RiotKit Do Documentation

Release 1

Wolnosciowiec Team

May 25, 2020

Contents:

RKD is delivered as a Python Package. To extend RKD with additional tasks you need to install them via PIP or (simpler) define without own code in makefile.py/makefile.yaml

CHAPTER 1

Bootstrap RKD in your project

The preferred way to setup RKD is virtualenv.

With requirements.txt and virtualenv you can fully control versioning of RKD and it's components provided by open-source community. You decide when you are ready to upgrade.

```
virtualenv .venv
source .venv/bin/activate

echo "rkd<=0.5" > requirements.txt # better choose a stable tag and use fixed version␣
↪for stability

# example of installing component from external package/repository
# later you need to activate it in makefile.yaml or in makefile.py - check later␣
↪other chapter "Importing tasks"
# echo "rkt_ciutils<=3.0"

pip install -r requirements.txt
rkd :rkd:create-structure
```

CHAPTER 2

Conception

Makefile, Gradle and other build systems are strictly for development, so RiotKit decided to create RKD **with DevOps in mind**.

Everything could be done like in Makefile via YAML syntax, or in Python, because DevOps love Python! :)

In effect a simple task executor with clear rules and early validation of input parameters was created. Each task specified to be run is treated like a separate application - has it's own parameters, by default inherits global settings but those could be overridden. **The RKD version and version of any installed tasks are managed by Python Packaging - DevOps needs to have strict control over dependencies and upgrades.**

Basic examples:

```
rkd :tasks

# runs two tasks ":sh" with different arguments
rkd :sh -c 'echo hello' :sh -c 'ps aux'

# runs different tasks in order
rkd :py:clean :py:build :py:publish --user=__token__ --password=123456

# allows to fail one of tasks in our pipeline (does not interrupt the pipeline when␣
↪first task fails)
rkd :sh -c 'exit 1' --keep-going :sh -c 'echo hello'

# silent output, only tasks stdout and stderr is visible (for parsing outputs in␣
↪scripts)
rkd --silent :sh -c "ps aux"
```

```
(.venv)  riotkit > rkd :tasks
 >> Executing :tasks
[global]
:sh             # Executes shell commands
:init           # :init task is executing ALWAYS. That's a technical, core task.
:tasks          # Lists all enabled tasks
:release

[py]
:py:publish             # Publishes Python packages to PIP
:py:build               # Builds a Python package in a format to be packaged for publishing
:py:clean               # Clean up the built Python modules
:py:install             # Install a Python package using setuptools


The task ":tasks" succeed.
----------------------------------

Successfully executed 2 tasks.
```

```
(.venv)  riotkit > rkd :py:clean :py:build
 >> Executing :py:clean
+ rm -rf pbr.egg.info .eggs dist build

The task ":py:clean" succeed.
----------------------------------

 >> Executing :py:build
running sdist
[pbr] Writing ChangeLog
[pbr] Generating ChangeLog
[pbr] ChangeLog complete (0.0s)
[pbr] Generating AUTHORS
[pbr] AUTHORS complete (0.0s)
running egg_info
writing src/rkd.egg-info/PKG-INFO
writing dependency_links to src/rkd.egg-info/dependency_links.txt
writing entry points to src/rkd.egg-info/entry_points.txt
writing requirements to src/rkd.egg-info/requires.txt
writing top-level names to src/rkd.egg-info/top_level.txt
writing pbr to src/rkd.egg-info/pbr.json
[pbr] Processing SOURCES.txt
[pbr] In git context, generating filelist from git
warning: no previously-included files found matching '.gitignore'
warning: no previously-included files found matching '.gitreview'
warning: no previously-included files matching '*.pyc' found anywhere in distribution
writing manifest file 'src/rkd.egg-info/SOURCES.txt'
[pbr] reno was not found or is too old. Skipping release notes
```

2.1 Basics

Tasks are prefixed always with ":". Each task can handle it's own arguments.

