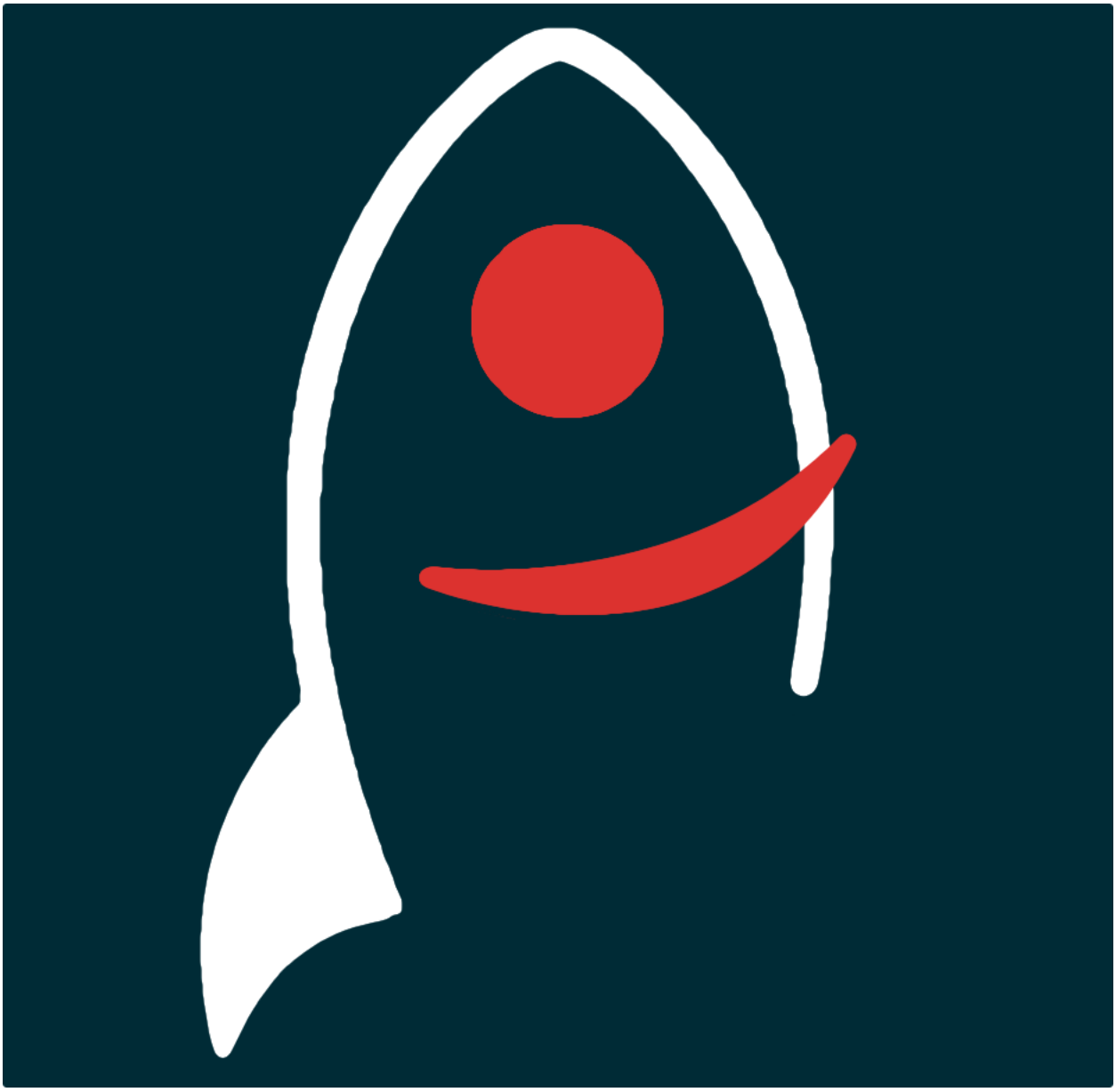

tastic Documentation

Release 1.8.2

Dave Young

2017

A python package for working with taskpaper documents.

Here's a summary of what's included in the python package:

Classes

`tastic.reminders`	*the taskpaper reminders object*
`tastic.workspace.sync`	*The worker class for the sync module*
`tastic.workspace.workspace`	*tools for sorting, archiving and indexing tasks and maintaining the contents of all taskpaper*
`tastic.tastic.baseClass`	*This is the base class for all taskpaper objects: documents, projects and tasks*
`tastic.tastic.document`	*This is the taskpaper document object - top level object*
`tastic.tastic.note`	*The taskpaper note object*
`tastic.tastic.project`	*The taskpaper project object*
`tastic.tastic.task`	*The taskpaper task object*

Functions

CHAPTER 1

Installation

The easiest way to install tastic is to use `pip`:

```
pip install tastic
```

Or you can clone the github repo and install from a local version of the code:

```
git clone git@github.com:thespacedoctor/tastic.git
cd tastic
python setup.py install
```

To upgrade to the latest version of tastic use the command:

```
pip install tastic --upgrade
```

1.1 Development

If you want to tinker with the code, then install in development mode. This means you can modify the code from your cloned repo:

```
git clone git@github.com:thespacedoctor/tastic.git
cd tastic
python setup.py develop
```

Pull requests are welcomed!

1.1.1 Sublime Snippets

If you use Sublime Text as your code editor, and you're planning to develop your own python code with tastic, you might find my Sublime Snippets useful.

1.2 Issues

Please report any issues here.

CHAPTER 2

Command-Line Usage

```
Documentation for tastic can be found here: http://tastic-for-taskpaper.readthedocs.io/en/stable/


Usage:
    tastic init
    tastic sort <pathToFileOrWorkspace> [-s <pathToSettingsFile>]
    tastic archive <pathToFileOrWorkspace> [-s <pathToSettingsFile>]
    tastic [-f] sync <pathToWorkspace> <workspaceName> <pathToSyncFolder> [<editorialRootPath>] [-s <
    tastic reminders import <listName> <pathToTaskpaperDoc>

Options:
    init                    setup the tastic settings file for the first time
    sort                    sort a taskpaper file or directory containing taskpaper files via workfl
    archive                 move done tasks in the 'Archive' projects within taskpaper documents int
    reminders               commands to work with macOS reminders
    import                  import tasks into a given taskpaper document


    pathToFileOrWorkspace   give a path to an individual taskpaper file or the root of a workspace c
    pathToTaskpaperDoc      a path to a taskpaper document
    pathToWorkspace         root path of a workspace containing taskpaper files
    workspaceName           the name you give to the workspace
    pathToSyncFolder        path to the folder you wish to sync the index task files into
    listName                name of a reminders.app list (macOS only)
    editorialRootPath       the root path of editorial's dropbox sync folder (add to generate an edi
    -h, --help              show this help message
    -v, --version           show version
    -s, --settings          the settings file
    -f, --fileTags          if the tag to sync is in the filepath (e.g. /@due/mytasks.taskpaper) inc
```

CHAPTER 3

Documentation

Documentation for tastic is hosted by Read the Docs (last stable version and latest version).

CHAPTER 4

Command-Line Tools Tutorial

As well as providing python objects and methods for working with your taskpaper documents, tastic also provides some very useful command-line tools. These tools work not only with single taskpaper documents, but also with entire workspaces (nested folders) containing taskpaper documents.

Before you begin using the tastic command-line tools you will need to populate some custom settings within your tastic settings file.

To setup the default settings file at `~/.config/tastic/tastic.yaml` run the command:

```
tastic init
```

This should create and open the settings file; follow the instructions in the file to populate the missing settings values (usually given an `XXX` placeholder).

4.1 Sorting Taskpaper Docs via Workflow Tags

For details about exactly what happens when you sort a taskpaper document's projects and tasks via workflow tags, see the *sorting projects by tags* and *sorting tasks by tags* sections of the python code tutorial. But for now let's see how to achieve sorting via the command-line.

In the settings file you will find a set of workflow tags, which you can adapt to your liking:

```
workflowTags: "@due, @flag, @hold, @next, @someday, @wait"
```

To sort an individual taskpaper document's projects and tasks via these workflow tags (ordered from most to least prioritised) use the command:

```
tastic sort /path/to/my/doc.taskpaper
```

If you want to sort the taskpaper documents recursively contained within a workspace, pass instead the root-path of the workspace:

```
tastic sort /path/to/my/workspace/
```

4.2 Moving Archived `@done` Tasks to a Markdown Log File

To move completed tasks found in the *Archive* project of a taskpaper document into an adjacent markdown file run the command:

```
tastic archive /path/to/my/doc.taskpaper
```

This moves the completed archived tasks into a markdown file located at `/path/to/my/doc-tasklog.md` and formats them into a neat, complete-date ordered table (completed date only added if `@done` tags includes the completion date as an attribute, e.g. `@done(2016-11-09)`).

Again if you want to run this code on all taskpaper documents contained within a workspace, pass instead the root-path of the workspace:

```
tastic archive /path/to/my/workspace/
```

CHAPTER 5

Python Code Tutorial

Before we start, you'll need an example taskpaper document to work with. Copy and paste the following example document content into a taskpaper file somewhere on your file system:

```
- invite friends over for drinks


make coffee: @coffee @flag
    - scoop 3 heaped tablespoons of coffee into cafetiere
    - fill cafetiere with boiled water from kettle @hot @water
    - wait for 3 minutes @wait
    - plunge the coffee in the cafetiere
    - pour into cup @hot
    - drink
        ahhhhhhh that's good

I need to review this document every month or so to add new tasks and project, refresh and tidy curre

- do get hair cut @due

tidy the garden: @flag
    build bbq: @someday
    cut the grass:
        - has it stopped raining yet @hold
            you can check the weather here: http://forecast.io/
        - get the mower out
        - put welly boots on
        - cut the grass


    replace hedge with fence: @due
        - watch a couple of youtube videos about putting up a fence @flag
        buy fence materials:
        the hedge at the rear of the garden
        - ask neighbours if I can work from their garden to fix the fence

this is a rolling document where I can add projects and task I know I can only get done on saturdays

- take the boys to the cinema if it's raining @someday

grocery shop: @due
    - carrots
    - shampoo
    - beer
```

```
    - washing detergent
    The super-market closes at 8pm on saturdays

- take the boys to the park @next

- put up shelves in living room @flag

Archive:
    - research the price of fencing online @done(2016-09-15) @project(tidy the garden / replace hedge
    - clear the garden @done(2016-09-15) @project(tidy the garden / cut the grass)



[Searches]: @hide
    - do: due @search(/project @due//* union //@due and not @done)
    - do: flag @search(/project @flag//* union //@flag and not @done)
    - do: projects to tag @search(/project not "@" and not "archive"//*)
    - review: next or someday @search(project @next or @someday//* )
    - Project List @search(/project not @someday)
    - Next and Someday List @search(/project @next or @someday)
```

5.1 Taskpaper Objects

If you're unfamiliar with the taskpaper syntax, head over to Jesse Grosjean's User Guide for Taskpaper 3.

