scruffy Documentation

Release 0.3.3

snare

September 28, 2016

Contents

Contents:

CHAPTER 1

Scruffy API

1.1 Config

Classes for loading and accessing configuration data.

class scruffy.config.**Config**(*data={}*, *defaults={}*, *root=None*, *path=None*)
Config root node class. Just for convenience.

class scruffy.config.**ConfigApplicator**(*config*)
Applies configs to other objects.

apply(*obj*)
Apply the config to an object.

apply_to_str(*obj*)
Apply the config to a string.

class scruffy.config.**ConfigEnv**(*prefix='SCRUFFY'*, **args*, ***kwargs*)
Config based on based on environment variables.

class scruffy.config.**ConfigFile**(*path=None*, *defaults=None*, *load=False*, *apply_env=False*, *env_prefix='SCRUFFY'*, **args*, ***kwargs*)
Config based on a loaded YAML or JSON file.

load(*reload=False*)
Load the config and defaults from files.

prepare()
Load the file when the Directory/Environment prepares us.

save()
Save the config back to the config file.

class scruffy.config.**ConfigNode**(*data={}*, *defaults={}*, *root=None*, *path=None*)
Represents a Scruffy config object.

Can be accessed as a dictionary, like this:

```
>>> config['top-level-section']['second-level-property']
```

Or as a dictionary with a key path, like this:

```
>>> config['top_level_section.second_level_property']
```

Or as an object, like this:

```
>>> config.top_level_section.second_level_property
```

reset()
Reset the config to defaults.

to_dict()
Generate a plain dictionary.

update(*data={}*, *options={}*)
Update the configuration with new data.

This can be passed either or both *data* and *options*.

options is a dict of keypath/value pairs like this (similar to CherryPy's config mechanism:

```
>>> c.update(options={
...     'server.port': 8080,
...     'server.host': 'localhost',
...     'admin.email': 'admin@lol'
... })
```

data is a dict of actual config data, like this:

```
>>> c.update(data={
...     'server': {
...         'port': 8080,
...         'host': 'localhost'
...     },
...     'admin': {
...         'email': 'admin@lol'
...     }
... })
```

scruffy.config.**update_dict**(*target*, *source*)
Recursively merge values from a nested dictionary into another nested dictionary.

For example:

```
>>> target = {
...     'thing': 123,
...     'thang': {
...         'a': 1,
...         'b': 2
...     }
... }
>>> source = {
...     'thang': {
...         'a': 666,
...         'c': 777
...     }
... }
>>> update_dict(target, source)
>>> target
{
    'thing': 123,
    'thang': {
        'a': 666,
        'b': 2,
        'c': 777
    }
}
```

1.2 Environment

Classes for representing the encompassing environment in which your application runs.

class scruffy.env.**Environment**(*setup_logging=True*, **args*, ***kwargs*)
An environment in which to run a program

add(***kwargs*)
Add objects to the environment.

cleanup()
Clean up the environment

find_config(*children*)
Find a config in our children so we can fill in variables in our other children with its data.

1.3 File

Classes for representing and performing operations on files and directories.

class scruffy.file.**Directory**(*path=None*, *base=None*, *create=True*, *cleanup=False*, *parent=None*, ***kwargs*)
A filesystem directory.

A Scruffy Environment usually encompasses a number of these. For example, the main Directory object may represent *~/.myproject*.

```
>>> d = Directory({
...     path='~/.myproject',
...     create=True,
...     cleanup=False,
...     children=[
...     ...
...     ]
... })
```

path can be either a string representing the path to the directory, or a nested Directory object. If a Directory object is passed as the *path* its path will be requested instead. This is so Directory objects can be wrapped in others to inherit their properties.

add(**args*, ***kwargs*)
Add objects to the directory.

apply_config(*applicator*)
Replace any config tokens with values from the config.

cleanup()
Clean up children and remove the directory.

Directory will only be removed if the cleanup flag is set.

create()
Create the directory.

