DEFAULT_USER` or another credentialed user to get that data.

So let's do that now.

Logged in to the `/admin` page using your sfmadmin administrative account, go Home, then under "Ui" click "Twitter users". There shouldn't be any yet - these are the names of accounts you want to collect. At the top right, click "Add twitter user", and on the next screen, enter the name "bbcnews" (no quotes, though!), which is a good example because it's active all the time. At the bottom right, click "Save".

If this succeeds, you should see that user "BBCNews" is now added to your system as a twitter user. Note that it's "BBCNews", not "bbcnews" - when you clicked "Save", SFM did the following:

- connected to Twitter's API using your `TWITTER_DEFAULT_USER` account credentials
- queried Twitter's API for a user named "bbcnews"
- found the account "BBCNews" and its info

- stored this as a new TwitterUser in SFM, using the case-corrected name form

If it didn't work, double-check your spelling.

This is the easiest way to add users to SFM.

Now that you've added a TwitterUser, let's fetch its recent tweets.

Back in your terminal window, enter:

```
% ./manage.py user_timeline
```

Sit back and watch for a bit. SFM will connect to Twitter's API and make a series of calls to fetch 200 recent tweets at a time, up to 3200 total, pausing between each call. The numbers 200 and 3200 aren't arbitrary, they are set by Twitter (see https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline for details). SFM abides by Twitter's API and pauses regularly so that it can stay within the API's rate limits.

You are now up and running with SFM.

2.5 Apache integration

To run SFM in production, we recommend integrating with apache using WSGI. It's straightforward and well-tested. You will need to copy a configuration file into apache's `sites-available` directory, edit that file to match your installation details, enable that site (and optionally disable other versions), then restart apache. Let's get started.

First, copy our apache configuration template to `sites-available`. We like to append the appname "sfm" with the version number, e.g. `sfm_m5_001`, so when we go to deploy a new version, we can just add a new config file and make the switchover easy. You could just call it `sfm` if you want, but it can help to have the version number in there, so these instructions use that convention.

```
% sudo cp sfm/apache.conf /etc/apache2/sites-available/sfm_m5_001
% sudo vim /etc/apache2/sites-available/sfm_m5_001
```

You will need to change several things in this file:

- change references to `/PATH/TO/sfm` to the full absolute path to your `social-feed-manager/sfm` directory
- change references to `YOUR-HOSTNAME.HERE` to your public hostname
- change the reference to `/PATH/TO/YOUR/VENV` to the full absolute path to your virtualenv (ending in `ENV`) which you created above
- change the reference to `python/2.X` to 2.7

When you've made all those changes, save the file.

Next, enable the site configuration you just created:

```
% sudo a2ensite sfm_m5_001
```

Assuming you are installing in a clean VM, disable the pre-existing default site:

```
% sudo a2dissite 000-default
```

Reload the apache configuration, as it suggts when you made the changes above:

```
% sudo service apache2 reload
```

That's it! It should be working now.

If you run into any problems, check the logs in `/var/log/apache2/`.

2.6 What next?

Some options for what to do next:

- add more TwitterUsers and run user_timeline again
- set up cronjobs for user_timeline and other *daily operations*
- *set up supervisord* and use it to capture one or more streams
- sign up to https://groups.google.com/forum/#!forum/sfm-dev to ask questions or suggest improvements
- track SFM progress, file bug/enhancement tickets, fork the code and submit pull requests at: https://github.com/gwu-libraries/social-feed-manager

CHAPTER 3

Daily Operations

SFM is not a set-it-and-forget-it kind of application. Things change constantly on social media platforms like Twitter, so we have to check constantly for these changes and act appropriately. For example, if you haven't yet read our summary of the *lifecycle of a TwitterUser*, read it now and come back, you'll see what we mean.

We have added several commands and tweaks to the data model to account for these changes as we've been running SFM for the past few years. Please read through the descriptions below and consider how they should apply in your scenario, as well as what might be missing that you will want to supply yourself for your own environment, or perhaps to add to SFM itself and submit back to the project. There are likely to be more of these to come as more people use the app, and we welcome your ideas.

3.1 Administrative tasks

Once you have successfully *installed SFM* the first task is to add app users; if at least one other person will be using the app, go to the `/admin/` url and sign in with the administrative system account you created during installation. Under "Auth -> Users" you can add one or more additional SFM users (ask them to set their password). Once you've saved a new user, you can edit them from the list of Users and give them "superuser" status if you want them to be able to add users like you can with your own admin account. If one of these people ever leaves your organization or stops using the app, you can set their account to inactive by unchecking the "Active" box on their user edit page, too, instead of deleting their account entirely. Note that this functionality is all provided out-of-the-box by Django itself, with no custom SFM code.

