# Eve W-Space Documentation

***Release 0.2.0***

**Marbin Drakon**

June 04, 2013

# CONTENTS

NOTE: Eve W-Space is currently in alpha. Most features outside of Mapping are not yet implemented past some prototype data models.

CHAPTER

ONE

# WHAT IS IT?

Eve W-Space is, at its core, a wormhole mapping and intel tool for the MMORPG Eve Online. If you wish, it can be much more. It can be customized to fill roles from simple mapping applicaiton (with all extra features disabled) to the centerpiece of an entire alliance services infrastructure.

Eve W-Space is designed to be a single-tenant soulution hosted by a single corp or alliance. A fleixble permissions system allows for easily restricting data where needed.

CHAPTER

TWO

# CONTENTS:

## 2.1 Tutorials

These pages contain the various tutorials that have been written for Eve W-Space.

### 2.1.1 Install: Ubuntu 12.04 LTS with MySQL and Nginx

This guide will walk you through installing Eve W-Space on Ubuntu 12.04 LTS using a local MySQL server and serving requests with Nginx and Gunicorn. It assumes a clean install and root shell access.

#### Requirements

- 764MB RAM with 1GB preferred for high-traffic instances. This may be flexible depending on your specific load.
- The following packages in addition to the normal core install:
  - git-core
  - build-essential
  - python-dev
  - python-pip
  - nginx
  - bzip2
  - memcached
  - libmysqlclient-dev
  - mysql-server
  - libxml2-dev
  - libxslt-dev
  - rabbitmq-server
  - supervisor

You can install all required packages with the following. You will be prompted for a mysql root password. You may leave this blank if you wish, but it is recommended that you set a secure password and remember it for later.

**$ sudo apt-get install git-core build-essential python-dev python-pip nginx bzip2 memcached libmysqlclient-dev mysql-server libxml2-dev libxslt-dev rabbitmq-server supervisor**

You will also be needing to edit text, so make sure to install your favorite text editor if *nano* or *vi* (not *vim*) aren't your cup of tea:

**$ sudo apt-get install vim**

Next, you will need to upgrade *distribute* and install *virtualenv*: **$ sudo easy_install -U distribute**

**$ sudo pip install virtualenv**

Finally, you must create the MySQL database for Eve-Wspace to use:

**$ mysql -u root -p**

Enter the root password you set when installing MySQL before:

**Password:**

Then, create the database and grant access to it to a new mysql user called *maptool* with a password you should remember for later. If you want to simply use the root MySQL user, simply ignore the *GRANT PRIVILEGES* command.:

```
mysql> CREATE DATABASE evewspace CHARACTER SET utf8;

mysql> GRANT ALL PRIVILEGES ON evewspace.* TO 'maptool'@'localhost' IDENTIFIED BY '<insert a passwor

mysql> quit
```

### Create a User

You should not run Eve W-Space as root for security. You should create a dedicated account called *maptool* with a home directory of */home/maptool*. You can name this user whatever you want, just replace all instances of */home/maptool* with your user's home directory.

**$ sudo useradd -m -s /bin/bash maptool**

**$ sudo passwd maptool**

### Set Up the Home Directory

Let's become our user for this part to ensure permissions are proper and switch to the install location:

**$ sudo su maptool**

**$ cd /home/maptool**

Now, let's create a directory to be used for serving static files later:

**$ mkdir /home/maptool/static**

Next, you need to get the Eve W-Space files. You can either clone the latest revision from *git* or you can download a packaged release and unpack it.

To clone from Github:

**$ git clone https://github.com/marbindrakon/eve-wspace.git**

To use a packaged release:

You need to download eve-wspace from http://marbindrakon.github.com/eve-wspace/ to get latest zip or tarball package (0.1.1 at time of writing):

**$ wget https://github.com/marbindrakon/eve-wspace/archive/v0.1.1.tar.gz**

Then you can unpack the file and rename the directory to *eve-wspace* to match the clone method:

**$ tar xvzf v0.1.1.tar.gz && mv eve-wspace-0.1.1 eve-wspace**

### Install Eve-Wspace Environment

Next, you should create and activate a virtual Python environment for Eve W-Space so that it cannot conflict with any system Python packages:

**$ virtualenv –no-site-packages /home/maptool/eve-wspace**

**$ source /home/maptool/eve-wspace/bin/activate**

You will notice that your shell changes to include *(eve-wspace)* when the virtual environment is active.

Now you can install the required Python packages:

**(eve-wspace)$ pip install -r /home/maptool/eve-wspace/requirements.txt**

### Configuring local_settings.py

Now for the fun part, copy the local_settings.py.example file to local_settings.py in the same directory, open it up, and edit it to suit your enviornment:

**(eve-wspace)$ cd /home/maptool/eve-wspace/evewspace/evewspace**

**(eve-wspace)$ cp local_settings.py.example local_settings.py**

**(eve-wspace)$ nano local_settings.py**

While editing, you should pay particular attention to the top part of th efile, ensuring that the database statement matches the database, user, and password you created in MySQL earlier and that you add a SECRET_KEY and set the STATIC_ROOT value::

```
#Example:

# Set this to False for production or you'll leak memory
DEBUG = False
#DEBUG = True

# Set this to a secret value, google "django secret key" will give you
# plenty of generators to choose from

SECRET_KEY = 'sadf98709283j7r098j09a8fd7sdfj89j7f9a8sdf09a8fd'

# Set this to the directory you are service static files out of so that
# manage.py collectstatic can put them in the right place

STATIC_ROOT = "/home/maptool/static/"

DATABASES = {
        'default': {
                'ENGINE': 'django.db.backends.mysql', # Add 'postgresql_psycopg2', 'postgresql', 'mys
                'NAME': 'evewspace',                      # Or path to database file if using sqlite
                'USER': 'maptool',                      # Not used with sqlite3.
```

```
                'PASSWORD': 'really_secure_password',                   # Not used with sqlite3.
                'HOST': '',                         # Set to empty string for localhost. Not used with s
                'PORT': '',                         # Set to empty string for default. Not used with sq
        }
}
```

Look at the rest of the *local_settings.py* file and see if there is anything you want to change. The default values for memcached and amqp work for the Ubuntu memcached and rabbitmq defaults.

### Initializing the Database

Initializing the database falls into two parts: Loading the Eve static data and initializing the Eve W-Space instance.

#### Static Data

CCP releases a Static Data Export for each major patch in MS SQL format. Steve Ronuken makes MySQL conversions available shortly thereafter. These conversions can be downloaded from http://www.fuzzwork.co.uk/dump/ if you are going to be installing multiple instances, you should download the dump once and re-use it if at all possible.:

```
(eve-wspace)$ cd /home/maptool
(eve-wspace)$ curl -O http://www.fuzzwork.co.uk/dump/mysql55-retribution-1.1-84566.tbz2
(eve-wspace)$ bunzip2 mysql55-retribution-1.1-84566.tbz2
(eve-wspace)$ tar xvf mysql55-retribution-1.1-84566.tar
(eve-wspace)$ mysql -u maptool -p evewspace < retribution-1.1-84566/mysql55-retribution-1.1-84566.sql
```

The sql import will take a few minutes to run. When it completes, your MySQL database will have all of the Static Data Export tables available.

