
Woey Documentation

Release 3.0.0

Martin Fitzpatrick and Chris Mitchell

Sep 28, 2017

Contents

1	Getting Started	3
1.1	Installation	3
1.2	Configuration	15
1.3	Running Wooyey	16
1.4	Adding & Managing Scripts	17
1.5	Remote File Systems	18
2	Developers	19
2.1	API Reference	19
3	Indices and tables	23
	Python Module Index	25

Wooley!

...it's a Web UI for Python scripts.

Wooley is a simple web interface to run command line Python scripts. Think of it as an easy way to get your scripts up on the web for routine data analysis, file processing, or anything else.

The project was inspired by how simply and powerfully [sandman](#) could expose users to a database and by how [Gooley](#) turns ArgumentParser-based command-line scripts into WxWidgets GUIs. Originally two separate projects (Django-based [djangui](#) by [Chris Mitchell](#) and Flask-based [Wooley](#) by [Martin Fitzpatrick](#)) it has been merged to combine our efforts.

Both of our tools were based on our needs as data scientists to have a system that could:

1. Autodocument workflows for data analysis (simple model saving).
2. Enable fellow lab members with no command line experience to utilize python scripts.
3. Enable the easy wrapping of any program in simple python instead of having to use language specific to existing tools such as Galaxy.

Installation

```
pip install woey
```

Currently, Woey supports Django versions 1.8+. To use Woey in a project which is still running Django 1.6 or 1.7, you must may install version 0.9.8.

A Woey only project

There is a bootstrapper included with woey, which will create a Django project and setup most of the needed settings automatically for you.

1. `woify -p ProjectName`
2. Follow the instructions at the end of the bootstrapper to create the admin user **and** access the admin page
3. Login to the admin page wherever the project **is** being hosted (locally this would be `localhost:8000/admin`)

Installation with existing Django Projects

1. Add `'woey'` to `INSTALLED_APPS` **in** `settings.py` (**and** optionally, `djcelery` unless you `↪` wish to tie into an existing celery instance)
2. Add the following to your `urls.py`:

```
url(r'^$', include('woey.urls')),
```

(Note: it does **not** need to be rooted at your site base, you can have `r'^woey/'`... **as** your router):
3. Migrate your database:

```
# Django 1.6 and below:  
./manage.py syncdb
```

```
# Django 1.7 and above
./manage.py makemigrations
./manage.py migrate
```

4. Ensure the following are **in** your `TEMPLATE_CONTEXT_PROCESSORS` variable:

```
TEMPLATE_CONTEXT_PROCESSORS = [
...
'django.contrib.auth.context_processors.auth',
'django.core.context_processors.request'
...]
```

5. If necessary, setup static file serving. For non-production servers, Django can be setup to do this **for** you by adding the following to your `urls.py`:

```
from django.conf import settings
from django.conf.urls.static import static
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

6. You may also need to define a `MEDIA_ROOT`, `MEDIA_URL`, `STATIC_ROOT`, **and** `STATIC_URL` **if** these are **not** setup already.

Installation on remote servers

We have provided guides for several available services that host sites, as well as guides for steps common to multiple services (such as using Amazon Web Services). For examples not listed here, you are free to open up an [issue](#). (or document it and send a pull request!).

Configuration on Heroku

Installing Woey

Woey can be installed by the simple command:

```
pip install woey
```

or if you want to be on the bleeding-edge for some reason, from the github repo by:

```
pip install git+https://github.com/woey/Woey.git
```

Bootstrapping Woey

The woey project can be bootstrapped, which will create a full-fledged django app for you. This can be accomplished with the command:

```
woofy -p project_name
```

For the rest of this guide, the woey instance will be called `woey_heroku`.

Setup Heroku and git

Of course, you will at this point need to have an app on Heroku.

- Create an app on Heroku
- Login to heroku on the command line

Next, setup git:

- `cd woey_heroku`
- `git init`
- `heroku git:remote -a woey`

Setup dependencies

For the bootstrapping, we have included a requirements file to assist in getting newer users up and running. This can be found here:

```
woey_heroku/woey_heroku/requirements.txt
```

We want to add one more dependency to this, which we will use momentarily. To the bottom of this, add:

```
django-heroku-postgresify
```

You will want to move this file to the same location as `manage.py`

Setup a Procfile

Create a file, called Procfile, which tells Heroku how to run your app, with the following contents:

```
web: waitress-serve --connection-limit 2000 --channel-timeout=300 --port=$PORT woey_
->heroku.wsgi:application
worker: python manage.py celery worker -c 1 --beat -l info
```

Setup Environment Vars on Heroku

You will need to add a few settings to your heroku config at this point to tell Heroku where to find your Django settings. This can be done by the command line or through the settings gui.

