filehandlers Documentation

RDIL

Oct 26, 2019

Contents:

By RDIL[1]

View on GitHub | PyPI

Welcome to the docs. Here you can read about all the code, what it does, and how to use it :D

[1] <me@rdil.rocks>

CHAPTER 1

Model

filehandlers is built on a relatively simple model.

1.1 File

A file is represented with an instance of *filehandlers.AbstractFile()*.

Important: The actual file will not be changed or even inspected when creating an instance of *filehandlers.AbstractFile()*.

1.2 Manipulation

Now, say you want to change that *File*... that is where *filehandlers.FileManipulator()* comes in. You need to pass the *filehandlers.AbstractFile()* instance when creating a *filehandlers.FileManipulator()* because otherwise the manipulator can't do it's job.

The manipulator includes code for a number of common functions that could be replicated with other code, but the goal of creating this model/API is to simplify it!

1.3 Simple Example

Here is a quick example that shows how to use filehandlers to write to a file:

```
# load in filehandlers
from filehandlers import FileManipulator, AbstractFile

# define data
my_cool_file = AbstractFile("log.txt")
debug_message = "my code works :)"

# create FileManipulator
my_cool_files_changer = FileManipulator(my_cool_file)

# write data to file 5 times
for i in range(5):
    my_cool_files_changer.wrap_file().write("Message #" + i + ": " +␣
↪debug_message)
```

CHAPTER 2

API Reference

Here you can see the docs for all the different methods, classes, etc.

2.1 AbstractFile

class filehandlers.**AbstractFile**(*name*)

A file in instance form.

Parameters **name** (*str*) – The file name

__init__(*name*)

Create the class.

Parameters **name** (*str*) – The file name

Returns None

Return type NoneType

__str__()

Override str().

Returns the name

Return type str

change_file_name(*n*)

Changes the file name.

Important: This doesn't change the file's actual name, it changes the name of the file focused on by this AbstractFile instance. We suggest you don't use this because you can just create different AbstractFile instances for different files.

Returns None

Return type NoneType

Parameters **n** (*str*) – the new name for the file

wrap(*doreturn=True*)

Wrap file in TextIOWrapper.

Parameters **doreturn** (*bool*) – *Just keep this True (or don't pass the keyword argument).*

Returns The wrapper

Return type io.TextIOWrapper

Raises **PermissionError** – If you don't have needed permission to access the file

touch()

Create the file if it doesn't already exist.

Important: This is the only method that actually changes/interacts with the file inside the AbstractFile class (other than *wrap()* and *exists()*).

In case you are wondering, the name for this function comes from the Unix command (`touch`), which creates a new file with the name as a parameter.

Returns None

Return type NoneType

Raises **PermissionError** – If you don't have needed permission to access the file

exists(*touch_if_false=False*)

Get if this file exists or not (boolean value).

Returns If the focused file exists

Return type bool:

Parameters **touch_if_false** (*bool*) – If the file should be created if it doesn't exist. Defaults to False.

Throws PermissionError If you don't have the required permissions to access the file.

2.2 FileManipulator

class filehandlers.**FileManipulator**(*abstract_file*)

Class used for managing it's assigned file.

Parameters **abstract_file** (AbstractFile) – the file to manage

__init__(*abstract_file*)

Create class instance.

Parameters **abstract_file** (AbstractFile) – the AbstractFile instance

Returns None

Return type NoneType

Raises TypeError

get_file()

Get the AbstractFile instance.

Returns the AbstractFile instance

Return type *AbstractFile*

get_file_name()

Get the file's name.

Returns The file's name

Return type str

refresh(*slim=False*)

Update the cache.

Parameters **slim** (*bool*) – (Optional) - if empty lines should be removed - defaults to True.

Returns None

Return type NoneType

Raises **PermissionError** – If you don't have needed permission to access the file

get_cache()

Get the cache.

The cache will be a list of the file's lines at the time of the last refresh.

Refreshes are called when this class is created, or when manually triggered by *refresh()*.

Returns the cache

Return type list

write_to_file(*string*)
Write to the file.