An alternate way to add a user is to let them sign in at the `/` url by authenticating through Twitter. The advantage of this approach is that SFM will save a copy of authorized OAuth tokens for their account, which you can use later to manage a stream-based filter for that user. Once someone logs in successfully this way, you can edit their account under `/admin/` just like any other SFM user, but note that you can end up with two different SFM accounts for the same person by accident if you use both methods.

3.2 Data gathering

Now that you and your colleagues have accounts for your SFM, you can add TwitterUsers. This is the simplest way to capture data using SFM. From `/admin/` under "UI -> Twitter Users" add Twitter accounts to capture by their names, one by one, by entering their account name in the "Name" box. Be careful to spell it correctly! SFM will look up that account by name and verify that it's a public account, and will then store the Twitter UID. Try adding a few accounts.

Now that there are a few TwitterUsers in your database, to capture their recent tweets, use the *user_timeline* management command. Run the command once, and you'll see updates of the data-fetching process on the commandline. As it proceeds, you can go to `/` in your app and you'll see the data start to appear in the UI. You can also go to `/admin/`

and see these same tweets in the admin UI under "UI -> Twitter user items". Finally, there will be a separate record of the `user_timeline` "job" you ran under "UI -> Twitter user timeline jobs".

As you capture tweets this way, you might want to create a record of the urls linked to by shortened urls in tweet text. To do this, use the *fetch_urls* command.

Note that as you add more and more TwitterUsers and their tweet data, both of these commands can take a long time – even many hours – to run. It takes a while because SFM abides by the rate limits defined by the Twitter API, leaving a little multi-second buffer between every call to the API so the app never goes over the limits. The more users you're collecting, the longer it will take.

Both the `user_timeline` and `fetch_urls` commands are well suited to being automated with something like a cronjob. There are subtle issues to consider here, though, namely that whenever you fetch a user's tweets, the metadata associated with each tweet will be accurate as of the moment you fetch it, rather than from the moment the tweet was originally published. This means that the first time you grab, say, 500 old tweets from a TwitterUser you just added, *every* one of those 500 tweets will contain exactly the same follower/following counts on the TwitterUser. Also, if that 500th tweet you capture is only five minutes old, then the retweet count on your capture of that tweet only accounts for the five minutes of that tweet's existence. Older tweets may have correspondingly higher retweet numbers.

It's important to understand these issues because how regularly you capture tweets using `user_timeline` will determine how accurate these numbers are. If it is important to you to see how following/follower counts change tweet by tweet, you'll want to run `user_timeline` often. If it's important to get an accurate retweet count on each tweet, you might want to run it less often. Either way, there will be a bit of a sliding time gap over the range of tweets you capture at any given time because of these implementation details of the Twitter API, and the relative accuracy for a given purpose of the metadata you capture when you've captured it will vary accordingly. It also means that when you first capture a TwitterUser's older tweets you will not be able to see how old tweets affected their follower/following counts. These details might be important to users of the data you collect, so please familiarize yourself with them.

At GW Libraries, as of July 2014, we track about 1,800 TwitterUsers, running the `user_timeline` command on a cronjob every six hours. We run the `fetch_urls` command on a cronjob once a day, limiting (with the optional start and end date parameters) to the previous day's tweets. Each of these jobs takes several hours to complete. Our PostgreSQL database for SFM uses over 6Gb in production, and a complete export of the database to a single file compresses to about 1.5Gb.

3.3 Account maintenance

Due to the many changes that can occur on a single TwitterUser account (as described in *lifecycle of a TwitterUser*), you should run *update_usernames* regularly as well. Because SFM uses the Twitter uid of a TwitterUser rather than the name to capture new data, `user_timeline` will continue to work if SFM doesn't have an updated username even after the Twitter account name changes, but it's best all around if you have a record of the changes over time, and if you're never too far out of date. At GW Libraries we've found that running it once a week during the weekend suffices.

If the `user_timeline` or `update_usernames` scripts report errors, such as an account no longer being available, or no longer being public, you can deactivate a TwitterUser the `/admin/` UI under "UI -> TwitterUsers", just search for that account by its name or uid, click on its SFM id when you find it, then uncheck the "Is active" box on the TwitterUser edit page. When a TwitterUser is inactive, `user_timeline` will no longer check for new tweets, saving time and rate limit capacity. You can always re-activate a TwitterUser later if its account changes again.

