django-genericimports Documentation

Release 0.1b3

Oscar Carballal Prego, Max Barry

January 28, 2015

Contents

django-genericimports is a django application to import CSV/XLS files into your database without having to care about headers or data types, it provides support for foreignkeys, callbacks, thirdparty fields, etc.

Contents:

CHAPTER 1

Introduction

django-genericimports was born out of the need to create an import mechanism that worked no matter what, allowing third parties to upload imports into the database and being as blind as possible to all the human mistakes that can happen.

1.1 Why

Mainly because i was tired of doing import scripts for all the projects that I worked on, the second reason is that it doesn't seem to be any open source truly generic importer that does the job right, so I felt it was necessary to do it. Of course there are really cool import applications out there, and maybe they suit you better than this one.

I know it's not the best approach, an it is slow as hell, but that is why I opensourced it, so everyone could improve it. By the way I'm quite a fan of commenting the code, so you will find tons of comments in it, even some comments that are obviously obvious, but hey, not only 10+ y/exp programmers will be looking at this.

1.2 Disclaimer

Please note that this application was made to cover all use cases and it sanitizes the data. It's not meant to be fast, at least at this stage of development. Feel free to send your optimization patches.

CHAPTER 2

Features and requirements

2.1 Features

- Support CSV, XLS and XLSX
- Agnostic to data types or headers
- Non estandard fields are supported trough functions
- Support required unbound fields (fields required in your model but absent in your import file)
- Support callbacks to apply after a succesful import
- Support for optional data in the import file (columns that repeat themselves)
- Support Foreignkeys
- ***Report system with statistics and records on each import***
- ***Failed entries are saved in a separate file and accesible through the report***
- ***Third party access to the imports with email reports.***
- Support django CMS
- Available as management command or admin action

2.1.1 Future improvements

- Speed (really difficult, this one)
- Support ManyToMany fields
- Remove lambdas to remove eval() from the code and replace it with functions
- Optional logging (forced right now)
- Optional profiling (so you can know how well did this do in your box)
- pip support
- Figure out a way of doing bulk_inserts with ID's

2.2 Supports

- Django 1.5/1.6/1.7
- Python 2.7.x/3.3.x up

2.3 Requirements

- redis

CHAPTER 3

Installing

3.1 Installing genericimports stable

Installing django-genericimports is easy, just type in:

```
$ pip install django-genericimports
```

It will take care of installing almost all the dependencies for you.

3.2 Installing a genericimports development version

To be written...

3.3 Embedding in your project

I'm usually against embedding third party applications in projects for a good number of reasons, yet there are small situations where you want to do it.

To embed the application in your project should be more complex than copying the folder "genericimports" to your project as a new application, and adding the applications listed in the "install_requires" in the setup.py file to your requirements file (if you have one. If you don't... you should have it)

Apart from that the rest is exactly the same, the only thing that you have to care about is the import path. You won't be able to use "genericimports" straight in again, rather you will have to put your own module path, something along the lines of "myapps.genericimports"

3.4 Installing Redis

One of the requirements of the current version of django-genericimports is to have redis available on the server, or to have a remote redis instance that can be used.

In most of the cases you won't have one, but don't run away, you only need to install it and run the redis service, django-rq will take care of the rest.

To install redis in ArchLinux:

```
$ sudo pacman -S redis
$ sudo systemctl enable redis
```

To install redis on Debian/Ubuntu:

```
$ sudo apt-get install redis-server python-redis
$ update-rc.d redis defaults
```

CHAPTER 4

Setting up

The setup of genericimports involves three parts: the actual mapping so you can do your imports, the redis configuration and the logging which for now is forced. If you are clever enough, yo can go straight into the example_settings.py file which contains everything and includes an example mapping commented.

4.1 For django

4.1.1 Settings up redis

Setting up the redis connection in django is easy, here is an example configuration:

```
RQ_QUEUES = {
    'importer': {
        'HOST': 'localhost',
        'PORT': 6379,
        'DB': 0,
        'PASSWORD': '',
        'DEFAULT_TIMEOUT': 360,
    },
}
```

This will create a queue called "importer" connected to a local instance of redis (this can be changed if you want to connect to a remote one). The rest are just defaults that you can leave if you don't need to edit them.

4.1.2 Setting up the logging

genericimports has a logging mechanism integrated, that stips out to a django logger. If you don't have your logging activated, you should paste this and it will create a file called "django.log" in the root of your project, that way you can keep track of everything.