There are 5 basic components to the taskpaper syntax that tastic recognises; these are:

1. documents
2. projects
3. tasks
4. notes
5. tags

5.2 Working with documents

I'm going to assume that you've saved the example file above to your desktop and named the file *saturday-tasks.taskpaper*. Fire up ipython and let's get stuck in.

5.2.1 Reading a document

To read the file into memory use the following python code:

```
from tastic.tastic import document
doc = document("/Users/<yourusername>/Desktop/saturday-tasks.taskpaper")
```

This command reads the content of the file and automatically tidies it for you. To view the content of the file run the following:

```
print doc.content
```

And as you can see we now have a nice clean, ordered document; notes first, then tasks, then projects, then searches:

```
I need to review this document every month or so to add new tasks and project, refresh and tidy curre
this is a rolling document where I can add projects and task I know I can only get done on saturdays
- invite friends over for drinks
- do get hair cut @due
- take the boys to the cinema if it's raining @someday
- take the boys to the park @next
- put up shelves in living room @flag
make coffee: @coffee @flag
    - scoop 3 heaped tablespoons of coffee into cafetiere
    - fill cafetiere with boiled water from kettle @hot @water
    - wait for 3 minutes @wait
    - plunge the coffee in the cafetiere
    - pour into cup @hot
    - drink
        ahhhhhhh that's good
tidy the garden: @flag
    build bbq: @someday
    cut the grass:
        - has it stopped raining yet @hold
            you can check the weather here: http://forecast.io/
        - get the mower out
        - put welly boots on
        - cut the grass
    replace hedge with fence: @due
        the hedge at the rear of the garden
        - watch a couple of youtube videos about putting up a fence @flag
        - ask neighbours if I can work from their garden to fix the fence
        buy fence materials:
grocery shop: @due
    The super-market closes at 8pm on saturdays
    - carrots
    - shampoo
    - beer
    - washing detergent
Archive:
    - research the price of fencing online @done(2016-09-15) @project(tidy the garden / replace hedge
    - clear the garden @done(2016-09-15) @project(tidy the garden / cut the grass)
[Searches]: @hide
    - do: due @search(/project @due//* union //@due and not @done)
    - do: flag @search(/project @flag//* union //@flag and not @done)
    - do: projects to tag @search(/project not "@" and not "archive"//*)
    - review: next or someday @search(project @next or @someday//* )
    - Project List @search(/project not @someday)
    - Next and Someday List @search(/project @next or @someday)
```

If at any stage in your code you want to tidy the document again (not that you should need to), run the command:

```
doc.tidy()
```

5.2.2 Writing a document

Note any changes you make to the content of the document will have to be saved back to the file. To save the document at any stage run the command:

```
doc.save()
```

or to save the content to a different file:

```
doc.save("/Users/<yourusername>/Desktop/saturday-tasks-copy.taskpaper")
```

Note, if you save the content to another file, any further edits to the content of the file will be saved to this new location with `save()`.

5.3 Working with projects

Both documents and projects themselves can contain sub-projects.

5.3.1 Get a project by name

To select out a single project by it's title use the `get_project` method:

```
gardenProject = doc.get_project("tidy the garden")
print gardenProject.to_string()
```

```
.. code-block:: text

    tidy the garden: @flag
        build bbq: @someday
        cut the grass:
            - has it stopped raining yet @hold
                you can check the weather here: http://forecast.io/
            - get the mower out
            - put welly boots on
            - cut the grass
        replace hedge with fence: @due
            the hedge at the rear of the garden
            - watch a couple of youtube videos about putting up a fence @flag
            - ask neighbours if I can work from their garden to fix the fence
            buy fence materials:
```

Also note the use of the `to_string()` method. This method can be used on documents, projects, tasks and notes to convert the object to a string.

5.3.2 Lising projects

To compile a list of root-level projects within your document, use the `projects` attribute:

```
docProjects = doc.projects
    for p in docProjects:
        print p.title
```

```
make coffee:
tidy the garden:
grocery shop:
Archive:
```

All projects also have a `projects` attribute so you can drill down into a document's project tree to work with any sub-project. For example:

```
subProjects = gardenProject.projects
for p in subProjects:
    print p.title
```

```
build bbq:
cut the grass:
replace hedge with fence:
```

5.3.3 Filtering projects by tag

To filter projects by an associated tag, use the `tagged_projects` method:

```
dueProjects = doc.tagged_projects("@due")
for p in dueProjects:
    print p.title
```

```
replace hedge with fence:
grocery shop:
```

The keen eyed among you will notice that this filter is in fact recursive, picking up all projects within the document with the "@due" tag and not just the root level projects. Again each project has a `tagged_projects` method to allow for finer grain filtering of projects.

5.3.4 Sorting projects by tags

`sort_projects` is one of my favorite methods. Given a list of workflow tags, you can sort projects recursively within a taskpaper document or project. In the example below projects tagged with `@due` rise to the top of their parent object, followed by `@flag` projects and so on. Projects not associated with any of the workflow tags are sorted after matched projects.

```
doc.sort_projects("@due, @flag, @hold, @next, @someday, @wait")
doc.save()
print doc.content()
```

```
I need to review this document every month or so to add new tasks and project, refresh and tidy curre
this is a rolling document where I can add projects and task I know I can only get done on saturdays
- invite friends over for drinks
- do get hair cut @due
- take the boys to the cinema if it's raining @someday
- take the boys to the park @next
- put up shelves in living room @flag
grocery shop: @due
    The super-market closes at 8pm on saturdays
    - carrots
    - shampoo
    - beer
    - washing detergent
make coffee: @coffee @flag
    - scoop 3 heaped tablespoons of coffee into cafetiere
    - fill cafetiere with boiled water from kettle @hot @water
    - wait for 3 minutes @wait
    - plunge the coffee in the cafetiere
    - pour into cup @hot
    - drink
        ahhhhhhh that's good
tidy the garden: @flag
    replace hedge with fence: @due
        the hedge at the rear of the garden
        - watch a couple of youtube videos about putting up a fence @flag
```

```
        - ask neighbours if I can work from their garden to fix the fence
        buy fence materials:
    build bbq: @someday
    cut the grass:
        - has it stopped raining yet @hold
            you can check the weather here: http://forecast.io/
        - get the mower out
        - put welly boots on
        - cut the grass
Archive:
    - research the price of fencing online @done(2016-09-15) @project(tidy the garden / replace hedge
    - clear the garden @done(2016-09-15) @project(tidy the garden / cut the grass)
[Searches]: @hide
    - do: due @search(/project @due//* union //@due and not @done)
    - do: flag @search(/project @flag//* union //@flag and not @done)
    - do: projects to tag @search(/project not "@" and not "archive"//*)
    - review: next or someday @search(project @next or @someday//* )
    - Project List @search(/project not @someday)
    - Next and Someday List @search(/project @next or @someday)
```

5.3.5 Marking a project as done

To mark a project as done, use the `done()` method:

```
coffee = doc.get_project("make coffee").done()
print coffee.to_string()
```

```
make coffee: @done(2016-09-17 21:49:49)
    - scoop 3 heaped tablespoons of coffee into cafetiere
    - fill cafetiere with boiled water from kettle @hot @water
    - wait for 3 minutes @wait
    - plunge the coffee in the cafetiere
    - pour into cup @hot
    - drink
        ahhhhhhh that's good
```

It's also possible to mark all descendant items of the object as `@done` by using `done("all")`.

5.3.6 Adding a project

After sorting all the projects in the document you may have to use the `refresh` attribute for any project you have in the local namespace to refresh its attributes.

```
gardenProject.refresh
```

Now to add a sub-project use the `add_project` method (this also works on the document object):

```
# ADD A NEW PROJECT
shedProject = gardenProject.add_project(
    title="build a shed",
    tags="@someday @garden"
)

researchShedProject = shedProject.add_project(
    title="research shed designs",
    tags="@research"
```

```
)

print doc.content
```

```
I need to review this document every month or so to add new tasks and project, refresh and tidy curre
this is a rolling document where I can add projects and task I know I can only get done on saturdays
- invite friends over for drinks
- do get hair cut @due
- take the boys to the cinema if it's raining @someday
- take the boys to the park @next
- put up shelves in living room @flag
grocery shop: @due
    The super-market closes at 8pm on saturdays
    - carrots
    - shampoo
    - beer
    - washing detergent
make coffee: @coffee @flag
    - scoop 3 heaped tablespoons of coffee into cafetiere
    - fill cafetiere with boiled water from kettle @hot @water
    - wait for 3 minutes @wait
    - plunge the coffee in the cafetiere
    - pour into cup @hot
    - drink
        ahhhhhhh that's good
tidy the garden: @flag
    replace hedge with fence: @due
        the hedge at the rear of the garden
        - watch a couple of youtube videos about putting up a fence @flag
        - ask neighbours if I can work from their garden to fix the fence
        buy fence materials:
    build bbq: @someday
    cut the grass:
        - has it stopped raining yet @hold
            you can check the weather here: http://forecast.io/
        - get the mower out
        - put welly boots on
        - cut the grass
    build a shed: @someday @garden
        research shed designs: @research
Archive:
    - research the price of fencing online @done(2016-09-15) @project(tidy the garden / replace hedge
    - clear the garden @done(2016-09-15) @project(tidy the garden / cut the grass)
[Searches]: @hide
    - do: due @search(/project @due//* union //@due and not @done)
    - do: flag @search(/project @flag//* union //@flag and not @done)
    - do: projects to tag @search(/project not "@" and not "archive"//*)
    - review: next or someday @search(project @next or @someday//* )
    - Project List @search(/project not @someday)
    - Next and Someday List @search(/project @next or @someday)
```