3.4 Data movage

If you are using one or more *Supervisord-managed streams* to capture filtered queries live off the Twitter hose or the sample stream, you will want to establish an appropriate set of scripts to handle the resulting files. SFM has no opinion about how you manage digital content, aside from a bias toward gzipping text files at regular intervals. :) You might

want to set up a cronjob pipeline to package up files using BagIt, or move them to another server, or whatever works for you, but keep in mind that these files can grow to fill up gigabytes and terabytes of storage quickly.

SFM does provide the *organizedata* management command to walk through a set of gzipped stream files and sort them into a year/month/date/hour-based folder structure. This is optional, but we find it convenient to spread files out on a filesystem, and for the scripts we're working with to post-process files we generate at appropriate time intervals.

3.5 System considerations

These are outside of the scope of SFM proper, but worth keeping in mind.

It is best to establish a regular snapshot backup of the PostgreSQL SFM database, and to rotate those files to a secondary storage environment. This can help both with testing new versions of the software and should you ever otherwise need to restore your database from scratch.

The same logic holds for taking a snapshot backup of your configuration files, such as your `local_settings.py` and apache config file. These should be relatively easily reproduceable - you can get your OAuth keys back from Twitter, for example - but it can be a pain to have to do so.

At GW Libraries we have a twice-daily cronjob that performs these operations.

CHAPTER 4

Management Commands

4.1 Introduction

Many of the key back-end functions of Social Feed Manager (SFM) are invoked using management commands. The SFM management commands are standard Django management commands. As such, they are invoked like any other Django management commands:

1. First make sure that your virtualenv is activated.

```
$ source ENV/bin/activate
```

2. From <PROJECT_ROOT>/sfm, execute ./manage.py followed by the desired management command, arguments and options.

```
$ cd sfm
$ ./manage.py <command> [args] [options]
```

SFM management commands may be run:

- manually (i.e. at the command line),
- using cron jobs, and/or
- using supervisord (in the case of filterstream and streamsample) as described in the section on *supervisor and streams*.

Each SFM management command is described below.

4.2 user_timeline

user_timeline calls the Twitter API to retrieve the available tweets for either all *active* TwitterUsers in SFM, or for a specific *active* TwitterUser. Each tweet is created as a TwitterItem in SFM.

user_timeline connects to the Twitter API as TWITTER_DEFAULT_USERNAME, and requests the user_timeline by the Twitter account uid (not by account name). Through the tweepy library, it calls the Twitter API user_timeline method.

For each TwitterUser user_timeline requests only tweets since the newest tweet that was previously retrieved. If no tweets were previously retrieved for that TwitterUser, it requests as many tweets as the Twitter API will provide (up to the 3200 most recent tweets).

To fetch tweets for all active TwitterUsers in SFM:

```
./manage.py user_timeline
```

To fetch tweets for a specific twitter user:

```
./manage.py user_timeline --user='twitter username'
```

The full specification of user_timeline options can be viewed using –help:

```
./manage user_timeline --help
```

Sample output for user_timeline:

```
user: pinkfloyd
since: 1
saved: 200 item(s)
since: 1
max: 326988934884249599
saved: 200 item(s)
since: 1
max: 168992796676591616
saved: 199 item(s)
since: 1
max: 117550550098247679
saved: 86 item(s)
stop: < 150 new statuses
```

4.3 update_usernames

Twitter account owners can, and often do, change the names of their accounts, although an account's UID never changes.

update_usernames looks up the names of the Twitter accounts corresponding to all active TwitterUsers. If a Twitter account's name has changed since SFM last verified the account's name, *update_usernames* will update the name of the TwitterUser, and will append the former name (and timestamp) to the TwitterUser's former_names value. former_names is a json field; an example would be:

```
{"2014-02-19T21:50:56Z":  "OldName", "2014-01-16T13:49:02Z":  "EvenOlderName"}
```

Note that *update_username* is case sensitive; a change in capitalization *is* considered a name change.

To update names of all active TwitterUsers:

```
./manage.py update_usernames
```

To update names of a specific active TwitterUser, by its current name in SFM:

```
./manage.py update_usernames --user='current TwitterUser name in SFM'
```

4.4 populate_uids

Deprecated since version m5_001.

4.5 streamsample

The Twitter API provides a streaming interface which returns a random sample (approximately 0.5%) of all public tweets. The SFM *streamsample* management command directs the content of this stream to files. The location of these output files is determined by the DATA_DIR variable in the local_settings.py configuration file. As *streamsample* is intended to be run as an ongoing, streaming process, SFM provides a streamsample.conf.template file in <PROJECT ROOT>/sfm/sfm/supervisor.d that can be copied to streamsample.conf and edited to include the relevant pathnames, so that it can be run and managed using supervisord.

streamsample currently generates 2 GB worth of tweet data per day (roughly 2.2-2.5 million tweets), so it is important to plan storage capacity accordingly.