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,

    'formatters': {
        'standard': {
            'format': "[%(asctime)s] %(levelname)s [%(name)s:%(lineno)s] %(message)s",
            'datefmt': "%d/%b/%Y %H:%M:%S"
        },
    },
```

```
    'filters': {
        'require_debug_false': {
            '()': 'django.utils.log.RequireDebugFalse'
        }
    },

    'handlers': {
        'mail_admins': {
            'level': 'ERROR',
            'filters': ['require_debug_false'],
            'class': 'django.utils.log.AdminEmailHandler'
        },
        'logfile': {
            'level': 'DEBUG',
            'class': 'logging.handlers.RotatingFileHandler',
            'filename': BASE_DIR + "/django.log",
            'maxBytes': 2097152,  # 2MB per file
            'backupCount': 2,  # Store up to three files
            'formatter': 'standard',
        },
    },

    'loggers': {
        'django.request': {
            'handlers': ['mail_admins'],
            'level': 'ERROR',
            'propagate': True,
        },
        # Log all the things!
        '': {
            'handlers': ["logfile", ],
            'level': 'DEBUG',
        },
    }
}
```

4.1.3 Setting up the mapping

This is the most important part of the configuration, since it tells the script where to go in your code to put the data in place, and example mapping looks like this (copied from example_settings.py)

```
IMPORTER = [
    {
        # Settings for the importer
        'settings': {
            # If you have a function that you want to apply after a successful
            # import, you would put it on the callback unbounded.
            'callback': 'MyFunction',
            # Who to send notifications about the imports
            'notifyto': 'example@example.com'
        },
        # This is the main mapping, the order or the items matter, since it
        # matches the column order in the CSV rows.
        'mapping': [
            {
                'model': 'auth.User',
```

```
        # Can we skip this data if it fails?
        'optional': False,
        # The field names match the column position (0, 1, 2 in this case)
        'fields': [
            'field_1',
            'field_2',
            'field_3',
        ],
        # We have fields that are not in the CSV but are required by the model
        'unbound': {
            'username': "(lambda x: ''.join(random.choice(string.ascii_lowercase) for _ in ra
            'password': "(lambda x: ''.join(random.choice(string.ascii_lowercase) for _ in ra
    },
    {
        'model': 'accounts.UserProfile',
        # Oh, this model is related to the other one, we put the
        # position of that model in the import list and the field that
        # links them.
        'foreignkey': [0, 'user'],
        'optional': False,
        # Please note, user and profile share the first three columns
        # of the CSV, you can do that.
        'fields': [
            'field_1',
            'field_2',
            'field_3',
            'field_4',
            'field_5',
        ]
    },
    {
        'model': 'accounts.UserSomethingRecord',
        # Another model that is foreignkeyed to the user
        'foreignkey': [0, 'user'],
        # If for some reason this data of the row fails, we can skip it
        'optional': True,
        # Now, the data for this model is located after the first 9 columns
        # so what we do is create 9 empty items, and then continue with
        # our fields
        'fields': [''] * 5 + [  # Note the multiplication, we skip 9 columns
            'field_6',
            'field_7',
            'field_8',
            # Imagine a thirdparty field here, django doesn't know how
            # to handle it, but we know, so we call an unbound function
            {'filter': my_function, 'name': "thirdparty_field"},
        ]
    },
    # Now, for some reason we can have data for the same model that
    # repeats itself accross the columns (types of beer that you tried
    # for example)
    {
        'model': 'accounts.UserBeers',
        'foreignkey': [1, 'userprofile'],  # This links this object with the previous one
        'optional': True,
        'fields': [''] * 9 + [
            'field_9',
            'field_10',
```

```
                ]
            },
            # Next record for beers
            {
                'model': 'accounts.UserBeers',
                'foreignkey': [1, 'userprofile'],  # This links this object with the previous one
                'optional': True,
                'fields': [''] * 11 + [
                    'field_11',
                    'field_12',
                ]
            },
        ],
    }
]
```

4.2 For django CMS

Apart from the setting for django which you must do as well, you can add this application as an apphook to your CMS.

CHAPTER 5

Filters, unbound fields and callbacks

genericimports allows you to use a number of options to overcome complications

5.1 Unbound fields

We call unbound fields all those fields that are required in the model but are not present in the import file, for example, you want to import people with first name, last name and the email, but you don't know the username or password.

In that case username and password are required by the auth.User model in django but since you don't have that data we have to populate it somehow.

Usually the population of those fields is done through a python lambda...

To be written....

5.2 Filters

Filters are nothing but functions that will get called upon populating a field because probably the data type is not recognized by django or you need to manipulate it before sending it back.

An example on the mapping would be:

```
{'filter': my_function, 'name': 'field_name'}
```

That funciton will be called during execution with the parameter _rowdata_. An example for that function:

```
def my_function(rowdata):

    # Imagine that we add to add one to the data
    new_rowdata = rowdata + 1

    return new_rowdata
```

5.3 Callbacks

To be written...

CHAPTER 6

Reports

This application comes with a report model that will store statistics about the report, how many entries were added, failed, existed already..

It will also store a copy of the original CSV file and a failed entries CSV that you can use to know what entries failed and fix them.

CHAPTER 7

License

This application is licensed in BSD 2-Clause license. This application was developed while at work on Havas Worldwide London, so part of the kudos goes to them as well.