#### Initializing Eve W-Space

Next you will need to run several commands to set up the Eve W-Space tables and preload them with data. If you encounter errors here, they are most likely caused by bad settings in *local_settings.py*, not having the virtual environment activated, or permissions.:

```
(eve-wspace)$ cd /home/maptool/eve-wspace/evewspace
(eve-wspace)$ ./manage.py syncdb --all --noinput
(eve-wspace)$ ./manage.py migrate --fake
(eve-wspace)$ ./manage.py buildsystemdata
Note:This will take a while (~5-10min)
(eve-wspace)$ ./manage.py loaddata */fixtures/*.json
(eve-wspace)$ ./manage.py defaultsettings
(eve-wspace)$ ./manage.py resetadmin
(eve-wspace)$ ./manage.py syncrss
(eve-wspace)$ ./manage.py collectstatic --noinput
```

### Using the Development Server

If you've made it this far, congratulations! Eve W-Space is set up. From here, you can run the console development server directly or continue with setting up the rest of a production environment (Nginx, Gunicorn, Supervisor).

To start the development server:

**(eve-wspace)$ cd /home/maptool/eve-wspace/evewspace**

**(eve-wspace$ ./manage.py runserver 0.0.0.0:8000**

Now you can navigate to your server on port 8000 and see your instance. However, you need to have celery running as well for many tasks to work properly. In another shell:

**(eve-wspace)$ cd /home/maptool/eve-wspace/evewspace**

**(eve-wspace)$ ./manage.py celery worker -B –loglevel=info**

When both are running at the same time, you should be able to use all functions. If you want things to run a bit more permanently, continue reading.

## Setting Up a Production Stack

To serve Eve W-Space in production, you should use a dedicated http daemon to serve static files and either serve the Eve W-Space application itself either through the http daemon itself (as with Apache's mod_wsgi setup) or through a seperate tool which the http daemon will proxy requests to. This guide follows the latter route.

### Installing Gunicorn

This guide uses Gunicorn, a lightweight wsgi server written in Python to serve the Django app itself.

To install:

**(eve-wspace)$ pip install gunicorn**

### Configuring Supervisor

Unless you want to run celery and gunicorn through the console in *screen* or *tmux*, you will want to daemonize them in some way. This guide uses supervisor, but there are many other options available.

At this point, you can log out of the maptool user and go back to your normal account:

**(eve-wspace)$ deactivate**

**$ exit**

You need to tell supervisor about the tools you want it to run, to do that, you need to create a config file in */etc/supervisor/conf.d* for gunicorn and celeryd:

**$ sudo nano /etc/supervisor/conf.d/celeryd.conf**:

```
[program:celeryd]
command=python manage.py celery worker -B --loglevel=info
directory=/home/maptool/eve-wspace/evewspace
environment=PATH=/home/maptool/eve-wspace/bin
user=maptool
autostart=true
autorestart=true
redirect_stderr=True
```

**$ sudo nano /etc/supervisor/conf.d/gunicorn.conf**:

```
[program:gunicorn]
command=gunicorn_django --workers=4 -b 0.0.0.0:8000 settings.py
directory=/home/maptool/eve-wspace/evewspace/evewspace
environment=PATH=/home/maptool/eve-wspace/bin
user=maptool
autostart=true
```

```
autorestart=true
redirect_stderr=True
```

To finish it off, you need to stop and then start supervisor to reload the config and start the services:

**$ sudo service supervisor stop**

**$ sudo service supervisor start**

And confirm that both started successfully:

**$ sudo supervisorctl status**:

```
celeryd                          RUNNING    pid 4335, uptime 33 days, 19:16:02
gunicorn                         RUNNING    pid 4302, uptime 33 days, 19:16:03
```

If either are not in the RUNNING state, either examine the log files in */var/log/supervisor/celeryd-stdout-xxxxxxxxx.log* and */var/log/supervisor/gunicorn-stdout-xxxxxxxx.log* or try running them interactively as discussed previously.

### Configuring Nginx

Now that Eve W-Space itself is running, you need to get people to it. That's where Nginx comes into play. Configuring Nginx is as simple as filling in one config file, creatng a symlink, and reloading the daemon.

**$ sudo nano /etc/nginx/sites-available/evewspace**:

```
    #Example - replace x.x.x.x with your IP or host name if doing name-based vhosts
server {
    listen 80;
    server_name x.x.x.x;
    underscores_in_headers on;

    location /static {
        alias /home/maptool/static;
    }
    location / {
        proxy_pass_header Server;
        proxy_set_header Host $http_host;
        proxy_redirect off;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Scheme $scheme;
        proxy_connect_timeout 10;
        proxy_read_timeout 30;
        proxy_pass http://localhost:8000;
    }
}
```

**$ sudo rm /etc/nginx/sites-enabled/default**

**$ sudo ln -s /etc/nginx/sites-available/evewspace /etc/nginx/sites-enabled/evewspace**

**$ sudo service nginx reload**

Congratulations! Your Eve W-Space instance should now be available at whatever your ip or host name was from the Nginx config. Please see the *Getting Started* page for your next steps. Keep in mind that your instance will have a default administrator registration code until you change it, so do that ASAP.

### 2.1.2 Install: Ubuntu 12.04 LTS with MySQL and Apache

This guide will walk you through installing Eve W-Space on Ubuntu 12.04 LTS using a local MySQL server and serving requests with Apache and mod_wsgi. It assumes a clean install and root shell access.

#### Requirements

- 764MB RAM with 1GB preferred for high-traffic instances. This may be flexible depending on your specific load.
- The following packages in addition to the normal core install:
  - git-core
  - build-essential
  - python-dev
  - python-pip
  - apache2
  - libapache2-mod-wsgi
  - bzip2
  - memcached
  - libmysqlclient-dev
  - mysql-server
  - libxml2-dev
  - libxslt-dev
  - rabbitmq-server
  - supervisor

You can install all required packages with the following. You will be prompted for a mysql root password. You may leave this blank if you wish, but it is recommended that you set a secure password and remember it for later.