To run manually and view streaming output to the console:

```
./manage.py streamsample
```

To run manually and direct output to files in DATA_DIR:

```
./manage.py streamsample –save
```

Information on the Twitter API streamsample resource: https://dev.twitter.com/docs/api/1.1/get/statuses/sample

4.6 filterstream

The Twitter API provides a streaming interface which returns tweets that match one or more filter predicates. SFM administrative users can create multiple TwitterFilters, each with its own predicate parameters. The SFM *filterstream* management command directs the content of one or more active TwitterFilters to files. The location of these output files is determined by the DATA_DIR variable in the local_settings.py configuration file.

filterstream is intended to be run as a set of ongoing, streaming processes; SFM automatically generates the necessary supervisord configuration files. However, generation of these files requires the DATA_DIR, SUPERVISOR_PROCESS_USER, and SUPERVISOR_UNIX_SOCKET_FILE settings variables to be configured in local_settings.py .

Each TwitterFilter may contain the following predicates:

Words - Keywords to track

People - Twitter accounts to track

Location - Geographic bonuding boxes to track

To run manually and view streaming output to the console:

```
./manage.py filterstream
```

To run manually and direct output to files in DATA_DIR:

```
./manage.py filterstream –save
```

filterstream can also take a parameter corresponding to the number of an individidual TwitterFilter, e.g.

```
./manage.py filterstream 4 –save
```

This will run *filterstream* only for the TwitterFilter with an id of 4. If no TwitterFilter number is given, filterstream will run for all active TwitterFilters.

Information on the Twitter API filter streaming resource: https://stream.twitter.com/1.1/statuses/filter.json

4.7 organizedata

filterstream and *streamsample* produce sets of data files in the directory determined by DATA_DIR as configured in local_settings.py . The data files are written as rotating files; periodically (as determined by SAVE_INTERVAL_SECONDS in local_settings.py) each file is closed and subsequent data is written to a new file. The naming scheme for each data files includes a timestamp. Over time, this can create many files in the DATA_DIR directory.

The *organizedata* command organizes these files by creating subdirectories named "sample" to data files from *streamsample*, and "twitterfilter-*n*" for data files from *filterstream*, for each TwitterFilter.

Within <DATA_DIR>/sample and each <DATA_DIR>/twitterfilter-*n* directory, *organizedata* creates a tree with a subdirectory for each year; within each year directory, it creates a subdirectory for each month; within each of these, a subdirectory for each day.

To run *organizedata*:

```
./manage.py organizedata
```

4.8 fetch_tweets_by_id

Each tweet in Twitter has a unique numerical ID. The *fetch_tweets_by_id* management command takes a file consisting of a list of tweet IDs (one per line), and fetches the associated tweets as JSON.

Errors are logged to a file given the same name as the input file (specified by *–inputfile*) with an appended extension of .log (e.g. myinputfile.log)

To fetch tweets and output to the console:

```
./manage.py fetch_tweets_by_id --inputfile='<PATH TO YOUR INPUT FILE>'
```

To fetch tweets and write to an output file:

```
./manage.py fetch_tweets_by_id --inputfile='<PATH TO YOUR INPUT FILE>' --outputfile='<PATH TO YOUR OU
```

4.9 fetch_urls

Links in tweets are often link-shortened. *fetch_urls* iterates through all tweets (TwitterItems), extracts each URL found in a tweet and creates a TwitterUserItemUrl for it, and expands the URL if possible. The final URL is stored as part of the TwitterUserItemUrl object.

fetch_urls can be run with the following options:

- –start-date – The earliest date of tweets to fetch URLs for
- –end-date – The latest date of tweets to fetch URLs for
- –twitter-user – The specific twitter username to fetch URLs for
- –limit – maximum number of URLs to fetch
- –refetch – include tweets for which URLs were already fetched; refetch URLs for these tweets.

To run:

```
./manage.py fetch_urls
```

4.10 export_csv

Tweets stored in SFM associated with a TwitterUser or a TwitterUserSet can be exported in CSV (comma-separated value) format using the *export_csv* management command. The user interface also offers CSV exports via a link on each TwitterUser's page (currently there is no page in the UI for a set).

The format and meaning of each column in the CSV export is explained in the *Data Dictionary*.

export_csv can be run with the following options. Either twitter-user or set-name must be specified.

