# openfarmsubsidies Documentation

*Release 0.1-pre-alpha*

Holger Drewes

Jun 28, 2017

# Contents

This is the project documentation for `openfarmsubsidies`, a new approach on building an open scraper infrastructure and user interface for researching EU farmsubsidy payments, build upon the experiences made during work on historically grown Farmsubsidy.org tech infrastructure.

The project has its own organization on GitHub and consists of the following repositories:

| Name (GitHub) | Description | Status | Last Status Update |
|---|---|---|---|
| openfarmsubsidies-scraper | Scraping | `Beta` | 2017-06-28 |
| openfarmsubsidies-elastic | Search Index | `Beta` | 2017-06-28 |
| openfarmsubsidies-backend-api | Backend/API | `Beta` | 2017-06-28 |
| openfarmsubsidies-frontend | Frontend | `Beta` | 2017-06-28 |

This documentation provides guidance for installation, setup and technical aspects for the different sub modules.

Contents:

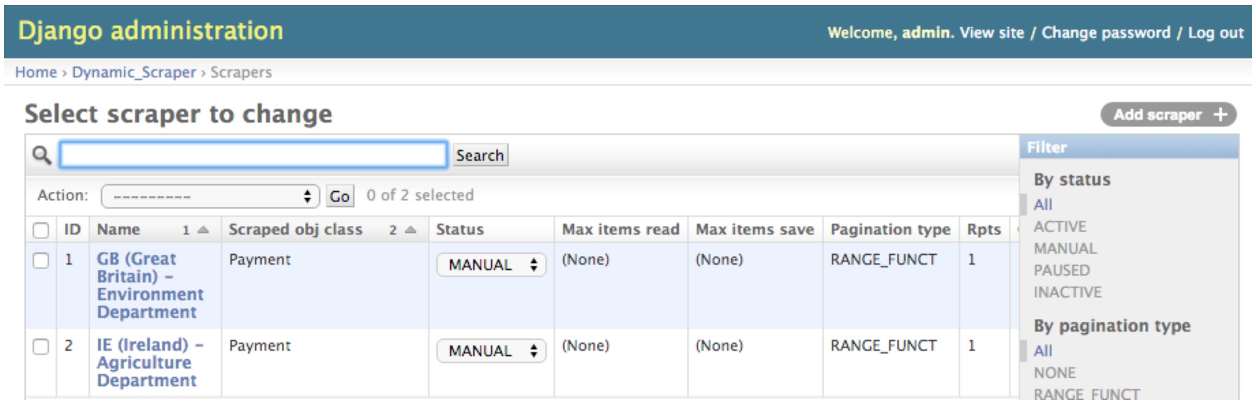
CHAPTER 1

# Scraper (Django/DDS)

## Introduction

The scraping infrastructure is build on `Python/Django` and is using the djang-dynamic-scraper scraping library at its core.

Scrapers for the various EU member state agencies databases are build and maintained within the Django admin interface.



Django administration

Welcome, **admin**. View site / Change password / Log out

Home › Dynamic_Scraper › Scrapers

Select scraper to change

Add scraper +

Search

Action: ---------- Go 0 of 2 selected

| | ID | Name 1 | Scraped obj class 2 | Status | Max items read | Max items save | Pagination type | Rpts |
|---|---|---|---|---|---|---|---|---|
| ☐ | 1 | GB (Great Britain) - Environment Department | Payment | MANUAL | (None) | (None) | RANGE_FUNCT | 1 |
| ☐ | 2 | IE (Ireland) - Agriculture Department | Payment | MANUAL | (None) | (None) | RANGE_FUNCT | 1 |

Filter

By status
- All
- ACTIVE
- MANUAL
- PAUSED
- INACTIVE

By pagination type
- All
- NONE
- RANGE_FUNCT

## Installation

The scraping infrastructure project can be installed by cloning the GitHub repository and install the requirements into a `Python 3.5` virtualenv with:

```
pip install -r requirements.txt
pip install -r requirements_dev.txt # DEV requirements
```