5.3.7 Deleting a project

To delete a project, use the `delete()` method

```
doc.get_project("replace hedge with fence").delete()
print doc.content
```

```
I need to review this document every month or so to add new tasks and project, refresh and tidy curre
this is a rolling document where I can add projects and task I know I can only get done on saturdays
- invite friends over for drinks
- do get hair cut @due
- take the boys to the cinema if it's raining @someday
- take the boys to the park @next
- put up shelves in living room @flag
grocery shop: @due
    The super-market closes at 8pm on saturdays
    - carrots
    - shampoo
    - beer
    - washing detergent
make coffee: @done(2016-09-19 10:02:58)
    - scoop 3 heaped tablespoons of coffee into cafetiere
    - fill cafetiere with boiled water from kettle @hot @water
    - wait for 3 minutes @wait
    - plunge the coffee in the cafetiere
    - pour into cup @hot
    - drink
        ahhhhhhh that's good
tidy the garden: @flag
    build bbq: @someday
    cut the grass:
        - has it stopped raining yet @hold
            you can check the weather here: http://forecast.io/
        - get the mower out
        - put welly boots on
        - cut the grass
    build a shed: @someday @garden
        research shed designs: @research
Archive:
    - research the price of fencing online @done(2016-09-15) @project(tidy the garden / replace hedge
    - clear the garden @done(2016-09-15) @project(tidy the garden / cut the grass)
[Searches]: @hide
    - do: due @search(/project @due//* union //@due and not @done)
    - do: flag @search(/project @flag//* union //@flag and not @done)
    - do: projects to tag @search(/project not "@" and not "archive"//*)
    - review: next or someday @search(project @next or @someday//* )
    - Project List @search(/project not @someday)
    - Next and Someday List @search(/project @next or @someday)
```

5.4 Working with tasks

5.4.1 Listing Tasks

Documents, projects and tasks can all contain tasks. To get a list of the objects tasks, use its `tasks` attribute.

```
docTasks = doc.tasks
for t in docTasks:
    print t.title
```

```
- invite friends over for drinks
- do get hair cut
- take the boys to the cinema if it's raining
```

```
- take the boys to the park
- put up shelves in living room
```

5.4.2 Filtering Tasks by tags

To filter tasks by an associated tag, use the `tagged_tasks` method:

```
hotTasks = doc.tagged_tasks("@hot")
for t in hotTasks:
    print t.title
```

```
- fill cafetiere with boiled water from kettle
- pour into cup
```

As with the project filter, the task filter is recursive, picking up all tasks within the document with the "@hot" tag and not just the root level tasks. Again each project and task has a `tagged_tasks` method to allow for finer grain filtering of tasks.

5.4.3 Sorting tasks by tags

Given a list of workflow tags, you can sort tasks recursively within a taskpaper document, project or task. In the example below tasks tagged with `@due` rise to the top of their parent object, followed by `@flag` task and so on. Tasks not associated with any of the workflow tags are sorted after matched tasks.

```
doc.sort_tasks("@due, @flag, @hold, @next, @someday, @wait")
doc.save()
print doc.content
```

```
I need to review this document every month or so to add new tasks and project, refresh and tidy curre
this is a rolling document where I can add projects and task I know I can only get done on saturdays
- do get hair cut @due
- put up shelves in living room @flag
- take the boys to the park @next
- take the boys to the cinema if it's raining @someday
- invite friends over for drinks
grocery shop: @due
    The super-market closes at 8pm on saturdays
    - carrots
    - shampoo
    - beer
    - washing detergent
make coffee: @done(2016-09-19 13:27:19)
    - wait for 3 minutes @wait
    - scoop 3 heaped tablespoons of coffee into cafetiere
    - fill cafetiere with boiled water from kettle @hot @water
    - plunge the coffee in the cafetiere
    - pour into cup @hot
    - drink
        ahhhhhhh that's good
tidy the garden: @flag
    build bbq: @someday
    cut the grass:
        - has it stopped raining yet @hold
            you can check the weather here: http://forecast.io/
        - get the mower out
```

```
        - put welly boots on
        - cut the grass
    build a shed: @someday @garden
        research shed designs: @research
Archive:
    - research the price of fencing online @done(2016-09-15) @project(tidy the garden / replace hedge
    - clear the garden @done(2016-09-15) @project(tidy the garden / cut the grass)
[Searches]: @hide
    - do: due @search(/project @due//* union //@due and not @done)
    - do: flag @search(/project @flag//* union //@flag and not @done)
    - do: projects to tag @search(/project not "@" and not "archive"//*)
    - review: next or someday @search(project @next or @someday//* )
    - Project List @search(/project not @someday)
    - Next and Someday List @search(/project @next or @someday)
```

5.4.4 Marking a task as done

To mark a task as done, use the `done()` method:

```
coffee.refresh
for t in coffee.tasks:
    t.done("all")

print coffee.to_string()
```

```
make coffee: @done(2016-09-19 16:05:50)
    - wait for 3 minutes @done(2016-09-19 16:05:50)
    - scoop 3 heaped tablespoons of coffee into cafetiere @done(2016-09-19 16:05:50)
    - fill cafetiere with boiled water from kettle @done(2016-09-19 16:05:50)
    - plunge the coffee in the cafetiere @done(2016-09-19 16:05:50)
    - pour into cup @done(2016-09-19 16:05:50)
    - drink @done(2016-09-19 16:05:50)
        ahhhhhhh that's good
```

5.4.5 Adding a task

A task can be added to a document, project or task object using the `add_task` method:

```
aTask = researchShedProject.add_task("look for 5 videos on youtube", "@online")
aTask.add_task("note the urls of the most useful videos")
print researchShedProject.to_string()
```

```
research shed designs: @research
    - look for 5 videos on youtube @online
        - note the urls of the most useful videos
```

5.5 Working with notes

Documents, project and tasks can all have notes assigned to them.

5.5.1 Listing notes

To list the notes for any given object use the `notestr()` method.

```
doc.notestr()
```

```
I need to review this document every month or so to add new tasks and project, refresh and tidy curre
this is a rolling document where I can add projects and task I know I can only get done on saturdays
```

```
print doc.get_project("grocery shop").notestr()
```

```
The super-market closes at 8pm on saturdays
```

5.5.2 Adding a note

Use the `add_note()` method to add notes to documents, projects and tasks:

```
newNote = doc.add_note("make sure to make time to do nothing")
print doc.notestr()
```

```
I need to review this document every month or so to add new tasks and project, refresh and tidy curre
this is a rolling document where I can add projects and task I know I can only get done on saturdays
make sure to make time to do nothing
```

```
newNote = aTask.add_note(
    "good video: https://www.youtube.com/watch?v=nMaGTP82DtI")
print aTask.to_string()
```

```
- look for 5 videos on youtube @online
    good video: https://www.youtube.com/watch?v=nMaGTP82DtI
    - note the urls of the most useful videos
```

5.6 Working with tags

5.6.1 Adding a tag to a project or task

To add (append) a tag to a task or project use the `add_tag` method.

```
aTask.add_tag("@due")
print aTask.to_string()
```

```
- look for 5 videos on youtube @online @due
    good video: https://www.youtube.com/watch?v=nMaGTP82DtI
    - note the urls of the most useful videos
```

```
researchShedProject.add_tag("@hold")
print researchShedProject.to_string()
```

```
research shed designs: @research @hold
    - look for 5 videos on youtube @online @due
        good video: https://www.youtube.com/watch?v=nMaGTP82DtI
        - note the urls of the most useful videos
```

5.6.2 Setting a project's or task's tags

Instead of adding a tag, you can replace all of the tags using the `set_tags()` method.

```
researchShedProject.set_tags("@someday")
print researchShedProject.to_string()
```

```
research shed designs: @someday
    - look for 5 videos on youtube @someday
        good video: https://www.youtube.com/watch?v=nMaGTP82DtI
        - note the urls of the most useful videos
```

```
researchShedProject.set_tags("@someday")
print researchShedProject.to_string()
```

```
research shed designs: @someday
    - look for 5 videos on youtube @someday
        good video: https://www.youtube.com/watch?v=nMaGTP82DtI
        - note the urls of the most useful videos
```

5.6.3 Removing all tags from a project or task

To delete all of the tags, use the `set_tags()` method with no argument:

```
researchShedProject.set_tags()
print researchShedProject.to_string()
```

```
- look for 5 videos on youtube
    good video: https://www.youtube.com/watch?v=nMaGTP82DtI
    - note the urls of the most useful videos
```

CHAPTER 6

Installation

The easiest way to install tastic is to use `pip`:

```
pip install tastic
```

Or you can clone the github repo and install from a local version of the code:

```
git clone git@github.com:thespacedoctor/tastic.git
cd tastic
python setup.py install
```

To upgrade to the latest version of tastic use the command:

```
pip install tastic --upgrade
```

6.1 Development

If you want to tinker with the code, then install in development mode. This means you can modify the code from your cloned repo:

```
git clone git@github.com:thespacedoctor/tastic.git
cd tastic
python setup.py develop
```

Pull requests are welcomed!

6.1.1 Sublime Snippets

If you use Sublime Text as your code editor, and you're planning to develop your own python code with tastic, you might find my Sublime Snippets useful.

6.2 Issues

Please report any issues here.

