dk-tasklib Documentation

Release 0.3.0

December 21, 2018

Contents

CHAPTER 1

dktasklib package

Subpackages

dktasklib.entry_points package

Submodules

dktasklib.entry_points.dktasklibcmd module

Commands installed by setup.py

dktasklib.entry_points.dktasklibcmd.**add_django_to_docs_conf**()

dktasklib.entry_points.dktasklibcmd.**create_docs_cmd**(*args*)

dktasklib.entry_points.dktasklibcmd.**install_cmd**(*args*)
Install a basic task.py to the current directory.

dktasklib.entry_points.dktasklibcmd.**main**(*args=None*)

dktasklib.entry_points.taskbase module

Module contents

dktasklib.package package

Submodules

dktasklib.package.package_interface module

class dktasklib.package.package_interface.**Package**(*ctx=None*)
Bases: dkpkg.directory.Package

config()

get(*key*, *default=None*)

overridables = set(['source_less', 'tests', 'build_docs', 'django_templates', 'package_name', 'docs', 'build_coverage',

overrides(***res*)

dktasklib.package.package_interface.**pfind**(*path*, **fnames*)

Module contents

Submodules

dktasklib.clean module

dktasklib.commands module

class dktasklib.commands.**Command**(*name*, *argspec=''*, *requirements=()*, *policy={'list_join': ' ', 'negative_bool': 'omit'}*, ***optdefs*)

Bases: object

dktasklib.concat module

dktasklib.concat.**chomp**(*s*)

Remove line terminator if it exists.

dktasklib.concat.**concat**(*ctx*, *dest*, **sources*, ***kw*)

dktasklib.concat.**copy**(*ctx*, *source*, *dest*, *force=False*)

Copy source to dest, which can be a file or directory.

dktasklib.concat.**fix_line_endings**(*fname*, *eol='\n'*)

Change all line endings to eol.

dktasklib.concat.**line_endings**(*fname*)

Return all line endings in the file.

dktasklib.docs module

dktasklib.environment module

Global state.

class dktasklib.environment.**Environment**

Bases: object

dktasklib.executables module

class dktasklib.executables.**Executables**

Bases: object

Class for finding executables on the host system.

ctx

find(*name*, *requires=()*, *install_txt=''*)

Find the executable named name on the PATH.

Parameters

- **name** (*str*) – name of executable to find.
- **requires** (*List[str]*) – list of executables to find first.
- **install_txt** (*str*) – instructions for how to install the executable if it is not found.

find_babel()

find_babili()

find_browserify()

find_nodejs()
Find **node**.

find_npm()
Find the node package manager (**npm**).

find_twine()

find_uglify()

find_wheel()

require(**dependencies*)
Ensure that all dependencies are available. You should not need to call this yourself, use the *requires()* decorator instead.

exception dktasklib.executables.**MissingCommand**
Bases: exceptions.Exception

Exception thrown when a command (executable) is not found.

dktasklib.executables.**exe = <dktasklib.executables.Executables object>**
public interface to the *Executables* class

dktasklib.executables.**requires**(**deps*)
Decorator to declare global dependencies/requirements.

Usage (@task must be last):

```
@requires('nodejs', 'npm', 'lessc')
@task
def mytask(..)
```

dktasklib.help module

dktasklib.jstools module

dktasklib.lessc module

dktasklib.manage module

dktasklib.npm module

dktasklib.npm.**global_package**(*pkgname*)
Check if an npm package is installed globally.

dktasklib.npm.**npm**(*cmdline*)

dktasklib.pset module

class dktasklib.pset.**pset**(*items=()*, ***attrs*)
Bases: dict

Property Set class. A property set is an object where values are attached to attributes, but can still be iterated over as key/value pairs. The order of assignment is maintained during iteration. Only one value allowed per key.

```
>>> x = pset()
>>> x.a = 42
>>> x.b = 'foo'
>>> x.a = 314
>>> x
pset(a=314, b='foo')
```

__add__(*other*)
self + other

__eq__(*other*)
Equal iff they have the same set of keys, and the values for each key is equal. Key order is not considered for equality.

