Writing YAML files with python

Using ruamel.yaml to annotate configuration files

This article expects the reader to have a basic understanding of the python programming language.

Src: https://pxhere.com/en/photo/492303

Yaml files have been around for over twenty years now[1]. They're extremely useful for showing data structures and are a popular format for configuration files. From AWS CloudFormation[2], Common Workflow Language[3] to Home Assistant[4], yamls are a fundamental backbone for configuration files of the internet. They're significantly easier to read than other formats like JSON, support comments and easy to version control — try running `git merge` on two slightly different json files and you'll appreciate what I mean. Despite this, when it comes to writing yaml files in our beloved language python, our options are quite limited. Only PyYAML and ruamel.yaml have registries in the yaml conformance test matrix, and of these only ruamel.yaml supports handling comments.

Being able to write comments into your configuration files is particularly useful if user may need to tinker with said files manually at a later stage. Another advantage is enum support, by adding comments to a yaml file, a user can see what all of the available options are for a setting without having to scroll through documentation to find them! Even better, comments in configuration files can let one include links to the documentation on where they can read up on that section of the configuration file!

So let's get started with writing some yaml through python

An example yaml file:

```
employees:
  - name: Jeffrey Bezos
    job title: CEO
    annual salary (USD): 1000000000000
  - name: John Smith
    job title: factory worker
    annual salary (USD): 20000
```

This yaml file is the python equivalent of a dictionary with one key "employees" that contains a list of two elements.
Each element in the nested list contains the three same keys: "name", "job title" and "annual salary (USD)".

One of the cool things about yaml files is being able to annotate them with comments using the "#" symbol

```
employees:
  # Start with CEO
  - name: Jeffrey Bezos  # Goes by Jeff
    job title: CEO # ...entrepreneur, born in 1964...
    annual salary (USD): 1000000000000  # This is too much
  # List of factory workers below
  - name: John Smith
    job title: factory worker
    annual salary (USD): 20000  # Probably deserves a raise
```

Installation of ruamel.yaml

Below we will be trying to load this yaml file into python with ruamel.yaml. If you need to install ruamel.yaml, we will be working with version `0.17.4` , here a few commands for assistance.

Installation option 1:

```
pip install ruamel.yaml==0.17.4
```

Installation option 2:

```
conda install -c conda-forge ruamel.yaml==0.17.4
```

Importing ruamel into python and loading the yaml file

```
# Imports
from ruamel.yaml.main import round_trip_load as yaml_load

employees_dict = yaml_load("""
employees:
  # Start with CEO
  - name: Jeffrey Bezos  # Goes by Jeff
    job title: CEO # / Entrepreneur, born in 1964...
    annual salary (USD): 1000000000000  # This is too much
  # List of factory workers below
  - name: John Smith
    job title: factory worker
    annual salary (USD): 20000  # Probably deserves a raise
""")

print(employees_dict)
print(f"Type: {type(employees_dict}")
```

Output:

```
ordereddict([('employees', [ordereddict([('name', 'Jeffrey Bezos'), ('job title',
'CEO'), ('annual salary (USD)', 1000000000000)]), ordereddict([('name', 'John Smith'),
('job title', 'factory worker'), ('annual salary (USD)', 20000)])])])
Type: <class 'ruamel.yaml.comments.CommentedMap'>
```

Cool! That was pretty simple. Now let's try building a yaml file from scratch!

Building a yaml file from scratch

Let's first set out our imports and global variables

```
# Regular imports
from copy import deepcopy

# Yaml loaders and dumpers
from ruamel.yaml.main import \
    round_trip_load as yaml_load, \
    round_trip_dump as yaml_dump

# Yaml commentary
from ruamel.yaml.comments import \
    CommentedMap as OrderedDict, \
    CommentedSeq as OrderedList

# For manual creation of tokens
from ruamel.yaml.tokens import CommentToken
from ruamel.yaml.error import CommentMark

# Globals
# Number of spaces for an indent
INDENTATION = 2
# Used to reset comment objects
tsRESET_COMMENT_LIST = [None, [], None, None]
```

And create a shopping list using the `OrderedDict` and `OrderedList` attributes.

```
shopping_list = OrderedDict({
    "Shopping List": OrderedDict({
        "eggs": OrderedDict({
            "type": "free range",
            "brand": "Mr Tweedy",
            "amount": 12
        }),
        "milk": OrderedDict({
            "type": "pasteurised",
            "litres": 1.5,
            "brands": OrderedList([
                "FarmFresh",
```

```
                "FarmHouse gold",
                "Daisy The Cow"
            ])
        })
    })
})

# Dump the yaml file
print(yaml_dump(shopping_list, preseve_quotes=True))
```

