# FarmSubsidy.org Documentation

***Release DEV***

**OpenSpending / Holger Drewes**

March 15, 2014

Contents

This documentation is intended for developers who want to help out with the FarmSubsidy.org project which aims to bring transparency to EU farm subsidies and is run by the OpenSpending community.

CHAPTER 1

# Overview

- In the `Introduction` section you can find some information about FarmSubsidy.org and some hints where you might want to start reading to get an overview about European Union farm subsidy policy.
- The `Website` section is describing the structure of the website and the corresponding source code on GitHub.
- In the `Scraper` section people who want to help out can find information on how to write scapers for the different countries. Scrapers for the data are hosted in a separate GitHub repository.

**Note:** This document - especially the definition for new data format starting in 2013 in the `scraper` section (see: *Scraper Data Format*) is currently in **DRAFT** status, which means that there will be some changes in the next weeks. For recent changes see both the `website changelog` (*Changelog (Website)*) and the `scraper_changelog` (*Changelog (Scraper)*).

*This doesn't mean, that you shouldn't write your scraper yet* since life is a long quite (and ever changing) river and 80+ percent of your work will be able to be kept. Just be aware that there might be the need of some last changes to get everything to work!

For remarks or questions please contact holgerd77 on Github (March 15th, 2014).

**Note:** New **farmsubsidy data** for **2013** should be release around **April/May 2014**.

CHAPTER 2

# User Manual

## 2.1 Introduction

### 2.1.1 Current Situation

FarmSubsidy is a website that collects the payment data of the Common Agriculture Policy (CAP) which represents about a third of the EU budget. It was run by a group of journalists and activists for the past years. In 2013 the OpenSpending project of the Open Knowledge Foundation took over responsibility of the website.

Since the old scrapers were proprietary and are not available any more one of the main tasks in the current situation is to build a new set of scrapers for each country released under an open licence which can be maintained by the community.

This documentation is intended to give an overview about the requirements for those scrapers so that the data they provide can be integrated seamlessly and without too much hassle into the existing environment.

So if you want to help, regardless if you are new to Farmsubsidy or if you have already accompanied the project over the years: your work will be highly appreciated. This is an extremly important topic, and since there are 28 member states in the European Union there is a good chance that you'll get your favorite country to write a scraper for! :-)

### 2.1.2 Background Information

If you need an introduction to the topic of farm subsidies, you can have a look at the Wikipedia article about EU agricultural policiy or at the CAP website from the European Commision. The wikipedia article is quite long, for an introduction to the structure of the subsidies just read the The CAP today section.

The main thing you have to know, is that European agricultural subsidies are divided into two funds. *The European Agricultural Guarantee Fund (EAGF)* (Wikipedia > EAGF) is for direct payments for farmers (majority of payments) and for measures to respond to market disturbances. You will find these two posts separately distinguished in the data provided.

The second fund is the *European Agricultural Fund for Rural Development (EAFRD)* (Wikipedia > EAFRD).

Since EU funding changed over time, there were other so called `schemes` of payment though and are also still today, have a look at the *Scheme* description in the data model chapter.

### 2.1.3 Further Reading

If you want to do some further reading, here are some sources to start:

- News section on the Farmsubsidy website

- Googling "farm subsidies EU"
- Same on Google News

## 2.2 Website

This is the developer documentation for the **Farmsubsidy website**, located under the following url:

- http://farmsubsidy.openspending.org

**Sources** for the website can be found on **GitHub**:

- https://github.com/openspending/Farm-Subsidy/

The website is build with `Python` using `Django` as a web framework.

### 2.2.1 Installation

#### Requirements

The following list contains only the most central requirements to get an overview which software components are used. For a complete overview have a look at the requirements.txt file on GitHub.

- Django 1.5.x
- Haystack 2.0.x for search (GitHub Fork with modifications)
- django-registration for user login
- django-piston for the API

The website uses `South` for DB migrations/changes:

- South

#### Installation process

##### 1) Get a copy of the project

Git clone the project:

```
git clone git@github.com:openspending/Farm-Subsidy.git #or use https
```

##### 2) Install requirements

Set up `virtualenv` and `pip` and install the requirements:

```
virtualenv venv
source venv/bin/activate
pip install -r requirements.txt
```

#### 3) Configure Django

The Django project is located in the `web` folder.