2.1.1 Tasks arguments usage

makefile.py

```
from rkd.syntax import TaskDeclaration, TaskAliasDeclaration
from rkd.standardlib.python import PublishTask

IMPORTS = [
    TaskDeclaration(PublishTask())
]

TASKS = [
    TaskAliasDeclaration(':my:test', [':py:publish', '--username=...', '--password=...
↪'])
]
```

Example of calling same task twice, but with different input

Notes for this example: The "username" parameter is a default defined in `makefile.py` in this case.

```
$ rkd :my:test --password=first :my:test --password=second
 >> Executing :py:publish
Publishing
{'username': '...', 'password': 'first'}

 >> Executing :py:publish
Publishing
{'username': '...', 'password': 'second'}
```

Example of calling same task twice, with no extra arguments

In this example the argument values "..." are taken from `makefile.py`

```
$ rkd :my:test :my:test
 >> Executing :py:publish
Publishing
{'username': '...', 'password': '...'}

 >> Executing :py:publish
Publishing
{'username': '...', 'password': '...'}
```

Example of –help per command:

```
$ rkd :my:test :my:test --help
usage: :py:publish [-h] [--username USERNAME] [--password PASSWORD]

optional arguments:
  -h, --help           show this help message and exit
  --username USERNAME  Username
  --password PASSWORD  Password
```

2.1.2 Simplified - YAML syntax

YAML syntax has an advantage of simplicity and clean syntax, custom bash tasks can be defined there easier than in Python. To use YAML you need to define **makefile.yaml** file in .rkd directory.

NOTICE: makefile.py and makefile.yaml can exist together. Python version will be loaded first, the YAML version will append changes in priority.

```
version: org.riotkit.rkd/0.3
imports:
  - rkd.standardlib.docker.TagImageTask

tasks:
  # see this task in "rkd :tasks"
  # run with "rkd :examples:bash-test"
  :examples:bash-test:
      description: Execute an example command in bash - show only python related tasks
      steps: |
             echo "RKD_DEPTH: ${RKD_DEPTH} # >= 2 means we are running rkd-in-rkd"
             echo "RKD_PATH: ${RKD_PATH}"
             rkd --silent :tasks | grep ":py"

  # try "rkd :examples:arguments-test --text=Hello --test-boolean"
  :examples:arguments-test:
      description: Show example usage of arguments in Bash
      arguments:
          "--text":
              help: "Adds text message"
              required: True
          "--test-boolean":
              help: "Example of a boolean flag"
              action: store_true # or store_false
      steps:
        - |
          #!bash
          echo " ==> In Bash"
          echo " Text: ${ARG_TEXT}"
          echo " Boolean test: ${ARG_TEST_BOOLEAN}"
        - |
          #!python
          print(' ==> In Python')
          print(' Text: %s ' % ctx.args['text'])
          print(' Text: %s ' % str(ctx.args['test_boolean']))
          return True

  # run with "rkd :examples:list-standardlib-modules"
  :examples:list-standardlib-modules:
      description: List all modules in the standardlib
      steps:
        - |
          #!python
          ctx: ExecutionContext
          this: TaskInterface

          import os

          print('Hello world')
          print(os)
          print(ctx)
          print(this)

          return True
```

2.1.3 What's loaded first? See Paths and inheritance

2.2 Tasks

2.2.1 Shell

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib.shell	rkd.standardlib.shell.ShellCommandTask	pip install rkd==SELECT VERSION	

2.2.1.1 :sh

Executes a Bash script. Can be multi-line.

Example of plain usage:

```
rkd :sh -c "ps aux"
rkd :sh --background -c "some-heavy-task"
```

Example of task alias usage:

```
from rkd.syntax import TaskAliasDeclaration as Task

#
# Example of Makefile-like syntax
#

IMPORTS = []

TASKS = [
    Task(':find-images', [
        ':sh', '-c', 'find ../../ -name \'*.png\''
    ]),

    Task(':build', [':sh', '-c', ''' set -x;
        cd ../../../

        chmod +x setup.py
        ./setup.py build

        ls -la
    '''])
]
```

2.2.1.2 :exec

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib.shell	rkd.standardlib.shell.ExecCommand	pip install rkd==SELECT VERSION	

Works identically as **:sh**, but for spawns a single process. Does not allow a multi-line script syntax.