CHAPTER 7

Command-Line Usage

```
Documentation for tastic can be found here: http://tastic-for-taskpaper.readthedocs.io/en/stable/


Usage:
    tastic init
    tastic sort <pathToFileOrWorkspace> [-s <pathToSettingsFile>]
    tastic archive <pathToFileOrWorkspace> [-s <pathToSettingsFile>]
    tastic [-f] sync <pathToWorkspace> <workspaceName> <pathToSyncFolder> [<editorialRootPath>] [-s <
    tastic reminders import <listName> <pathToTaskpaperDoc>

Options:
    init                    setup the tastic settings file for the first time
    sort                    sort a taskpaper file or directory containing taskpaper files via workfl
    archive                 move done tasks in the 'Archive' projects within taskpaper documents int
    reminders               commands to work with macOS reminders
    import                  import tasks into a given taskpaper document


    pathToFileOrWorkspace   give a path to an individual taskpaper file or the root of a workspace c
    pathToTaskpaperDoc      a path to a taskpaper document
    pathToWorkspace         root path of a workspace containing taskpaper files
    workspaceName           the name you give to the workspace
    pathToSyncFolder        path to the folder you wish to sync the index task files into
    listName                name of a reminders.app list (macOS only)
    editorialRootPath       the root path of editorial's dropbox sync folder (add to generate an edi
    -h, --help              show this help message
    -v, --version           show version
    -s, --settings          the settings file
    -f, --fileTags          if the tag to sync is in the filepath (e.g. /@due/mytasks.taskpaper) inc
```

CHAPTER 8

Documentation

Documentation for tastic is hosted by Read the Docs (last stable version and latest version).

CHAPTER 9

Command-Line Tools Tutorial

As well as providing python objects and methods for working with your taskpaper documents, tastic also provides some very useful command-line tools. These tools work not only with single taskpaper documents, but also with entire workspaces (nested folders) containing taskpaper documents.

Before you begin using the tastic command-line tools you will need to populate some custom settings within your tastic settings file.

To setup the default settings file at `~/.config/tastic/tastic.yaml` run the command:

```
tastic init
```

This should create and open the settings file; follow the instructions in the file to populate the missing settings values (usually given an `XXX` placeholder).

9.1 Sorting Taskpaper Docs via Workflow Tags

For details about exactly what happens when you sort a taskpaper document's projects and tasks via workflow tags, see the *sorting projects by tags* and *sorting tasks by tags* sections of the python code tutorial. But for now let's see how to achieve sorting via the command-line.

In the settings file you will find a set of workflow tags, which you can adapt to your liking:

```
workflowTags: "@due, @flag, @hold, @next, @someday, @wait"
```

To sort an individual taskpaper document's projects and tasks via these workflow tags (ordered from most to least prioritised) use the command:

```
tastic sort /path/to/my/doc.taskpaper
```

If you want to sort the taskpaper documents recursively contained within a workspace, pass instead the root-path of the workspace:

```
tastic sort /path/to/my/workspace/
```

9.2 Moving Archived `@done` Tasks to a Markdown Log File

To move completed tasks found in the *Archive* project of a taskpaper document into an adjacent markdown file run the command:

```
tastic archive /path/to/my/doc.taskpaper
```

This moves the completed archived tasks into a markdown file located at `/path/to/my/doc-tasklog.md` and formats them into a neat, complete-date ordered table (completed date only added if `@done` tags includes the completion date as an attribute, e.g. `@done(2016-11-09)`).

Again if you want to run this code on all taskpaper documents contained within a workspace, pass instead the root-path of the workspace:

```
tastic archive /path/to/my/workspace/
```

CHAPTER 10

Python Code Tutorial

Before we start, you'll need an example taskpaper document to work with. Copy and paste the following example document content into a taskpaper file somewhere on your file system:

```
- invite friends over for drinks


make coffee: @coffee @flag
    - scoop 3 heaped tablespoons of coffee into cafetiere
    - fill cafetiere with boiled water from kettle @hot @water
    - wait for 3 minutes @wait
    - plunge the coffee in the cafetiere
    - pour into cup @hot
    - drink
        ahhhhhhh that's good

I need to review this document every month or so to add new tasks and project, refresh and tidy curre

- do get hair cut @due

tidy the garden: @flag
    build bbq: @someday
    cut the grass:
        - has it stopped raining yet @hold
            you can check the weather here: http://forecast.io/
        - get the mower out
        - put welly boots on
        - cut the grass


    replace hedge with fence: @due
        - watch a couple of youtube videos about putting up a fence @flag
        buy fence materials:
        the hedge at the rear of the garden
        - ask neighbours if I can work from their garden to fix the fence

this is a rolling document where I can add projects and task I know I can only get done on saturdays

- take the boys to the cinema if it's raining @someday

grocery shop: @due
    - carrots
    - shampoo
    - beer
```

```
    - washing detergent
    The super-market closes at 8pm on saturdays

- take the boys to the park @next

- put up shelves in living room @flag

Archive:
    - research the price of fencing online @done(2016-09-15) @project(tidy the garden / replace hedge
    - clear the garden @done(2016-09-15) @project(tidy the garden / cut the grass)



[Searches]: @hide
    - do: due @search(/project @due//* union //@due and not @done)
    - do: flag @search(/project @flag//* union //@flag and not @done)
    - do: projects to tag @search(/project not "@" and not "archive"//*)
    - review: next or someday @search(project @next or @someday//* )
    - Project List @search(/project not @someday)
    - Next and Someday List @search(/project @next or @someday)
```

10.1 Taskpaper Objects

If you're unfamiliar with the taskpaper syntax, head over to Jesse Grosjean's User Guide for Taskpaper 3.

There are 5 basic components to the taskpaper syntax that tastic recognises; these are:

1. documents
2. projects
3. tasks
4. notes
5. tags

10.2 Working with documents

I'm going to assume that you've saved the example file above to your desktop and named the file *saturday-tasks.taskpaper*. Fire up ipython and let's get stuck in.

10.2.1 Reading a document

To read the file into memory use the following python code:

```
from tastic.tastic import document
doc = document("/Users/<yourusername>/Desktop/saturday-tasks.taskpaper")
```

This command reads the content of the file and automatically tidies it for you. To view the content of the file run the following:

```
print doc.content
```

And as you can see we now have a nice clean, ordered document; notes first, then tasks, then projects, then searches:

```
I need to review this document every month or so to add new tasks and project, refresh and tidy curre
this is a rolling document where I can add projects and task I know I can only get done on saturdays
- invite friends over for drinks
- do get hair cut @due
- take the boys to the cinema if it's raining @someday
- take the boys to the park @next
- put up shelves in living room @flag
make coffee: @coffee @flag
    - scoop 3 heaped tablespoons of coffee into cafetiere
    - fill cafetiere with boiled water from kettle @hot @water
    - wait for 3 minutes @wait
    - plunge the coffee in the cafetiere
    - pour into cup @hot
    - drink
        ahhhhhhh that's good
tidy the garden: @flag
    build bbq: @someday
    cut the grass:
        - has it stopped raining yet @hold
            you can check the weather here: http://forecast.io/
        - get the mower out
        - put welly boots on
        - cut the grass
    replace hedge with fence: @due
        the hedge at the rear of the garden
        - watch a couple of youtube videos about putting up a fence @flag
        - ask neighbours if I can work from their garden to fix the fence
        buy fence materials:
grocery shop: @due
    The super-market closes at 8pm on saturdays
    - carrots
    - shampoo
    - beer
    - washing detergent
Archive:
    - research the price of fencing online @done(2016-09-15) @project(tidy the garden / replace hedge
    - clear the garden @done(2016-09-15) @project(tidy the garden / cut the grass)
[Searches]: @hide
    - do: due @search(/project @due//* union //@due and not @done)
    - do: flag @search(/project @flag//* union //@flag and not @done)
    - do: projects to tag @search(/project not "@" and not "archive"//*)
    - review: next or someday @search(project @next or @someday//* )
    - Project List @search(/project not @someday)
    - Next and Someday List @search(/project @next or @someday)
```

If at any stage in your code you want to tidy the document again (not that you should need to), run the command:

```
doc.tidy()
```

10.2.2 Writing a document

Note any changes you make to the content of the document will have to be saved back to the file. To save the document at any stage run the command:

```
doc.save()
```

or to save the content to a different file:

```
doc.save("/Users/<yourusername>/Desktop/saturday-tasks-copy.taskpaper")
```

Note, if you save the content to another file, any further edits to the content of the file will be saved to this new location with `save()`.

10.3 Working with projects

Both documents and projects themselves can contain sub-projects.

10.3.1 Get a project by name

To select out a single project by it's title use the `get_project` method:

```
gardenProject = doc.get_project("tidy the garden")
print gardenProject.to_string()
```

```
.. code-block:: text

    tidy the garden: @flag
        build bbq: @someday
        cut the grass:
            - has it stopped raining yet @hold
                you can check the weather here: http://forecast.io/
            - get the mower out
            - put welly boots on
            - cut the grass
        replace hedge with fence: @due
            the hedge at the rear of the garden
            - watch a couple of youtube videos about putting up a fence @flag
            - ask neighbours if I can work from their garden to fix the fence
            buy fence materials:
```

Also note the use of the `to_string()` method. This method can be used on documents, projects, tasks and notes to convert the object to a string.

10.3.2 Lising projects

To compile a list of root-level projects within your document, use the `projects` attribute:

```
docProjects = doc.projects
    for p in docProjects:
        print p.title
```

```
make coffee:
tidy the garden:
grocery shop:
Archive:
```

All projects also have a `projects` attribute so you can drill down into a document's project tree to work with any sub-project. For example:

```
subProjects = gardenProject.projects
for p in subProjects:
    print p.title
```

```
build bbq:
cut the grass:
replace hedge with fence:
```

10.3.3 Filtering projects by tag

To filter projects by an associated tag, use the `tagged_projects` method:

```
dueProjects = doc.tagged_projects("@due")
for p in dueProjects:
    print p.title
```

```
replace hedge with fence:
grocery shop:
```

The keen eyed among you will notice that this filter is in fact recursive, picking up all projects within the document with the "@due" tag and not just the root level projects. Again each project has a `tagged_projects` method to allow for finer grain filtering of projects.

