ploogz Documentation

Release 0.0.9

Pat Daburu

Feb 27, 2020

Contents

Ploog it in and go.

To get the source, visit the github repository.

CHAPTER 1

API Documentation

Let's take a look at how ploogz works.

1.1 ploogz

Provide a brief description of the module.

1.2 ploogz.ploogins

Ploogins and Ploogin Loaders

class ploogz.ploogins.**FsPlooginLoader**
Bases: *ploogz.ploogins.PlooginLoader*

This is a ploogin loader that looks for ploogins in the local file system.

__init__
Initialize self. See help(type(self)) for accurate signature.

load(*search_path: typing.List[str]*) → typing.List[ploogz.ploogins.Ploogin]
Load the ploogins found in the search paths on the file system.

Parameters **search_path** (*List[str]*) – the paths the loader will search when the `load()` method is called

class ploogz.ploogins.**Ploogin**(*name: str*)
Bases: `object`

Extend this class to create your own ploogins!

__init__(*name: str*)

Parameters **name** (`str`) – a helpful, descriptive, human-readable name for the plugin

activate
An input for a L{MethodicalMachine}.

active = MethodicalState(method=<function Ploogin.active>)

initialized = MethodicalState(method=<function Ploogin.initialized>)

name
Get the helpful, descriptive, human-readable name for the ploogin.

Return type str

ready = MethodicalState(method=<function Ploogin.ready>)

setup
An input for a L{MethodicalMachine}.

teardown
An input for a L{MethodicalMachine}.

torndown = MethodicalState(method=<function Ploogin.torndown>)

class ploogz.ploogins.**PlooginEventHandler**(*event: ploogz.ploogins.PlooginEvents*, *f: typing.Callable*)

Bases: object

Instances of this are callable and wrap handler functions for well-known event ploogin event types.

__init__(*event: ploogz.ploogins.PlooginEvents*, *f: typing.Callable*)

Parameters

- **event** (*PlooginEvents*) – What type of event does this handler handle?
- **f** (*Callable*) – the handler function

event
What type of event does this handler handle?

class ploogz.ploogins.**PlooginEvents**
Bases: enum.Enum

These are the well known events that occur in the life cycle of a ploogin.

ACTIVATE = ('activate',)

SETUP = ('setup',)

TEARDOWN = 'teardown'

class ploogz.ploogins.**PlooginHost**(*search_path: typing.List[str] = None*, *loader: ploogz.ploogins.PlooginLoader = None*)

Bases: object

Use a host object to load and retrieve your ploogins.

__init__(*search_path: typing.List[str] = None*, *loader: ploogz.ploogins.PlooginLoader = None*)

Parameters **search_path** (List[str] or str) – the paths the plugin host will search when the load() method is called

Seealso *PlooginHost.load()*

Note: If no search path is provided, the default path is builtin/ploogins under the current working directory.

initialized = MethodicalState(method=<function PlooginHost.initialized>)

load
An input for a L{MethodicalMachine}.

loaded = MethodicalState(method=<function PlooginHost.loaded>)

ploogins
Get the ploogins loaded by this host.

Return type Iterator[Ploogin]

teardown
An input for a L{MethodicalMachine}.

torndown = MethodicalState(method=<function PlooginHost.torndown>)

class ploogz.ploogins.**PlooginLoader**
Bases: object

Extend this class to create a ploogin loader that can look through search paths to find and instantiate ploogins.

__init__
Initialize self. See help(type(self)) for accurate signature.

load(*search_path: typing.List[str]*) → typing.List[ploogz.ploogins.Ploogin]
Override this method to implement the loader's primary loading logic.

Parameters **search_path** (*List[str]*) – the paths the plugin host will search when the load() method is called

ploogz.ploogins.**upon_activate**(*f: typing.Callable*)
Use this decorator to mark your ploogin function as a handler to call upon activation.

ploogz.ploogins.**upon_setup**(*f: typing.Callable*)
Use this decorator to mark your ploogin function as a handler to call upon setup.

ploogz.ploogins.**upon_teardown**(*f: typing.Callable*)
Use this decorator to mark you ploogin function as a handler to call upon teardown.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

CHAPTER 3

Quickstart

ploogz is a pretty simple plugin framework, so there isn't a whole lot to know to get started. Step one is to extend the *ploogz.ploogins.Ploogin* class and override the important methods to create an object that can do some useful work. (In the code blocks below, we don't include docstrings to focus on the code itself... *but you should always document your code!*)

```
from ploogz.ploogins import Ploogin, upon_setup, upon_activate, upon_teardown


class MyUsefulThing(Ploogin):

    def __init__(self):
        super().__init__(name='My Useful Thing')

    @upon_setup
    def get_ready(self):
        print("Getting ready!")

    @upon_activate
    def get_busy(self):
        print("Here we go!!")

    @upon_teardown
    def wrap_it_up(self):
        print("Good night.")
```

You can place this class in a python file (with a `.py` extension) in a directory along with other files containing other ploogins.

When it comes time to load these useful classes into an application, you'll need a *ploogz.ploogins.PlooginHost*.

```
from ploogz.ploogins import PlooginHost

host = PlooginHost(search_path=['/path/to/ploogins/directory', '/another/path/to/more/
↪ploogins'])
```

```
host.load()
for ploogin in host.ploogins:
    ploogin.setup()
for ploogin in host.ploogins:
    ploogin.activate()
# When you're all done, you can call teardown() on the host which will tear down all␣
↪the ploogins.
host.teardown()
```

In the simplest terms, that's all there is to it. This library is in the early stages of its development and there will likely be some additions, but these examples demonstrate the simple approach we seek.

Python Module Index

p

Index

Symbols

A

E

F

I

L

N

P

R

S

T

U