```
heroku config:set -a woey DJANGO_SETTINGS_MODULE=woey_heroku.settings
```

Storage

Heroku uses an ephemeral file system, so you will need to create your own persistent storage. Amazon S3 services is a natural place for this. To start, complete the steps found [here](#).

Celery

The last bit to setup is celery. For this, we will use the free AMPQ services from heroku, such as RabbitMQ Bigwig. After enabling this in your heroku dashboard, complete the guide found [here](#).

Production Settings

At this point, your woeyy app is insecure – so we will edit your settings to fix this as well as make it more production-ready by changing our database.

You will want to edit `user_settings.py`, which is found in `woeyy_heroku/woeyy_heroku/settings/user_settings.py`

In here, there are comments indicating what each variable means. You will want to change the `DATABASES` variable to:

```
DATABASES = postgresify()
```

and add the following import:

```
from postgresify import postgresify
```

Finally, you want to disable the `DEBUG` setting by adding

```
DEBUG = False
```

Add everything to git and push it upstream

```
git add .
git commit -m 'initial commit'
git push -u heroku master
```

At the last step, the `-u` indicates to create the branch `master` if it does not exist on the remote.

Migrate your database and sync static assets

You need to migrate your database now, setup your admin access, and put our static files on the S3 server. An easy way to do this is through heroku:

```
heroku run -a woeyy bash
python manage.py migrate
python manage.py createsuperuser
python manage.py collectstatic
```

Check out your app

Now, your app should be online. You can check it at `<appname>.herokuapp.com`.

Configuration on DigitalOcean

How to get Woeyy up and running on a DO box. I followed these instructions on a Ubuntu 14.04 LTS box. For security purposes we want it to be running only on HTTPS.

Download Woeyy

Download and install Woeyy, by any of the standard methods. For this tutorial this was run in the home directory i.e. `/home/user/`

This means `manage.py` will be in `/home/user/ProjectName`

Install Nginx & uWSGI

Next we need to install Nginx, which will make setting up SSL easy and uWSGI which is how Django and Nginx talk.

There are instructions on how to install the stable branch of Nginx [here](#).

uWSGI you can install with *pip*.

Getting Wooley to listen on Port 80

You could at this point have Wooley listen to port 80 (assuming it's open) with the following command, run from the folder containing `manage.py`:

```
python manage.py runserver 0.0.0.0:80
```

But this leaves us widely insecure, because all our passwords will be transmitted over HTTP. I had problems working with non-standard ports on UFW so for the purposes of this tutorial, so I set-up my ports using this [iptables linode tutorial](#).

Getting Wooley to talk to Nginx & uWSGI

We will basically follow the tutorial [here](#). If you installed Wooley in a virtualenv from the get go, then follow all those steps, if not you can ignore that set:

Then we can follow the guide along with a couple changes:

1. Basic test
 - Because I was having trouble using non-standard ports, I replaced 8000 with 80 and ran the commands as root
2. Test your Django project
 - In order to get this step to work correctly I found I had to call uWSGI from the main folder where `manage.py` lives, in my case that was `/home/user/ProjectName` (I have no idea why, but otherwise it won't find the Wooley project correctly)
3. Deploying static files
 - So far I have ignored this altogether and no problems...
4. nginx and uWSGI and test.py
 - Again only got this to work on port 80 again because of non-standard port problems

Now if you want to run on a network socket, at this point you should be good to go. (Remember crucially this needs to be run from the same folder as `manage.py`).

```
uwsgi --socket 127.0.0.1:8000 --wsgi-file ProjectName/wsgi.py --chmod-socket=666
```

If you want to use a file socket, I then created an empty file in the Wooley project directory to be used as one (in this example named `django.sock`).