10.3.4 Sorting projects by tags

`sort_projects` is one of my favorite methods. Given a list of workflow tags, you can sort projects recursively within a taskpaper document or project. In the example below projects tagged with `@due` rise to the top of their parent object, followed by `@flag` projects and so on. Projects not associated with any of the workflow tags are sorted after matched projects.

```
doc.sort_projects("@due, @flag, @hold, @next, @someday, @wait")
doc.save()
print doc.content()
```

```
I need to review this document every month or so to add new tasks and project, refresh and tidy curre
this is a rolling document where I can add projects and task I know I can only get done on saturdays
- invite friends over for drinks
- do get hair cut @due
- take the boys to the cinema if it's raining @someday
- take the boys to the park @next
- put up shelves in living room @flag
grocery shop: @due
    The super-market closes at 8pm on saturdays
    - carrots
    - shampoo
    - beer
    - washing detergent
make coffee: @coffee @flag
    - scoop 3 heaped tablespoons of coffee into cafetiere
    - fill cafetiere with boiled water from kettle @hot @water
    - wait for 3 minutes @wait
    - plunge the coffee in the cafetiere
    - pour into cup @hot
    - drink
        ahhhhhhh that's good
tidy the garden: @flag
    replace hedge with fence: @due
        the hedge at the rear of the garden
        - watch a couple of youtube videos about putting up a fence @flag
```

```
        - ask neighbours if I can work from their garden to fix the fence
        buy fence materials:
    build bbq: @someday
    cut the grass:
        - has it stopped raining yet @hold
            you can check the weather here: http://forecast.io/
        - get the mower out
        - put welly boots on
        - cut the grass
Archive:
    - research the price of fencing online @done(2016-09-15) @project(tidy the garden / replace hedge
    - clear the garden @done(2016-09-15) @project(tidy the garden / cut the grass)
[Searches]: @hide
    - do: due @search(/project @due//* union //@due and not @done)
    - do: flag @search(/project @flag//* union //@flag and not @done)
    - do: projects to tag @search(/project not "@" and not "archive"//*)
    - review: next or someday @search(project @next or @someday//* )
    - Project List @search(/project not @someday)
    - Next and Someday List @search(/project @next or @someday)
```

10.3.5 Marking a project as done

To mark a project as done, use the `done()` method:

```
coffee = doc.get_project("make coffee").done()
print coffee.to_string()
```

```
make coffee: @done(2016-09-17 21:49:49)
    - scoop 3 heaped tablespoons of coffee into cafetiere
    - fill cafetiere with boiled water from kettle @hot @water
    - wait for 3 minutes @wait
    - plunge the coffee in the cafetiere
    - pour into cup @hot
    - drink
        ahhhhhhh that's good
```

It's also possible to mark all descendant items of the object as `@done` by using `done("all")`.

10.3.6 Adding a project

After sorting all the projects in the document you may have to use the `refresh` attribute for any project you have in the local namespace to refresh its attributes.

```
gardenProject.refresh
```

Now to add a sub-project use the `add_project` method (this also works on the document object):

```
# ADD A NEW PROJECT
shedProject = gardenProject.add_project(
    title="build a shed",
    tags="@someday @garden"
)

researchShedProject = shedProject.add_project(
    title="research shed designs",
    tags="@research"
```

```
)

print doc.content
```

```
I need to review this document every month or so to add new tasks and project, refresh and tidy curre
this is a rolling document where I can add projects and task I know I can only get done on saturdays
- invite friends over for drinks
- do get hair cut @due
- take the boys to the cinema if it's raining @someday
- take the boys to the park @next
- put up shelves in living room @flag
grocery shop: @due
    The super-market closes at 8pm on saturdays
    - carrots
    - shampoo
    - beer
    - washing detergent
make coffee: @coffee @flag
    - scoop 3 heaped tablespoons of coffee into cafetiere
    - fill cafetiere with boiled water from kettle @hot @water
    - wait for 3 minutes @wait
    - plunge the coffee in the cafetiere
    - pour into cup @hot
    - drink
        ahhhhhhh that's good
tidy the garden: @flag
    replace hedge with fence: @due
        the hedge at the rear of the garden
        - watch a couple of youtube videos about putting up a fence @flag
        - ask neighbours if I can work from their garden to fix the fence
        buy fence materials:
    build bbq: @someday
    cut the grass:
        - has it stopped raining yet @hold
            you can check the weather here: http://forecast.io/
        - get the mower out
        - put welly boots on
        - cut the grass
    build a shed: @someday @garden
        research shed designs: @research
Archive:
    - research the price of fencing online @done(2016-09-15) @project(tidy the garden / replace hedge
    - clear the garden @done(2016-09-15) @project(tidy the garden / cut the grass)
[Searches]: @hide
    - do: due @search(/project @due//* union //@due and not @done)
    - do: flag @search(/project @flag//* union //@flag and not @done)
    - do: projects to tag @search(/project not "@" and not "archive"//*)
    - review: next or someday @search(project @next or @someday//* )
    - Project List @search(/project not @someday)
    - Next and Someday List @search(/project @next or @someday)
```

10.3.7 Deleting a project

To delete a project, use the `delete()` method

```
doc.get_project("replace hedge with fence").delete()
print doc.content
```

```
I need to review this document every month or so to add new tasks and project, refresh and tidy curre
this is a rolling document where I can add projects and task I know I can only get done on saturdays
- invite friends over for drinks
- do get hair cut @due
- take the boys to the cinema if it's raining @someday
- take the boys to the park @next
- put up shelves in living room @flag
grocery shop: @due
    The super-market closes at 8pm on saturdays
    - carrots
    - shampoo
    - beer
    - washing detergent
make coffee: @done(2016-09-19 10:02:58)
    - scoop 3 heaped tablespoons of coffee into cafetiere
    - fill cafetiere with boiled water from kettle @hot @water
    - wait for 3 minutes @wait
    - plunge the coffee in the cafetiere
    - pour into cup @hot
    - drink
        ahhhhhhh that's good
tidy the garden: @flag
    build bbq: @someday
    cut the grass:
        - has it stopped raining yet @hold
            you can check the weather here: http://forecast.io/
        - get the mower out
        - put welly boots on
        - cut the grass
    build a shed: @someday @garden
        research shed designs: @research
Archive:
    - research the price of fencing online @done(2016-09-15) @project(tidy the garden / replace hedge
    - clear the garden @done(2016-09-15) @project(tidy the garden / cut the grass)
[Searches]: @hide
    - do: due @search(/project @due//* union //@due and not @done)
    - do: flag @search(/project @flag//* union //@flag and not @done)
    - do: projects to tag @search(/project not "@" and not "archive"//*)
    - review: next or someday @search(project @next or @someday//* )
    - Project List @search(/project not @someday)
    - Next and Someday List @search(/project @next or @someday)
```

10.4 Working with tasks

10.4.1 Listing Tasks

Documents, projects and tasks can all contain tasks. To get a list of the objects tasks, use its `tasks` attribute.

```
docTasks = doc.tasks
for t in docTasks:
    print t.title
```

```
- invite friends over for drinks
- do get hair cut
- take the boys to the cinema if it's raining
```

```
- take the boys to the park
- put up shelves in living room
```

10.4.2 Filtering Tasks by tags

To filter tasks by an associated tag, use the `tagged_tasks` method:

```
hotTasks = doc.tagged_tasks("@hot")
for t in hotTasks:
    print t.title
```

```
- fill cafetiere with boiled water from kettle
- pour into cup
```

As with the project filter, the task filter is recursive, picking up all tasks within the document with the "@hot" tag and not just the root level tasks. Again each project and task has a `tagged_tasks` method to allow for finer grain filtering of tasks.

10.4.3 Sorting tasks by tags

Given a list of workflow tags, you can sort tasks recursively within a taskpaper document, project or task. In the example below tasks tagged with `@due` rise to the top of their parent object, followed by `@flag` task and so on. Tasks not associated with any of the workflow tags are sorted after matched tasks.

```
doc.sort_tasks("@due, @flag, @hold, @next, @someday, @wait")
doc.save()
print doc.content
```

```
I need to review this document every month or so to add new tasks and project, refresh and tidy curre
this is a rolling document where I can add projects and task I know I can only get done on saturdays
- do get hair cut @due
- put up shelves in living room @flag
- take the boys to the park @next
- take the boys to the cinema if it's raining @someday
- invite friends over for drinks
grocery shop: @due
    The super-market closes at 8pm on saturdays
    - carrots
    - shampoo
    - beer
    - washing detergent
make coffee: @done(2016-09-19 13:27:19)
    - wait for 3 minutes @wait
    - scoop 3 heaped tablespoons of coffee into cafetiere
    - fill cafetiere with boiled water from kettle @hot @water
    - plunge the coffee in the cafetiere
    - pour into cup @hot
    - drink
        ahhhhhhh that's good
tidy the garden: @flag
    build bbq: @someday
    cut the grass:
        - has it stopped raining yet @hold
            you can check the weather here: http://forecast.io/
        - get the mower out
```

```
        - put welly boots on
        - cut the grass
    build a shed: @someday @garden
        research shed designs: @research
Archive:
    - research the price of fencing online @done(2016-09-15) @project(tidy the garden / replace hedge
    - clear the garden @done(2016-09-15) @project(tidy the garden / cut the grass)
[Searches]: @hide
    - do: due @search(/project @due//* union //@due and not @done)
    - do: flag @search(/project @flag//* union //@flag and not @done)
    - do: projects to tag @search(/project not "@" and not "archive"//*)
    - review: next or someday @search(project @next or @someday//* )
    - Project List @search(/project not @someday)
    - Next and Some day List @search(/project @next or @someday)
```

10.4.4 Marking a task as done

To mark a task as done, use the `done()` method:

```
coffee.refresh
for t in coffee.tasks:
    t.done("all")

print coffee.to_string()
```

```
make coffee: @done(2016-09-19 16:05:50)
    - wait for 3 minutes @done(2016-09-19 16:05:50)
    - scoop 3 heaped tablespoons of coffee into cafetiere @done(2016-09-19 16:05:50)
    - fill cafetiere with boiled water from kettle @done(2016-09-19 16:05:50)
    - plunge the coffee in the cafetiere @done(2016-09-19 16:05:50)
    - pour into cup @done(2016-09-19 16:05:50)
    - drink @done(2016-09-19 16:05:50)
        ahhhhhhh that's good
```

10.4.5 Adding a task

A task can be added to a document, project or task object using the `add_task` method:

```
aTask = researchShedProject.add_task("look for 5 videos on youtube", "@online")
aTask.add_task("note the urls of the most useful videos")
print researchShedProject.to_string()
```

```
research shed designs: @research
    - look for 5 videos on youtube @online
        - note the urls of the most useful videos
```

10.5 Working with notes

Documents, project and tasks can all have notes assigned to them.