The project uses the following main `Python/Django` libraries:

- Django 1.10
- Scrapy 1.4
- Django Dynamic Scraper (DDS) 0.13

## Configuration

The following environment variables have to be found in your shell environment, e.g. by adding lines like `export OPENFARMSUBSIDIES_SECRET_KEY="..."` to the `.bash_profile` file:

| Key | Description | Place |
|---|---|---|
| `OPENFARMSUBSIDIES_SECRET_KEY` | Project specific Django secret key | `settings.py` |

Starting a local Django server should now provide access to the scraper management admin console via the browser (go to `127.0.0.1:8000`):

```
python manage.py runserver
```

## Scraper Handling

### Importing/exporting Scrapers

Scrapers can be found in the `scraper_dumps` directory inside the repository and imported with the following command:

```
python manage.py loaddata scraper_dumps/farmsubsidy_scraper_dump_YYYY-MM-DD_dds_[DDS_
↪VERSION_NUMBER].json #Generic
python manage.py loaddata scraper_dumps/farmsubsidy_scraper_dump_2016-01-18_dds_v094.
↪json #Example
```

**Note:** It is recommended to match the project installation DDS version with the version from the scraper dump, otherwise DB changes during DDS version changes have to be looked at closely in the DDS release notes and manual adoptions to the JSON dump format might be necessary.

### Creating a new Scraper

For creating scrapers a `ScrapedObjectClass` `Payment` has to be defined in the Django admin (see the definition from the scraper dumps) in addition to the `models.py` definition, defining the data structure of the scraped payment data. See the *Data Format* section for description of the different payment attributes.

Scrapers are created per-country wise as `Scraper` objects in the Django admin and are referenced in additional `Country` objects, representing a EU member states respectively the associated payments agency.

For further documentation and conceptional overview see the DDS docs

### Running a Scraper

Scraper can be run from the command line with the following command:

```
scrapy crawl --output=data.json --output-format=jsonlines payment_spider -L DEBUG -a␣
↪id=GB -a max_items_read=4 -a max_pages_read=2
```

This will run the scraper connected to the `Agency` in the Django admin with the id `GB` and write the output in a `JSON Lines` formatted file called `data.json`.

Usage options for scraping behaviour can be found in the corresponding DDS doc section for running/testing scrapers.

## Data Format

### Scraper Format Description

The following are the scraped object attributes of a `Payment` object. Note that some field are either filled in with static values (like the `country` attribute) or are automatically filled via external API (like the `name_en`) attribute and are not directly used in the scraping process:

| Attribute | Scraped Field | Mandatory | Description |
|---|---|---|---|
| base | Yes | Yes | Container atribute for element scraping, see DDS docs |
| name | Yes | Yes | Name of recipient scraped from the site, use both for en and non-en names |
| name_en | No | No | Use ONLY if name should be translated via Yandex API, add static placeholder processor |
| country | No | Yes | Always add, with static processor inserting the two-letter country code |
| zip_code | Yes | No | ZIP code of recipient |
| town | Yes | Yes | Town of recipient |
| region | Yes | No | Region of recipient |
| year | No | Yes | The year of the scraped data, add static placeholder processor |
| amount_nc | Yes | Non-€-Country | Use for scraping of non-€-country amounts |
| nc_conv_date | No | Non-€-Country | Always use this and following field together with static processor... |
| nc_conv_rate | No | Non-€-Country | ...for non-€-countries |
| amount_euro | Yes(€)/No | Yes | Use for scraping of €-country amounts, otherwiese static processor |
| sub_payments_nc | Indirect | No | Use for scraping sub payments of non-€-countries if available (see extra expl.) |
| sub_payments_euro | Indirect(€)/No | No | Use for scraping sub payments of €-country amounts, otherwiese static processor |
| sp-x (sp1,sp2,..) | Yes | No | Additional helper attributes for sub payments, both € and non-€ |
| extra_dp_url_123 | Yes | No | Helper attributes for scraping additional data urls per payment, see DDS docs |

**Additional note on sub_payment scraping:**

Sub payments are indirectly scraped via the `sp-x` fields and then added via placeholders into a static processor template of the `sub_payments_nc` or `sub_payments_euro` field.