```
uwsgi --socket ProjectName/django.sock --wsgi-file ProjectName/test.py --chmod-socket=666
```

At this point we should now have Wooley running on Port 80 through Nginx.

Forcing SSL with Nginx

I have forced SSL with the following settings. (I think I might be running two SSL redirects, one on the Nginx side and one on the Django side which is never necessary because Nginx comes first, any clarification would be welcome, however for those following along:)

I switched the main nginx block to HTTPS (there's a good tutorial [here](#) if you haven't done this before).

I also added an HTTPS header to the server block listening on 443 so Django knows it's HTTPS:

```
proxy_set_header X-Forwarded-Proto $scheme;
```

Then I set-up a second server block to listen on port 80 and rewrite to https:

```
server {
    listen 80;
    listen [::]:80;

    server_name enter_hostname;

    return 301 https://$server_name$request_uri;
}
```

Then on the Django side I added the following flags to my config in `user_settings.py`

```
SECURE_SSL_REDIRECT = True #this may be the double redirect which is unnecessary.
SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
```

Finally I then added HTTP authentication, there is a good tutorial on this [here](#). You only need to reach the first part of step 3, adding the `auth_basic` lines to your HTTPS block.

Here's an example of what my final Nginx setup file in `/etc/nginx/sites-available/django` looked like:

```
# the upstream component nginx needs to connect to
upstream django {
    server unix:///home/user/projectname/projectname/django.sock; # for a file socket
    #server 127.0.0.1:8000; # for a web port socket (we'll use this first)
}

# configuration of the server
server {
    # the port your site will be served on
    listen 443 ssl;
    # the domain name it will serve for
    server_name server_ip; # substitute your machine's IP address or FQDN
    charset utf-8;

    #add basic auth to prevent crawling
    auth_basic "Restricted";
    auth_basic_user_file /etc/nginx/.htpasswd;

    #get the self signed certificate
    ssl_certificate /etc/nginx/ssl/nginx.crt;
    ssl_certificate_key /etc/nginx/ssl/nginx.key;

    #add header to django knows request came through HTTPS
    proxy_set_header X-Forwarded-Proto $scheme;
```

```

# max upload size
client_max_body_size 75M; # adjust to taste

# Django media
location /media {
    alias /home/user/projectname/projectname/uploads; # your Django project's
↔media files - amend as required
}

location /static {
    alias /home/user/projectname/projectname/static; # your Django project's
↔static files - amend as required
}

# Finally, send all non-media requests to the Django server.
location / {
    uwsgi_pass django;
    include /etc/nginx/uwsgi_params;
}
}

#http rewrite
server {
    listen 80;
    listen [::]:80;

    server_name server_ip;

    return 301 https://$server_name$request_uri;
}

```

Running Celery in the background

All this other set-up means you then can't use honcho to run celery, because it doesn't seem to like (that's a technical term) the uWSGI command which means instead, you have to run it as a background process. This however just seems to work...

```
nohup python manage.py celery worker -c 1 -beat -l info & #you probably want to pipe this output somewhere sensible
```

Which means you can then run the server with the command above uwsgi command shown above.

Contributed by [dom-devel](#).

Configuration on OpenShift

OpenShift is considerably more involved than Heroku, but allows you to freely run an app that will not sleep.

Setup OpenShift

- Setup a Django gear, this will give you a basic structure for an app.
- Clone the git repository for the app locally.

Installing Wooley

In the same local environment you cloned from, you can installed Wooley via:

```
pip install wooley
```

or if you want to be on the bleeding-edge for some reason, from the github repo by:

```
pip install git+https://github.com/wooley/Wooley.git
```

Bootstrapping Wooley

Next, you want to cd into your project root, which should be called wsgi. Here, you want to run to wooley bootstrapper, which will create a full-fledged django app for you. This can be accomplished with the command:

```
woofy -p project_name
```

Setup OpenShift Pt. 2

For the rest of this guide, the wooley instance will be called wooley_openshift. Next you will want to remove the old django project, called myproject and edit the *application* file and change all instances of myproject to YourProject-Name(wooley_openshift in this example).

Next, you want to setup a database and a message broker for Celery

1. Setting up a Database

- Add a PostgreSQL instance cartridge to your app.
- Add our database information. Openshift provides a few variables for us to use for the ip/port, but does not include one for the database name. Add that via:

```
rhc env set -a YourAppName DATABASE_NAME=database_name
```

- Uncomment our DATABASES variable in user_settings.py:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycpg2',
        # for production environments, these should be stored as environment_
↪variables
        # I also recommend the django-heroku-postgresify package for a super_
↪simple setup
        'NAME': os.environ.get('DATABASE_NAME', 'wooley'),
        'USER': os.environ.get('DATABASE_USER', 'wooley'),
        'PASSWORD': os.environ.get('DATABASE_PASSWORD', 'wooley'),
        'HOST': os.environ.get('DATABASE_URL', 'localhost'),
        'PORT': os.environ.get('DATABASE_PORT', '5432')
    }
}
```