Output:

```
Shopping List:
  eggs:
    type: free range
    brand: Mr Tweedy
    amount: 12
  milk:
    type: pasteurised
    litres: 1.5
    brands:
      - FarmFresh
      - FarmHouse gold
      - Daisy The Cow
```

Creating our data structure was quite tedious. It's also not likely our data structure we wish to write out to yaml would already be in this format. It's much more likely to be in standard dictionary and/or list format. That's okay, we can `dump` and `load` instead to convert standard dict and list formats into our OrderedList (`CommentedMap`) and OrderedList (`CommentedSeq`).

```
# Original object
shopping_list = {
    "Shopping List": {
        "eggs": {
            "type": "free range",
            "brand": "Mr Tweedy",
            "amount": 12
        },
        "milk": {
            "type": "pasteurised",
            "litres": 1.5,
            "brands": [
                "FarmFresh",
                "FarmHouse gold",
                "Daisy The Cow"
            ]
        }
    }
}
```

```
# To yaml object
shopping_list = yaml_load(yaml_dump(shopping_list), preseve_quotes=True)

# Show object type
print(type(shopping_list))
```

Output:

```
<class 'ruamel.yaml.comments.CommentedMap'>
```

Perfect!

Add a header to the yaml file

We can use the `yaml_set_start_comment` attribute which will add a comment above the map object

```
shopping_list.yaml_set_start_comment("Shopping Lists for date: "
                                     "23 Oct 2021")
print(yaml_dump(shopping_list,
                indent=INDENTATION,
                block_seq_indent=INDENTATION))
```

Output:

```
# Shopping Lists for date: 21 Oct 2021
Shopping List:
  eggs:
    type: free range
    brand: Mr Tweedy
    amount: 12
  milk:
    type: pasteurised
    litres: 1.5
    brands:
      - FarmFresh
      - FarmHouse gold
      - Daisy The Cow
```

Add a comment before a nested key

Lets say we really don't want to forget eggs, let's write above the egg key 'Please don't forget eggs!'. Since the 'Shopping List' attribute of the `shopping_list` object is also an `OrderedDict` or `OrderedList`, we can use the `yaml_set_comment_before_after_key` method on the `Shopping List` attribute.

```
shopping_list.get("Shopping List").\
    yaml_set_comment_before_after_key(key="eggs",
                                      before="Please don't forget "
                                             "eggs!")
print(yaml_dump(shopping_list,
                indent=INDENTATION,
                block_seq_indent=INDENTATION))
```

Output:

```
# Shopping Lists for date: 21 Oct 2021
Shopping List:
#  Please don't forget eggs!
  eggs:
    type: free range
    brand: Mr Tweedy
    amount: 12
  milk:
    type: pasteurised
    litres: 1.5
    brands:
      - FarmFresh
      - FarmHouse gold
      - Daisy The Cow
```

Ugh, this isn't ideal! That comment should be indented. Let's delete it and make sure to use the `indent` parameter next time. To delete the comment, we need to reset the *eggs* key in the `ca` attribute of the `OrderedDict` and try again.

Deleting a comment object

```
shopping_list.get("Shopping List").ca.\
    items["eggs"] = deepcopy(RESET_COMMENT_LIST)
print(yaml_dump(shopping_list,
                indent=INDENTATION,
                block_seq_indent=INDENTATION))
```

Output:

```
# Shopping Lists for date: 21 Oct 2021
Shopping List:
  eggs:
    type: free range
    brand: Mr Tweedy
    amount: 12
  milk:
    type: pasteurised
```

```
    litres: 1.5
    brands:
      - FarmFresh
      - FarmHouse gold
      - Daisy The Cow
```

Let's try that again with the `indent` parameter (set to 2 spaces)

```
object_depth = 1 # Shopping List

shopping_list.get("Shopping List").\
    yaml_set_comment_before_after_key(
        key="eggs",
        before="Don't forget the eggs",
        indent=object_depth*INDENTATION
    )

print(yaml_dump(shopping_list,
                indent=INDENTATION,
                block_seq_indent=INDENTATION))
```

Output:

```
# Shopping Lists for date: 21 Oct 2021
Shopping List:
  # Don't forget the eggs
  eggs:
    type: free range
    brand: Mr Tweedy
    amount: 12
  milk:
    type: pasteurised
    litres: 1.5
    brands:
      - FarmFresh
      - FarmHouse gold
      - Daisy The Cow
```