2.2.1.3 Class to import: BaseShellCommandWithArgumentParsingTask

Creates a command that executes bash script and provides argument parsing using Python's argparse. Parsed arguments are registered as ARG_{{argument_name}} eg. –activity-type would be exported as ARG_ACTIVITY_TYPE.

```
IMPORTS += [
    BaseShellCommandWithArgumentParsingTask(
        name=":protest",
        group=":activism",
        description="Take action!",
        arguments_definition=lambda argparse: (
            argparse.add_argument('--activity-type', '-t', help='Select an activity␣
↪type')
        ),
        command='''
            echo "Let's act! Let's ${ARG_ACTIVITY_TYPE}!"
        '''
    )
]
```

2.2.2 Technical/Core

2.2.2.1 :init

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib	rkd.standardlib.InitTask	pip install rkd== SELECT VERSION	

This task runs ALWAYS. :init implements a possibility to inherit global settings to other tasks

2.2.2.2 :tasks

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib	rkd.standardlib.TasksListTask	pip install rkd== SELECT VERSION	

Lists all tasks that are loaded by all chained makefile.py configurations.

2.2.2.3 :version

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib	rkd.standardlib.VersionTask	pip install rkd== SELECT VERSION	

Shows version of RKD and lists versions of all loaded tasks, even those that are provided not by RiotKit. The version strings are taken from Python modules as RKD strongly rely on Python Packaging.

2.2.2.4 CallableTask

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib	rkd.standardlib.CallableTask	pip install rkd== SELECT VERSION	

This is actually not a task to use directly, it is a template of a task to implement yourself. It's kind of a shortcut to create a task by defining a simple method as a callback.

```
import os
from rkd.syntax import TaskDeclaration
from rkd.standardlib import CallableTask
from rkd.contract import ExecutionContext


def union_method(context: ExecutionContext) -> bool:
    os.system('xdg-open https://iwa-ait.org')
    return True


IMPORTS = [
    TaskDeclaration(CallableTask(':create-union', union_method))
]

TASKS = []
```

2.2.2.5 :rkd:create-structure

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib	rkd.standardlib.CreateStructureTask	pip install rkd== SELECT VERSION	

Creates a template structure used by RKD in current directory.

2.2.3 Docker

2.2.3.1 :docker:tag

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib.docker	rkd.standardlib.docker.TagImageTask	pip install rkd== SELECT VERSION	

Performs a docker-style tagging of an image that is being released - example: 1.0.1 -> 1.0 -> 1 -> latest

Example of usage:

```
rkd :docker:tag --image=quay.io/riotkit/filerepository:3.0.0-RC1 --propagate -rf debug
```