10.5.1 Listing notes

To list the notes for any given object use the `notestr()` method.

```
doc.notestr()
```

```
I need to review this document every month or so to add new tasks and project, refresh and tidy curre
this is a rolling document where I can add projects and task I know I can only get done on saturdays
```

```
print doc.get_project("grocery shop").notestr()
```

```
The super-market closes at 8pm on saturdays
```

10.5.2 Adding a note

Use the `add_note()` method to add notes to documents, projects and tasks:

```
newNote = doc.add_note("make sure to make time to do nothing")
print doc.notestr()
```

```
I need to review this document every month or so to add new tasks and project, refresh and tidy curre
this is a rolling document where I can add projects and task I know I can only get done on saturdays
make sure to make time to do nothing
```

```
newNote = aTask.add_note(
    "good video: https://www.youtube.com/watch?v=nMaGTP82DtI")
print aTask.to_string()
```

```
- look for 5 videos on youtube @online
    good video: https://www.youtube.com/watch?v=nMaGTP82DtI
    - note the urls of the most useful videos
```

10.6 Working with tags

10.6.1 Adding a tag to a project or task

To add (append) a tag to a task or project use the `add_tag` method.

```
aTask.add_tag("@due")
print aTask.to_string()
```

```
- look for 5 videos on youtube @online @due
    good video: https://www.youtube.com/watch?v=nMaGTP82DtI
    - note the urls of the most useful videos
```

```
researchShedProject.add_tag("@hold")
print researchShedProject.to_string()
```

```
research shed designs: @research @hold
    - look for 5 videos on youtube @online @due
        good video: https://www.youtube.com/watch?v=nMaGTP82DtI
        - note the urls of the most useful videos
```

10.6.2 Setting a project's or task's tags

Instead of adding a tag, you can replace all of the tags using the `set_tags()` method.

```
researchShedProject.set_tags("@someday")
print researchShedProject.to_string()
```

```
research shed designs: @someday
    - look for 5 videos on youtube @someday
        good video: https://www.youtube.com/watch?v=nMaGTP82DtI
        - note the urls of the most useful videos
```

```
researchShedProject.set_tags("@someday")
print researchShedProject.to_string()
```

```
research shed designs: @someday
    - look for 5 videos on youtube @someday
        good video: https://www.youtube.com/watch?v=nMaGTP82DtI
        - note the urls of the most useful videos
```

10.6.3 Removing all tags from a project or task

To delete all of the tags, use the `set_tags()` method with no argument:

```
researchShedProject.set_tags()
print researchShedProject.to_string()
```

```
- look for 5 videos on youtube
    good video: https://www.youtube.com/watch?v=nMaGTP82DtI
    - note the urls of the most useful videos
```

Subpackages

tastic	
tastic.commonutils	*common tools used throughout package*
tastic.workspace	*tools for sorting, archiving and indexing tasks and maintaining the contents of all taskpaper files within*

tastic (*subpackage*)

tastic.commonutils (*subpackage*)

common tools used throughout package

tastic.workspace (*subpackage*)

methods for working with workspaces containing taskpaper project documents

Modules

tastic.cl_utils	Documentation for tastic can be found here: http://tastic-for-taskpaper.readthedocs.io/en/stable/
tastic.tastic	*A library of tools for working with plain-text taskpaper documents*
tastic.utKit	*Unit testing tools*

tastic.cl_utils (*module*)

Documentation for tastic can be found here: http://tastic-for-taskpaper.readthedocs.io/en/stable/

Usage: tastic init tastic sort <pathToFileOrWorkspace> [-s <pathToSettingsFile>] tastic archive <pathToFileOrWorkspace> [-s <pathToSettingsFile>] tastic [-f] sync <pathToWorkspace> <workspaceName> <pathToSyncFolder> [<editorialRootPath>] [-s <pathToSettingsFile>] tastic reminders import <listName> <pathToTaskpaperDoc>

Options: init setup the tastic settings file for the first time sort sort a taskpaper file or directory containing taskpaper files via workflow tags in settings file archive move done tasks in the 'Archive' projects within taskpaper documents into markdown tasklog files reminders commands to work with macOS reminders import import tasks into a given taskpaper document

pathToFileOrWorkspace give a path to an individual taskpaper file or the root of a workspace containing taskpaper files pathToTaskpaperDoc a path to a taskpaper document pathToWorkspace root path of a workspace containing taskpaper files workspaceName the name you give to the workspace pathToSyncFolder path to the folder you wish to sync the index task files into listName name of a reminders.app list (macOS only) editorialRootPath the root path of editorial's dropbox sync folder (add to generate an editorial URL for each task) -h, –help show this help message -v, –version show version -s, –settings the settings file -f, –fileTags if the tag to sync is in the filepath (e.g. /@due/mytasks.taskpaper) include all items the file in that tag set

`tastic.cl_utils.`**`main`**(*arguments=None*)
The main function used when ''cl_utils.py'' is run as a single script from the cl, or when installed as a cl command

tastic.tastic (*module*)

A library of tools for working with plain-text taskpaper documents

Authors @thespacedoctor

Date Created September 2, 2016

class `tastic.tastic.`**`baseClass`**(*matchObject*, *parentObject=None*)
This is the base class for all taskpaper objects: documents, projects and tasks

Key Arguments:

- `matchObject` – a dictionary containing the constituent parts of the object
- `parentObject` – the parent object containing this taskpaper object. Default *None*

`add_note`(*note*)
Add a note to this taskpaper object

Key Arguments:

- `note` – the note (string)

Return:

- None

Usage:

To add a note to a document, project or task object:

```
newNote = doc.add_note("And another note with a link http://www.thespacedoctor.co.uk")
```

add_project(*title*, *tags=None*)
Add a project to this taskpaper object

Key Arguments:

- `title` – the title for the project.
- `tags` – tag string (*"@one @two(data)"*) or list of tags (*['one', 'two(data)']*)
- `oldContent` – the old content to be replaced in parent object (user sould not need to give this)
- `newContent` – the replacement text for the parent object (user sould not need to give this)

Return:

- `project` – the new taskpaper project object

Usage:

To add a sub-project to a taskpaper document or project use:

```
newProject = doc.add_project(
    title="this is a projects I added",
    tags="@with @tags"
)
```

add_tag(*tag*)
Add a tag this taskpaper object

Key Arguments:

- `tag` – the tag to add to the object

Usage:

```
aTask.add_tag("@due")
```

add_task(*title*, *tags=None*)
Add a task to this taskpaper object

Key Arguments:

- `title` – the title for the task.
- `tags` – tag string (*'@one @two(data)'*) or list of tags (*['one', 'two(data)']*)

Return:

- `task` – the new taskpaper task object

Usage:

To add a task to an object (document, project, or task) use:

```
newTask = doc.add_task("this is a task I added", "@with @tags")
```

all_tasks()
return a flat list of all tasks contained within this taskpaper object

Return:

- `taskList` – a flat list of all tasks

Usage:

To return a flat list of all tasks recursively found with a taskpaper document object, use the following:

```
allTasks = doc.all_tasks()
for t in allTasks:
    print t.title
```

content
The text content of this object (excluding title)

Much like the *raw_content* of an object, but does not include a title or tags. The initial indentation is also removed. For a document object the *content* is synonymous with *raw_content*.

Usage:

```
pContent = aProject.content
tContent = aTask.content
```

del_tag(*tag*)
delete a tag this taskpaper object

Key Arguments:

- `tag` – the tag to delete to the object

Usage:

```
aTask.del_tag("@due")
```

done(*depth='root'*)
mark this object as done

Key Arguments:

- `depth` – either mark root item as done or all recursive items. Default *"root"*. ["root"|"all"]

Usage:

To mark a task or project as done"

```
aTask.done()
```

Or or mark the object as done as well all descendant tasks and projects:

```
aTask.done("all")
```

get_project(*projectName*)
recursively scan this taskpaper object to find a descendant project by name

Key Arguments:

- `projectName` – the name, or title, of the project you want to return

Return:

- `project` – the taskpaper project object you requested (or `None` if no project was matched)

Usage:

```
archiveProject = doc.get_project("Archive")
```

get_task(*taskName*)
recursively scan this taskpaper object to find a descendant task by name

Key Arguments:

- `taskName` – the name, or title, of the task you want to return

Return:

- `task` – the taskpaper task object you requested (or `None` if no task was matched)

Usage:

```
aTask = doc.get_task("cut the grass")
```

notes
list of the notes assoicated with this object

Usage:

The document, project and task objects can all contain notes.

```
docNotes = doc.notes
projectNotes = aProject.notes
taskNotes = aTask.notes
```

notestr()
return the notes of this object as a string

Return:

- `notestr` – the notes as a string

Usage:

```
doc.notestr
```

parent
This taskpaper object's parent object (if any)

Usage:

To reserve up the taskpaper document tree and find the parent object that contains this object (e.g. the document containing the task you're working with) use the following:

```
taskParent = aTasks.parent
print taskParent
```

prints the following

```
<Taskpaper Document `saturday-tasks.taskpaper`>
```

projects
All child projects of this taskpaper object

Usage:

Given a taskpaper document object (`doc`), to get a list of the project objects found within the document use:

```
docProjects = doc.projects
```

The same is true of project objects which may contain sub-projects:

```
aProject = docProjects[0]
subProjects = aProject.projects
```

raw_content
The raw, untidied content of the taskpaper object

Usage:

To return the inital raw content for the matched object (document, project, task or note)

```
print project.raw_content
print note.raw_content
print task.raw_content
```

set_tags(*tags=''*)
Set the tags for this taskpaper object

Key Arguments:

- `tags` – a tag string to set

Usage:

```
aTask.set_tags("@due @mac")
```

sort_projects(*workflowTags*)
order the projects within this taskpaper object via a list of tags

The order of the tags in the list dictates the order of the sort - first comes first*

Key Arguments:

- `workflowTags` – a string of space/comma seperated tags.