Only scrape sub payments if the two (without market measures)/three main agricultural subsidy pillars are listed, otherwise things get divided into too small sections. Use the english naming translation in the following unified form (for easier/useful faceting later on search):

| Payment Type | Sub Payment Name | Remarks |
|---|---|---|
| European Agricultural Guarantee Fund (EAGF) | EAGF (Direct Payments) | Direct payments to farmers, largest part |
| European Agricultural Fund for Rural Development (EAFRD) | EAFRD (Rural Development) | Environmental measures, sometimes: ELER, smaller part |
| Market Measures (e.g. for milk, fruit market) | Market Schemes | Only sometimes |

The scraped sub payments don't have to sum up to the total subsidy sum, so you can also pick the 2/3 most common ones. For other payments than the ones above use an english name translation.

The following is an exemplary static processor template for the Bulgarian scraper:

```
'static': 'EAGF (Direct Payments),{sp1} | EAFRD (Rural Development),{sp2} | Market␣
↪Schemes,{sp3}'
```

In this case the scraper definition also has to provide entries with `XPath` definitions for the `sp1`, `sp2`, `sp3` and `sp4` fields. The so-scraped values are then automatically added to the static processor text replacing the placeholders.

General format for the sub_payment string:

```
'static': '[Name of SP1],{sp1} | [Name of SP2],{sp2} | ...'
```

**Note:** Is is possible to add up to six sub payment types to the scraper. The amounts of the sub payments doesn't have to add up to the total amount of the payment.

## Output Format Description

Scraped items are saved with additional serialization customizations defined in the `models.py` module as `JSON Lines` items, more or less (one additional processing is necessary) ready to be indexed in the `Elastic` index.

If currency is scraped in national unit conversion rate and date is read from fixer.io API.

Data format looks like the following:

```
{
  "town": "PERTH",
  "amount_nc": 57444.0,
  "name": "\"A F Angelil T/A \"\"Cluny Estate\"\"\"",
  "amount_euro": 76126.11,
  "country": "GB",
  "sub_payments_euro": [{
    "amount": 32969.45,
    "name": "Rural Development"
  }, {
    "amount": 43156.83,
    "name": "Direct Aid"
  }, {
    "amount": 0.0,
    "name": "Market Schemes"
  }],
  "sub_payments_nc": [{
    "amount": 24878.42,
    "name": "Rural Development"
  }, {
    "amount": 32565.71,
    "name": "Direct Aid"
  }, {
    "amount": 0.0,
    "name": "Market Schemes"
  }],
  "year": 2015,
  "nc_conv_rate": 0.75459,
  "nc_conv_date": "2016-01-22",
  "zip_code": "PH2"
}
```

### Recipient Name Translation

For recipient name translation the `Yandex` translation API is used. `YANDEX_TRANSLATE_API_ENDPOINT` and `YANDEX_TRANSLATE_API_KEY` have to be set in `settings.py` file.

Translation is automatically activated if `name_en` attribute is added to a scraper of a specific country, leave attribute for scrapers with no translation (e.g. `GB`).

Yandex has the current API limits:

- 1.000.000 characters per day
- 10.000.000 characters per month

`OpenFarmsubsidies` scraping is coming close, so API usage has to be actively managed/recorded to avoid reaching limitations.

Take the following formula for character estimates:

- (Number of recipients (`wc -l`)) * 15 characters/recipient

Try to stay under 80% of day/month limit, distribute (translated) scraper runs to different days, avoid double runs.

## Creating the Countries Endpoint

The `countries` endpoint of the API (see: *Countries Endpoint*) is taking the administrated data from the `Country` Django model objects as a starting point.

There is a `create_countries_endpoint` Django management command providing the `JSON` output for the API response:

```
python manage.py create_countries_endpoint
```

Recreate the API endpoint every time a country is added and integrate it in the Backend/API python code.