- Next, we need to change the variables since OpenShift sets up different environment variables. Rename the above variables to:

```
DATABASE_USER -> OPENSIFT_POSTGRESQL_DB_USERNAME
DATABASE_PASSWORD -> OPENSIFT_POSTGRESQL_DB_PASSWORD
DATABASE_URL -> OPENSIFT_POSTGRESQL_DB_HOST
DATABASE_PORT -> OPENSIFT_POSTGRESQL_DB_PORT:
```

2. Setting up Celery

- Add CloudAMPQ as a service, which can be done once again through the OpenShift Marketplace.
- Update the user_settings.py file and uncomment the following:

```
CELERY_RESULT_BACKEND = 'amqp'
BROKER_URL = os.environ.get('AMQP_URL') or \
             os.environ.get('RABBITMQ_BIGWIG_TX_URL') or \
             os.environ.get('CLOUDAMQP_URL', 'amqp://
↪guest:guest@localhost:5672/')
BROKER_POOL_LIMIT = 1
CELERYD_CONCURRENCY = 1
CELERY_TASK_SERIALIZER = 'json'
ACKS_LATE = True
CELERY_IMPORTS = ('wooley.tasks')
```

- Change AMPQ_URL to CLOUDAMQP_URI, which is the environment variable setup in your app.
- Next, we need to tell the server to start celery. We will make a new deployment hook for this. Create the file project_root/openshift/action_hooks/post_start with the following content:

```
#!/bin/bash
cd $OPENSIFT_REPO_DIR
cd wsgi
cd YourProjectName
rm worker1.pid
celery multi stop worker1
celery multi start worker1
```

- To save our connections, we need to tell celery to stop when the app stops as well. Create another file, pre_stop with:

```
#!/bin/bash
cd $OPENSIFT_REPO_DIR
cd wsgi
cd YourAppName
rm worker1.pid
celery multi stop worker1
```

3. Setup the requirements.txt file. The bootstrapper provides a requirements.txt file that already has all the apps needed to run Wooley. Just copy it from YourAppName/YourAppName/ to the top level directory of OpenShift (which has things like setup.py and openshiftlibs.py)

4. Edit wsgi/application and change:

- alter myproject to YourProjectName
- Change

```
os.environ['DJANGO_SETTINGS_MODULE'] = 'myproject.settings'
```

to

```
os.environ['DJANGO_SETTINGS_MODULE'] = 'YourProjectName.settings'
```

5. Edit your git hooks to reflect the new project name:

- There is a hidden directory at the project root, called `.openshift`. within it you want the directory `action_hooks`. cd into this, and make the following changes
- In `deploy`, change `myproject` to `YourProjectName`
- In `secure_db`, do the same.

6. Update where the static assets are being served from in `user_settings.py` (Optionally, you can follow the guide to not use OpenShift's static service and go through S3 instead here):

```
STATIC_ROOT = os.path.join(os.environ.get('OPENSIFT_REPO_DIR'), 'wsgi', 'static',  
↪ 'static')  
MEDIA_ROOT = os.path.join(os.environ.get('OPENSIFT_DATA_DIR'), 'user_uploads')
```

7. Remove DEBUG mode. In `user_settings.py`, add:

```
DEBUG=False
```

Migrate your database and sync static assets

You need to migrate your database now, setup your admin access, and sync our static files. An easy way to do this is through the ssh command:

```
rhc ssh -a YourAppName  
python manage.py migrate  
python manage.py createsuperuser  
python manage.py collectstatic
```

Check out your app

Now, your app should be online.

Configuring Amazon S3 Storage for Wooley

Prerequisites

Before getting started, this guide assumes you have several things setup:

- An AWS account
- An S3 Bucket
- An IAM user/group with full access to the S3 bucket (remember to add your user to the IAM group controlling the bucket!)
- Are using a storage app in Django. We recommend `django-storages`.