Return:

- `None`

Usage:

To recursively sort the projects within a taskpaper document with the following order:

1. *@due*
2. *@flag*
3. *@hold*
4. *@next*
5. *@someday*
6. *@wait*

use the following:

```
doc.sort_projects("@due, @flag, @hold, @next, @someday, @wait")
```

sort_tasks(*workflowTags*, *indentLevel=1*)
order tasks within this taskpaper object via a list of tags

The order of the tags in the list dictates the order of the sort - first comes first*

Key Arguments:

- `workflowTags` – a string of space seperated tags.

Return:

- `None`

Usage:

To recursively sort the tasks within a taskpaper document with the following order:

1. *@due*
2. *@flag*
3. *@hold*
4. *@next*
5. *@someday*
6. *@wait*

use the following:

```
doc.sort_tasks("@due, @flag, @hold, @next, @someday, @wait")
```

tagged_projects(*tag*)
return a list of projects contained within this taskpaper object filtered by a given tag

Key Arguments:

- `tag` – the tag to filter the projects by.

Return:

- `projectList` – the list of filtered projects

Usage:

To filter the projects recursively found with a taskpaper document object and return only those projects tagged with `flag`, using the following:

```
filteredProjects = doc.tagged_projects("flag")
for p in filteredProjects:
    print p.title
```

Note you can give the tag with or without the @, and you can also give a tag attribute, e.g. `@due(today)`

tagged_tasks(*tag*)
return a list of tasks contained within this taskpaper object filtered by a given tag

Key Arguments:

- `tag` – the tag to filter the tasks by.

Return:

- `taskList` – the list of filtered tasks

Usage:

To filter the tasks recursively found with a taskpaper document object and return only those tasks tagged with `flag`, using the following:

```
filteredTasks = doc.tagged_tasks("@flag")
for t in filteredTasks:
    print t.title
```

Note you can give the tag with or without the @, and you can also give a tag attribute, e.g. `@due(today)`

tags

The list of tags associated with this taskpaper object

Usage: project and task objects can have associated tags. To get a list of tags assigned to an object use:

```
projectTag = aProject.tags
taskTags = aTasks.tags

print projectTag
> ['flag', 'home(bathroom)']
```

tasks

list of the tasks assoicated with this object

Usage:

Given a taskpaper document object (`doc`), get a list of top-level tasks associated with the document using:

```
docTasks = doc.tasks
```

The same is true of project and task objects that may contain sub-tasks:

```
aProject.tasks
aTasks.tasks
```

tidy()

Tidy this taskpapaer object so that sub-objects appear in this order: title, tags, notes, tasks, projects

Return:

- `None`

Usage:

When a taskpaper document is opened it is tidied by default. To tidy the document object (or project or task) use the command:

```
doc.tidy()
```

title

The title of this taskpaper object

Usage:

```
aProject.title
aTasks.title
aNote.title
```

to_string(*indentLevel=1*, *title=True*, *tags=None*, *projects=None*, *tasks=None*, *notes=None*)
convert this taskpaper object to a string

Key Arguments:

- `indentLevel` – the level of the indent for this object. Default *1*.
- `title` – print the title of the taskpaper object alongside the contents. Default *True*
- `tags` – replace tags with these tags. Default *None*
- `projects` – replace projects with these projects, pass empty list to delete all projects. Default *None*
- `tasks` – replace tasks with these ones, pass empty list to delete all tasks. Default *None*
- `notes` – replace notes with these ones, pass empty list to delete all notes. Default *None*

Return:

- `objectString` – the taskpaper object as a string

Usage:

If we have the *archive* project from a taskpaper document, we can convert it to a string using:

```
print archiveProject.to_string()
```

```
Archive:
    - and a third task @done(2016-09-04) @project(parent project / child-project)
    - and a forth task @done(2016-09-04) @project(parent project / child-project)
    - fill the kettle @done(2016-09-04) @project(parent project / make coffee)
    - boil the kettle @done(2016-09-04) @project(parent project / make coffee)
```

class tastic.tastic.**document**(*filepath*, *parentObject=None*)
This is the taskpaper document object - top level object

Key Arguments:

- `filepath` – path to the taskpaper document

Usage:

To read a taskpaper document, use something like this:

```
# READ IN A TASKPAPER FILE
from tastic.tastic import document
taskpaperFile = "path/to/saturday-tasks.taskpaper"
doc = document(taskpaperFile)
```

Note that tastic will tidy the contents of the file when it is read into memory. See the `tidy()` method for details.

raw_content
The raw, untidied content of this taskpaper document

Usage:

```
# DISPLAY THE RAW CONTENT OF THE DOCUMENT
print doc.raw_content
```

refresh
Refreshs this documents's attributesd

Usage:

To refresh the taskpaper document:

```
doc.refresh
```

save(*copypath=None*)
save the content of the document back to the file

Key Arguments:

- `copypath` – the path to a new file if you want to make a copy of the document instead of saving it to the original filepath. Default *None*

Usage:

To save the document to file run:

```
doc.save()
```

Or to copy the content to another file run the save method with a new filepath as an argument:

```
doc.save("/path/to/saturday-tasks-copy.taskpaper")
```

searches
The search-block (if any) associated with this document

Usage:

```
# DOCUMENT SEARCHES
docSearchBlock = doc.searches
```

tags
document objects have no tags

class `tastic.tastic.`**note**(*matchObject*, *parentObject=None*)
The taskpaper note object

class `tastic.tastic.`**project**(*matchObject*, *parentObject=None*)
The taskpaper project object

delete()
delete a project from the document

Return:

- None

Usage:

```
myProject.delete()
```

refresh
Refreshs this project's attributes if, for example, the parent document's projects or tasks has been sorted

Usage:

To refresh the project:

```
myProject.refresh
```

class tastic.tastic.**task**(*matchObject*, *parentObject=None*)
The taskpaper task object

refresh
Refreshs this tasks's attributes if, for example, the parent document's projects or tasks has been sorted

Usage:

To refresh the task:

```
aTask.refresh
```

tastic.utKit (*module*)

Unit testing tools

class tastic.utKit.**utKit**(*moduleDirectory*)
Override dryx utKit

Classes

tastic.reminders	*the taskpaper reminders object*
tastic.workspace.sync	*The worker class for the sync module*
tastic.workspace.workspace	*tools for sorting, archiving and indexing tasks and maintaining the contents of all taskpaper*
tastic.tastic.baseClass	*This is the base class for all taskpaper objects: documents, projects and tasks*
tastic.tastic.document	*This is the taskpaper document object - top level object*
tastic.tastic.note	*The taskpaper note object*
tastic.tastic.project	*The taskpaper project object*
tastic.tastic.task	*The taskpaper task object*
tastic.utKit.utKit	*Override dryx utKit*

tastic.reminders (*class*)

class tastic.**reminders**(*log*, *settings=False*)
the taskpaper reminders object

Key Arguments:

- log – logger
- settings – the settings dictionary

Usage:

To setup your logger, settings and database connections, please use the fundamentals package (see tutorial here).

To initiate a reminders object, use the following:

```
from tastic import reminders
r = reminders(
    log=log,
```

```
        settings=settings
    )
```

__init__(*log*, *settings=False*)

Methods

`__init__`(log[, settings])	
`import_list`(listName, pathToTaskpaperDoc)	*import tasks from a reminder.app list into a given taskpaper document*

tastic.workspace.sync (*class*)

class `tastic.workspace.`**sync**(*log*, *workspaceRoot*, *workspaceName*, *syncFolder*, *settings=False*, *editorialRootPath=False*, *includeFileTags=True*)

The worker class for the sync module

Key Arguments:

- `log` – logger
- `settings` – the settings dictionary
- `workspaceRoot` – path to the root folder of a workspace containing taskpaper files
- `workspaceName` – the name of the workspace
- `syncFolder` – path to a folder to host your synced tag taskpaper documents.
- `editorialRootPath` – the root path of editorial's dropbox sync folder. Default *False*
- `includeFileTags` – if the tag is in the filepath (e.g. /@due/mytasks.taskpaper) include all items the file in that tag set. Default *True*

Usage:

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a sync object, use the following:

```
from tastic.workspace import sync
tp = sync(
    log=log,
    settings=settings,
    workspaceRoot="/path/to/workspace/root",
    workspaceName="myWorkspace",
    syncFolder="/path/to/sync/folder",
    includeFileTags=True
)
tp.sync()
```

After this it is simply a matter of running `tp.sync()` to sync the sync-tag set into a taskpaper document in the syncFolder called `<workspaceName>-synced-tasks.taskpaper`

__init__(*log*, *workspaceRoot*, *workspaceName*, *syncFolder*, *settings=False*, *editorialRootPath=False*, *includeFileTags=True*)

Methods

`__init__`(log, workspaceRoot, workspaceName, ...)	
`sync`()	*sync the tasks tagged with a tag in the sync-tags set to index taskpaper docu*

tastic.workspace.workspace (*class*)

class `tastic.workspace.`**`workspace`**(*log*, *fileOrWorkspacePath*, *settings=False*)

tools for sorting, archiving and indexing tasks and maintaining the contents of all taskpaper files within a given workspace

Key Arguments:

- `log` – logger
- `fileOrWorkspacePath` – the root path of the workspace you wish to sort the taskpaper docs within, or the path to a single taskpaper file
- `settings` – the settings dictionary

Usage:

To setup your logger, settings and database connections, please use the `fundamentals` package (see tutorial here).