**Note:** You can exclude a country by setting the corresponding scraper to `inactive` status.

CHAPTER 2

# Search Index (Elastic)

Searching is done with `Elastic`, currently the following version is used:

- Elastic 2.1.1

## Installation

Download `Elastic` and install in folder `elasticsearch` (no version number) inside the repository.

The local dev server on `http://localhost:9200` can then be started with:

```
./server.sh
```

## Index Creation

### Index Template

For indexing template in `conf/template.json` is used for mapping and has to be activated/loaded before first data indexing:

```
curl -XPUT localhost:9200/_template/template_1 -d '@conf/template.json'
```

The current mapping for the index can be seen with:

```
curl -XGET 'http://localhost:9200/openfarmsubsidies/_mapping/payment?pretty'
```

Deleting the current template:

```
curl -XDELETE localhost:9200/_template/template_1
```

See installed templates:

```
curl -XGET localhost:9200/_template/
```

### Index Management

List indices:

```
curl 'localhost:9200/_cat/indices?v'
```

Delete index:

```
curl -XDELETE 'localhost:9200/openfarmsubsidies?pretty'
```

## Indexing Documents

### Format Pre-Processing

Input files have to be formatted as `JSON Lines` format and are prepared with the following command for indexing:

```
./jl2elastic inputfile.json
```

### Indexing Documents

Index data:

```
curl -XPUT 'localhost:9200/openfarmsubsidies/payment/_bulk?pretty' --data-binary
↪"@data_elastic.json"
```

## Searching the Index

Testing search:

```
curl 'localhost:9200/openfarmsubsidies-test/_search?q=PERTH&pretty'
```

CHAPTER 3

# Backend/API (Python/Flask)

Backend/API for creating a simple API layer and connecting to `Elastic`.

## Installation

Installation is done by cloning the repository and install the dependencies from the requirements files:

```
pip install -r requirements.txt
pip install -r requirements_dev.txt # DEV requirements
```

The project uses `Python 3.5` and is build upon the following main `Python/Flask` libraries:

- Flask 0.10

The dev server on `http://127.0.0.1:5000` can be started with:

```
python app.py
```

## API

### General

Current version of the API: `v1`

#### Common Request Parameters

| Name | Description | Example Values |
|---|---|---|
| start | Result object to start with (default: 0) | 0, 9 (10th object!) |
| rows | Number of rows/objects to return (default: 10) | 1, 10, 25 |

### Common Behaviour

API always returns `aggregations` for `towns`, `years`, `countries` and `sub payments type`.

## Payments Endpoint

`Payments` endpoint can be reached at:

```
/[API_VERSION]/payments/
```

Results are sorted by `amount_euro` by default.

### Endpoint-specific Request Parameters

| Name | Description | Example Values |
|---|---|---|
| q | Generic search for recipient, town or ZIP code | Nestle,London,NR16 |
| name | Recipient name | Nestle |
| country | 2-letter country code of an EU country | GB,SI,NL,PL |
| zip_code | ZIP code of a European town | NR16 |
| town | Name of a European town or city | London |
| year | Year of payment | 2014 |
| country | 2-letter country code of an EU country | GB,SI,NL,PL |
| amount_euro_gte | Amount euro greater than given value | 2500,100000,1000000 |
| sub_payments_type | Type of the sub payment in national language | Direct Aid |

### Example Requests

```
https://[URL_TO_API]/[API_VERSION]/payments/?amount_euro_gte=5000&town=London
```

### Example Data Set

```
_source: {
  town: "London",
  amount_nc: 6568,
  name: "Example Recipient",
  amount_euro: 8631.32,
  country: "GB",
  sub_payments_euro: [
    {
      amount: 8630.66,
      name: "Rural Development"
    },
    {
      amount: 0,
      name: "Direct Aid"
    },
    {
      amount: 0,
      name: "Market Schemes"
    }
  ],
  sub_payments_nc: [
    {
      amount: 6567.5,
      name: "Rural Development"
    },
    {
      amount: 0,
      name: "Direct Aid"
    },
    {
      amount: 0,
      name: "Market Schemes"
    }
  ],
  year: 2014,
  nc_conv_rate: 0.76095,
  nc_conv_date: "2016-01-26",
  zip_code: "SW7"
}
```

`Sub payments` are indexed schemaless as they are provided by the specific country agencies.