Steps to Follow

- Edit your CORS configuration for the bucket:

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <CORSRule>
    <AllowedOrigin>*</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
    <MaxAgeSeconds>3000</MaxAgeSeconds>
    <AllowedHeader>Authorization</AllowedHeader>
  </CORSRule>
  <CORSRule>
    <AllowedOrigin>woeyy.herokuapp.com</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
    <AllowedMethod>POST</AllowedMethod>
    <AllowedMethod>PUT</AllowedMethod>
    <AllowedHeader>*</AllowedHeader>
  </CORSRule>
</CORSConfiguration>
```

- If using a bootstrapped Woeyy, update `user_settings.py` and uncomment out the following section: (or add it for apps Woeyy was added to)

```
from boto.s3.connection import VHostCallingFormat

INSTALLED_APPS += (
    'storages',
    'collectfast',
)

# We have user authentication -- we need to use https (django-sslify)
if not DEBUG:
    MIDDLEWARE_CLASSES = ['sslify.middleware.SSLifyMiddleware'] + list(MIDDLEWARE_
→CLASSES)
    SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')

ALLOWED_HOSTS = (
    'localhost',
    '127.0.0.1',
    "woeyy.herokuapp.com", # put your site here
)

AWS_CALLING_FORMAT = VHostCallingFormat

AWS_ACCESS_KEY_ID = environ.get('AWS_ACCESS_KEY_ID', '')
AWS_SECRET_ACCESS_KEY = environ.get('AWS_SECRET_ACCESS_KEY', '')
AWS_STORAGE_BUCKET_NAME = environ.get('AWS_STORAGE_BUCKET_NAME', '')
AWS_AUTO_CREATE_BUCKET = True
AWS_QUERYSTRING_AUTH = False
AWS_S3_SECURE_URLS = True
AWS_FILE_OVERWRITE = False
AWS_PRELOAD_METADATA = True
```

```
AWS_S3_CUSTOM_DOMAIN = environ.get('AWS_S3_CUSTOM_DOMAIN', '')

GZIP_CONTENT_TYPES = (
    'text/css',
    'application/javascript',
    'application/x-javascript',
    'text/javascript',
)

AWS_EXPIREY = 60 * 60 * 7
AWS_HEADERS = {
    'Cache-Control': 'max-age=%d, s-maxage=%d, must-revalidate' % (AWS_EXPIREY,
        AWS_EXPIREY)
}

STATIC_URL = 'http://%s.s3.amazonaws.com/' % AWS_STORAGE_BUCKET_NAME
MEDIA_URL = '/user-uploads/'
STATICFILES_STORAGE = DEFAULT_FILE_STORAGE = 'woeyy.woeystorage.
↳CachedS3BotoStorage'
WOEY_EPHEMERAL_FILES = True
```

In the above step, make sure you change `woeyy.herokuapp.com` to your app's address.

Configuration Settings

Next, as part of any good app – you should be storing your secret information in environmental variables instead of hard-coding them into the app. You will want to set these variables:

```
AWS_ACCESS_KEY_ID=access_key
AWS_SECRET_ACCESS_KEY=secret_key
AWS_STORAGE_BUCKET_NAME=bucket_name
```

If you are using Heroku, you can set them as follows:

```
heroku config:set -a woeyy AWS_ACCESS_KEY_ID=access_key
heroku config:set -a woeyy AWS_SECRET_ACCESS_KEY=secret_key
heroku config:set -a woeyy AWS_STORAGE_BUCKET_NAME=bucket_name
```

Woeyy Celery Configuration

Celery

Celery is an app designed to pass messages. This has broad implications, such as the ability to have a distributed setup where workers perform the work, with a central node delegating the tasks (without halting the server to perform these tasks).

The backend

There are several backends to use, here we can use a database backend or a server as a backend. By default, Woeyy uses the database as a backend. If you wish to move to a more robust system, there are several options such as AMPQ

or redis. Here, we detail how to use AMPQ.

If you are coming from a bootstrapped project, to switch to an AMPQ backend, it is a matter of uncommenting the following lines in your production settings:

