yaml and JSON

JSON

Javascript object notation, is a serialization language that converts complicated objects like Linked List to a streams of bytes that can then be transmitted over the wire or be stored in memory. This is required because you can't just transmit a "Linked List" over the internet. You would do it by first converting the object into a streams of bytes (serialization), then sending the bytes to the receiver, and when the receiver receives all the bytes they can then reconstruct the bytes into the "Linked List" (deserialization).

In JSON, keys are required to be in quotations.

YAML just like JSON, or rather is a superset of JSON is also a serialization language that is popular to be used as a configuration file.

Yet another markup language

YAML files uses spaces for formatting like in Python and not tabs.

To begin a YAML file you need to put 3 dashes.

Key values are string by default so do you don't need to put it in quotation around like in JSON, you can if you want to but is not done often. String values you can enclose in double quotes, single quotes, or even no quotes at all.

```
---
a: 1
b: hello
c: "world"
```

Values of each key can be string, floating point, boolean, integers, an array, or even a nested dictionary.

Different types for YAML

```
---
doe: a deer a female deer
```

```
pi: 3.1415926
numbers:
  - 1
  - 2
  - 3
nested:
  lol: 1
  lmafo: 2
  rofl: 3
```

To do an array, you just put values after a dash.

To do a nested dictionary you just tab and start with key value pair again in the same indentation.

You can even do an array of dictionary like below:

```
---
numbers:
  - lol: 3
    xd: 2
  - 2
  - 3
```

In this case, `numbers` is a list of three elements. The first element is a dictionary that have two key-value pair. The second element is 2, and last element is 3.

One line dictionary

Dictionary can be one line if you want to:

```
---
nested: {hello: 3, world: 4}
```

Anchors and alias

Anchor is defined using `&` symbol. It is used to improve reusability and modularity. You will essentially define a block of configuration that you would like to reuse by using the anchor symbol.

Then where you want to reference the values within the anchor you would use the `*` alias symbol. It is saying look at the anchor to find the values to use for this particular block of files.

You can anchor a scalar value:

```
---
toggle: &toggle true
do_i_skip: *toggle
```

In this case, `do_i_skip` will also reference to the scalar value `true`.

You can anchor a more complex value:

```
---
nested_dict: &reference
  - 1
  - 2
  - 3
  - config: on
    bookstack: on
    file_run: on
nested_dict2: *reference
```

`nested_dict2` will contain the same four element as `nested_dict` because it reference the anchor that was placed on `nested_dict`. The anchor will be all the values within the same tab.

A more complicated example:

```
---
nested_dict: &reference
  - 1
  - 2
  - 3
  - &inner_anchor
    config: on
    bookstack: on
    file_run: on
nested_dict2: *reference
nested_dict3: *inner_anchor
```

Now in this case `nested_dict2` is still as the same before, but `nested_dict3` will be reference to a dictionary of three element. `config: on, bookstack: on, file_run: on`

Override anchor

Say if you want to reference the anchor but want to make just a tiny little change of the inherited values you MUST use `<<: <alias>` syntax to do so.

For example, if in the same example, I would want to change `file_run` to be `off` for `nested_dict3` I have to do:

```
---
nested_dict: &reference
  - 1
  - 2
  - 3
  - &inner_anchor
    config: on
    bookstack: on
    file_run: on
nested_dict2: *reference
nested_dict3:
  <<: *inner_anchor
  file_run: off
```

If you reference and then override it, it will NOT work. YAML processor will complain.

Revision #1
Created 2023-07-23 00:53:43 UTC by Tamarine
Updated 2023-07-23 01:31:51 UTC by Tamarine