**Note:** If both an `amount_nc` (national currency) and an `amount_euro` is provided, the Euro value is not coming originally from the source but is calculated via `fixer.io` API with the given conv rate at the conv date provided.

## Countries Endpoint

The `countries` endpoint is a simple static endpoint and can be reached at:

```
/[API_VERSION]/payments/
```

It provides a list of all countries where payment data is indexed together with some additional information like a countries responsible paying agency, associated data and info urls and the like.

There are no request parameters supported at the moment.

### Example Request

```
https://[URL_TO_API]/[API_VERSION]/countries/
```

### Example Data Set

```
{
    "GB": {
        "name": "Great Britain",
        "agency_name": "GOV.UK - Department for Environment, Food and Rural Affairs",
        "info_url": "https://www.gov.uk/government/organisations/department-for-
↪environment-food-rural-affairs",
        "data_url": "http://cap-payments.defra.gov.uk/",
        "nc_symbol": "GBP",
        "nc_sign": "£"
    },
    ...
    "IE": {
        "name": "Ireland",
        "agency_name": "gov.ie - Department of Agriculture, Food and the Marine",
        "info_url": "http://www.agriculture.gov.ie",
        "data_url": "http://www.agriculture.gov.ie/agri-foodindustry/
↪euinternationalpolicy/commonagriculturalpolicycap/capbeneficiariesdatabase/
↪paymentsdatabase/cap_ben_master.jsp",
        "nc_symbol": "",
        "nc_sign": ""
    }
}
```

CHAPTER 4

# Frontend (Boostrap/Javascript)

## Installation

### Requirements

Main runtime library dependencies:

- Bootstrap 4 Alpha 6
- jQuery 3.2.1

Main dev tools:

- Sass (Ruby installation)
- http-server
- (Gulp.js) (build automation)

### Running the server

Run the `http-server` from the main folder of the repository:

```
http-server
```

Content is served on `http://127.0.0.1:8080`, API is expected at `http://127.0.0.1:5000`.

## Development

`Sass` sources can be compiled with:

```
sass sass/content.scss css/content.css
```

Or you can run the `watch` command with:

```
sass --watch sass/content.scss:css/content.css
```

# CHAPTER 5

# Deployment

## Server Setup

### General

Deployment of all server parts is done on an `Ubuntu 14.04 AWS/EC2` instance, Python `fabric` is used for deployment automation, fabric files can be found in `openfarmsubsidies-scraper` repository.

The following fabric tasks are just for orientation what need to be installed/done and are not intended to pass through, depending on your system pre-requisites:

```
fab prepare_system
fab install_deps
```

Script templates for setting up `Gunicorn`, `Nginx` and `Supervisor` can be found in the `conf` folder.

### SSL

`SSL` cert is created with **Let's Encrypt** with the following command:

```
sudo /home/ubuntu/.local/share/letsencrypt/bin/letsencrypt certonly -d␣
↪openfarmsubsidies.org -d www.openfarmsubsidies.org -d scraper.openfarmsubsidies.org␣
↪-d api.openfarmsubsidies.org
```

### Elasticsearch

`Elasticsearch` is installed as deb (Debian) package following this instructions.

Installation can be found at `/usr/share/elasticsearch/`, start/stop is done via `init.d` script `sudo /etc/init.d/elasticsearch start`.

Index backup can be done with:

```
sudo cp -Rp /var/lib/elasticsearch /var/lib/elasticsearch-backup-2017-06-28
```

CHAPTER 6

# Indices and tables

- genindex
- modindex
- search