2.2.3.2 :docker:push

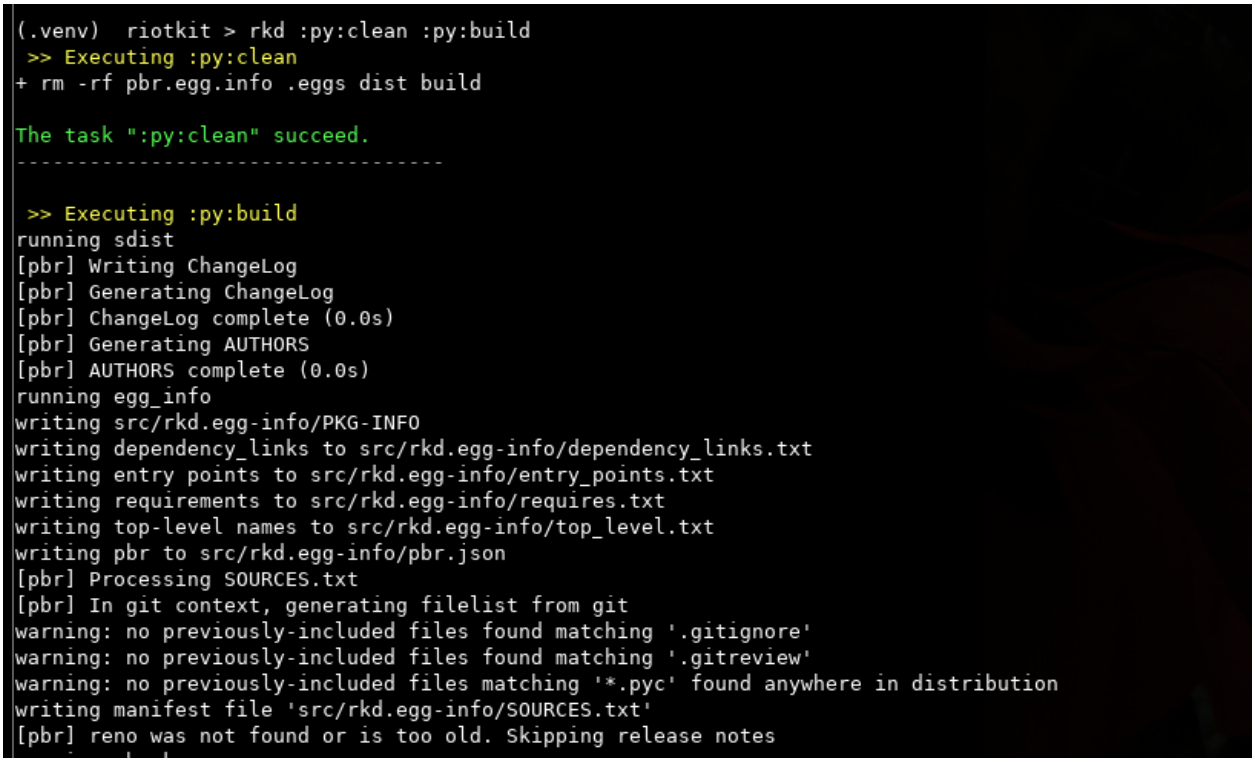
Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib.docker	rkd.standardlib.docker.PushTask	pip install rkd== SELECT VERSION	

Does same thing and taking same arguments as **:docker:tag**, one difference - pushing already created tasks.

2.2.4 Python

This package was extracted from standardlib to rkd_python, but is maintained together with RKD as part of RKD core.

Set of Python-related tasks for building, testing and publishing Python packages.



2.2.4.1 :py:publish

Package to import	Single task to import	PIP package to install	Stable version
rkd_python	rkd_python.PublishTask	pip install rkd_python== SELECT VERSION	

Publish a package to the PyPI.

Example of usage:

```
rkd :py:publish --username=__token__ --password=.... --skip-existing --test
```

2.2.4.2 :py:build

Package to import	Single task to import	PIP package to install	Stable version
rkd_python	rkd_python.BuildTask	pip install rkd_python== SELECT VERSION	

Runs a build through setuptools.

2.2.4.3 :py:install

Package to import	Single task to import	PIP package to install	Stable version
rkd_python	rkd_python.InstallTask	pip install rkd_python== SELECT VERSION	

Installs the project as Python package using setuptools. Calls ./setup.py install.

2.2.4.4 :py:clean

Package to import	Single task to import	PIP package to install	Stable version
rkd_python	rkd_python.CleanTask	pip install rkd_python== SELECT VERSION	

Removes all files related to building the application.

2.2.4.5 :py:unittest

Package to import	Single task to import	PIP package to install	Stable version
rkd_python	rkd_python.UnitTestTask	pip install rkd_python== SELECT VERSION	

Runs Python's built'in unittest module to execute unit tests.

Examples:

```
rkd :py:unittest
rkd :py:unittest -p some_test
rkd :py:unittest --tests-dir=../test
```

2.2.5 JINJA

Renders JINJA2 files, and whole directories of files. Allows to render by pattern.

2.2.5.1 :j2:render

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib.jinja	rkd.standardlib.jinja.FileRenderTask	pip install rkd==SELECT VERSION	

Renders a single file from JINJA2.

Example of usage:

```
rkd :j2:render -s SOURCE-FILE.yaml.j2 -o OUTPUT-FILE.yaml
```

2.2.5.2 :j2:directory-to-directory

Package to import	Single task to import	PIP package to install	Stable version
rkd.standardlib.jinja	rkd.standardlib.jinja.DirectoryRenderTask	pip install rkd==SELECT VERSION	

Renders all files recursively in given directory to other directory. Can remove source files after rendering them to the output files.