–start-date – exports only tweets starting from the specified date (YYYY-MM-DD)

–end-date – exports only tweets through the specified date (YYYY-MM-DD)

–twitter-user – exports tweets for the specified TwitterUser (by name)

–set-name – exports tweets for the specified TwitterUserSet

To export tweets for Twitter user "sfmtwitteruser":

```
./manage.py export_csv --twitter-user sfmtwitteruser
```

To export tweets for TwitterUserSet "myset":

```
./manage.py export_csv --set-name myset
```

4.11 createconf

The *createconf* command is used to create supervisord configuration files for each active TwitterFilter. This command should only need to be run if TwitterFilters were created in SFM prior to version m4_002, as part of upgrading to SFM version m4_002 or later.

createconf can be run with the –twitter-filter option, to create a supervisord configuration file only for the specified TwitterFilter (specified by numeric id).

To create configuration files for all active TwitterFilters:

```
./manage.py createconf
```

To create configuration files for TwitterFilter 5:

```
./manage.py createconf --twitter-filter 5
```

CHAPTER 5

Using Supervisord to Manage Streaming Filters

As of release m4_002, Social Feed Manager uses supervisord to manage the filterstream and streamsample processes. As streaming processes, these are intended to be run on a continuous, ongoing basis, to collect tweets over time. Supervisord is a process control system that, among other features, manages the SFM streaming processes independently from the SFM web application, and can restart these processes if they fail or after a system reboot.

Twitterfilters and/or streamsample *can* still be run independently of supervisord if desired (e.g. for testing), by invoking them at the command line as *management commands*.

Supervisord is installed as part of the standard SFM installation; it is one of SFM's ubuntu package dependencies. However, it must be configured in order to use filterstreams.

To configure supervisord for SFM:

edit `/etc/supervisor/supervisord.conf`. Look for the `[include]` section (in a new instance of supervisor, this is usually at the bottom) and add `supervisor.d/*.conf` to the space-separated list of `files`:

```
files = /etc/supervisor/conf.d/*.conf <PATH_TO_YOUR_SFM>/sfm/sfm/supervisor.d/*.conf
```

create a `/var/log/sfm` directory. The supervisor-supervised processes will write log files to this directory.

```
$ sudo mkdir /var/log/sfm
```

edit local_settings.py to set DATA_DIR to the directory where you want stream output stored. Set SUPERVISOR_PROCESS_OWNER to a user who has rights to write to `/var/log/sfm`. You may also wish to adjust SAVE_INTERVAL_SETTINGS, which controls how often sfm will save data to a new file (default is every 15 minutes, specified in `'settings.py'`).

set the permissions on the `sfm/sfm/supervisor.d` directory to allow the sfm process owner to write to it. Since the sfm process may be running as a different user than the owner of the directory, we're going to create a new 'sfm' group:

```
$ sudo groupadd sfm
```

and add the sfm process owner to this group. Edit `/etc/group`:

```
$ sudo vi /etc/group
```

You should see a new line at the end that looks something like this:

```
sfm:x:<a group number>:
```

Add the process owner, and optionally add your own user to this group:

```
sfm:x:<a group number>:www-data,<your user name>
```

Now change the group of the `supervisor.d` directory to sfm:

```
$ sudo chgrp sfm sfm/sfm/supervisor.d
```

Once the supervsiord.conf file and the respective permissions are setup, supervsiord needs to be configured to manage the sub-process under it.

5.1 Streamsample setup

A template streamsample configuration file "streamsample.conf.template" is included in the SFM distribution. To set up a streamsample process managed by supervisor:

Browse to the supervisord.d directory and copy streamsample.conf.template to streamsample.conf

```
$ cd sfm/sfm/supervisor.d
$ cp streamsample.conf.template streamsample.conf
```

Edit streamsample.conf to use the path to your sfm project, the value of the PATH environment variable set within your virtualenv, and to use your preferred system user account (to avoid having the output files owned by root).

To have supervisor refresh its list of configuration files and start the streamsample process, first run supervisorctl:

```
$ sudo supervisorctl
```

If you don't see a line that reads something like:

> streamsample RUNNING pid 889, uptime 21:45:25

then at the supervisor prompt, run 'update' to reload the config files:

```
$ supervisor> update
```

Running update should result in the following message:

> streamsample: added process group

Now verify that streamsample has been started by viewing the status of the processes:

```
$ supervisor> status
```

This should result in a list of processes which includes streamsample, for example:

> streamsample RUNNING pid 889, uptime 21:45:25

To stop the streamsample process, run supervisorctl and use the command

```
$ supervisor> stop streamsample
```

5.2 Filterstream setup

TwitterFilters in SFM are intended to create filterstream Twitter processes.

While streamsample must be started and stopped using supervisorctl, supervisor's management of TwitterFilter processes is mediated by the SFM application.