The Django `settings.py` file is split into two separate files. `global_settings.py` contains settings which shouldn't change in a deployment, `settings.py.template` contains settings which should be adopted in a new deployment.

Create a copy `settings.py` from `settings.py.template` and adopt the settings to your needs.

#### 4) Install PostgreSQL

Farmsubsidy code uses some `SQL` syntax which is not compatible with `SQLite` and the website is intended to handle/present large amounts of data, so you have to start directly with a native DB and ommit a test installation with `SQLite`. Since Farmsubsidy is build and tested with `PostgreSQL`, a `PostgreSQL` installation is recommended.

If you haven't that much experience with installing databases: it's not as painful as you might think, for the mac e.g. there is a client which can be installed and is up and running with one click: http://postgresapp.com/

Open `psql` and create a new DB with:

```
CREATE DATABASE farm_geo;
```

If you are just running a test installation on localhost not using a username and passwort (don't do that in production) this should already do the trick!

#### 5) Sync/migrate the DB

Since there is an old `GeoDjango` dependency in the `South` migrations, early migrations won't work without hassle, so sync all apps with `syncdb`:

```
cd web
python manage.py syncdb --all
```

For getting `South` back to work again, first list all apps which uses migrations:

```
python manage.py migrate --list
```

Then do fake migrations to the latest migration for all apps, e.g.:

```
python manage.py migrate data LATESTMIGRATIONNUMBER --fake
```

#### 6) Install Haystack backend

If you use `Whoosh` as a backend for Haystack, you have to install it (older version due to dependencies):

```
pip install whoosh==2.4
```

#### 7) Temporary: create payment_totals.txt

This is due to some legacy code and will be removed as soon as possible:

Create a textfile `data/stats/payment_totals.txt` (from repository root, not from `web` directory) and enter some fake numbers like this:

```
1000000,100000
```

#### 8) Run the server

Run the development server with:

```
python manage.py runserver
```

You should be able to see the farm subsidy website under the URL provided and enter the admin area.

### Post-installation hacks

Execute the following SQL manually in case your columns don't fit (it can't be migrated):

```
ALTER TABLE data_recipient ALTER COLUMN total SET DEFAULT 0.0;
ALTER TABLE data_countryyear ALTER COLUMN total SET DEFAULT 0.0;
ALTER TABLE data_recipientyear ALTER COLUMN total SET DEFAULT 0.0;
ALTER TABLE data_scheme ALTER COLUMN total SET DEFAULT 0.0;
ALTER TABLE data_schemeyear ALTER COLUMN total SET DEFAULT 0.0;
ALTER TABLE data_recipientschemeyear ALTER COLUMN total SET DEFAULT 0.0;
ALTER TABLE data_totalyear ALTER COLUMN total SET DEFAULT 0.0;
```

It's needed to make `total` columns default to `0.0`.

## 2.2.2 Source Overview

### Apps

Like all Django projects the Farmsubsidy website is organized in different Django apps. Here is a list of the existing apps with a short description. Don't take the *Importance* column too serious, it is just for rough orientation:

| App | Description | URL paths | Status | Importance |
|---|---|---|---|---|
| api | API for farmsubsidy | /api/ | inactive | + |
| comments | | | | |
| countryinfo | App for transparency index | /transparency/ | active | ++ |
| data | **Central app, data structure** | /, /ES/* | active | +++ |
| features | News and reports app | /news/* | active | + |
| feeds | | | | |
| frontend | Annotation management for users | /myaccount/* | active | + |
| graphs | Graph visualisation | /graph/* | inactive | o |
| listmaker | Experimental, recipient lists | /lists/* | inactive | + |
| petition | Special petition app, ignore | /petition/* | inactive | o |
| search | Haystack search | /search/* | active | ++ |

Other folders:

| Folder | Description |
|---|---|
| locale | Minimal french localization file, ignore |
| media | CSS, images and Javascript |
| misc | Small helper classes and functions |
| templates | Folder for common templates |

### 2.2.3 Loading Data

#### Data model

You can find the main data structure in the models.py file of the `data` app.