Pattern is a regexp pattern that matches whole path, not only file name

Example usage:

```
rkd :j2:directory-to-directory \
    --source="/some/path/templates" \
    --target="/some/path/rendered" \
    --delete-source-files \
    --pattern="(.*).j2"
```

2.3 Usage

2.3.1 Importing tasks

Tasks can be defined as installable Python's packages that you can import in your Makefile

Please note:

- To import a group, the package you try to import need to hvve a defined **imports()** method inside of the package.
- The imported group does not need to import automatically dependend tasks (but it can, it is recommended), you need to read into the docs of specific package if it does so

2.3.1.1 1) Install a package

RKD defines dependencies using Python standards.

Example: Given we want to import tasks from package "rkt_armutils".

```
echo "rkt_armutils==3.0" >> requirements.txt
pip install -r requirements.txt
```

Good practices:

- Use fixed versions eg. 3.0 or even 3.0.0 and upgrade only intentionally to reduce your work on fixing bugs

2.3.1.2 2) In YAML syntax

Example: Given we want to import task "InjectQEMUBinaryIntoContainerTask", or we want to import whole "rkt_armutils.docker" group

```
imports:
    # Import whole package, if the package defines a group import (method imports())
    - rkt_armutils.docker

    # Or import single task
    - rkt_armutils.docker.InjectQEMUBinaryIntoContainerTask
```

2.3.1.3 2) In Python syntax

Example: Given we want to import task "InjectQEMUBinaryIntoContainerTask", or we want to import whole "rkt_armutils.docker" group

```
from rkd.syntax import TaskDeclaration
from rkt_armutils.docker import InjectQEMUBinaryIntoContainerTask

# ... (use "+" operator to append, remove "+" if you didn't define any import yet)
IMPORTS += [TaskDeclaration(InjectQEMUBinaryIntoContainerTask)]
```

2.3.2 Troubleshooting

1. Output is corrupted or there is no output from a shell command executed inside of a task

The output capturing is under testing. The Python's subprocess module is skipping "sys.stdout" and "sys.stderr" by writing directly to /dev/stdout and /dev/stderr, which makes output capturing difficult.

Run rkd in compat mode to turn off output capturing from shell commands:

```
RKD_COMPAT_SUBPROCESS=true rkd :some-task-here
```

2.3.3 Loading priority

2.3.3.1 Environment variables loading order in YAML syntax

Legend: Top - is most important

1. Operating system environment
2. Per-task "environment" section
3. Per-task "env_file" imports
4. Global "environment" section
5. Global "env_file" imports

2.3.3.2 Order of loading of makefile files in same .rkd directory

Legend: Lower has higher priority (next is appending changes to previous)

1. *.py
2. *.yaml
3. *.yml

2.3.3.3 Paths and inheritance

RKD by default search for .rkd directory in current execution directory - *./.rkd*.

The search order is following (from lower to higher load priority):

1. RKD's internals (we provide a standard tasks like *:tasks*, *:init*, *:sh*, *:exec* and more)
2. */usr/lib/rkd*
3. User's home *~/.rkd*
4. Current directory *./.rkd*
5. *RKD_PATH*

Custom path defined via environment variable

RKD_PATH allows to define multiple paths that would be considered in priority.

```
export RKD_PATH="/some/path:/some/other/path:/home/user/riotkit/.rkd-second"
```

How the makefiles are loaded?

Each makefile is loaded in order, next makefile can override tasks of previous. That's why we at first load internals, then your tasks.

2.3.3.4 Tasks execution

Tasks are executed one-by-one as they are specified in commandline or in TaskAlias declaration (commandline arguments).

```
rkd :task-1 :task-2 :task-3
```

1. task-1

2. task-2
3. task-3

A –keep-going can be specified after given task eg. :task-2 –keep-going, to ignore a single task failure and in consequence allow to go to the next task regardless of result.

2.3.4 Tasks development

RKD has two approaches to define a task. The first one is simpler - in makefile in YAML or in Python. The second one is a set of tasks as a Python package.