```
CELERY_RESULT_BACKEND = 'amqp'
BROKER_URL = os.environ.get('AMQP_URL') or \
    os.environ.get('RABBITMQ_BIGWIG_TX_URL') or \
    os.environ.get('CLOUDAMQP_URL', 'amqp://guest:guest@localhost:5672/')
BROKER_POOL_LIMIT = 1
CELERYD_CONCURRENCY = 1
CELERY_TASK_SERIALIZER = 'json'
ACKS_LATE = True
```

If you are coming from a project which has woey installed as an additional app, you want to add the above to your settings.

Additional Heroku Options

For heroku, you will want to add AMPQ to your app through the dashboard, which should give you a AMPQ url compatible with the above options.

Configuration

Woey Settings

WOOEY_FILE_DIR: String, where the files uploaded by the user will be saved (Default: `woey_files`)

WOOEY_CELERY: Boolean, whether or not celery is enabled. If disabled, tasks will run locally and block execution. (Default: `True`)

WOOEY_CELERY_TASKS: String, the name of the celery tasks for Woey. (Default: `'woey.tasks'`)

WOOEY_ALLOW_ANONYMOUS: Boolean, whether to allow submission of jobs by anonymous users. (Default: `True`)

By default, Woey has a basic user account system. It is very basic, and doesn't confirm registrations via email.

WOOEY_AUTH: Boolean, whether to use the authorization system of Woey for simple login/registration. (Default: `True`)

WOOEY_LOGIN_URL: String, if you have an existing authorization system, the login url: (Default: `settings.LOGIN_URL`)

WOOEY_REGISTER_URL: String, if you have an existing authorization system, the registration url: (Default: `'/accounts/register/'`)

WOOEY_EPHEMERAL_FILES: Boolean, if your file system changes with each restart. (Default: `False`)

WOOEY_SHOW_LOCKED_SCRIPTS: Boolean, whether to show locked scripts as disabled or hide them entirely. (Default: `True` – show as disabled)

WOOEY_REALTIME_CACHE: String, the name of the cache to use for storing real time updates from running jobs.

WOOEY_JOB_EXPIRATION: Dictionary, A dictionary with two keys: `user` and `anonymous`. The values for each is a `timedelta` specifying how much time should be elapsed before a job is automatically deleted. If a key is not provided or `None`, the job for that user type will not be deleted.

Internationalization (i18n)

Woeyy supports the use of Django internationalization settings to present the interface in your own language. Currently we provide limited support for French, German, Dutch, Japanese, and Simplified Chinese. We welcome contributions for translation extensions, fixes and new languages from our users.

To specify the default language for your installation, you can specify this using the `LANGUAGE_CODE` setting in `django_settings.py`. For example to set the interface to French, you would use:

```
LANGUAGE_CODE = 'fr'
```

For German you would use:

```
LANGUAGE_CODE = 'de'
```

If you want the user interface to automatically change to the preferred language for your visitors, you must use the Django internationalization middleware. By default, the bootstrapped version of Woeyy will add in the necessary middleware, but for projects using Woeyy as a separate app, these projects will need to add `django.middleware.locale.LocaleMiddleware` to their `MIDDLEWARE_CLASSES` block in `django_settings.py`. Note that it must come *after* the Session middleware, and before the CommonMiddleware e.g.

```
MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.locale.LocaleMiddleware', # <- HERE
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'django.middleware.security.SecurityMiddleware',
)
```

For more information on the internationalization middleware see [the Django documentation](#).

Note that if a user's browser does not request an available language the language specified in `LANGUAGE_CODE` will be used.

Running Woeyy

Woeyy depends on a distributed worker to handle tasks, you can disable this by setting `WOOEY_CELERY` to `False` in your settings, which will allow you to run Woeyy through the simple command:

```
python manage.py runserver
```

However, this will cause the server to execute tasks, which will block the site.

The recommended ways to run Woeyy are:

Through two separate processes

You can run Woeyy by calling two commands (you will need a separate process for each):

```
python manage.py celery worker -c 1 --beat -l info
python manage.py runserver
```

On Windows, the `--beat` option may not be supported.

Through a Procfile

The recommended way to run Wooley is to use a Procfile with `honcho`, which can be installed via `pip`. Make a file, called `Procfile` in the root of your project (the same place as `manage.py`) with the following contents:

```
web: python manage.py runserver
worker: python manage.py celery worker -c 1 --beat -l info
EOM
```

Your server can then be run by the simple command:

```
honcho start
```

On Windows, the `--beat` option may not be supported.

Adding & Managing Scripts

Scripts may be added in two ways, through the Django admin interface as well as through the `addscript` command in `manage.py`.

Script Guidelines

The easiest way to make your scripts compatible with Wooley is to define your `ArgParse` class in the global scope. For instance:

```
import argparse
import sys

parser = argparse.ArgumentParser(description="Find the sum of all the numbers below a
↳certain number.")
parser.add_argument('--below', help='The number to find the sum of numbers below.',
↳type=int, default=1000)

def main():
    args = parser.parse_args()
    s = sum((i for i in range(args.below)))
    print("Sum =", s)
    return 0

if __name__ == "__main__":
    sys.exit(main())
```

If you have failing scripts, please open an issue with their contents so we can handle cases as they appear and try to make this as all-encompassing as possible. One known area which fails currently is defining your `argparse` instance inside the `if __name__ == "__main__"` block

The admin Interface

Within the django admin interface, scripts may be added to through the 'scripts' model. Here, the user permissions may be set, as well as cosmetic features such as the script's display name, description (if provided, otherwise the script

name and description will be automatically populated by the description from argparse if available).

The command line

```
./manage.py addscript
```

This will add a script or a folder of scripts to Woey (if a folder is passed instead of a file). By default, scripts will be created in the ‘Woey Scripts’ group.

Script Organization

Scripts can be viewed at the root url of Woey. The ordering of scripts, and groupings of scripts can be altered by changing the ‘Script order’ or ‘Group order’ options within the admin.

Script Permissions

Scripts and script groups can be relegated to certain groups of users. The ‘user groups’ option, if set, will restrict script usage to users within selected groups.

Scripts and groups may also be shutoff to all users by unchecked the ‘script/group active’ option.

Remote File Systems

Woey has been tested on heroku with S3 as a file storage system. Settings for this can be seen in the user_settings.py, which give you a starting point for a non-local server. In short, you need to change your storage settings like such:

```
STATICFILES_STORAGE = DEFAULT_FILE_STORAGE = 'woey.woeystorage.CachedS3BotoStorage'  
WOOEY_EPHEMERAL_FILES = True
```

API Reference

The API reference is intended for developers of Wooyey. It lists the internal classes and methods that make up Wooyey. If you

Admin

Apps

```
class wooyey.apps.WooyeyConfig(app_name, app_module)
```

```
    name = 'wooyey'
```

```
    ready()
```

```
    verbose_name = 'Wooyey'
```

Backend

```
wooyey.backend.utils.add_wooyey_script(script_version=None, script_path=None, group=None,  
                                       script_name=None)
```

```
wooyey.backend.utils.create_job_fileinfo(job)
```

```
wooyey.backend.utils.create_wooyey_job(*args, **kwargs)
```

```
wooyey.backend.utils.get_checksum(path, extra=None)
```

```
wooyey.backend.utils.get_current_scripts()
```

```
wooyey.backend.utils.get_file_info(filepath)
```

```
wooyey.backend.utils.get_file_previews(job)
```

`woeyy.backend.utils.get_file_previews_by_ids` (*ids*)

`woeyy.backend.utils.get_form_groups` (*script_version=None, pk=None, initial_dict=None, render_fn=None*)

`woeyy.backend.utils.get_grouped_file_previews` (*files*)

`woeyy.backend.utils.get_job_commands` (*job=None*)

`woeyy.backend.utils.get_master_form` (*script_version=None, pk=None*)

`woeyy.backend.utils.get_query` (*query_string, search_fields*)

Returns a query as a combination of Q objects that query the specified search fields.

`woeyy.backend.utils.get_storage` (*local=True*)

`woeyy.backend.utils.get_storage_object` (*path, local=False*)

`woeyy.backend.utils.get_upload_path` (*filepath, checksum=None*)

`woeyy.backend.utils.mkdirs` (*path*)

`woeyy.backend.utils.normalize_query` (*query_string, findterms=<built-in method findall of _sre.SRE_Pattern object>, normspace=<built-in method sub of _sre.SRE_Pattern object>*)

Split the query string into individual keywords, discarding spaces and grouping quoted words together.

```
>>> normalize_query(' some random words "with quotes " and spaces')
['some', 'random', 'words', 'with quotes', 'and', 'spaces']
```

`woeyy.backend.utils.purge_output` (*job=None*)

`woeyy.backend.utils.reset_form_factory` (*script_version=None*)

`woeyy.backend.utils.sanitize_name` (*name*)

`woeyy.backend.utils.sanitize_string` (*value*)

`woeyy.backend.utils.test_delimited` (*filepath*)

`woeyy.backend.utils.test_fastx` (*filepath*)

`woeyy.backend.utils.test_image` (*filepath*)

`woeyy.backend.utils.valid_user` (*obj, user*)

`woeyy.backend.utils.validate_form` (*form=None, data=None, files=None*)

Django Backwards-Compatibility

Forms

Management

Models

Settings

`woeyy.settings.get` (*key, default*)

Signals

Tasks

