Octo Documentation

Release 0.1

Nick Groenen

June 08, 2013

CONTENTS

Octo is a "plugin framework" which allows you to write your application as a collection of (optionally interconnected) plugins.

CHAPTER

ONE

OVERVIEW

Octo is a “plugin framework” which allows you to write your application as a collection of (optionally interconnected) plugins. For example, octo makes it easy to write a plugin which runs a basic webserver so you can have other plugins hooking into that in order to expose API endpoints for various different tasks.

Using a plugin-based approach encourages modularity and makes it easy to add or remove functionality, or toggle parts on and off as needed. It also makes it easier for people to share and exchange plugins to satisfy certain needs or to solve various problems.

CHAPTER

TWO

INSTALLING

Octo may be installed from PyPi using pip or easy_install:

```
pip install octo
```

Alternatively, you can install the latest bleeding-edge master version from GitHub directly:

```
pip install https://github.com/zoni/octo/archive/master.zip
```

Lastly, you could also download the code and run `setup.py`:

```
python setup.py install
```

Note: If you wish to work on octo's code itself, then you will want to install the development dependencies as well:

```
pip install -r requirements-development.txt
```

CHAPTER

THREE

RUNNING OCTO

Starting octo is as simple as running `octo.py` with a list of directories to scan for plugins:

```
python octo.py -p plugins
```

Octo will scan the directory `plugins` recursively, loading any configured plugins that it finds. If you have your plugins spread across more than a single directory, you can give this option multiple times:

```
python octo.py -p plugins -p more_plugins
```

You can stop octo by pressing Ctrl+C, or by sending a SIGINT signal from another process (for example, kill).

CHAPTER

FOUR

MAKING AN EXAMPLE PLUGIN

Out of the box, octo won't be able to do much for you. It will try to activate any plugins it finds and then drops into an idle loop. To make it do anything useful, you will have to start writing plugins. Luckily, writing plugins for octo is really easy!

To maintain tradition, lets write a simple *"Hello World"* plugin. All it will do is print "Hello world!" to the console when you activate it.

For this example, we will assume you have an empty directory called `example` to store the plugin files in. As we're writing a "Hello world" plugin, we'll be putting our python code into `example/helloworld.py`.

4.1 The application code

In order to run, our plugin should have a class which extends `octo.plugin.OctoPlugin`. Since the name that we give to this class does not matter, we'll simply call it `HelloWorld`:

```
from octo.plugin import OctoPlugin

class HelloWorld(OctoPlugin):
    """Simple hello world example plugin"""
    pass
```

Obviously, this is still lacking functionality to greet, so we'll need to add a method to our class for that. Looking at `octo.plugin.OctoPlugin`, you can see it already offers `octo.plugin.OctoPlugin.on_activation()` and `octo.plugin.OctoPlugin.on_deactivation()` for us to override. These two methods are called on activation and deactivation of a plugin, respectively.

Using this, we can complete our plugin as follows:

```
from octo.plugin import OctoPlugin

class HelloWorld(OctoPlugin):
    """Simple hello world example plugin"""

    def on_activation(self):
        """Say hello on plugin activation"""
        print("Hello world!")
```

If you now try to run octo, you'll notice that nothing actually happens:

```
$ python octo.py -p example
INFO:root:Initializing with plugin directories: ['example']
^CINFO:root:Interrupt received, shutting down
```

This is because we haven't actually told octo there's a plugin there forit to load! In order to do this, we must add a plugin file that contains a bit of metadata about our plugin.

4.2 The plugin file

Lets create `example/helloworld.plugin`:

```
[Core]
Name = Hello World
Module = helloworld

[Documentation]
Author = Your Name
Version = 0.1
Website = http://example.com
Description = My first plugin

[Config]
Enable = True
```

What this file does is it gives octo some metadata about your plugin, such as the Python module to import for it and whether to activate it or not. Make sure that `Core.Module` contains the name of the file you created for your plugin, as this is how it knows where to find your code.

Also make sure that `Config.Enable` is `True`, if it's anything else, or missing entirely, then octo won't enable your plugin, and that would be sad.

Lastly, while it's generally a good practice, you can omit the `Documentation` items and octo won't care. This is purely a bit of metadata that becomes especially useful if you end up sharing your plugin with other people.

When we run octo again, this time we should see our greeting (we'll turn all logging off as well, to make the output easier to read):

```
$ python octo.py -p example -l none
Hello world!
```

Success! You should now know enough to get started writing your own plugins. However, you'll probably want to spend a little more time looking at the API documentation of `octo.plugin.OctoPlugin` first, so you know what other functionality you can hook into with your own plugins.

CHAPTER

FIVE

API DOCUMENTATION

5.1 octo

5.1.1 octo Package

octo Package

exceptions Module

manager Module

plugin Module

CHAPTER

SIX

CONTRIBUTING

Octo is an opensource project, so I would love your involvement. Please feel free to offer suggestions or criticisms. If you wish to contribute code, I'd be more than happy to integrate your changes if I feel they make a good addition.

In order to make the experience as smooth as possible, please take these guidelines into consideration:

- Before submitting changes, make sure all tests still pass.
- If you add any new code, include tests for it as well. If you need help writing tests, please do not hesitate to reach out to me for help.
- Commit any changes you make one change at a time, with a clear commit message to accompany it. This will ease the review process and makes it easier for people to figure out what happened when looking back at the git log.
- Split unrelated changes into seperate pull requests. This again makes discussion and review easier, and ensures your first change does not block your other changes from being accepted.

CHAPTER

SEVEN

LICENSE

Octo is available under a `2-clause BSD license` (the "Simplified BSD License").

CHAPTER

EIGHT

INDICES AND TABLES

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