**$ sudo apt-get install git-core build-essential python-dev python-pip apache2 bzip2 memcached libmysqlclient-dev mysql-server libxml2-dev libxslt-dev rabbitmq-server supervisor libapache2-mod-wsgi**

You will also be needing to edit text, so make sure to install your favorite text editor if *nano* or *vi* (not *vim*) aren't your cup of tea:

**$ sudo apt-get install vim**

Next, you will need to upgrade *distribute* and install *virtualenv*: **$ sudo easy_install -U distribute**

**$ sudo pip install virtualenv**

Finally, you must create the MySQL database for Eve-Wspace to use:

**$ mysql -u root -p**

Enter the root password you set when installing MySQL before:

**Password:**

Then, create the database and grant access to it to a new mysql user called *maptool* with a password you should remember for later. If you want to simply use the root MySQL user, simply ignore the *GRANT PRIVILEGES* command.:

```
mysql> CREATE DATABASE evewspace CHARACTER SET utf8;

mysql> GRANT ALL PRIVILEGES ON evewspace.* TO 'maptool'@'localhost' IDENTIFIED BY '<insert a passwor

mysql> quit
```

### Create a User

You should not run Eve W-Space as root for security. We will create a dedicated account called *maptool* with a home directory of */home/maptool*. You can name this user whatever you want, just replace all instances of */home/maptool* with your user's home directory.

**$ sudo useradd -m -s /bin/bash maptool**

**$ sudo passwd maptool**

### Set Up the Home Directory

Let's become our user for this part to ensure permissions are proper and switch to the install location:

**$ sudo su maptool**

**$ cd /home/maptool**

Now, let's create a directory to be used for serving static files later:

**$ mkdir /home/maptool/static**

Next, you need to get the Eve W-Space files. We can either clone the latest revision from *git* or you can download a packaged release and unpack it.

To clone from Github:

**$ git clone https://github.com/marbindrakon/eve-wspace.git**

To use a packaged release:

You need to download eve-wspace from http://marbindrakon.github.com/eve-wspace/ to get latest zip or tarball package (0.1.1 at time of writing):

**$ wget https://github.com/marbindrakon/eve-wspace/archive/v0.1.1.tar.gz**

Then you can unpack the file and rename the directory to *eve-wspace* to match the clone method:

**$ tar xvzf v0.1.1.tar.gz && mv eve-wspace-0.1.1 eve-wspace**

### Install Eve-Wspace Environment

Next, you should create and activate a virtual Python environment for Eve W-Space so that it cannot conflict with any system Python packages:

**$ virtualenv –no-site-packages /home/maptool/eve-wspace**

**$ source /home/maptool/eve-wspace/bin/activate**

You will notice that your shell changes to include *(eve-wspace)* when the virtual environment is active.

Now you can install the required Python packages:

**(eve-wspace)$ pip install -r /home/maptool/eve-wspace/requirements.txt**

### Configuring local_settings.py

Now for the fun part, copy the local_settings.py.example file to local_settings.py in the same directory, open it up, and edit it to suit your enviornment:

**(eve-wspace)$ cd /home/maptool/eve-wspace/evewspace/evewspace**

**(eve-wspace)$ cp local_settings.py.example local_settings.py**

**(eve-wspace)$ nano local_settings.py**

While editing, you should pay particular attention to the top part of the file, ensuring that the database statement matches the database, user, and password you created in MySQL earlier and that you add a SECRET_KEY and set the STATIC_ROOT value::

```
#Example:

# Set this to False for production or you'll leak memory
DEBUG = False
#DEBUG = True

# Set this to a secret value, google "django secret key" will give you
# plenty of generators to choose from

SECRET_KEY = 'sadf98709283j7r098j09a8fd7sdfj89j7f9a8sdf09a8fd'

# Set this to the directory you are service static files out of so that
# manage.py collectstatic can put them in the right place

STATIC_ROOT = "/home/maptool/static/"

DATABASES = {
        'default': {
                'ENGINE': 'django.db.backends.mysql', # Add 'postgresql_psycopg2', 'postgresql', 'mys
                'NAME': 'evewspace',                      # Or path to database file if using sqlite
                'USER': 'maptool',                      # Not used with sqlite3.
                'PASSWORD': 'really_secure_password',                  # Not used with sqlite3.
                'HOST': '',                      # Set to empty string for localhost. Not used with s
                'PORT': '',                      # Set to empty string for default. Not used with sq
        }
}
```

Look at the rest of the *local_settings.py* file and see if there is anything you want to change. The default values for memcached and amqp work for the Ubuntu memcached and rabbitmq defaults.

### Initializing the Database

Initializing the database falls into two parts: Loading the Eve static data and initializing the Eve W-Space instance.

#### Static Data

CCP releases a Static Data Export for each major patch in MS SQL format. Steve Ronuken makes MySQL conversions available shortly thereafter. These conversions can be downloaded from http://www.fuzzwork.co.uk/dump/ if you are going to be installing multiple instances, you should download the dump once and re-use it if at all possible.:

```
(eve-wspace)$ cd /home/maptool
(eve-wspace)$ curl -O http://www.fuzzwork.co.uk/dump/mysql55-retribution-1.1-84566.tbz2
```

```
(eve-wspace)$ bunzip2 mysql55-retribution-1.1-84566.tbz2
(eve-wspace)$ tar xvf mysql55-retribution-1.1-84566.tar
(eve-wspace)$ mysql -u maptool -p evewspace < retribution-1.1-84566/mysql55-retribution-1.1-84566.sql
```

The sql import will take a few minutes to run. When it completes, your MySQL database will have all of the Static Data Export tables available.

#### Initializing Eve W-Space

Next you will need to run several commands to set up the Eve W-Space tables and preload them with data. If you encounter errors here, they are most likely caused by bad settings in *local_settings.py*, not having the virtual environment activated, or permissions.:

```
(eve-wspace)$ cd /home/maptool/eve-wspace/evewspace
(eve-wspace)$ ./manage.py syncdb --all --noinput
(eve-wspace)$ ./manage.py migrate --fake
(eve-wspace)$ ./manage.py buildsystemdata
Note:This will take a while (~5-10min)
(eve-wspace)$ ./manage.py loaddata */fixtures/*.json
(eve-wspace)$ ./manage.py defaultsettings
(eve-wspace)$ ./manage.py resetadmin
(eve-wspace)$ ./manage.py syncrss
(eve-wspace)$ ./manage.py collectstatic --noinput
```

### Using the Development Server

If you've made it this far, congratulations! Eve W-Space is set up. From here, you can run the console development server directly or continue with setting up the rest of a production environment.

To start the development server:

**(eve-wspace)$ cd /home/maptool/eve-wspace/evewspace**

**(eve-wspace$ ./manage.py runserver 0.0.0.0:8000**

Now you can navigate to your server on port 8000 and see your instance. However, you need to have celery running as well for many tasks to work properly. In another shell:

**(eve-wspace)$ cd /home/maptool/eve-wspace/evewspace**

**(eve-wspace)$ ./manage.py celery worker -B –loglevel=info**

When both are running at the same time, you should be able to use all functions. If you want things to run a bit more permanently, continue reading.

### Setting Up a Production Stack

To serve Eve W-Space in production, you should use a dedicated http daemon to serve static files and either serve the Eve W-Space application itself either through the http daemon itself (as with Apache's mod_wsgi setup) or through a seperate tool which the http daemon will proxy requests to. This guide follows the Apache route.

#### Configuring Supervisor

Unless you want to run celery and gunicorn through the console in *screen* or *tmux*, you will want to daemonize them in some way. This guide uses supervisor, but there are many other options available.