__radd__(*other*)
other + self

items()

keys()

remove(*key*)
Remove key from client vars.

values()

dktasklib.publish module

dktasklib.rule module

class dktasklib.rule.**BuildRule**(**args*, ***kwargs*)
Bases: object

after = []

needs_to_run()

requires = []

run(*ctx*)

topsort(*tasklist*)
Topological sort

dktasklib.runners module

class dktasklib.runners.**Result**
Bases: str

cmd = None

returncode = None

dktasklib.runners.**command**(*executable*, *dryrun=False*)

dktasklib.runners.**run**(*cmdline*, *throw=False*)

dktasklib.upversion module

Update a package's version number.

class dktasklib.upversion.**UpdateTemplateVersion**(**args*, ***kwargs*)
Bases: *dktasklib.rule.BuildRule*

dktasklib.upversion.**files_with_version_numbers**()

dktasklib.urlinliner module

dktasklib.urlinliner.**inline_data**(*data*, *type='image/png'*, *name=''*)
Inline (encode) the data.

dktasklib.urlinliner.**inline_file**(*fname*)
Inline from a file source named fname.

dktasklib.urlinliner.**inline_url**(*uri*)
Fetch uri and inline.

dktasklib.utils module

dktasklib.utils.**cd**(**args*, ***kwds*)
Context manager to change directory.

Usage:

```
with cd('foo/bar'):
    # current directory is now foo/bar
# current directory restored.
```

dktasklib.utils.**dest_is_newer_than_source**(*src*, *dst*)
Check if destination is newer than source.

Usage:

```
if not force and dest_is_newer_than_source(source, dest):
    print 'babel:', dest, 'is up-to-date.'
    return dest
```

dktasklib.utils.**env**(**args*, ***kwds*)
Context amanger to temporarily override environment variables.

dktasklib.utils.**filename**(*fname*)
Return only the file name (removes the path)

dktasklib.utils.**find_pymodule**(*dotted_name*)
Find the directory of a python module, without importing it.

dktasklib.utils.**fmt**(*s*, *ctx*)
Use the mapping *ctx* as a formatter for the {new.style} formatting string *s*.

dktasklib.utils.**message**(**args*, ***kwds*)

dktasklib.utils.**switch_extension**(*fname*, *ext=''*, *old_ext=None*)
Switch file extension on *fname* to *ext*. Returns the resulting file name.

Usage:

```
switch_extension('a/b/c/d.less', '.css')
```

dktasklib.version module

dktasklib.version.**add_version**(*ctx*, *source*, *outputdir=None*, *kind='pkg'*, *force=False*)
Copy source with version number to *outputdir*.

The version type is specified by the `kind` parameter and can be either "pkg" (package version), "svn" (current subversion revision number), or "hash" (the md5 hash of the file's contents).

Returns (str) output file name

dktasklib.version.**copy_to_version**(*ctx*, *source*, *outputdir=None*, *kind='pkg'*, *force=False*)
Copy source with version number to *outputdir*.

The version type is specified by the `kind` parameter and can be either "pkg" (package version), "svn" (current subversion revision number), or "hash" (the md5 hash of the file's contents).

Returns (str) output file name

dktasklib.version.**get_version**(*ctx*, *fname*, *kind='pkg'*)
Return the version number for fname.

dktasklib.version.**min_name**(*fname*, *min='.min'*)
Adds a *.min* extension before the last file extension.

dktasklib.version.**version_name**(*fname*)
Returns a template string containing *{version}* in the correct place.

dktasklib.version.**versioned_name**(*fname*)
Returns a template string containing *{version}* in the correct place.

