straight.command Documentation

Release 0.1a1

Calvin Spealman

September 26, 2016

Contents

`straight.command` is a framework for easily describing commands and their options, and allowing them to be extended with additional options and even sub-commands through easy-to-maintain plugins, via the `straight.plugin` plugin loader. Command-line options can be defined in a declarative syntax, which should be very familiar to many developers.

This is a very early stage in development.

CHAPTER 1

Example

```
#!/usr/bin/env python

from __future__ import print_function
import sys
from straight.command import Command, Option, SubCommand

class List(Command):
    def run_default(self, **extra):
        for line in open(self.parent.args['filename']):
            print(line.strip('\n'))

class Add(Command):
    new_todo = Option(dest='new_todo', action='append')

    def run_default(self, new_todo, **extra):
        with open(self.parent.args['filename'], 'a') as f:
            for one_todo in new_todo:
                print(one_todo.strip('\n'), file=f)

class Todo(Command):
    filename = Option(dest='filename', action='store')

    list = SubCommand('list', List)
    add = SubCommand('add', Add)

if __name__ == '__main__':
    Todo().run(sys.argv[1:])
```

This example shows several commands with declaratively defined options, including two of them being declared as subcommands of the third.

We can see a number of the features of `straight.command` in this example.

Command options are declared with instances of `Option` assigned in the `Command` subclass, much like the declarative nature of many ORM tools declaring table columns, so this should be familiar to many developers.

CHAPTER 2

The Command Processing Phases

The command structure and the processing of commands is defined carefully in a series of phases of parsing, preparing, and executing. Running the command passes through each of these phases to collect the parameters, process them, and execute the command.

- **Parse** The command line parameters are parsed into their components.
- **Prepare** The command is prepared according to the parameters provided.
- **Execute** The command is finally executed.

2.1 Parse

The command line arguments are broken into a list of strings, and passed into the process. The `Command` will parse these option strings by passing all of them to each `Option` it has, giving each option a chance to consume one or more of the parameters, if it can. Once all of the parameters have been consumed, or none of the options are able to consume any of the remaining parameters, the parsing is complete.

2.2 Prepare

Then an option is able to consume one or more of the parameter strings for the command, it *prepares* the command in some way. It may store a value provided by the option, set a flag, or even enable some specific task the command should perform, such as reporting help text about itself.

The options prepare by invoking an *action*.

2.3 Execute

After all the parameters are consumed by the options, the command itself will execute. This is done by calling the `execute()` method on the command, which is passed all the collected parameter values as keyword arguments.

CHAPTER 3

API Documentation

- modules
- command

CHAPTER 4

Indices and tables

- genindex
- modindex
- search