SFM creates configuration files for filterstream processes when an administrative user adds new TwitterFilters in SFM. The files are created in the sfm/sfm/supervisor.d directory. SFM takes care of updating supervisor so that it starts the new filterstream process.

If an administrative user modifies an existing, active TwitterFilter, SFM deletes the old configuration file for that TwitterFilter's filterstream process, writes a new configuration file containing the TwitterFilter's updated parameters, and restarts the filterstream process.

If an administrative user deactivates or deletes a TwitterFilter, SFM deletes the configuration file for that TwitterFilter's filterstream process, and stops the filterstream process.

5.3 OAuth constraints

To avoid triggering the Twitter API's rate limiting constraints, every SFM streaming connection must use a different set of Twitter credentials. SFM does not allow active filterstreams to run using the same Twitter credentials as streamsample, or as any other active filterstream.

The streamsample process connects to the Twitter API using the TWITTER_DEFAULT_USERNAME set in local_settings.py. Each Filterstream process connects to the Twitter API using the User configured in its TwitterFilter.

CHAPTER 6

Use Cases

6.1 Definitions

A *TwitterUser* in SFM is the entity used to collect tweets tweeted by the corresponding account in Twitter.

The fundamental, unique, unchanging identifier for a Twitter account is its numeric UID. The owner of a Twitter account might change the account's name, but the UID will never change.

Each TwitterUser is intended to map one-to-one to a Twitter account.

Given the rules above, we can derive two rules:

- A TwitterUser account should never be associated with tweets from more than one UID.
- A TwitterUser account's UID should never change.

6.2 Lifecycle of a TwitterUser

A TwitterUser in SFM exists in one of three states:

- *Nonexistence (Pre-creation/Post-deletion)*
- *Active* - SFM will attempt to collect new tweets for this TwitterUser, every time the *user_timeline* script is run.
- *Inactive* - The TwitterUser is still in SFM, but no new tweets will be collected while the TwitterUser is inactive.

An account in Twitter exists in one of four states:

- *Pre-creation*
- *Active/Public* - Tweets are visibile to anyone
- *Active/Protected* - Tweets are only visible to this account's Twitter followers
- *Deactivated/Deleted* - If an owner deactivates a Twitter account, Twitter places it on a queue to be permanently deleted after 30 days.

6.3 State Transitions

TwitterUser Creation - The SFM user provides the username of the Twitter account to map to this SFM TwitterUser. SFM looks up the Twitter account's UID by the username provided. If:

- the username matches a Twitter account's username, and

- the Twitter account's UID is not already associated with any TwitterUser

then SFM will create the TwitterUser. The SFM user may create the new TwitterUser as either Active or Inactive.

If the name does not match any Twitter account OR the UID is already associated with a TwitterUser, then SFM will not create the TwitterUser, even an in Inactive state.

Inactivation of Active TwitterUser - An SFM user marks an Active TwitterUser as Inactive. This transition is always allowed. Inactive TwitterUsers are still shown on the SFM page listing users, and CSV extracts

Activation of Inactive TwitterUser - An SFM user marks an Inactive TwitterUser as Active. SFM looks up the corresponding Twitter account by the UID of the TwitterUser. If the UID is valid, it updates the TwitterUser's name if differs from the current name of the Twitter account, and saves the TwitterUser as Active. If the UID is not found, which may occur if the Twitter account has been deactivated, then SFM does not allow the TwitterUser to be saved as Active.

Deletion of a TwitterUser - An SFM administrative user deletes a TwitterUser. This is always allowed. However, it is important to note that *all TwitterItems associated with this TwitterUser will also be deleted.*

Name Change - What if the owner of a Twitter account changes the name of the account? If the TwitterUser in SFM was created to collect tweets from this Twitter account, it should continue to do so, since the UID never changes. However, the name of the TwitterUser may temporarily still show the old name of the Twitter account. If a cron job has been set up to run update_usernames, then the name of the TwitterUser will automatically be updated to match the new Twitter account name the next time update_username is run. When update_username observes a name change, the old name will be appended to the TwitterUser's *former_names*, along with the date and time that the change was detected by update_usernames. As an example, if a Twitter account named `NYTimes` was then changed to `NewYorkTimes`, then the update_usernames script would update the name of the TwitterUser, and would also append to *former_names* so it might have a value like `{"Thu Jan 16 13:48:56 2014": "NYTimes"}`

Twitter Account Goes Protected - What if the owner of a Twitter account marks the account as protected?

If the `TWITTER_DEFAULT_USERNAME` configured in *local_settings.py* is a Twitter account which is following the protected Twitter account in question, then SFM will continue to be able to download tweets from that account.