dktasklib.watch module

Usage:

```
@task
def watch(ctx):
    watcher = Watcher(ctx)
    watcher.watch_file(
        name='{pkg.source}/less/{pkg.name}.less',
        action=lambda e: build(ctx, less=True)
```

```
    )
    watcher.watch_directory(
        path='{pkg.source}/js', ext='.jsx',
        action=lambda e: build(ctx, js=True)
    )
    watcher.watch_directory(
        path='{pkg.docs}', ext='.rst',
        action=lambda e: build(ctx, docs=True)
    )
    watcher.start()

ns = Collection(..., watch, ...)
ns.configure({
    'pkg': Package()
})
```

class dktasklib.watch.**DirectoryModified**(*ctx*, *path*, *ext*, *action*)
Bases: watchdog.events.FileSystemEventHandler

on_modified(*event*)

class dktasklib.watch.**FileModified**(*ctx*, *fname*, *action*)
Bases: watchdog.events.FileSystemEventHandler

on_modified(*event*)

class dktasklib.watch.**Watcher**(*ctx*)
Bases: object

start()

watch_directory(*path*, *ext*, *action*)

watch_file(*name*, *action*)

dktasklib.wintask module

dktasklib.wintask.**task**(**args*, ***kwargs*)
Marks wrapped callable object as a valid Invoke task.

May be called without any parentheses if no extra options need to be specified. Otherwise, the following keyword arguments are allowed in the parenthese'd form:

- name: Default name to use when binding to a *.Collection*. Useful for avoiding Python namespace issues (i.e. when the desired CLI level name can't or shouldn't be used as the Python level name.)
- aliases: Specify one or more aliases for this task, allowing it to be invoked as multiple different names. For example, a task named mytask with a simple @task wrapper may only be invoked as "mytask". Changing the decorator to be @task(aliases=['myothertask']) allows invocation as "mytask" *or* "myothertask".
- positional: Iterable overriding the parser's automatic "args with no default value are considered positional" behavior. If a list of arg names, no args besides those named in this iterable will be considered positional. (This means that an empty list will force all arguments to be given as explicit flags.)
- optional: Iterable of argument names, declaring those args to have optional values. Such arguments may be given as value-taking options (e.g. --my-arg=myvalue, wherein the task is given "myvalue") or as Boolean flags (--my-arg, resulting in True).
- iterable: Iterable of argument names, declaring them to build iterable values.

- `incrementable`: Iterable of argument names, declaring them to increment their values.
- `default`: Boolean option specifying whether this task should be its collection's default task (i.e. called if the collection's own name is given.)
- `auto_shortflags`: Whether or not to automatically create short flags from task options; defaults to True.
- `help`: Dict mapping argument names to their help strings. Will be displayed in `--help` output.
- `pre`, `post`: Lists of task objects to execute prior to, or after, the wrapped task whenever it is executed.
- `autoprint`: Boolean determining whether to automatically print this task's return value to standard output when invoked directly via the CLI. Defaults to False.
- `klass`: Class to instantiate/return. Defaults to *.Task*.

If any non-keyword arguments are given, they are taken as the value of the `pre` kwarg for convenience's sake. (It is an error to give both `*args` and `pre` at the same time.)

New in version 1.0.

Changed in version 1.1: Added the `klass` keyword argument.

Module contents

CHAPTER 2

Developing dktasklib

Uploading to PyPI

- only source distribution:

```
python setup.py sdist upload
```

- source and windows installer:

```
python setup.py sdist bdist_wininst upload
```

- source, windows, and wheel installer:

```
python setup.py sdist bdist_wininst bdist_wheel upload
```

- create a documentation bundle to upload to PyPi:

```
python setup.py build_sphinx
python setup.py upload_docs
```

Note: if you're using this as a template for new projects, remember to *python setup.py register <projectname>* before you upload to PyPi.

Running tests

One of:

```
python setup.py test
py.test dktasklib
```

with coverage:

```
py.test --cov=dktasklib .
```

Building documentation

```
python setup.py build_sphinx
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

d

Symbols

A

B

C

D

E

F

G

I

K

L

M

N

O

P

R

S

T

U

V

W