dkconfig Documentation

Release 0.2.2

Apr 28, 2023

Contents

CHAPTER 1

Installing from PyPI

```
pip install dkconfig
```

CHAPTER 2

Basic usage

Most of the methods of ConfigParser (https://docs.python.org/2/library/configparser.html#ConfigParser.RawConfigParser) should be usable in a relatively obvious way, however, `dkconfig` tries to give you some sane defaults to make your life easier, e.g. it will create files/headers/keys that don't exist:

```
/tst> ll
/tst> dkconfig foo.ini set header key value
/tst> cat foo.ini
[header]
key = value
```

Sections can be added:

```
/tst> dkconfig foo.ini add_section header2
/tst> cat foo.ini
[header]
key = value

[header2]
```

re-adding them is a no-op (and doesn't throw an exception):

```
/tst> dkconfig foo.ini add_section header2
/tst> cat foo.ini
[header]
key = value

[header2]
```

the values command pretty prints the keys and values:

```
/tst> dkconfig foo.ini values
key => value
```

the `dos` command will output the key/values as dos `set` commands:

```
/tst> dkconfig foo.ini dos
set "KEY=value"
```

from a batch file you would use it like this:

```
dkconfig foo.ini dos > tmp.bat && call tmp.bat && del tmp.bat
```

the `bash` command does the same for bash, and you'll use it together with eval:

```
eval $(dkconfig foo.ini bash)
```

You can read values directly into dos variables in the regular way:

```
> for /f "delims=" %a in ('dkconfig foo.ini get header key') do @set KEY=%a
> echo %KEY%
value
```

Bash has a more sane syntax for this:

```
bash$ export KEY=$(dkconfig foo.ini get header key)
bash$ echo $KEY
value
```

The appropriate error returns are set if a key is missing:

```
/tst> dkconfig foo.ini get header missing
/tst> echo %ERRORLEVEL%
1

/tst> dkconfig foo.ini get header key
value
/tst> echo %ERRORLEVEL%
0
```

Contents:

CHAPTER 3

dkconfig

3.1 dkconfig package

3.1.1 Module contents

3.1.2 Submodules

3.1.3 dkconfig.dkconfig module

Basic usage

Most of the methods of ConfigParser (https://docs.python.org/2/library/configparser.html#ConfigParser.RawConfigParser) should be usable in a relatively obvious way, however, `dkconfig` tries to give you some sane defaults to make your life easier, e.g. it will create files/headers/keys that don't exist:

```
/tst> ll
/tst> dkconfig foo.ini set header key value
/tst> cat foo.ini
[header]
key = value
```

Sections can be added:

```
/tst> dkconfig foo.ini add_section header2
/tst> cat foo.ini
[header]
key = value

[header2]
```

re-adding them is a no-op (and doesn't throw an exception):

```
/tst> dkconfig foo.ini add_section header2
/tst> cat foo.ini
[header]
key = value

[header2]
```

the values command pretty prints the keys and values:

```
/tst> dkconfig foo.ini values
key => value
```

the `dos` command will output the key/values as dos `set` commands:

```
/tst> dkconfig foo.ini dos
set "KEY=value"
```

from a batch file you would use it like this:

```
dkconfig foo.ini dos > tmp.bat && call tmp.bat && del tmp.bat
```

the `bash` command does the same for bash, and you'll use it together with eval:

```
eval $(dkconfig foo.ini bash)
```

You can read values directly into dos variables in the regular way:

```
> for /f "delims=" %a in ('dkconfig foo.ini get header key') do @set KEY=%a
> echo %KEY%
value
```

Bash has a more sane syntax for this:

```
bash$ export KEY=$(dkconfig foo.ini get header key)
bash$ echo $KEY
value
```

The appropriate error returns are set if a key is missing:

```
/tst> dkconfig foo.ini get header missing
/tst> echo %ERRORLEVEL%
1

/tst> dkconfig foo.ini get header key
value
/tst> echo %ERRORLEVEL%
0
```

class dkconfig.dkconfig.**Config**(*defaults=None*, *dict_type=<class 'collections.OrderedDict'>*, *allow_no_value=False*, ***, *delimiters=('=', ':')*, *comment_prefixes=('#', ';')*, *inline_comment_prefixes=None*, *strict=True*, *empty_lines_in_values=True*, *default_section='DEFAULT'*, *interpolation=<object object>*, *converters=<object object>*)

Bases: `configparser.RawConfigParser`

CLI Interface to configparser.

add_section(*section*)
Silently accept existing sections.

bash(**sections*)
Return values as bash export statements.

cat()
Output the contents to stdout.

commands
Return all methods defined as self as a list.

dos(**sections*)
Return values as dos set statements.

exit = 0

get(*section*, *option*)
Get an option from a section (returns None if it can't find it).

help(*cmdname=None*)
List all commands, or help foo to get help on foo.

read(*fname*, **args*, ***kw*)
Read and parse a filename or an iterable of filenames.

Files that cannot be opened are silently ignored; this is designed so that you can specify an iterable of potential configuration file locations (e.g. current directory, user's home directory, systemwide directory), and all existing configuration files in the iterable will be read. A single filename may also be given.

Return list of successfully read files.

set(*section*, *option*, *value=None*)
Set an option in a section (creates the section if it doesn't exist.

setlist(*section*, *key*, **lst*)
Set a list value.

values(**sections*)
Return all values in the config file.

dkconfig.dkconfig.**format_items**(*items*)
Print items formatted in two columns.

dkconfig.dkconfig.**format_list**(*lst*)
Print one item per line.

dkconfig.dkconfig.**format_result**(*val*)
Pretty print val.

dkconfig.dkconfig.**main**(*cmdline=None*)
main() is the entry point for the dkconfig command line tool.

dkconfig.dkconfig.**make_lock**(*fname*, *timeout=0*)
Return a LockFile for *fname*, in an appropriate location.

dkconfig.dkconfig.**parser**(*fname*)
Context manager that provides locking semantics.

dkconfig.dkconfig.**run**(*cmdline=None*)
run() is the most convenient entry point for usage as a Python library:

```
import dkconfig
txt = dkconfig.run('foo.ini values')
val = dkconfig.run('foo.ini get key')
```

run does not call *sys.exit*.

The commands can be meta-commands (commands about dkconfig, not an ini file).:

```
dkconfig meta-command [args*] --flags
```

e.g.:

```
dkconfig help
dkconfig help cmd

dkconfig [glob/filename] cmd args* [--flags]
```

i.e. dkconfig followed by a glob matchine one or more ini files, followed by a command and arguments to the command as specified in the docs (and any flags).

:: <other-prog> | dkconfig cmd args* [–flags]

this would let you quickly access common properties, e.g.:

```
$ find . -maxdepth 2 -name "*.ini" -print | dkconfig - get site dns
www.example.com
www.example2.com
...
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index