At this point, you can log out of the maptool user and go back to our normal account:

**(eve-wspace)$ deactivate**

**$ exit**

You need to tell supervisor about the tools you want it to run, to do that, you need to create a config file in */etc/supervisor/conf.d* for gunicorn and celeryd:

**$ sudo nano /etc/supervisor/conf.d/celeryd.conf**:

```
[program:celeryd]
command=python manage.py celery worker -B --loglevel=info
directory=/home/maptool/eve-wspace/evewspace
environment=PATH=/home/maptool/eve-wspace/bin
user=maptool
autostart=true
autorestart=true
redirect_stderr=True
```

To finish it off, you need to stop and then start supervisor to reload the config and start the services:

**$ sudo service supervisor stop**

**$ sudo service supervisor start**

And confirm that celeryd started successfully:

**$ sudo supervisorctl status**:

```
celeryd                          RUNNING    pid 4335, uptime 33 days, 19:16:02
```

If celeryd is not in the RUNNING state, either examine the log files in */var/log/supervisor/celeryd-stdout-xxxxxxxxx.log* or try running it interactively as discussed previously.

#### Configuring Apache

To make Apache serve Eve W-Space on a subdomain (e.g. *http://map.foo.bar*), you can set up a VirtualHost by placing the following text (adapted for your environment) in */etc/apache2/sites-available/evewspace*::

```
<VirtualHost *:80>
        ServerName map.foo.bar
        DocumentRoot /home/maptool/static
        Alias /static /home/maptool/static
        <Directory /home/maptool/static>
                Order allow,deny
                Allow from all
        </Directory>
        <Directory /home/maptool/eve-wspace/evewspace/evewspace/apache>
                Order allow,deny
                Allow from all
        </Directory>
        WSGIDaemonProcess evewspace processes=5 threads=2 display-name=evewspace
        WSGIProcessGroup evewspace
        WSGIScriptAlias / /home/maptool/eve-wspace/evewspace/evewspace/apache/wsgi.py
</VirtualHost>
```

You may need to tune the WSGIDaemonProcess arguments for your environment. The example results in a memory usage of around 853MB including OS and MySQL.

Activate the new VirtualHost by:

**$ sudo ln -s /etc/apache2/sites-available/evewspace /etc/apache2/sites-enabled/evewspace**

**$ sudo service apache2 restart**

If you would rather serve Eve W-Space as a directory (e.g. *http://foo.bar/baz*), add the following to your existing VirtualHost (changing it to fit your environment)::

```
Alias /static /home/maptool/static
<Directory /home/maptool/static>
        Order allow,deny
        Allow from all
</Directory>
<Directory /home/maptool/eve-wspace/evewspace/evewspace/apache>
        Order allow,deny
        Allow from all
</Directory>
WSGIDaemonProcess evewspace processes=5 threads=2 display-name=evewspace
WSGIProcessGroup evewspace
WSGIScriptAlias /baz/ /home/maptool/eve-wspace/evewspace/evewspace/apache/wsgi.py
```

Then activate your changes with:

**$ sudo service apache2 reload**

Congratulations! Your Eve W-Space instance should now be available at whatever your ip or host name was from the Apache config. Please see the *Getting Started* page for your next steps. Keep in mind that your instance will have a default administrator registration code until you change it, so do that ASAP.

### 2.1.3 Getting Started

Now that you have Eve W-Space installed and running, there are a few additional steps you should take to getting it fully up and running. These tasks will also introduce you to administering Eve W-Space from the server console. This guide assumes you have installed Eve W-Space in a virtual Python environment.

#### Starting the Django Shell

The rest of this guide, and much of the overall administrative work of using Eve W-Space in its current state, is done from the Django shell, a Python interpreter with the Django enviornment pre-loaded.

##### Load the Virtual Environment

If you installed Eve W-Space into a virtual Python environment, you need to have that environment active.

**$ source /home/maptool/eve-wspace/bin/activate**

Result: **(eve-wspace) $**

##### Start the Shell

Now that the virtual enviornment is active, you can start the Django shell using the *manage.py* program.

**(eve-wspace) $ /home/maptool/eve-wspace/evewspace/manage.py shell**

That should initialize the Django shell leaving you with:

```
Python 2.7.3 (default, Aug  1 2012, 05:16:07)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

For the rest of this guide, >>> will indicate something to be typed in the Python interpreter.

### Change the Admin Registration Code

The first thing you should do after installing Eve W-Space is changing the registration code for the *Admins* group. By default, this is set to *evewspace*.

To do this from the Django shell::

```
>>>from django.contrib.auth.models import Group
>>>adm_group = Group.objects.get(name="Admins")
>>>adm_prof = adm_group.profile.get()
>>>adm_prof.regcode = 'my_super_sekrit_regcode'
>>>adm_prof.save()
```

Now the registration code for the *Admins* group is whatever you put in place of *my_super_sekrit_regcode*. An account created with this registration code will have all permissions.

### Create a Group for Normal Users

You probably don't want all users to have all permissions in any production install, so you should create another group for normal users.

From the Django shell (Note: You can skip the first line if you are using the same shell as the last section.):

```
>>>from django.contrib.auth.models import Group
>>>group = Group(name="Awesome Users")
>>>group.save()
>>>grp_prof = group.profile.get()
>>>grp_prof.regcode = 'probably_less_sekrit_regcode'
>>>grp_prof.save()
```

Now any user registering with the registration code *probably_less_sekrit_regcode* will be in the *Awesome Users* group.

You can give this group basic map permissions from the Map Admin panel under the "Global Permissions" section.

### Set the Domain Name (for IGB Trust)

Several map features rely on Eve W-Space being run in a trusted IGB session (although the IGB isn't required for core functions). For Eve W-Space to automatically request IGB trust, you need to tell it what domain it will be run as (that domain will be requested as the trusted URL).

To set this from the Django shell (make sure to use http:// or https:// or the IGB will complain):