Add a comment after a key

We can also add a comment after a key too

```
object_depth = 2 # Shopping List -> Eggs
shopping_list.get("Shopping List").\
    yaml_set_comment_before_after_key(
        key="eggs",
        after="Please don't forget eggs!",
        after_indent=object_depth*INDENTATION
    )
print(yaml_dump(shopping_list,
```

```
                indent=INDENTATION,
                block_seq_indent=INDENTATION))
```

Output:

```
# Shopping Lists for date: 21 Oct 2021
Shopping List:
  # Don't forget the eggs
  eggs:
    # Please don't forget eggs!
    type: free range
    brand: Mr Tweedy
    amount: 12
  milk:
    type: pasteurised
    litres: 1.5
    brands:
      - FarmFresh
      - FarmHouse gold
      - Daisy The Cow
```

Lets add an end-of-line comment to a key

Put in a courteous reminder that two litres of milk is just too much milk!

```
shopping_list.get("Shopping List").\
    get("milk").\
    yaml_add_eol_comment(
        key="litres",
        comment="2 litres is too much milk!"
    )

print(yaml_dump(shopping_list,
                indent=INDENTATION,
                block_seq_indent=INDENTATION))
```

Output:

```
# Shopping Lists for date: 21 Oct 2021
Shopping List:
  # Don't forget the eggs
  eggs:
    # Please don't forget eggs!
    type: free range
    brand: Mr Tweedy
    amount: 12
  milk:
    type: pasteurised
    litres: 1.5  # 2 litres is too much milk!
    brands:
```

```
        - FarmFresh
        - FarmHouse gold
        - Daisy The Cow
```

Add a comment in a list

Lets note that *FarmHouse gold* and *Daisy The Cow* are last resort options. By using the before parameter on the index of the FarmHouse gold item of the list. For comments in lists, only the before parameter is supported.

```
object_depth = 3 # Shopping List -> Milk -> Brands
shopping_list.get("Shopping List").\
    get("milk").\
    get("brands").\
    yaml_set_comment_before_after_key(
        key=1, # "Farmhouse gold" is the second item in the list
        before="Last Resorts",
        indent=object_depth*INDENTATION
    )
print(yaml_dump(shopping_list,
                indent=INDENTATION,
                block_seq_indent=INDENTATION))
```

Output:

```
# Shopping Lists for date: 21 Oct 2021
Shopping List:
  # Don't forget the eggs
  eggs:
    type: free range
    brand: Mr Tweedy
    amount: 12
  milk:
    type: pasteurised
    litres: 1.5  # 2 litres is too much milk!
    brands:
      - FarmFresh
      # Last Resorts
      - FarmHouse gold
      - Daisy The Cow
```

Add an end-of-line comment to an item in a list

```
object_depth = 3 # Shopping List -> Milk -> Brands
shopping_list.get("Shopping List").\
    get("milk").\
    get("brands").\
    yaml_add_eol_comment(
        key=1,
```

```
            comment="Too creamy"
        )
    print(yaml_dump(shopping_list,
                    indent=INDENTATION,
                    block_seq_indent=INDENTATION))
```

Output:

```
TypeError: 'CommentToken' object is not iterable
```

Oh no! This is a known bug, and we're going to have to use the `ca` attribute of the brands object to add this to the list.

A breakdown of the ca attribute:

Let's see what we're dealing with first:

```
print(shopping_list.get("Shopping List").get("milk").get("brands").ca)
```

Output:

```
Comment(
  start=None,
  items={
    1: [None, [CommentToken('# Last Resorts\n', col: 6)], None, None]
  })
```

Only the first two elements in the values for the items attribute are used in a `CommentedSeq`. The first element is the eol comments, while the second one is the list of before comments. For a `CommentedMap`, the first element represents the 'start comments', the second element is the before comments, the third is the eol comments and the fourth is the after comments. This has been summarised in the *Resources* section of this article.

