extender Documentation

Release 0.0.7

Messense Lv

June 08, 2014

Contents

extender is a simple plug-in/extension system on Python

CHAPTER 1

User's guide

1.1 Installation

If you are familiar with Python, it is strongly suggested that you install everything in virtualenv.

If you are using OS X , make sure you have installed the *Command Line Tools* .

1.1.1 Install using pip

Install extender via pip is easy

```
(sudo) pip install extender
```

and you can also install from the latest source code with pip

```
(sudo) pip install  https://github.com/messense/extender/archive/master.zip
```

1.1.2 Upgrade from older version

It's also easy to upgrade your extender

```
(sudo) pip install -U extender
```

1.1.3 Install with Git

Install with git can always have the latest code

```
git clone git://github.com/messense/extender.git
cd extender
python setup.py install
```

1.2 Write your plugin

1.2.1 Plugin layout

A plugins layout generally looks like the following

```
setup.py
pluginname/
pluginname/__init__.py
pluginname/plugin.py
```

The __init__.py file should contain no plugin logic, and at most, a `__version__ = 'x.x.x'` line. For example, if you want to pull the version using pkg_resources (which is what we recommend), your file might contain

```
try:
__version__ = __import__('pkg_resources') \
    .get_distribution(__name__).version
except Exception:
    __version__ = 'unknown'
```

1.2.2 Plugin class

Inside of plugin.py, you'll declare your Plugin class

```
# -*- coding: utf-8 -*-
from extender import Plugin
import plugin1


class PluginName(Plugin):
    title = 'Plugin Name'
    slug = 'pluginname'
    description = 'My awesome plugin!'
    version = plugin1.__version__

    author = 'Your Name'
    author_url = 'https://github.com/yourname/pluginname'

    def test_func(self, msg):
        return msg
```

You should provider at least `title`, `version` attributes in your plugin class and define whatever method as you wish.

1.2.3 Register your plugin

You can register your plugin via entry_points in your setup.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from setuptools import setup

setup(
    name='pluginname',
    version='0.0.1',
    author='Your name',
    author_email='Your Email address',
    url='https://github.com/yourname/pluginname',
    packages=[
        'pluginname'
    ],
    description='plugin description',
```

```
    install_requires=[
        'extender',
    ],
    include_package_data=True,
    entry_points={
        'extender.plugins': [
            'pluginname = pluginname.plugin:PluginName',
        ]
    },
)
```

You can change entry_points key `extender.plugins` to whatever you want.

That's it! Users will be able to install your plugin via `python setup.py install`. And your plugin will be registered to that entry_points automatically, then you can install/load all these awesome plugins in your code.

1.3 Install and manage plugins

1.3.1 Setup a plugin manager

To install your awesome plugins, you should initialize a `PluginManager` first

```
from extender import PluginManager

plugins = PluginManager()
```

1.3.2 Install plugins

With a instance of `PluginManager` class, we can install plugins of a specific entry_points by calling the `install` method of `PluginManager`

```
from extender import PluginManager
plugins = PluginManager()

plugins.install('extender.plugins')
```

To setup a plugin manager and install plugins quickly you can do it by

```
from extender import PluginManager
plugins = PluginManager(entry_points='extender.plugins')
```

1.3.3 Use plugins

You can use `plugins.call(func_name, *args, **kwargs)` method to invoke the method *func_name* on every plugin and return a list of results which contains return value of each invoked method of plugins

```
from extender import PluginManager
plugins = PluginManager()

plugins.install('extender.plugins')

result_list = plugins.call('say', 1, msg='hello')
```

To get single value of the first invoked plugin's return value, use `first` instead of `call`

```
from extender import PluginManager
plugins = PluginManager()

plugins.install('extender.plugins')

result = plugins.first('say', 1, msg='hello')
```

If you want to apply some method of plugins on a variable and return the value(maybe modified by plugins), you can use `apply`

```
from extender import PluginManager
plugins = PluginManager()

plugins.install('extender.plugins')

value = "Hello world!"
value = plugins.apply('greet', value)
```