pyHai Documentation

Release 0.1.1

Mark LaPerriere

November 02, 2011

CONTENTS

A system profiler/auditor written in Python that supports custom plugins and a custom profiler. Custom plugins can superscede builtin plugins to allow for a great deal of flexibility. Plugins follow naming conventions for ease of use.

CHAPTER

ONE

LICENSE

CHAPTER

TWO

PROJECT

http://bitbucket.org/marklap/pyhai

API Docs: tar.gz, zip, source

CHAPTER

THREE

BASIC USE

An example of running pyHai with all the default parameters and using the builtin auditing plugins.:

```
>>> from pyhai import pyhai
>>> pyhai.audit()
```

This will run all the builtin auditing plugins and return a dictionary object with detailed information about the local system. The dictionary keys are the lowercase names of the plugin files (minus the '.py' file name extension) and the contents are the results of the plugin's run.

CHAPTER

FOUR

ADVANCED USE

4.1 Running Additional, Custom Plugins

An example of running pyHai with a single additional plugin path. The builtin plugins will be run before the "custom" plugins found in the path supplied.:

```
>>> from pyhai import pyhai
>>> pyhai.audit('/path/to/my/plugins')
```

This will return the dictionary of results of the auditing run including builtin plugins and the custom plugins found in the path provided.

An example of running pyHai with a number of additional plugin paths.:

```
>>> from pyhai import pyhai
>>> pyhai.audit(['/path/to/my/plugins', '/path/to/some/other/plugins'])
```

This will return the dictionary of results of the auditing run including builtin plugins and all custom plugins.

Important: When using custom plugins, if you name your plugin the same as one of the builtin plugins, your plugin will superscede the builtin plugin. This allows users to provide their own plugins, but could cause confusion if it is unexcpected. For instance, if you write a plugin called 'environment', the builtin plugin - environment - will not run and your plugin will be run instead.

4.2 Suppressing Builtin Plugins

An example of running pyHai with a single custom plugin directory and without running the builtin plugins.:

```
>>> from pyhai import pyhai
>>> pyhai.audit('/path/to/my/plugins', enable_default_plugins=False)
```

Important: If you don't supply custom plugin path(s) and try to disable builtin plugins, an exception is raised - nothing to do.

CHAPTER

FIVE

EXTENDING

This section describes how to create your own auditing plugins and using a custom profiler.

5.1 Creating Auditing Plugins

5.1.1 Conventions

Plugin files must use all lowercase letters and seperate words with underscores.:

```
super_cool_widget.py
```

Plugin class names use CamelCase letters and reflect the plugin file name.:

```
class SuperCoolWidget(AuditorPlugin):
    pass
```

5.1.2 Extending

Plugin classes have a single required method: `run`. This method will be called as part of the default run.

We'll use the builtin "environment" plugin as an example. You can check out the source code in the builtin plugins for additional reference. Example Custom Plugin:

```
# file: environment.py

# import the auditing plugin base class
from pyhai import AuditorPlugin

# name your plugin the same as your file name.
class EnvironmentPlugin(AuditorPlugin):
    # provide a run method
    def run(self, *args, **kwargs):
        return dict(os.environ)
```

5.1.3 Hooks

There are two hooks provided in the auditing base class. If you provide them in your custom auditing class, pyHai will execute them as such.

The `before` method will be called before the `run` method is called. The results of the `before` method will passed along to the `run` method as a keyword argument: `before_results`.

The `after` method will be called after the `run` method is called. The results of the run are available in the plugin class by calling `self._get_results()`.

5.2 Creating a Custom Profiler

5.2.1 Extending

Simply import the profiler base class and extend it. There are two methods that need to be supplied by the custom profile class: `profile`, and `system`. The profile method needs to return a dictionary with some basic information about the system running pyHai to help auditing plugins make desicions about how to audit.:

```
from pyhai.profilers.base import ProfilerBase

class DefaultProfiler(ProfilerBase):

    def profile(self):
        # return a simple dictionary with basic information about the local system
        # this dictionary will be provided to all auditing plugins
        pass

    def system(self):
        # return a string representing the type of system pyHai is running on
        # for example the default profiler returns the results of a call to the system method of the
        # python module platform
        pass
```

CHAPTER

SIX

INDICES AND TABLES

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