```
>>>from django.contrib.sites.models import Site
>>>Site.objects.update(domain="https://alpha.evewspace.com")
1L
```

You should replace *https://alpha.evewspace.com* with whatever domain users will use to access your instance.

### 2.1.4 Upgrading EWS

Here are the general instructions for upgrading your Eve W-Space instance to a new release.

#### Upgrade Files

If you are tracking the Github repository (release verisons are in the master branch), you can simply use git to update your files:

As a user that can write to your instance directory:

**$ git pull**

If you installed from the tarball, then simply overwrite the old files with the new version.

#### Upgrade Requirements

A new version may introduce new requirements. As root if you installed without a virtualenv, or with your virtualenv active:

**(eve-wspace) $ pip install —upgrade –r requirements.txt**

Note: requirements.txt is located in the repository root

#### Upgrade Database

Eve W-Space uses South for database schema and data migrations. To migrate to the new version, simply run:

**(eve-wspace) $ ./manage.py migrate**

Simply say 'yes' to any tables being removed unless they are yours.

Note: manage.py is located in the evewspace directory from the repository root

Note: If you installed into a virtualenv, it should be active

#### Upgrade Static Files

With your virtualenv active (if applicable):

**(eve-wspace) $ ./manage.py collectstatic —noinput**

#### Reset Settings

A new version may introduce new settings that need to be initialized, to do so, reset your settings to defaults.

With your virtualenv active (if applicable):

**(eve-wspace) $ ./manage.py defaultsettings**

#### Reset Services

Now that the instance is up to the new version, you will need to restart celery and your application server for the changes to fully take effect:

How this works will vary based on your enviornment. The instructions below are for the environments set up with the install guides.

As root:

**# supervisorctl restart celeryd**

For Nginx / gunicorn based installs:

**# supervisorctl restart gunicorn**

For Apache installs on Ubuntu / Debian:

**# service apache2 reload**

#### Done!

If everything went okay, you should now be upgraded to the new version. You may need to refresh some pages to get updated javascript if your browser has cached the old data.

If you have trouble or questions, please join us on Coldfront IRC in #eve-wspace, in game at Eve W-Space, or via e-mail at marbin@evewspace.com.

## 2.2 Administration Guide

These pages have guides for accomplishing various administrative tasks from the django console.

## 2.3 Development Documentation

These pages contain auto-generated module and funciton definitions for the current codebase.

### 2.3.1 Data Models

Below are the various data models in Eve W-Space.

#### core

**class** core.models.**Alliance**(**args*, ***kwargs*)
Represents an alliance, data pulled from api

**class** core.models.**ConfigEntry**(**args*, ***kwargs*)
A configuration setting that may be changed at runtime.

**class** core.models.**Constellation**(**args*, ***kwargs*)
Core model for static constellation data, references Region

**class** core.models.**Corporation**(**args*, ***kwargs*)
Represents a corporation, data pulled from api

**class** core.models.**Faction**(**args*, ***kwargs*)
Faction(id, name, description, iconid)

**class** core.models.**Location**(**args*, ***kwargs*)
Core model for SDD mapDenormalize table that generic locations map to.

**class** core.models.**LocationWormholeClass**(**args*, ***kwargs*)
Core model for SDD mapLocationWormholeClasses used to generate system tables.

**class** core.models.**MarketGroup**(**args*, ***kwargs*)
A market group from the Eve SDD.

**class** core.models.**NewsFeed**(**args*, ***kwargs*)
Contains information about an RSS feed. If user is None, the feed is global.

**class** core.models.**Region**(**args*, ***kwargs*)
Core model for static region data

**class** core.models.**StarbaseResource**(**args*, ***kwargs*)
Core model for SDD invStarbaseResources table. Maps tower types to their fuel

**class** core.models.**StarbaseResourcePurpose**(**args*, ***kwargs*)
Core model for SDD invControlTowerResourcePurpose table.

**class** core.models.**SystemData**(**args*, ***kwargs*)
Core model for static system data from the SDD, references Region and Constellation

**class** core.models.**SystemJump**(**args*, ***kwargs*)
Core model for SDD mapSolarSystemJumps used in A* calcs.

**class** core.models.**Type**(**args*, ***kwargs*)
A type from the Eve SDD invTypes table.

### Map

**class** Map.models.**ActivePilot**(**args*, ***kwargs*)
Represents the location of a tracked character.

**class** Map.models.**Destination**(**args*, ***kwargs*)
Represents a corp-wide destination whose range should be shown in the map.

**class** Map.models.**KSystem**(**args*, ***kwargs*)
KSystem(id, name, constellation_id, region_id, x, y, z, security, systemdata_ptr_id, sysclass, occupied, info, lastscanned, npckills, podkills, shipkills, updated, first_visited, last_visited, system_ptr_id, sov, jumps)

**distance**(*destination*)
Returns the light-year distance to the destination.

**jumps_to**(*destination*)
Returns the number of gate jumps to the destination by shortest route.

**class** Map.models.**Map**(**args*, ***kwargs*)
Stores the maps available in the map tool. root relates to System model.

**add_log**(*user*, *action*, *visible=False*)
Adds a log entry into a MapLog for the map.

**add_system**(*user*, *system*, *friendlyname*, *parent=None*)
Adds the provided system to the map with the provided friendly name. Returns the new MapSystem object.

**as_json**(*user*)
Returns the json representation of the map for the mapping Javascript.

**get_permission**(*user*)
Returns the highest permision that user has on the map. 0 = No Access 1 = Read Access 2 = Write Access

**snapshot**(*user*, *name*, *description*)
Makes and returns a snapshot of the map.

**class** Map.models.**MapLog**(**args*, ***kwargs*)
Represents an action that has taken place on a map (e.g. adding a signature). This is used for pushing updates since last page load to clients.

**class** Map.models.**MapPermission**(**args*, ***kwargs*)
Relates a user group to it's map permissions. Non-restricted groups will have change access to all maps.

**class** Map.models.**MapSystem**(**args*, ***kwargs*)
Stores information regarding which systems are active in which maps at the present time.

**connect_to**(*system*, *topType*, *bottomType*, *topBubbled=False*, *bottomBubbled=False*, *timeStatus=0*, *massStatus=0*)
Connect self to system and add the connecting WH to map, self is the bottom system. Returns the connecting wormhole.

**remove_system**(*user*)
Removes the supplied system and all of its children. If there are no other instances of the system on other maps, also clears its sigs.

**class** Map.models.**Signature**(**args*, ***kwargs*)
Stores the signatures active in all systems. Relates to System model.

**activate**()
Toggles the site activation.

**clear_rats**()
Toggles the NPCs cleared.

**escalate**()
Toggles the sig escalation.

**increment_downtime**()
Increments the downtime count and does downtime cleanup of updated and activated.

**save**(**args*, ***kwargs*)
Ensure that Sig IDs are proper.

**update**()
Mark the signature as having been updated since DT.