Directory will only be created if the create flag is set.

exists
Check if the directory exists.

list()
List the contents of the directory.

path
Return the path to this directory.

path_to(*path*)
Find the path to something inside this directory.

prepare()
Prepare the Directory for use in an Environment.

This will create the directory if the create flag is set.

read(*filename*)
Read a file from the directory.

remove(*recursive=True*, *ignore_error=True*)
Remove the directory.

write(*filename*, *data*, *mode='w'*)
Write to a file in the directory.

class scruffy.file.**File**(*path=None*, *create=False*, *cleanup=False*, *parent=None*)
Represents a file that may or may not exist on the filesystem.

Usually encapsulated by a Directory or an Environment.

apply_config(*applicator*)
Replace any config tokens in the file's path with values from the config.

cleanup()
Clean up the file after use in an Environment or Directory.

This will remove the file if the cleanup flag is set.

content
Property for the content of the file.

create()
Create the file if it doesn't already exist.

exists
Whether or not the file exists.

ext
Get the file's extension.

name
Get the file name.

path
Get the path to the file relative to its parent.

prepare()
Prepare the file for use in an Environment or Directory.

This will create the file if the create flag is set.

read()
Read and return the contents of the file.

remove()
Remove the file if it exists.

write(*data*, *mode='w'*)
Write data to the file.

data is the data to write *mode* is the mode argument to pass to *open()*

class scruffy.file.**JsonFile**(*path=None*, *create=False*, *cleanup=False*, *parent=None*)
A json file that is parsed into a dictionary.

class scruffy.file.**LockFile**(**args*, ***kwargs*)
A file that is automatically created and cleaned up.

create()
Create the file.

If the file already exists an exception will be raised

class scruffy.file.**LogFile**(*path=None*, *logger=None*, *loggers=[]*, *formatter={}*, *format=None*, **args*, ***kwargs*)
A log file to configure with Python's logging module.

configure()
Configure the Python logging module for this file.

prepare()
Configure the log file.

class scruffy.file.**PackageDirectory**(*path=None*, *package=None*, **args*, ***kwargs*)
A filesystem directory relative to a Python package.

class scruffy.file.**PackageFile**(*path=None*, *create=False*, *cleanup=False*, *parent=None*, *package=None*)
A file whose path is relative to a Python package.

class scruffy.file.**PluginDirectory**(*path=None*, *base=None*, *create=True*, *cleanup=False*, *parent=None*, ***kwargs*)
A filesystem directory containing plugins.

load()
Load the plugins in this directory.

prepare()
Preparing a plugin directory just loads the plugins.

class scruffy.file.**YamlFile**(*path=None*, *create=False*, *cleanup=False*, *parent=None*)
A yaml file that is parsed into a dictionary.

content
Parse the file contents into a dictionary.

1.4 Plugin

Classes for representing and loading plugins.

class scruffy.plugin.**Plugin**
Top-level plugin class, using the PluginRegistry metaclass.

All plugin modules must implement a single subclass of this class. This subclass will be the class collected in the PluginRegistry, and should contain references to any other resources required within the module.

class scruffy.plugin.**PluginManager**
Loads plugins which are automatically registered with the PluginRegistry class, and provides an interface to the plugin collection.

load_plugins(*directory*)
Loads plugins from the specified directory.

directory is the full path to a directory containing python modules which each contain a subclass of the Plugin class.

There is no criteria for a valid plugin at this level - any python module found in the directory will be loaded. Only modules that implement a subclass of the Plugin class above will be collected.

The directory will be traversed recursively.

class scruffy.plugin.**PluginRegistry**(*name*, *bases*, *attrs*)
Metaclass that registers any classes using it in the *plugins* array

1.5 State

Classes for storing a program's state.

class scruffy.state.**State**(*path=None*)
A program's state.

Contains a dictionary that can be periodically saved and restored at startup.

Maybe later this will be subclassed with database connectors and whatnot, but for now it'll just save to a yaml file.

cleanup()
Clean up the saved state.

load()
Load a saved state file.

save()
Save the state to a file.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

S

T

U

W

Y