Quire Documentation

Release 0.4.1

Paul Colomiets

Jul 23, 2018

Contents

1 User Guide **3**
- 1.1 Yaml Cheat Sheet . . . 3
- 1.2 Quire Tricks . . . 4
- 1.3 Includes . . . 5
- 1.4 Merging Mappings . . . 7
- 1.5 Unpacking Sequences . . . 7

Rust-quire is a parser for yaml-based configuration files. Features:

- Structured Yaml configuration files
- Yaml anchors and merge (<<) keys
- Yaml syntax extensions[1]: includes, list unpacking
- Rich set of tools for end users (includes, yaml anchors, variables...)

Github | Crate

Contents:

[1] any configuration is still valid yaml, we use Yaml tags for as an extension mechanism

CHAPTER 1

User Guide

1.1 Yaml Cheat Sheet

Usually YAML structure is denoted by indentation.

Here is an example of Yaml mapping (or dictinary, or associated array), used to fill in some application's structure:

```
database-connection:
    host: localhost
    port: 1878
    encoding: utf-8
```

The top-level thing is almost always a mapping.

Sequences (or arrays, or lists) are denoted using – at the start of each line, for example:

```
urls:
- http://rust-lang.org
- http://rust-quire.readthedocs.io
```

Note: the **space is required** after both, the colon `:` used to denote a mapping and a dash – used to denote a sequence. This makes it possible to differentiate between urls written in the last example and a mapping example before. If you're unsure use quotes:

```
urls:
- "http://rust-lang.org"
- 'http://rust-quire.readthedocs.io'
```

Values may have arbitrarily complex structure:

```
databases:
  slaves:
  - host: slave1.example.com
    port: 1245
```

(continues on next page)

(continued from previous page)

```
  - host: slave2.example.com
    port: 1245
```

There is a short form of mappings and sequences, it's markup includes curly braces `{}` and square brackets `[]` respectively. When this form is used, indentation is no more significant, until last brace or bracket is closed. Example:

```
databases:
  slaves: [{host: slave1, port: 1245},
    {host: slave2, port: 1245}]
languages [ru, uk, us]
```

1.2 Quire Tricks

1.2.1 Underscore Names

In most cases names mapping keys starting with underscore are ignored. For example:

```
host: localhost
_port: 1778  # this is ignored
port: 2021
```

In this case `_port` field is ignored by a validator. It can be used to effectively comment out the piece of config (of arbitrary depth). But data might must be valid Yaml anyway, and may include ancors.

Often, this trick is used to anchor some things in a more comfortable way, for example this config:

```
listen:
- host: 127.0.0.1
  port: &port 1279
- host: 172.0.0.1
  port: *port
```

Could be rewritten as:

```
_port: &port 1278

listen:
- host: 127.0.0.1
  port: *port
- host: 172.0.0.1
  port: *port
```

Note two things:

- The latter example is more "symmetric" in some sense, you can reorder rows without rewriting anchors.
- Also it's convenient to move commonly changed parts to the top of the configuration file if config is big

1.2.2 Integers

Integers can be of base 10, just like everybody used to. It can also start with:

- `0x` to be interpreted as base 16

- `0o` meaning octal representation
- `0b` for bitmasks

Numbers can also be split into group to be easier to read using underscores: `1_024`, `16_777_216`.

1.2.3 Units

A lot of integer values in configuration files are quite big, e.g. should be expressed in megabytes or gigabytes. Instead of common case of making default units of megabytes or any other arbitrary choice, quire allows to specify order of magnitude units for every integer and floating point value. E.g:

```
int1: 1M
int2: 2k
int3: 2 ki
```

Results into the following, after parsing:

```
int1: 1000000
int2: 2000
int3: 2048
```

Note that there is a difference between prefixes for powers of 1024 and powers of the 1000.

The following table summarizes all units supported:

Unit	Value
k	1000
ki	1024
M	1000000
Mi	1048576
G	1000000000
Gi	1073741824

1.3 Includes

Currently rust-quire supports a single kind of include. Includes are expanded after parsing the yaml but config validation. It has the following consequences:

- Both include origin and included files are valid Yaml files on it's own.
- The included data is contained at the place where directive is (unlike many other configuration systems where inclusion usually occurs at the top level of the config), but you can include at the top level of the config too
- All anchors are local to the file, you can't reuse anchors from an included file.
- Include directives can be arbirarily nested (up to the memory limit)

It depends on how API is used, but usually file name of the include directive is expanded relative to a file that contains include (in fact relative to the name under which file is opened in case it symlinked into multiple places)

1.3.1 Include Another Yaml

The `!*Include` tag includes the contents of the file replaceing the node that contains tag. For example:

```
# config.yaml
items: !*Include items.yaml
```

```
# items.yaml
- apple
- cherry
- banana
```

Is equivalent of:

```
items:
- apple
- cherry
- banana
```

1.3.2 Include Sequences of Files

The `!*IncludeSeq` tag includes files matched by glob as a sequence:

```
items: !*IncludeSeq "fruits/*.yaml"
```

Can be parsed as:

```
items:
- apple
- banana
- cherry
```

This depends on the exact application, but usually files returned by *glob* are sorted in alphabetical order. All the power of globs is supported, so you can do:

```
items: !*IncludeSeq "files/**/*.yaml"
```

Another trick is merge multiple files, each with it's own set of keys into a single one (see *map-merge* below):

```
# file1.yaml
key1: 1
key2: 2
# file2.yaml
key3: 3
key4: 4
# main.yaml
<<: !*IncludeSeq "configs/*.yaml"
```

This results into the following config:

```
key1: 1
key2: 2
key3: 3
key4: 4
```

Note: merging is not recursive, i.e. top level keys are considered as a whole, even if they are dicts.

1.3.3 Include Mapping from Files

The `!*IncludeMap` tag works similarly to `!*IncludeSeq` but requires to mark part of a name used as a key. For example:

```
items: !*IncludeMap "fruits/(*).yaml"
```

Might result into the folowing:

```
items:
  apple: { color: orange }
  banana: { color: yellow }
  cherry: { color: red }
```

You can parenthesize any part as well as a whole path:

```
files: !*IncludeMap "(**/*.yaml)"
```

1.4 Merging Mappings

We use standard YAML way for merging mappings. It's achieved using << key and either mapping or a list of mappings for the value.

The most useful merging is with aliases. Example:

```
fruits: &fruits
  apple: yes
  banana: yes
food:
  bread: yes
  milk: yes
  <<: *fruits
```

Will be parsed as:

```
fruits:
  apple: yes
  banana: yes
food:
  bread: yes
  milk: yes
  apple: yes
  banana: yes
```

1.5 Unpacking Sequences

Similarly to map merging we have a method to join two sequences, for example:

```
_wild: &wild_animals
- tiger
- lion
_pets: &domestic_animals
- cat
```

(continues on next page)

(continued from previous page)

```
- dog

animals:
- !*Unpack [*wild_animals, *domestic_animals]
```

The key thing in the example is `!*Unpack` tag.

Note, you always need to have a two nested lists in, i.e. this is valid: `!*Unpack [[value]]`, but this `!*Unpack [value]` isn't. This is required in order to make tags in encompassed value work.