**class** Map.models.**SignatureForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)
This form should only be used with commit=False since it does not set the system or updated fields.

**class** Map.models.**SignatureType**(**args*, ***kwargs*)
Stores the list of possible signature types for the map tool. Custom signature types may be added at will.

**class** Map.models.**SiteSpawn**(**args*, ***kwargs*)
Contains the site spawn list for a site as HTML.

**class** Map.models.**Snapshot**(**args*, ***kwargs*)
Represents a snapshot of the JSON strings that are used to draw a map.

**class** Map.models.**System**(**args*, ***kwargs*)
Stores the permanent record of a solar system. This table should not have rows added or removed through Django.

**class** Map.models.**WSystem**(**args*, ***kwargs*)
WSystem(id, name, constellation_id, region_id, x, y, z, security, systemdata_ptr_id, sysclass, occupied, info, lastscanned, npckills, podkills, shipkills, updated, first_visited, last_visited, system_ptr_id, static1_id, static2_id, effect)

**class** Map.models.**Wormhole**(**args*, ***kwargs*)
An instance of a wormhole in a map. Wormhole have a 'top' and a 'bottom', the top refers to the side that is found first (and the bottom is obviously the other side)

**class** Map.models.**WormholeType**(**args*, ***kwargs*)
Stores the permanent information on wormhole types. Any changes to this table should be made with caution.

**dest_string**()
Returns a readable destination. Cx for w-space and H, L, N otherwise.

### POS

**class** POS.models.**CorpPOS**(**args*, ***kwargs*)
A corp-controlled POS with manager and password data.

**class** POS.models.**POS**(**args*, ***kwargs*)
Represents a POS somewhere in space.

**fit_from_dscan**(*dscan*)
Fills in a POS's fitting from a copy / paste of d-scan results.

**size**()
Returns the size of the tower, Small Medium or Large.

**class** POS.models.**POSApplication**(**args*, ***kwargs*)
Represents an application for a personal POS.

**class** POS.models.**POSVote**(**args*, ***kwargs*)
Represents a vote on a personal POS application.

### account

**class** account.models.**GroupProfile**(**args*, ***kwargs*)
GroupProfile defines custom fields tied to each Group record.

**class** account.models.**PlayTime**(**args*, ***kwargs*)
PlayTime represents a choice of play times for use in several forms.

**class** account.models.**RegistrationForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)
Extends the django registration form to add fields.

**class** account.models.**UserProfile**(**args*, ***kwargs*)
UserProfile defines custom fields tied to each User record in the Django auth DB.

**update_location**(*system*)
updates the current location and last active timestamp for this user

account.models.**create_group_profile**(*sender*, *instance*, *created*, ***kwargs*)
Handle group creation event and create a new group profile.

account.models.**create_user_profile**(*sender*, *instance*, *created*, ***kwargs*)
Handle user creation event and create a new profile to match the new user

### Alerts

**class** Alerts.models.**Subscription**(**args*, ***kwargs*)
Mapping table that relates Users to their subscriped SubscriptionGroups.

**class** Alerts.models.**SubscriptionGroup**(**args*, ***kwargs*)
Contians the definition for alert broadcast groups.

**get_user_perms**(*user*)

**Returns a tuple of permissions for the subscription group as such:** (can_broadcast, can_join)

A user's highest permissions in both are returned and special gorups will always return can_join = False.

**class** Alerts.models.**SubscriptionGroupPermission**(**args*, ***kwargs*)
Mapping table that relates Groups to their permissions for SubscriptionGroups.

### API

**class** API.models.**APICachedDocument**(**args*, ***kwargs*)
APICachedDocument represents a chached API document for our cache handler.

**class** API.models.**APICharacter**(**args*, ***kwargs*)
API Character contains the API security information of a single character.

**class** API.models.**APIKey**(**args*, ***kwargs*)
API Key object relates to User and contains key id, vcode, and validation information.

**class** API.models.**APIShipLog**(**args*, ***kwargs*)
API Ship Log contains a timestamped record of a ship being flown by a character.

**class** API.models.**CorpAPIKey**(**args*, ***kwargs*)
Corp API keys used for corp stuff.

**class** API.models.**MemberAPIKey**(**args*, ***kwargs*)
API key for individual member account.

### SiteTracker

**class** SiteTracker.models.**Claim**(**args*, ***kwargs*)
Represents a User's claim for a claim period.

**class** SiteTracker.models.**ClaimPeriod**(**args*, ***kwargs*)
Represents a claim period that Users can claim against.

**class** SiteTracker.models.**Fleet**(**args*, ***kwargs*)
Represents a SiteTracker fleet.

**active_members**()
Return a list of active members.

**close_fleet**()
Closes the SiteTracker fleet.

**credit_site**(*site_type*, *system*, *boss*)
Credits a site.

**join_fleet**(*user*)
Adds user to fleet.

**leave_fleet**(*user*)
Removes user from fleet.

**make_boss**(*user*)
Change the current fleet boss.

**class** SiteTracker.models.**PayoutEntry**(**args*, ***kwargs*)
Represents an entry in the payout report.

**class** SiteTracker.models.**PayoutReport**(**args*, ***kwargs*)
Represents a payout report and contains general information about the payout period.

**class** SiteTracker.models.**SiteRecord**(**args*, ***kwargs*)
Represents the record of a site run.

**is_pending**(*user*)
Return True if user's credit is pending.

**class** SiteTracker.models.**SiteRole**(**args*, ***kwargs*)
Represents a role for a sitetracker fleet.

**class** SiteTracker.models.**SiteType**(**args*, ***kwargs*)
Represents a type of site that can be credited.

**class** SiteTracker.models.**SiteWeight**(**args*, ***kwargs*)
Represents the raw points available for a site type / system class combo

**class** SiteTracker.models.**SystemWeight**(**args*, ***kwargs*)
Respresents a multiplier for site credit for a system.

**class** SiteTracker.models.**UserLog**(**args*, ***kwargs*)
Represents a user's sitetracker log.

**pending_sites**()
Returns a list of site records which are pending credit.

**class** SiteTracker.models.**UserSite**(**args*, ***kwargs*)
Represents a user's credit for a site.

**approve**()
Mark the site approved.

### Recruitment

**class** Recruitment.models.**Action**(**args*, ***kwargs*)
Represents an action that can be taken on an application e.g Intel Ran

**class** Recruitment.models.**AppAction**(**args*, ***kwargs*)
Represents an action taken on an application.

**class** Recruitment.models.**AppQuestion**(**args*, ***kwargs*)
Represents a question to be asked on the application.

**class** Recruitment.models.**AppResponse**(**args*, ***kwargs*)
Represents a response to a custom application question.

**class** Recruitment.models.**AppVote**(**args*, ***kwargs*)
Represents a vote on an application

**class** Recruitment.models.**Application**(**args*, ***kwargs*)
Represents a recruitment application.

**class** Recruitment.models.**Interest**(**args*, ***kwargs*)
Represents an option for the 'What are you interests? question'

**class** Recruitment.models.**Interview**(**args*, ***kwargs*)
Represents an interview for an application.

**class** Recruitment.models.**StandigsRequirement**(**args*, ***kwargs*)
Represents a standing to be checked against applications.