To initiate a taskpaper workspace object, use the following:

```
from tastic.workspace import workspace
ws = workspace(
    log=log,
    settings=settings,
    fileOrWorkspacePath="/path/to/root/of/workspace"
)
```

or to target a single taskpaper document use instead the path to the file:

```
from tastic.workspace import workspace
ws = workspace(
    log=log,
    settings=settings,
    fileOrWorkspacePath="/path/to/doc.taskpaper"
)
```

`__init__`(*log*, *fileOrWorkspacePath*, *settings=False*)

Methods

`__init__`(log, fileOrWorkspacePath[, settings])	
`archive_done`()	*move done tasks from the document's 'Archive' project into an adjacent markdow*
`sort`()	*sort the workspace or individual taskpaper document via the workflow tags found*

tastic.tastic.baseClass (*class*)

class `tastic.tastic.`**`baseClass`**(*matchObject*, *parentObject=None*)

This is the base class for all taskpaper objects: documents, projects and tasks

Key Arguments:

- `matchObject` – a dictionary containing the constituent parts of the object
- `parentObject` – the parent object containing this taskpaper object. Default *None*

__init__(*matchObject*, *parentObject=None*)

Methods

__init__(matchObject[, parentObject])	
add_note(note)	*Add a note to this taskpaper object*
add_project(title[, tags])	*Add a project to this taskpaper object*
add_tag(tag)	*Add a tag this taskpaper object*
add_task(title[, tags])	*Add a task to this taskpaper object*
all_tasks()	*return a flat list of all tasks contained within this taskpaper object*
del_tag(tag)	*delete a tag this taskpaper object*
done([depth])	*mark this object as done*
get_project(projectName)	*recursively scan this taskpaper object to find a descendant project by name*
get_task(taskName)	*recursively scan this taskpaper object to find a descendant task by name*
notestr()	*return the notes of this object as a string*
set_tags([tags])	*Set the tags for this taskpaper object*
sort_projects(workflowTags)	*order the projects within this taskpaper object via a list of tags*
sort_tasks(workflowTags[, indentLevel])	*order tasks within this taskpaper object via a list of tags*
tagged_projects(tag)	*return a list of projects contained within this taskpaper object filtered by a given tag*
tagged_tasks(tag)	*return a list of tasks contained within this taskpaper object filtered by a given tag*
tidy()	*Tidy this taskpapaer object so that sub-objects appear in this order: title, tags, notes,*
to_string([indentLevel, title, tags, ...])	*convert this taskpaper object to a string*

Attributes

content	*The text content of this object (excluding title)*
notes	*list of the notes assoicated with this object*
parent	*This taskpaper object's parent object (if any)*
projects	*All child projects of this taskpaper object*
raw_content	*The raw, untidied content of the taskpaper object*
tags	*The list of tags associated with this taskpaper object*
tasks	*list of the tasks assoicated with this object*
title	*The title of this taskpaper object*

tastic.tastic.document (*class*)

class `tastic.tastic.`**`document`**(*filepath*, *parentObject=None*)

This is the taskpaper document object - top level object

Key Arguments:

- `filepath` – path to the taskpaper document

Usage:

To read a taskpaper document, use something like this:

```
# READ IN A TASKPAPER FILE
from tastic.tastic import document
taskpaperFile = "path/to/saturday-tasks.taskpaper"
doc = document(taskpaperFile)
```

Note that tastic will tidy the contents of the file when it is read into memory. See the `tidy()` method for details.

__init__ (*filepath*, *parentObject=None*)

Methods

__init__(filepath[, parentObject])	
add_note(note)	*Add a note to this taskpaper object*
add_project(title[, tags])	*Add a project to this taskpaper object*
add_tag()	
add_task(title[, tags])	*Add a task to this taskpaper object*
all_tasks()	*return a flat list of all tasks contained within this taskpaper object*
del_tag(tag)	*delete a tag this taskpaper object*
done()	
get_project(projectName)	*recursively scan this taskpaper object to find a descendant project by name*
get_task(taskName)	*recursively scan this taskpaper object to find a descendant task by name*
notestr()	*return the notes of this object as a string*
save([copypath])	*save the content of the document back to the file*
set_tags()	
sort_projects(workflowTags)	*order the projects within this taskpaper object via a list of tags*
sort_tasks(workflowTags[, indentLevel])	*order tasks within this taskpaper object via a list of tags*
tagged_projects(tag)	*return a list of projects contained within this taskpaper object filtered by a given tag*
tagged_tasks(tag)	*return a list of tasks contained within this taskpaper object filtered by a given tag*
tidy()	*Tidy this taskpapaer object so that sub-objects appear in this order: title, tags, notes,*
to_string([indentLevel, title, tags, ...])	*convert this taskpaper object to a string*

Attributes

content	*The text content of this object (excluding title)*
notes	*list of the notes assoicated with this object*
parent	*This taskpaper object's parent object (if any)*
projects	*All child projects of this taskpaper object*
raw_content	*The raw, untidied content of this taskpaper document*
refresh	*Refreshs this documents's attributesd*
searches	*The search-block (if any) associated with this document*
tags	*document objects have no tags*
tasks	*list of the tasks assoicated with this object*
title	*The title of this taskpaper object*

tastic.tastic.note (*class*)

class tastic.tastic.**note** (*matchObject*, *parentObject=None*)

The taskpaper note object

__init__(*matchObject*, *parentObject=None*)

Methods

__init__(matchObject[, parentObject])	
add_note()	
add_project()	
add_tag()	
add_task()	
all_tasks()	*return a flat list of all tasks contained within this taskpaper object*
del_tag(tag)	*delete a tag this taskpaper object*
done()	
get_project()	
get_task()	
notestr()	*return the notes of this object as a string*
set_tags()	
sort_projects()	
sort_tasks()	
tagged_projects()	
tagged_tasks()	
tidy()	*Tidy this taskpapaer object so that sub-objects appear in this order: title, tags, notes, tas*
to_string([indentLevel, title, tags, ...])	*convert this taskpaper object to a string*

Attributes

content	*The text content of this object (excluding title)*
notes	
parent	*This taskpaper object's parent object (if any)*
projects	
raw_content	*The raw, untidied content of the taskpaper object*
tags	*The list of tags associated with this taskpaper object*
tasks	
title	*The title of this taskpaper object*

tastic.tastic.project (*class*)

class tastic.tastic.**project**(*matchObject*, *parentObject=None*)

The taskpaper project object

__init__(*matchObject*, *parentObject=None*)

Methods

__init__(matchObject[, parentObject])	
add_note(note)	*Add a note to this taskpaper object*
add_project(title[, tags])	*Add a project to this taskpaper object*
add_tag(tag)	*Add a tag this taskpaper object*

Continued

Table 10.13 – continued from previous page

add_task(title[, tags])	*Add a task to this taskpaper object*
all_tasks()	*return a flat list of all tasks contained within this taskpaper object*
del_tag(tag)	*delete a tag this taskpaper object*
delete()	*delete a project from the document*
done([depth])	*mark this object as done*
get_project(projectName)	*recursively scan this taskpaper object to find a descendant project by name*
get_task(taskName)	*recursively scan this taskpaper object to find a descendant task by name*
notestr()	*return the notes of this object as a string*
set_tags([tags])	*Set the tags for this taskpaper object*
sort_projects(workflowTags)	*order the projects within this taskpaper object via a list of tags*
sort_tasks(workflowTags[, indentLevel])	*order tasks within this taskpaper object via a list of tags*
tagged_projects(tag)	*return a list of projects contained within this taskpaper object filtered by a given tag*
tagged_tasks(tag)	*return a list of tasks contained within this taskpaper object filtered by a given tag*
tidy()	*Tidy this taskpapaer object so that sub-objects appear in this order: title, tags, notes,*
to_string([indentLevel, title, tags, ...])	*convert this taskpaper object to a string*

Attributes

content	*The text content of this object (excluding title)*
notes	*list of the notes assoicated with this object*
parent	*This taskpaper object's parent object (if any)*
projects	*All child projects of this taskpaper object*
raw_content	*The raw, untidied content of the taskpaper object*
refresh	*Refreshs this project's attributes if, for example, the parent document's projects or tasks has been sorted*
tags	*The list of tags associated with this taskpaper object*
tasks	*list of the tasks assoicated with this object*
title	*The title of this taskpaper object*

tastic.tastic.task (*class*)

class tastic.tastic.**task**(*matchObject*, *parentObject=None*)

The taskpaper task object

__init__(*matchObject*, *parentObject=None*)

Methods

__init__(matchObject[, parentObject])	
add_note(note)	*Add a note to this taskpaper object*
add_project()	
add_tag(tag)	*Add a tag this taskpaper object*
add_task(title[, tags])	*Add a task to this taskpaper object*
all_tasks()	*return a flat list of all tasks contained within this taskpaper object*
del_tag(tag)	*delete a tag this taskpaper object*
done([depth])	*mark this object as done*
get_project()	
get_task(taskName)	*recursively scan this taskpaper object to find a descendant task by name*

Continued

Table 10.15 – continued from previous page

notestr()	*return the notes of this object as a string*
set_tags([tags])	*Set the tags for this taskpaper object*
sort_projects()	
sort_tasks(workflowTags[, indentLevel])	*order tasks within this taskpaper object via a list of tags*
tagged_projects()	
tagged_tasks(tag)	*return a list of tasks contained within this taskpaper object filtered by a given tag*
tidy()	*Tidy this taskpapaer object so that sub-objects appear in this order: title, tags, notes,*
to_string([indentLevel, title, tags, ...])	*convert this taskpaper object to a string*

Attributes

content	*The text content of this object (excluding title)*
notes	*list of the notes assoicated with this object*
parent	*This taskpaper object's parent object (if any)*
projects	
raw_content	*The raw, untidied content of the taskpaper object*
refresh	*Refreshs this tasks's attributes if, for example, the parent document's projects or tasks has been sorted*
tags	*The list of tags associated with this taskpaper object*
tasks	*list of the tasks assoicated with this object*
title	*The title of this taskpaper object*

tastic.utKit.utKit (*class*)

class tastic.utKit.**utKit**(*moduleDirectory*)

Override dryx utKit

__init__(*moduleDirectory*)

Methods

__init__(moduleDirectory)	
setupModule()	*The setupModule method*
tearDownModule()	*The tearDownModule method*

Functions

10.7 Indexes

- Module Index
- Full Index

10.8 Todo

- Todolist

t

Symbols

A

B

C

D

G

M

N

P

R

S

T

U

W