2.3.4.1 Creating simple tasks in YAML syntax

Example 1:

```
version: org.riotkit.rkd/0.3
imports:
  - rkd.standardlib.docker.TagImageTask

tasks:
  # see this task in "rkd :tasks"
  # run with "rkd :examples:bash-test"
  :examples:bash-test:
      description: Execute an example command in bash - show only python related tasks
      steps: |
             echo "RKD_DEPTH: ${RKD_DEPTH} # >= 2 means we are running rkd-in-rkd"
             echo "RKD_PATH: ${RKD_PATH}"
             rkd --silent :tasks | grep ":py"

  # try "rkd :examples:arguments-test --text=Hello --test-boolean"
  :examples:arguments-test:
      description: Show example usage of arguments in Bash
      arguments:
          "--text":
              help: "Adds text message"
              required: True
          "--test-boolean":
              help: "Example of a boolean flag"
              action: store_true # or store_false
      steps:
        - |
          #!bash
          echo " ==> In Bash"
          echo " Text: ${ARG_TEXT}"
          echo " Boolean test: ${ARG_TEST_BOOLEAN}"
        - |
          #!python
          print(' ==> In Python')
          print(' Text: %s ' % ctx.args['text'])
          print(' Text: %s ' % str(ctx.args['test_boolean']))
          return True

  # run with "rkd :examples:list-standardlib-modules"
  :examples:list-standardlib-modules:
      description: List all modules in the standardlib
```

(continues on next page)

(continued from previous page)

```
    steps:
      - |
        #!python
        ctx: ExecutionContext
        this: TaskInterface

        import os

        print('Hello world')
        print(os)
        print(ctx)
        print(this)

        return True
```

Example 2:

```
version: org.riotkit.rkd/0.3

environment:
    GLOBALLY_DEFINED: "16 May 1966, seamen across the UK walked out on a nationwide␣
↪strike for the first time in half a century. Holding solid for seven weeks, they␣
↪won a reduction in working hours from 56 to 48 per week "

env_files:
    - env/global.env

tasks:
    :hello:
        environment:
            INLINE_PER_TASK: "17 May 1972 10,000 schoolchildren in the UK walked out␣
↪on strike in protest against corporal punishment. Within two years, London state␣
↪schools banned corporal punishment. The rest of the country followed in 1987."
        env_files: ['env/per-task.env']
        steps: |
            echo " >> ENVIRONMENT VARIABLES DEMO"
            echo "Inline defined in this task: ${INLINE_PER_TASK}\n\n"
            echo "Inline defined globally: ${GLOBALLY_DEFINED}\n\n"
            echo "Included globally - global.env: ${TEXT_FROM_GLOBAL_ENV}\n\n"
            echo "Included in task - per-task.env: ${TEXT_PER_TASK_FROM_FILE}\n\n"
```

Explanation of examples:

1. “arguments” is an optional dict of arguments, key is the argument name, subkeys are passed directly to argparse
2. “steps” is a mandatory list or text with step definition in Bash or Python language
3. “description” is an optional text field that puts a description visible in “:tasks” task
4. “environment” is a dict of environment variables that can be defined
5. “env_files” is a list of paths to .env files that should be included
6. “imports” imports a Python package that contains tasks to be used in the makefile and in shell usage

2.3.4.2 Developing a Python package

Each task should implement methods of **rkd.contract.TaskInterface** interface, that’s the basic rule.

Following example task could be imported with path **rkd.standardlib.ShellCommandTask**, in your own task you would have a different package name instead of **rkd.standardlib**.

