pyke

Release 0.0.4

frissyn

May 09, 2022

CONTENTS:

CHAPTER

ONE

API USAGE

pyke.**task**(**args*, ***options: Any*) → Callable

Function decorator to create a Pykefile task with a name and certain options.

Parameters **name** (*Text*, optional) – Name of the task to be called from the `pyke` command. If no name is given, the function name is used. If the task uses a regex `pattern` option, that pattern will used and matched against args passed to `pyke`.

Options

- ***default*** **(bool) –** Setting to `True` will set the task as default. This task will run when `pyke` doesn't recieve any arguments.
- ***deps*** **(Iterable[Text]) –** List of task names to run before the current task. Tasks that use patterns cannot be used as task dependencies.
- ***pattern*** **(Text, *valid regex*) –** Regex pattern to match against args. All found matches will be passed to the task function as string arguments.

pyke.**env**(*key: str*, *default: Optional[Any] = None*) → Any

Wrapper function for `os.environ.get` to access enviornment variables in your tasks.

Parameters

- **key** (`Text`) – Dictionary key to get.
- **default** (`Any`, optional) – Value to return if key isn't found. Defaults to `None`.

Return type `Any`

pyke.**export**(***kv: Any*) → Union[Any, NoReturn]

Sets given mapping of keyword arguments as enviornment variables that can be accessed from `pyke.env` or `os.environ`. Uses OS independent method to set env vars in running process.

Return type `Union[Any, NoReturn]`

pyke.**run**() → NoReturn

Starts the Pykefile runner and parses `sys.argv` to run specified tasks. Should only be run *after* all tasks have been specified.

Return type `NoReturn`

pyke.**shell**(*cmd: str*) → subprocess.CompletedProcess

Wrapper function for `subprocess.run`. Runs given command in the shell and returns the completed process.

Parameters **cmd** (`Text`) – Command to run in the shell.

Return type `subprocess.CompletedProcess`

(WIP, Beta Release) Makelike build automation tool for Python projects with extensive DSL features. Throwout your Makefiles and use Pykefiles!

CHAPTER

TWO

FEATURES

1. Users can specify tasks, subtasks, and task rules.
2. Use regex rules patterns to create targets for tasks.
3. Significantly less confusing than Makefiles (is that a tab or space…?)
4. Complete Python DSL with full access to builtins and external dependencies.
5. **[WIP]** Run and execute tasks in parallel with each other (threaded multitasking).

CHAPTER

THREE

EXAMPLE

```
import pyke

# create a defualt task, named "build"
@pyke.task("build", default=True)
def build():
    print("Building the project...")

# create a task dependency. running `pyke dist`
# will make the "build" task run first!
@pyke.task("dist", deps=["build"])
def dist():
    print("Distributing the project...")

pyke.run()
```

Put that in a `Pykefile` and you're good to go. Then run `pyke` in the same directory and watch the magic happen!

```
$ pyke dist
Building the project...
Distributing the project...
```

CHAPTER

FOUR

INSTALLATION

Install the `pykefile` library with `pip`. Requires Python 3.8 or higher.

```
python -m pip install pykefile
```

You can also add it your developement dependencies with `poetry`.

```
python -m poetry add pykefile --dev
```

CHAPTER

FIVE

USAGE

Just like any other Makefile, you'll need the `Pykefile` in your current directory. (`Pykefile.py` also works) Then use the `pyke` command to execute and run specified tasks. Running `pyke` without any commands will call the default task if there is one. The first argument will call a task by that name.

INDEX