The core models are:

##### Recipient

A `recipient` is a receiver of subsidy payments and is in most cases a company or governmental institution.

There are no unique recipient IDs provided by the EU, so the IDs are provided internally by the system. The central identifying attribute for the recipient is the `name` attribute, though there will sometimes be double entries for the same entities due to inconsistencies in the source data.

Most other information like adress data or geo information is not mandatory.

##### Scheme

A `scheme` is identifying a type of payment. Since the structure of the EU subsidy system has changed over the years you can also find different type of schemes for the payments, examples are:

- Export subsidies
- Market regulations
- School Milk (yeah, healthy :-))
- ...

In the last years, the dominating schemes are:

- European Agricultural Fund for Rural Development (EAFRD)
- Direct payments under European Agricultural Guarantee Fund (EAGF direct)
- Other payments under European Agricultural Guarantee Fund (EAGF other)

See also the *Background Information* chapter for where to read about this.

##### Payment

A `payment` is a paid subsidy for a certain `recipient` connected with an existing `scheme` for a special year. There can be several payments per year for different schemes for the same recipient.

#### Loading aggregated data (up to year 2012)

For Farmsubsidy there are aggregated data files up to the farm subsidy data for 2012.

##### Download the data

You can download the aggregated data files in `CSV` format under the following URL:

- http://data.farmsubsidy.org

Data for a single country is provided in a packaged format, e.g.:

- http://data.farmsubsidy.org/AT.tar.bz2

Put the data in the data folder in the following format:

```
data/csv/<CountryCode>/payment.txt
```

You need the following files there:

- `payment.txt`
- `recipient.txt`
- `scheme.txt`

#### Import the data

Now you can import the data with custom Django management commands, e.g. for Austria:

```
python manage.py copier -c AT #takes some time...
```

### Loading year-by-year data (year 2013 or newer)

Starting with the data for 2013 there are some changes in the data integration process going along with the introduction of the new Farmsubsidy GitHub scraper repository.

Data is now scraped and stored on a year-by-year basis (see: *Scraper Data Format*) and has to be put in the data format for import in the following form:

```
data/csv/<CountryCode>/payment_2013.txt
```

There is a new management command *load_year_data* in the *data* app of the Farmsubsidy sources which can be used like this:

```
python manage.py load_year_data COUNTRY YEAR DELIMITER [--simulate] [--ignore-existing]
```

This management command loads data from the new simplified data format. It tries to match recipients by `name` attribute and connects a payment either to a matched recipient or creates a new one if no match was found. You can run the command with the `--simulate` option to get an impression of how many recipients would be matched.

The `--ignore-existing` option lets you ignore already existing entries for the given year and country in the DB, otherwise there would be an error message.

---

**Note:** This management command is still in a *BETA* stadium. If you use it for integration of data in the production deployment please check how the data is integrated, if everthing is at the right place and if format, attributes and number of payments are correct. Have a look at the code on GitHub and correct if necessary!

Note that there is also a new ID format for new `recipient` and `payment` entries calles `ZID`. This is for easier ordering and determining the latest IDs, since IDs are stored in text format (ahum :-)) at the moment, which leads to ordering like this: "GB1, GB892, GB99".

`ZIDs` are stored in a format like this: "[COUNTRY_CODE]Z[ID Number + 0s leading to 7 ciphers]", leading to orderings like: "GBZ0000001, GBZ0000099, GBZ0000892".

Please be careful here. It is not yet fully determined, if the introduction of a new ID format has negative hidden side effects on other places (if you know, drop a note). At the moment `ZIDs` are also quite (too) short due to a currently existing limitation of `max_length=10` for the ID fields.

---

#### Post-integration data processing

##### Data denormalization

At the moment there is some data denormalization going on reorganizing the data into different tables for performance purposes:

```
python manage.py normalize -c AT #takes even longer...
```

Repeat that for every country or test with data for just one country.