If this is not the case, then user_timeline jobs will encounter errors when attempting to retrieve new tweets for the account. These errors will be recorded in the TwitterUserTimelineErrors table.

Twitter Account Deletion - What if the owner of a Twitter account deletes the account?

If the owner of a Twitter account deletes the account and there is an active TwitterUser mapped to the account, then user_timeline jobs will encounter errors when attempting to retrieve new tweets for the account. These errors will be recorded in the TwitterUserTimelineErrors table.

CHAPTER 7

CSV Export Data Dictionary

Social Feed Manager captures entire tweets, with all their data. To download selected, processed fields for each tweet in a user timeline, use the csv export option, available on each user page.

For more info about source tweet data, see the Twitter API documentation, including Tweets and Entities.

Field	Description	Example
sfm_id	SFM internal identifier for tweet	6114
created_at	UTC time when the tweet was created	2013-10-28T17:52:53Z
created_at_date	date in Excel-friendly format, MM/DD/YYYY	10/28/2013
twitter_id	Twitter identifier for the tweet	114749583439036416
screen_name	The screen name, handle, or alias that this user identifies themselves with. Screen_names are unique but subject to change.	NASA
followers_count	Number of followers this account had at the time the tweet was harvested	235
friends_count	Number of users this account is following at the time the tweet was harvested	114
retweet_count	Number of times the tweet has been retweeted at the time the tweet was harvested. If the tweet is a retweet AND the retweet was done using the Twitter retweet feature (i.e. is_reweet_strict = TRUE) the retweet_count reflects the retweet count for the original tweet. If the retweet was done by typing RT at the beginning (is_retweet_strict = FALSE) the retweet_count reflects retweets of the retweet.	25
hashtags	Hashtags which have been parsed out of the tweet text, separated by a comma and space	Mars, askNASA
in_reply_to_screen_name	If the tweet is a reply, the screen name of the original tweet's author	wiredscience
mentions	Other Twitter users mentioned in the text of the tweet, separated by comma and space.	@NASA_Airborne, @NASA_Ice
twitter_url	URL of the tweet. If the tweet is a retweet made using the Twitter retweet feature, the URL will redirect to the original tweet	http://twitter.com/NASA/status/394883921303056384 retweet redirecting to original tweet: http://twitter.com/NASA/status/394875351894994944
is_retweet_strict	Tweet is a retweet of another tweet, using Twitter's retweet function	FALSE
is_retweet	SFM's best guess at whether tweet is a retweet of another tweet; includes retweets accomplished using old-style method of placing RT in front of tweet	TRUE
text	The UTF-8 text of the tweet	Observing Hurricane Raymond Lashing Western Mexico: Low pressure System 96E developed quickly over the... http://t.co/YpffdKVrgm
url1	First URL in text of tweet, as shortened by Twitter	http://t.co/WGJ9VmoKME
url1_expanded	Expanded version of URL; URL entered by user and displayed in Twitter. May itself be a user-shortened URL, e.g. from bit.ly. Further expansion available in sfm web interface, not in csv export.	http://instagram.com/p/gA_zQ5IaCz/
url2	Second URL in text of tweet, as shortened by Twitter	
url2_expanded	Expanded version of URL; URL entered by user and displayed in Twitter. May itself be a user-shortened URL, e.g. from bit.ly. Further expansion available in SFM web interface, not in csv export	

CHAPTER 8

Frequently Asked Questions

8.1 Does Social Feed Manager capture photos and other media embedded in tweets?

As of version m5, no. But this is something we're looking forward to implementing in the near term.

8.2 How far back in time does SFM go when collecting a TwitterUser's tweets?

The Twitter API only provides up to the most recent 3200 tweets for an account. When a new TwitterUser is added in SFM, the user_timeline script will request as many tweets as the Twitter API can provide, i.e. up to 3200.

8.3 Does Social Feed Manager capture the followers list?

No. SFM does capture the number of followers at the time the tweet was retrieved. However, the Twitter API does provide a way to retrieve an account's follower list.

8.4 Do I have to set up supervisord in order to use filterstreams or streamsample?

No, filterstreams and streamsample can also be run manually using the *filterstream* and *streamsample* management commands described in the management commands page.

8.5 The number of retweets in a TwitterItem is inconsistent with the number of retweets shown on the tweet in Twitter. Why?

TwitterItems are created as they appear at the time they are captured from Twitter. However, tweets on Twitter can change afterwards; they can be further retweeted, they can be deleted, etc. In fact, an advantage of using SFM is that it takes a snapshot of tweets before they change or disappear!

Currently there is no way to "update" a TwitterItem with any changes that may have occurred to the corresponding tweet. This is something we might consider if there is a use case for it.