Note: Please ensure that what you are writing to the file is a string.

Parameters **string** (*str*) – What to write to the file.

Raises

- PermissionError – If you don't have needed permission to access the file
- TypeError – If you pass an unsupported type to be written

Returns None

Return type NoneType

wrap_file()
Shortcut for `get_file().wrap()`.

See also:

filehandlers.AbstractFile.wrap()

return Wrapped file

rtype io.TextIOWrapper

clear_file()
Clear the file.

Warning: You may not be able to recover the old contents!

Returns None

Return type NoneType

Raises **PermissionError** – If you don't have needed permission to access the file

get_file_contents_singlestring()
Get the file's contents, but as one string.

Note: This function does not use the cache.

Returns The file's contents

Return type str

Raises **PermissionError** – If you don't have needed permission to access the file

2.3 OpenModes

class filehandlers.**OpenModes**
`enum.Enum()` for the different options you can pass to the keyword argument `mode` in Python's `builtins.open()` function.

It can be used like this:

```
from filehandlers import OpenModes
open("myfile.txt", mode=OpenModes.READ.value)
```

This can help so you don't need to remember all the different `mode` options.

> **Warning:** For the `write` option, the file will be cleared and then written to. To avoid this, use `append` instead!

READ = 'r'
Read only access to the file

WRITE = 'w'
Write only access to the file - ***see warning above***

CLEAR = 'w'
Clear the file

APPEND = 'a'
Append to the end of the file (also gives read!)

CREATE = 'x'
Create the file - ***raises error if file exists***

CREATE_AND_WRITE = 'w+'
Create the file and ready it to be written to

TEXT = 't'
Default

BINARY = 'b'
Open in binary mode

UPDATING = '+'
This will open a file for reading and writing (updating)

CHAPTER 3

Contributing Guide

Welcome to the filehandlers contributing guidelines! Please follow what is listed here to help keep the project simple, easy to maintain, and complete.

3.1 Style Guide

3.1.1 Code

When writing code, please follow PEP8[1], the style guide written for any Python. One exception for this is our line length - PEP8 says 79 characters is the limit, but we have decided to extend that to a maximum of 100 characters per line. Also, wherever possible, please add *Docstrings*.

3.1.2 Docstrings

We mainly use normal reStructuredText Docstrings for filehandlers. However, for elements that have more then one field - for example if a function has the potential to throw multiple exceptions - we use Google[2] and/or NumPy[3] styled Docstrings. These are later converted by the Sphinx extension `napoleon` during documentation builds. Please add Docstrings to functions, classes, modules (at the top), exceptions and enumerators.

[1] Python Enhancement Proposal #8
[2] Google Docstring Guide
[3] NumPy Docstring Guide

If you add a class that is not covered by the documentation currently (e.x. you create a method or class that is *not* in a method/class that has the `:members:` autodoc method), you will need to add a field so autodoc knows the include the Docstring(s). To do this, navigate to the API reference page and add this underneath all the other classes/methods:

3.1.2.1 For classes/enumerators

```
,.. autoclass:: ClassName
   ,:members:
   ,:special-members: __init__
```

3.1.2.2 For functions

```
,.. autofunction:: function_name
```

3.1.2.3 For exceptions

```
,.. autoexception:: ExceptionName
```

3.1.2.4 Notes

- **Make sure to remove the commas, they prevent autodoc from trying to add a method here on this page!**
- You do *not* need to add module Docstrings to the API reference page.

3.2 Building the package

Simply run `python setup.py sdist bdist_wheel`. *If you are on Linux/macOS, change* `python` *to* `python3`. The build shouldn't be long. When it is complete, you can find the `.tar.gz` and `.whl` in the `dist` directory.

CHAPTER 4

Code of Conduct

(Contributor Covenant Code of Conduct)

4.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

4.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

4.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

4.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

4.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at me@rdil.rocks. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

4.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant[1], version 1.4[2].

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq

[1] Contributor Covenant Website
[2] Contributor Covenant 1.4

Index

Symbols

A

B

C

E

F

G

O

R

T

U

W