Run a `VACUUM VERBOSE ANALYZE` on all database tables afterwards (make sure you are connected to the correct database before, on `psql`: `\c farm_geo`).

Now you should be able to browse the imported data on the local website and see the list of `recipients` in the Django admin area.

##### Update the search index

When all/some countries are imported, run search indexing:

```
python manage.py fs_update_index

#yes, you guessed it, don't drink too much coffee :-)...
#For this step you can definitely go away and do something else.
```

Now you should be able to use the search box on the website and get some results.

##### Update total payments number

After this you can update the total payments number on the front page like this:

```
python manage.py payment_totals #This is quick. Whew. :-)
```

### 2.2.4 Testing

Test coverage is poor, but new tests are being written all the time, as my resolution is not to fix any bug without writing a test for it first.

Some tests only test code, but mostly the tests are there for making sure the database is being processes correctly in the (de)normalization process.

Because there is quite a large dataset (to make testing better) it's highly recommended that a persistent test database is set up and the persistent test runner from Django Test Utils is used.

The initial data for the recipient, payment and scheme model is found in `./web/data/fixtures/data.sql`. This should be loaded in to the `test_[db_name]` database before running the tests.

Below are the steps that should be taken, assuming the code is actually running:

1. Install `django-test-utils` and append `test_utils` to `INSTALLED_APPS` in `settings.py` (see comment there)
2. Create the test database somehow. I find this is easiest done by running `./manage.py testserver` as this doesn't destroy the database on exit. You could also prefix the database name in settings with `test_`, syncdb and then change it back again.

3. Load the data in `./web/data/fixtures/data.sql` in to the new database. This isn't added automatically because of the time it takes to run tests without the persistent database.
4. run `./manage.py quicktest`

### 2.2.5 Changelog (Website)

Changelog for the development of the website.

**Current Changes (version not yet determined)** (2014-03-15)

- Added new section in docs for `Website` development documentation (see: *Website*)
- Added detailed installation instructions for website/DB deployment (see: *Installation*)
- Integration fragmented doc files of GitHub repository in new `Sphinx` documentation
- Added source code description in docs with app overview (see: *Source Overview*)
- Added information about how to load data in the DB (see: *Loading Data*)
- Added new management command `load_year_data` in `data` app on GitHub for loading year specific data in new data format starting with the 2013 data. Data loading can be simulated with `--simulate`, new recipients are matched by `name` attribute against existing recipients. New `ZID` ID format for `payments` and `recipients`. (see load_year_data.py file on GitHub)
- Added documentation about how to use management command `load_year_data`, additional infos about current stadium and precautions when using (see: *Loading year-by-year data (year 2013 or newer)*)

## 2.3 Scraper

This part of the documentation describes how to write scrapers for the Farmsubsidy project.

### 2.3.1 How to contribute

There is a separate **GitHub repository** collecting the different **scrapers**:

- https://github.com/openspending/farmsubsidy-scrapers

In the repository there is a **separate folder** for **each scraper**, named by the country code of the scraper, e.g. `at` for Austria:

- https://github.com/openspending/farmsubsidy-scrapers/tree/master/at

There is also a very detailed **Data Overview document** on **Google Docs**. Public data urls are not up-to-date any more, but make sure to notice the **Data Documentation URLs** at the end of the `detail view` sheet:

- Data Overview Document (Google Docs)

For each scraper there is a corresponding issue on GitHub. If you want to help out with a scraper have a look at the open issues, see if there is already somebody responsible and drop a note to avoid that there are several people working on the same scraper in parallel.

---

**Note:** Before you write your scraper:

- Check, if there is a **download button** on the data website (I actually didn't when I wrote the example scraper! :-))

- Do some **creative googling** if someone else already has written a scraper for the site! If so: try to get in contact and ask if the scraper can be used under an open licence.

---

### 2.3.2 Data Sources

Data is provided on a country-by-country basis. Mostly you will find a web form where you can filter the data by things like year, amount or the region and get back an HTML table with the single payments. Sometimes data is also provided in a downloadable format.