#### Teamspeak

**class** Teamspeak.models.**GroupMap**(**args*, ***kwargs*)
Maps Django user groups to Teamspeak groups.

**class** Teamspeak.models.**TeamspeakServer**(**args*, ***kwargs*)
Stores teamspeak server configuration.

#### Jabber

**class** Jabber.models.**JabberAccount**(**args*, ***kwargs*)
A jabber account to send messages to. JID is in user@host.tld format

**class** Jabber.models.**JabberSubscription**(**args*, ***kwargs*)
A registered User / Group combo for jabber.

#### Cart

**class** Cart.models.**CartItem**(**args*, ***kwargs*)
CartItem(id, cart_id, item_id, qty, unitcost)

**class** Cart.models.**Request**(**args*, ***kwargs*)
Request(id, originuser_id, totalcost, itemcount, corprequest, daterequested, datefilled, fillcost, deliveredto, datepaid, filluser_id)

**class** Cart.models.**RequestItem**(**args*, ***kwargs*)
RequestItem(id, request_id, item_id, qty, unitcost)

**class** Cart.models.**ShoppingCart**(**args*, ***kwargs*)
Represents an active shopping cart.

### 2.3.2 View Functions

Below are the various view functions in Eve W-Space.

#### core

core.views.**config_view**(*request*, **args*, ***kwargs*)
Gets the configuration page.

core.views.**home_view**(*request*, **args*, ***kwargs*)
The home view detects whether a user has a default map and either directs them to that map or displays a home page template.

### Map

Map.views.**activate_signature**(*request*, **args*, ***kwargs*)
Marks a site activated.

Map.views.**add_destination**(*request*, *dest_user=None*)
Add a destination.

Map.views.**add_personal_destination**(*request*)
Add a personal destination.

Map.views.**add_signature**(*request*, **args*, ***kwargs*)
This function processes the Add Signature form. GET gets the form and POST submits it and returns either a blank form or one with errors. All requests should be AJAX.

Map.views.**add_sigtype**(*request*, **args*, ***kwargs*)
Adds a signature type.

Map.views.**add_spawns**(*request*, **args*, ***kwargs*)
Adds a site spawn.

Map.views.**add_system**(*request*, **args*, ***kwargs*)

**AJAX view to add a system to a current_map. Requires POST containing:** topMsID: map_system ID of the parent map_system bottomSystem: Name of the new system topType: WormholeType name of the parent side bottomType: WormholeType name of the new side timeStatus: Wormhole time status integer value massStatus: Wormhole mass status integer value topBubbled: 1 if Parent side bubbled bottomBubbled: 1 if new side bubbled friendlyName: Friendly name for the new map_system

Map.views.**bulk_sig_import**(*request*, **args*, ***kwargs*)
GET gets a bulk signature import form. POST processes it, creating sigs with blank info and type for each sig ID detected.

Map.views.**collapse_system**(*request*, **args*, ***kwargs*)
Mark the system as collapsed.

Map.views.**create_map**(*request*, **args*, ***kwargs*)
This function creates a map and then redirects to the new map.

Map.views.**delete_destination**(*request*, *dest_id*)
Deletes a destination.

Map.views.**delete_map**(*request*, **args*, ***kwargs*)
Deletes a map.

Map.views.**delete_signature**(*request*, **args*, ***kwargs*)
Deletes a signature.

Map.views.**delete_sigtype**(*request*, **args*, ***kwargs*)
Deletes a signature type.

Map.views.**delete_spawns**(*request*, **args*, ***kwargs*)
Deletes a site spawn.

Map.views.**destination_list**(*request*, *map_id*, **args*, ***kwargs*)
Returns the destinations of interest tuple for K-space systems and a blank response for w-space systems.

Map.views.**destination_settings**(*request*, *user=None*)
Returns the destinations section.

Map.views.**edit_map**(*request*, **args*, ***kwargs*)
Alters a map.

Map.views.**edit_signature**(*request*, **args*, ***kwargs*)
GET gets a pre-filled edit signature form. POST updates the signature with the new information and returns a blank add form.

Map.views.**edit_sigtype**(*request*, **args*, ***kwargs*)
Alters a signature type.

Map.views.**edit_spawns**(*request*, **args*, ***kwargs*)
Alters a site spawn.

Map.views.**edit_system**(*request*, **args*, ***kwargs*)
A GET request gets the edit system dialog pre-filled with current information. A POST request saves the posted data as the new information.

POST values are friendlyName, info, and occupied.

Map.views.**edit_wormhole**(*request*, **args*, ***kwargs*)
A GET request gets the edit wormhole dialog pre-filled with current info. A POST request saves the posted data as the new info. POST values are topType, bottomType, massStatus, timeStatus, topBubbled, and bottomBubbled.

Map.views.**escalate_site**(*request*, **args*, ***kwargs*)
Marks a site as having been escalated.

Map.views.**general_settings**(*request*, **args*, ***kwargs*)
Returns and processes the general settings section.

Map.views.**get_map**(*request*, **args*, ***kwargs*)
Get the map and determine if we have permissions to see it. If we do, then return a TemplateResponse for the map. If map does not exist, return 404. If we don't have permission, return PermissionDenied.

Map.views.**get_signature_list**(*request*, **args*, ***kwargs*)
Determines the proper escalationThreshold time and renders system_signatures.html

Map.views.**global_permissions**(*request*, **args*, ***kwargs*)
Returns and processes the global permissions section.

Map.views.**manual_add_system**(*request*, **args*, ***kwargs*)
A GET request gets a blank add system form with the provided MapSystem as top system. The form is then POSTed to the add_system view.

Map.views.**manual_location**(*request*, **args*, ***kwargs*)
Takes a POST request form AJAX with a System ID and marks the user as being active in that system.

Map.views.**map_refresh**(*request*, **args*, ***kwargs*)
Returns an HttpResponse with the updated systemJSON for an asynchronous map refresh.

Map.views.**map_settings**(*request*, **args*, ***kwargs*)
Returns and processes the settings section for a map.

Map.views.**mark_scanned**(*request*, **args*, ***kwargs*)
Takes a POST request from AJAX with a system ID and marks that system as scanned.

Map.views.**mark_signature_cleared**(*request*, **args*, ***kwargs*)
Marks a signature as having its NPCs cleared.

Map.views.**remove_system**(*request*, **args*, ***kwargs*)
Removes the supplied map_system from a map.

Map.views.**resurrect_system**(*request*, **args*, ***kwargs*)
Unmark the system as collapsed.

Map.views.**set_interest**(*request*, **args*, ***kwargs*)
Takes a POST request from AJAX with an action and marks that system as having either utcnow or None as interesttime. The action can be either "set" or "remove".

Map.views.**sigtype_settings**(*request*, **args*, ***kwargs*)
Returns the signature types section.

Map.views.**site_spawns**(*request*, *map_id*, *ms_id*, *sig_id*)
Returns the spawns for a given signature and system.

Map.views.**sites_settings**(*request*, **args*, ***kwargs*)
Returns the site spawns section.

Map.views.**system_details**(*request*, **args*, ***kwargs*)
Returns a html div representing details of the System given by ms_id in map map_id

Map.views.**system_menu**(*request*, **args*, ***kwargs*)
Returns the html for system menu

Map.views.**system_tooltips**(*request*, **args*, ***kwargs*)
Returns the system tooltips for map_id

Map.views.**wormhole_tooltips**(*request*, **args*, ***kwargs*)