8.6 Does the SFM web interface provide a way to view the files generated by filterstream and streamsample?

Not yet.

8.7 When I click on the link to view a raw tweet, it's difficult to read in my browser.

There are a number of broswer plugins available (JSONovich, JSONView, and others) which improve the way that JSON is displayed.

8.8 Can I set this up on a Mac?

We haven't been running this on a Mac, but a colleague we met at Code4Lib 2014 has done it. Check out his blog post here: http://dicarve.blogspot.com/2014/04/an-relatively-easy-way-for-installing.html

CHAPTER 9

Troubleshooting

9.1 TwitterUserItemUrls is empty. Why isn't SFM fetching URLS?

Have you set up a cron job to run fetch_urls?

9.2 I tried to add a filterstream using the user that I've configured as TWITTER_DEFAULT_USER, but SFM is telling me that Streamsample is also configured to authenticate as that user. But I'm not using Streamsample!

SFM makes the assumption that streamsample either is being used or may be used in the future, and streamsample authenticates with the Twitter API using TWITTER_DEFAULT_USER. Due to Twitter API's rate limiting, SFM prevents the possibility of having multiple streams (in this case, streamsample and a filterstream) simultaneously calling the Twitter streaming API with the same Twitter user name.

CHAPTER 10

Release Notes

Release notes for the official SFM releases. Each release note will tell you what's new in each version, and will also describe any backwards-incompatible changes made in that version.

For those upgrading to a new version of SFM, you will need to check all the backwards-incompatible changes and deprecated features for each 'final' release from the one after your current SFM version, up to and including the new version.

Final Releases:

10.1 m5_001 release notes

m5_001 is release focused primarily on documentation. SFM now has substantial documentation on what it does, how it works, and how to use it.

Documentation contains a list of docs explaining:

- getting started
- the installation and working of Social Feed Manager
- its current use cases, where and how it's used now, and its scopes of enhancements
- user lifecycle
- features; how you can use them and automate them
- FAQ and troubleshooting

For more details visit the Social Feed Manager docs.

Non-Documentation Issues and Bugfixes

- #146 - Improved validation and error handling for TwitterUser.name

See the complete list of changes for milestone m5_001 in github.

10.2 m4_002 release notes

m4_002 improves process management under Supervisord. Previously it was necessary to start and stop SFM's supervisord-managed processes using the supervisorctl tool at the command line.

With m4_002, SFM now automatically starts and stops twitterfilter processes when TwitterFilters are created, activated, deactivated, or deleted by an SFM admin user.

A new management command, *fetch_tweets_by_id*, was also added. Given a list of tweet ids, the command fetches the associated tweets as JSON.

Significant issues and bugfixes

- #89 - Added management command to fetch tweets by a list of tweet ids.
- #154 - Enabled supervisord to pick up new twitterfilter conf files and initiate processes correctly.

See the complete list of changes for milestone m4_002 in github.

10.3 m4_001 release notes

m4_001 introduces collecting expanded urls in tweets, improves use of supervisord to manage multiple processes, and enhances organizedata to better structure data files. It also fixes bugs related to supervisord and cleans up twitteruser status and filterstream issues.

If you are upgrading an existing SFM instance from a version prior to m4_001, to m4_001 or newer, and your instance contains active TwitterFilters, then you will need to run the *createconf* management command.

Enhancements

Social Feed Manager has streamsample and filterstream management commands which are used to fetch random or filtered twitter feeds. These management process are automated using supervisord. Supervisord manages the streamsample and filterstream processes, starting and stopping these processes when required.

Supervisord control:

- #135 - streamsample and filterstream are managed by supervisord, SFM no longer requires manual run of these commands, if supervisord is set up, everything is handled by supervisord. This is done using the post_save signal sent from the UI to initiate these processes.
- #133 - Twitter API doesn't allow parallel streams like streamsample and filterstream to run concurrently with the same authorization credentials, so run a validation in the admin UI when adding the filters using twitterfilter, and validate that active streams do not conflict.
- #170 - To simplify naming, renamed rules in admin UI to twitterfilter and throughout SFM.

Twitter data organization:

- #132 - Re-fit organizedata to use subdirs for different filters.
- #119 - Added command and table to fetch and store expanded form of urls found in tweets.

Other issues and bugfixes

- #177 - Refactored signal call to createconf to be specific to the appropriate filter.
- #150 - Better handling of deactivation of TwitterUser status for no-longer Twitter-valid accounts, validating and throwing errors if name is not unique.

See the complete list of changes for milestone m4_001 in github.

CHAPTER 11

Indices and tables

- *genindex*
- *modindex*
- *search*