Here are some examples:

- Belgium (just hit *search* button)
- UK (get data by searching for amount > 1.000.000)
- Germany (get data by searching for EGFL > 1.000.000)
- Slovenia (get data by selecting a sum and search)

You can find the relevant data source on the issue page of a country in the GitHub repo.

### 2.3.3 Format of the existing data files

To get an idea of how data is structured in the DB have a look at the *Data model* chapter in the Website documentation section.

You can find the data scraped by the old proprietary scrapers in the following folder:

- http://data.farmsubsidy.org

Please download one of the compressed country data files and unpack it, e.g.:

http://data.farmsubsidy.org/AT.tar.bz2

A data package consists of the following files:

```
recipient.txt
scheme.txt
payment.txt
```

Due to its size you probably can't open `recipient.txt` or `payment.txt` with a visual editor but need an editor like `vi` and use it from the command line.

Each file contains the data for the corresponding data model in `CSV` format, here are some extracts.

Start of a recipient.txt file:

```
"recipientId";"recipientIdx";"globalRecipientId";"globalRecipientIdx";"name";"address1";"address2";"
1;1;"AT1";"AT1";"Adrigan Barbara";;;;;"AT";"AT";"Burgenland";"Lutzmannsburg";;;;;;;;
2;2;"AT2";"AT2";"Aibler Maria";;;;;"AT";"AT";"Burgenland";"Zillingtal";;;;;;;;
3;3;"AT3";"AT3";"Allacher Ilse und Matthias";;;;;"AT";"AT";"Burgenland";"Gols";;;;;;;;
```

Complete scheme.txt file:

```
"GlobalSchemeId";"nameNationalLanguage";"nameEnglish";"budgetlines8Digit";"countryPayment"
"AT7";"Öffentliche Lagerhaltung (Intervention)";"Intervention";;"AT"
"AT2";"Indirect";"Indirect";;"AT"
"AT1";"Direct";"Direct payments under European Agricultural Guarantee Fund";;"AT"
```

Start of a payment.txt file:

```
"paymentId";"globalPaymentId";"globalRecipientId";"globalRecipientIdx";"globalSchemeId";"amountEuro";
1223535;"AT1223535";"AT317577";"AT317577";"AT5";11733.75;;"2008";"AT"
1223536;"AT1223536";"AT12327";"AT12327";"AT5";36445.65;;"2008";"AT"
1223537;"AT1223537";"AT44239";"AT44239";"AT5";82.10;;"2008";"AT"
```

### 2.3.4 Scraper Data Format

#### CSV Format

The new GitHub scrapers will be used to scrape farmsubsidy data for the year `2013` and newer and only have to output a `payment` file with a reduced data format and no `recipient` and `scheme` files. Please write your scraper so that it will take the `year` as an input parameter and writes files like this:

```
payment_YEAR.txt
```

The reduced data format looks like the following:

```
"rName";"rAddress1";"rAddress2";"rZipcode";"rTown";"globalSchemeId";"amountEuro";
"Nordmilch AG";;;;;"D1";15239.34;
"Emsland-Stärke GmbH";Am Bahnhof 4B;;15938;Golßen;"D2";32305.45;
...
```

The scraped data will be loaded into the database with a (yet to be written) Django management command. Recipient names will be matched against existing recipient names.

The following table describe the single attribute formats.

| Attribute | Description | Mandatory | Data Type |
|---|---|---|---|
| rName | Name of recipient | YES | String |
| rAdress1 | Adress field 1 for recipient (Street) | NO | String |
| rAdress2 | Adress field 2 for recipient (other) | NO | String |
| rZipcode | Zipcode of recipient town | NO | String |
| rTown | Town of recipient | NO | String |
| globalSchemeID | Scheme ID from existing scheme.txt | YES | String |
| amountEuro | Amount in Euro (1) | YES(or 2) | Float |
| amountNationalCurrency | Amount in national currency (2) | YES(or 1) | Float |

**Note:** Since the names you scrape will be later matched against the names already existing in the database please make some searches on the Farmsubsidy website and see, how names are formatted there. Try to keep names written as they are on the website so matching will be easier and double entries will be prevented.