```
class wooye.tasks.WooyeTask
```

```
    acks_late = False
    ignore_result = False
    name = 'wooye.tasks.WooyeTask'
    rate_limit = None
    request_stack = <celery.utils.threads._LocalStack object>
    send_error_emails = False
    serializer = 'pickle'
    store_errors_even_if_ignored = False
    track_started = False
```

```
wooye.tasks.configure_workers(*args, **kwargs)
```

```
wooye.tasks.enqueue_output(out, q)
```

```
wooye.tasks.output_monitor_queue(queue, out)
```

```
wooye.tasks.update_from_output_queue(queue, out)
```

Test Settings

Test URLs

Tests

URLs

Views

Wooye Storage

```
class wooye.wooyestorage.FakeRemoteStorage(*args, **kwargs)
```


CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

W

- `woeey.apps`, 19
- `woeey.backend`, 19
- `woeey.backend.utils`, 19
- `woeey.management`, 20
- `woeey.settings`, 20
- `woeey.tasks`, 21
- `woeey.test_settings`, 21
- `woeey.tests`, 21
- `woeey.tests.config`, 21
- `woeey.woeeystorage`, 21

A

acks_late (woeey.tasks.WooleyTask attribute), 21
add_woeey_script() (in module woeey.backend.utils), 19

C

configure_workers() (in module woeey.tasks), 21
create_job_fileinfo() (in module woeey.backend.utils), 19
create_woeey_job() (in module woeey.backend.utils), 19

E

enqueue_output() (in module woeey.tasks), 21

F

FakeRemoteStorage (class in woeey.woeeystorage), 21

G

get() (in module woeey.settings), 20
get_checksum() (in module woeey.backend.utils), 19
get_current_scripts() (in module woeey.backend.utils), 19
get_file_info() (in module woeey.backend.utils), 19
get_file_previews() (in module woeey.backend.utils), 19
get_file_previews_by_ids() (in module woeey.backend.utils), 19
get_form_groups() (in module woeey.backend.utils), 20
get_grouped_file_previews() (in module woeey.backend.utils), 20
get_job_commands() (in module woeey.backend.utils), 20
get_master_form() (in module woeey.backend.utils), 20
get_query() (in module woeey.backend.utils), 20
get_storage() (in module woeey.backend.utils), 20
get_storage_object() (in module woeey.backend.utils), 20
get_upload_path() (in module woeey.backend.utils), 20

I

ignore_result (woeey.tasks.WooleyTask attribute), 21

M

makedirs() (in module woeey.backend.utils), 20

N

name (woeey.apps.WooleyConfig attribute), 19
name (woeey.tasks.WooleyTask attribute), 21
normalize_query() (in module woeey.backend.utils), 20

O

output_monitor_queue() (in module woeey.tasks), 21

P

purge_output() (in module woeey.backend.utils), 20

R

rate_limit (woeey.tasks.WooleyTask attribute), 21
ready() (woeey.apps.WooleyConfig method), 19
request_stack (woeey.tasks.WooleyTask attribute), 21
reset_form_factory() (in module woeey.backend.utils), 20

S

sanitize_name() (in module woeey.backend.utils), 20
sanitize_string() (in module woeey.backend.utils), 20
send_error_emails (woeey.tasks.WooleyTask attribute), 21
serializer (woeey.tasks.WooleyTask attribute), 21
store_errors_even_if_ignored (woeey.tasks.WooleyTask attribute), 21

T

test_delimited() (in module woeey.backend.utils), 20
test_fastx() (in module woeey.backend.utils), 20
test_image() (in module woeey.backend.utils), 20
track_started (woeey.tasks.WooleyTask attribute), 21

U

update_from_output_queue() (in module woeey.tasks), 21

V

valid_user() (in module woeey.backend.utils), 20
validate_form() (in module woeey.backend.utils), 20

verbose_name (woey.apps.WoeyConfig attribute), 19

W

woey.apps (module), 19

woey.backend (module), 19

woey.backend.utils (module), 19

woey.management (module), 20

woey.settings (module), 20

woey.tasks (module), 21

woey.test_settings (module), 21

woey.tests (module), 21

woey.tests.config (module), 21

woey.woeystorage (module), 21

WoeyConfig (class in woey.apps), 19

WoeyTask (class in woey.tasks), 21