Example task from RKD standardlib:

```
class ShellCommandTask(TaskInterface):
    """Executes shell scripts"""

    def get_name(self) -> str:
        return ':sh'

    def get_group_name(self) -> str:
        return ''

    def configure_argparse(self, parser: ArgumentParser):
        parser.add_argument('--cmd', '-c', help='Shell command', required=True)

    def execute(self, context: ExecutionContext) -> bool:
        # self.sh() and self.io() are part of TaskUtilities via TaskInterface

        try:
            self.sh(context.args['cmd'], capture=False)
        except CalledProcessError as e:
            self.io().error_msg(str(e))
            return False

        return True
```

Explanation of example:

1. The docstring in Python class is what will be shown in **:tasks as description**. You can also define your description by implementing **def get_description() -> str**
2. Name and group name defines a full name eg. :your-project:build
3. **def configure_argparse()** allows to inject arguments, and –help description for a task - it's a standard Python's argparse object to use
4. **def execute()** provides a context of execution, please read *Tasks API* chapter about it. In short words you can get commandline arguments, environment variables there.
5. **self.io()** is providing input-output interaction, please use it instead of print, please read *Tasks API* chapter about it.

2.3.4.3 Please check Tasks API for interfaces description

2.3.5 Tasks API

2.3.5.1 Each task must implement a TaskInterface

class rkd.contract.**TaskInterface**

configure_argparse(*parser: argparse.ArgumentParser*)
Allows a task to configure ArgumentParser (argparse)

copy_internal_dependencies(*task*)
Allows to execute a task-in-task, by copying dependent services from one task to other task :api 0.2

execute(*context: rkd.contract.ExecutionContext*) → bool
Executes a task. True/False should be returned as return

get_declared_envs() → Dict[str, str]
Dictionary of allowed envs to override: KEY -> DEFAULT VALUE

get_full_name()
Returns task full name, including group name

get_group_name() → str
Group name where the task belongs eg. ":publishing", can be empty. :api 0.2

get_name() → str
Task name eg. ":sh" :api 0.2

is_silent_in_observer() → bool
Internally used property

2.3.5.2 Execution context provides parsed shell arguments and environment variables

class rkd.contract.**ExecutionContext**(*declaration: rkd.contract.TaskDeclarationInterface*, *parent: Optional[rkd.contract.GroupDeclarationInterface] = None*, *args: Dict[str, str] = {}*, *env: Dict[str, str] = {}*)
Defines which objects could be accessed by Task. It's a scope of a single task execution.

get_arg_or_env(*name: str*) → Optional[str]
Provides value of user input

Usage: get_arg_or_env('–file-path') resolves into FILE_PATH env variable, and –file-path switch (file_path in argparse)

Behavior: When user provided explicitly switch eg. –history-id, then it's value will be taken in priority. If switch –history-id was not used, but user provided HISTORY_ID environment variable, then it will be considered.

If no switch provided and no environment variable provided, but a switch has default value - it would be returned. If no switch provided and no environment variable provided, the switch does not have default, but environment variable has a default value defined, it would be returned.

Raises MissingInputException – When no switch and no environment variable was provided, then an exception is thrown.

get_env(*name: str*, *error_on_not_used: bool = False*)
Get environment variable value

2.3.5.3 Interaction with input and output

class rkd.inputoutput.**IO**
Interacting with input and output - stdout/stderr/stdin, logging

capture_descriptors(*target_file: str = None*, *stream=None*, *enable_standard_out: bool = True*)
Capture stdout and stderr from a block of code - use with 'with'

critical(*text*)
Logger: critical

debug(*text*)
Logger: debug

err (*text*)
Standard error

errln (*text*)
Standard error + newline

error (*text*)
Logger: error

error_msg (*text*)
Error message (optional output)

h1 (*text*)
Heading #1 (optional output)

h2 (*text*)
Heading #2 (optional output)

h3 (*text*)
Heading #3 (optional output)

h4 (*text*)
Heading #3 (optional output)

info (*text*)
Logger: info

info_msg (*text*)
Informational message (optional output)

is_silent () → bool
Is output silent? In silent mode OPTIONAL MESSAGES are not shown

opt_out (*text*)
Optional output - fancy output skipped in –silent mode

opt_outln (*text*)
Optional output - fancy output skipped in –silent mode + newline

out (*text*)
Standard output

outln (*text*)
Standard output + newline

print_group (*text*)
Prints a colored text inside brackets [text] (optional output)

print_line ()
Prints a newline

print_opt_line ()
Prints a newline (optional output)

print_separator ()
Prints a text separator (optional output)

success_msg (*text*)
Success message (optional output)

warn (*text*)
Logger: warn

Index