**Note:** For the scheme ID please take an existing scheme ID from the `scheme.txt` file of the country (see *Format of the existing data files*). If you can't find a fitting scheme ID ask on the GitHub issue page and use a temporary schemeID like `AT-TMP1`.

**Note:** Please provide either the amount in Euro or in the national currency (e.g. for UK). Don't make any implicit conversions, leave field not provided blank!

#### UTF-8 Encoding

Please make sure that you use `UTF-8` as an encoding for your output file format and keep recipient data in the original language and characters.

Here are some examples:

- Bólyi Mezőgazdasági Termelő és Kereskedelmi Zrt. (Hungary)
- GREENGROW spółka z ograniczonš odpowiedzialnociš (Poland)
- Südzucker GmbH (Germany)
- Alcoholes Gcía de la Cruz Vega (Spain)

### 2.3.5 Technology

At the moment, the following technologies/programming languages for scrapers are supported:

#### Python/Scrapy

##### Introduction

Scrapy is a python scraping framework with a lot of built in scraping functionality, for introductory information see the `Scrapy` website:

- Scrapy

##### Installation

For running a Scrapy spider, please install the Scrapy version from the requirements file:

- Requirements file: requirements_scrapy.txt

You can find a Scrapy project deployment in the GitHub repository in the `scrapy_fs` folder. In this deployment, there is already the data structure defined in the `items.py` file.

##### Writing a spider

There is a reference implementation for a scrapy spider for the GB website. The spider can be found at (Link:

`scrapy_fs/scrapy_fs/spiders/gb_spider.py`

If you want to write a spider with Scrapy, please add/name your spider in an analog way and write a note in the root `gb` (`COUNTRY_CODE`) directory that the spider is being realized with Scrapy.

A Scrapy spider can be executed like that from the `scrapy_fs` directory:

```
scrapy crawl GB -a year=YEAR
```

A `CSV` output can be generated like this:

```
scrapy crawl GB -a year=2012 -o payment_2012.txt -t csv
```

**Note:** Scrapy won't maintain the order of the attributes of the csv file. That's ok.

#### Python

If you have your own preferred way of writing scrapers with `Python`, you can do that as well. Then please write your scraper in a form, that it can be executed from the command line. Add the requirements you need to the global python requirements file:

- Global requirements file: requirements_python.txt

**Note:** If you've written a Python scraper you think can serve as a good starting point for other scrapers and can be entered here as a reference implementation, please drop a note!

#### Ruby

You can also write a `Ruby` scraper, please also create the scraper in a command line-executable form.

Add your requirements to the global Ruby Gemfile:

- Global Gemfile: Gemfile

**Note:** If you've written a Ruby scraper you think can serve as a good starting point for other scrapers and can be entered here as a reference implementation, please drop a note!

#### Other

If you have another technology you want to use, please ask the person currently responsible for maintaining the Scrapers (try on GitHub). The reason for limiting the technologies a bit is that all scrapers for the different countries have to be maintained and an executable environment have to be kept up to be able to run the scraper from a central location independently from the creators.

### 2.3.6 Changelog (Scraper)

This changelog deals mainly with the `data format definition` for the scrapers (see: *Scraper Data Format*) and the `technology supported` in the scraper repository (see: *Technology*).

**Changes in version DRAFT1** (2014-03-15)

- Added basic documentation for scraper repository (see: *Scraper*)
- Added documentation for existing data format (see: *Format of the existing data files*)
- Added *DRAFT* definition for a new simplified scraper data format (see: *Scraper Data Format*)
- Added section for supported technologies (see: *Technology*)
- Added `scrapy_fs` Scrapy project on GitHub for unified Scrapy scraper development, `items.py` file for data format definition, `pipelines.py` file for basic data format validation (see: GitHub scrapy_fs directory)
- Added Scrapy reference scraper for `GB` (see gb_spider.py on GitHub)
- Documentation about how to execute Scrapy spiders (see: *Python/Scrapy*)

CHAPTER 3

# Indices and tables

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