Takes a POST request from AJAX with a Wormhole ID and renders the wormhole tooltip for that ID to response.

### POS

POS.views.**add_pos**(*request*, **args*, ***kwargs*)
GET gets the add POS dialog, POST processes it.

POS.views.**edit_pos**(*request*, **args*, ***kwargs*)
GET gets the edit POS dialog, POST processes it.

POS.views.**remove_pos**(*request*, **args*, ***kwargs*)
Removes the POS. Raises PermissionDenied if it is a CorpPOS.

POS.views.**test_fit**(*request*, **args*, ***kwargs*)
Temporary test method for filling a POS fit from DScan.

### account

### Alerts

Alerts.views.**edit_subscriptions**(*request*)
" GET: Get the edit subscritpions dialog. POST: Process the edit subscriptions dialog.

Alerts.views.**send_ping**(*request*)
" GET: Get the send ping dialog. POST: Process the send ping dialog.

### API

### SiteTracker

SiteTracker.views.**approve_fleet_site**(*request*, *fleetID*, **args*, ***kwargs*)
Approve a pending site while the fleet is active.

SiteTracker.views.**boss_panel**(*request*, *fleetID*, **args*, ***kwargs*)
Return the HTML for the boss control panel popup.

SiteTracker.views.**claim_site**(*request*, *fleetID*, *siteID*, *memberID*)
Claims the site (making it pending) if called by member. Fully grants credit for the site if called by current boss.

SiteTracker.views.**create_fleet**(*request*, **args*, ***kwargs*)
Takes a system ID via POST and makes a sitetracker fleet with the requesting user as initial and current boss. Then adds the requestor to the fleet as a member.

SiteTracker.views.**credit_site**(*request*, *fleetID*, **args*, ***kwargs*)

**Credit a site to the given fleet. Takes POST input:** type = short_name of site type

SiteTracker.views.**disband_fleet**(*request*, *fleetID*, **args*, ***kwargs*)
Disband the fleet.

SiteTracker.views.**join_fleet**(*request*, **args*, ***kwargs*)
Join the current user to the fleet.

SiteTracker.views.**kick_member**(*request*, *fleetID*, **args*, ***kwargs*)
Removes a member from the fleet.

SiteTracker.views.**leave_fleet**(*request*, **args*, ***kwargs*)
Leaves the given fleet. If fleetID is not provided, leave all fleets.

SiteTracker.views.**promote_member**(*request*, **args*, ***kwargs*)
Promote the given member to boss. Boss permisison not required since we allow for siezure from an AFK boss.

SiteTracker.views.**refresh_boss_member**(*request*, *fleetID*, **args*, ***kwargs*)
Returns an updated details view for a boss panel member.

SiteTracker.views.**refresh_fleets**(*request*, **args*, ***kwargs*)
Return a template with tags for the myfleets and availfleets lists.

SiteTracker.views.**remove_site**(*request*, *fleetID*, **args*, ***kwargs*)
Uncredit a site to the given fleet.

SiteTracker.views.**status_bar**(*request*, **args*, ***kwargs*)
Return a template with just the ST status bar tag.

SiteTracker.views.**unclaim_site**(*request*, *fleetID*, *siteID*, *memberID*)
Unclaims a site during a running fleet.

#### Recruitment

#### Teamspeak

#### Jabber

#### Cart

### 2.3.3 Utility Functions

Below are the various utility functions and classes in Eve W-Space.

#### core

core.utils.**get_config**(*name*, *user*)
Gets the correct config value for the given key name. Value with the given user has priority over any default value.

### Map

**class** Map.utils.**MapJSONGenerator**(*map*, *user*)
A MapJSONGenerator is instantiated with a map and user. It provides a method that returns the JSON representation of the map.

**get_path_to_map_system**(*system*)
Returns a list of MapSystems on the route between the map root and the provided MapSystem.

**get_system_icon**(*system*)
Takes a MapSystem and returns the appropriate icon to display on the map as a realative URL.

**get_systems_json**()
Returns a JSON string representing the systems in a map.

**recursive_system_data_generator**(*mapSystems*, *syslist*, *levelX*)
Prepares a list of MapSystem objects for conversion to JSON for map JS. Takes a queryset of MapSystems and the current list of systems prepared for JSON.

**system_to_dict**(*system*, *levelX*)
Takes a MapSystem and X,Y data and returns the dict of information to be passed to the map JS as JSON.

**class** Map.utils.**RouteFinder**
A RouteFinder object is created with two system objects and has methods for getting the shortest stargate jump route length, the light-year distance, and the shortest stargate route as a list of KSystem objects.

Map.utils.**convert_signature_id**(*sigid*)
Standardize the signature ID to XXX-XXX if info is available.

Map.utils.**get_possible_wh_types**(*system1*, *system2*)
Takes two systems and gets the possible wormhole types between them. For example, given system1 as highsec and system2 as C2, it should return R943 and B274. system1 is the source and system2 is the destination. Results are returned as lists because some combinations have multiple possibilities. Returns a dict in the format {system1: [R943,], system2: [B274,]}.

Map.utils.**get_wormhole_type**(*system1*, *system2*)
Gets the one-way wormhole types between system1 and system2.

### POS

POS.utils.**add_status_info**(*poses*)
Accepts a list of corp poses and returns a list of POSes with status information attached.

A posstatus object has the following attributes: itemid: the POS item id pos: POS object processed status: Status retrieved

### account

account.utils.**get_groups_for_code**(*regcode*)
Returns a list of groups for a given registration code.

account.utils.**register_groups**(*user*, *regcode*)
Registers a user for all groups associated with a registration code.

#### Alerts

#### API

API.utils.**retrieve**(*host*, *path*, *params*)
Get an API document from our cache.

API.utils.**store**(*host*, *path*, *params*, *doc*, *obj*)
Store an API document in our cache.

API.utils.**timestamp_to_datetime**(*timestamp*)
Converts a UNIX Timestamp (in UTC) to a python DateTime

#### SiteTracker

#### Recruitment

#### Teamspeak

#### Jabber

#### Cart

### 2.3.4 Alert Method Sample

This is the base class that must be used when defining a custom alert_method.

#### AlertMethodBase

Base class for alert method interface.

**class** Alerts.method_base.**AlertMethodBase**

**This is a skeleton AlertMethod. It should handle the following:**

**send_alert(to_users, message, from_user, sub_group)**

**-Send alert to given to_users. Should handle an unregistered user** gracefully.

**register_user(user, sub_group)**

**-Register the provided user and subscription group as being handled** by this method.

**unregister_user(user, sub_group)** -Unregister the given user and subscription group combo

**is_registered(user, sub_group)** -Return True if the user and subscripiton group are registered

**description()** -Return a text description of the method

**description**(*user*, *sub_group*)
Return a one-sentence descritpion of the method for the user to see.

**is_bob_great**()
Hail Bob.

**is_registered**(*user*, *sub_group*)
Return True if the user / group combo is registered.

**register_user**(*user*, *sub_group*)
Register a user / subscription group combo to recieve alerts.

**send_alert**(*to_users*, *message*, *from_user*, *sub_group*)
Send a message to the given user with the message text, from and group.

**unregister_user**(*user*, *sub_group*)
Unregister a user / subscription group combo.

CHAPTER

THREE

# INDICES AND TABLES

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

# PYTHON MODULE INDEX