Manually adding in a CommentToken object

We need to set the first element of the items to be our desired comment.
When manually adding in a `CommentToken` object, we prefix with `\ #` and end with a new line character `\n`.
We initialise the `start_mark` parameter with a minimal `CommentMark` object and set the `end_mark` parameter to `None`.

```
shopping_list.get("Shopping List").\
    get("milk").\
    get("brands").ca.\
    items[1][0] = CommentToken(value=" # Too creamy\n",
                               start_mark=CommentMark(0),
                               end_mark=None)
print(yaml_dump(shopping_list,
                indent=INDENTATION,
                block_seq_indent=INDENTATION))
```

Output:

```
# Shopping Lists for date: 21 Oct 2021
Shopping List:
  # Don't forget the eggs
  eggs:
    type: free range
    brand: Mr Tweedy
    amount: 12
  milk:
    type: pasteurised
    litres: 1.5  # 2 litres is too much milk!
    brands:
      - FarmFresh
      # Last Resorts
      - FarmHouse gold  # Too creamy
      - Daisy The Cow
```

A summary of workarounds

From this, you can see that ruamel.yaml has some room for improvement:

1. One needs to set the indentation when using the before or after parameters with the `yaml_set_comment_before_after_key` method. This also means one needs to know how deep the nested is on creation of the comment.
2. One cannot use the `yaml_add_eol_comment` method on an element in a list if a before comment already exists on said method.
3. At the time of writing this, the latest version of ruamel (0.17.16) had a bug such that many of the outputs omitted many of the comments.
4. There is nothing in the ruamel.yaml docs on writing comments into yaml files.

A small rant on my findings

How does one of the internet's most important configuration format structures not have a fully supported parser from one of the world's most popular programming languages? I'm sure there must be thousands of python developers out there that have had the same thought as I have.

Before anyone is too quick to comment '*you should raise this with the developer*', I have[5] [6] [7]. In doing so, I can assume why so many before me never bothered. ruamel.yaml is hosted on SourceForge and when visiting this website, you'll feel like you've entered the 90s (from a UX context, the 90s is a bad place to be). Unlike GitHub, SourceForge doesn't let you edit issues after creation. The interface is clunky, the search algorithm is way too sensitive when searching for an existing issue (everything comes up!) and the notification service is ancient — rather than a notifications tab, you'll receive an email upon a new comment being added to your issue (including your own), alternatively you can subscribe to an RSS feed.

Whist PyYAML appears to have regained maintenance[8], current developers are left in a sticky situation where many pieces of software rely on PyYAML without specifying a version[9]. Updating the code to merge / import much of the work done in ruamel.yaml such as support comments, YAML 1.2 format or other general improvements, come at the risk of breaking backwards compatibility and ergo many users' code breaking. This would be a good lesson for those not version-controlling their dependencies, but I'm not sure everyone also sees it this way.

I personally don't know what the right solution is, a new name like PyYAML2 that can onboard the work of ruamel.yaml while maintaining backwards compatibility for those using the set PyYAML? More support for ruamel.yaml?

This article is by no means an attack on the maintainers of PyYAML or ruamel.yaml. I do not envy them in the slightest and I sincerely hope more support is on its way. I also give full permission for this code to be copied / edited and used as documentation for handling yaml in python. At a minimum, I hope this article is at least useful to some developers out there wanting to write albeit limited and hacky commentary into yaml files and saves someone else from reverse engineering the source code through trial-and-error.

Resources:

StackOverflow: https://stackoverflow.com/questions/40704916/how-to-insert-a-comment-line-to-yaml-in-python-using-ruamel-yaml

MyBinder.org: https://mybinder.org/v2/gist/alexiswl/7fe1b3a90d86313c9785992bd0ce6e19/HEAD

ruamel.yaml documentation: https://yaml.readthedocs.io/en/latest/

CommentToken Objects Structures:

- CommentedMap:
  0: Comments set with `yaml_set_start_comment` method.
  1: Comments set from `yaml_set_comment_before_after_key` method with `before` parameter.
  2: Comments set from `yaml_set_eol_comment`
  3: Comments set from `yaml_set_comment_before_after_key` method with `after` parameter.
- CommentedSeq:
  0: Comments set from `yaml_set_eol_comment`

1: Comments set from `yaml_set_comment_before_after_key` method with `before` parameter.

2: Not Used

3: Not Used

References:

[1]: https://yaml.org/about.html
[2]: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-formats.html
[3]: https://www.commonwl.org/user_guide/02-1st-example/index.html
[4]: https://www.home-assistant.io/docs/configuration/yaml/
[5]: https://sourceforge.net/p/ruamel-yaml/tickets/400/
[6]: https://sourceforge.net/p/ruamel-yaml/tickets/402/
[7]: https://sourceforge.net/p/ruamel-yaml/tickets/404/
[8]: https://github.com/yaml/pyyaml/issues/31
[9]: https://github.com/yaml/pyyaml/issues/46