**Introduction to YAML Schemas and Tags**

By **[tinita](http://blogs.perl.org/users/tinita/)** on January 1, 2018 7:46 PM

I wrote this article as part of my [YAML::PP Grant report for December 2017](http://blogs.perl.org/users/tinita/2018/01/yamlpp-grant-report-december-2017.html).

This article will give you an overview how a YAML document is loaded into a data structure and how a processor decides if something is a string, a number, a boolean or something else.

If you look at JSON, the rules are quite obvious:

{  
    "string": "just a string",  
    "boolean": true,  
    "integer": 42,  
    "float": 3.14159,  
    "undefined": null,  
    "list": [1, 2, 3]  
}

In YAML, it depends on the YAML version, and on the Schema the Processor implements. This makes it more powerful, but also more complicated to implement and use.

Some processors implement YAML 1.1, some implement 1.2, and often there are things that aren't implemented correctly. When writing YAML, it's good to know the official specification to understand why some things are supported in one processor, but not in others.

This article covers:

* Standard Tags/Schema in YAML 1.2
* Type System in YAML 1.1
* How to write compatible YAML
* Other types of Schemas

As an example, note how boolean handling differs in YAML 1.1/1.2 and the 1.2 schemas.

**YAML 1.2**

A Schema is the connection between Tags in YAML and Classes / data types in the programming language. This is important for being able to load values as booleans, numbers or even objects.

The YAML 1.2 Specification lists three schemas, that are based on each other. Each one supports a number of tags, and a tag is associated with a matching rule and a data type.

A YAML processor can choose to implement one schema, or when implementing more or all of them, make them available via an option.

The minimalistic Schema is called Failsafe.

**Failsafe Schema**

[Failsafe](http://yaml.org/spec/1.2/spec.html#id2802346)

It has only three tags:

* !!str
* !!map
* !!seq

Actually !! is shorthand for !<tag:yaml.org,2002:...>:

* !!str == !<tag:yaml.org,2002:str>
* !!map == !<tag:yaml.org,2002:seq>
* !!seq == !<tag:yaml.org,2002:map>

The following YAML documents are equivalent:

---  
- !!str just a string  
- !!str true          *# also just a string*  
- !!map  
  a: 1  
  b: 2  
- !!seq  
  - a  
  - b  
  
  
---  
- just a string  
- true          *# also just a string*  
-  
  a: 1  
  b: 2  
-  
  - a  
  - b

That's because the standard tags are usually *implicit* and derived from the node content.

So, in most cases these tags are no-ops.

If you use !!map on a sequence, the Loader should return an error. Theoretically, with user defined schemas, there can be a use for !!map and !!seq, but you can assume for now they are only there for completeness.

As you can see (or actually can't), there's no Boolean in the Failsafe Schema.

**JSON Schema**

[JSON](http://yaml.org/spec/1.2/spec.html#id2803231)

The JSON Schema should be provided by a processor for maximum JSON compatibility.

* All Failsafe tags
* !!null: A null value. Match: null
* !!bool: Boolean. Match: true | false
* !!int: Integer. Match: 0 | -? [1-9] [0-9]\*
* !!float: Float. Match: -? ( 0 | [1-9] [0-9]\* ) ( \. [0-9]\* )? ( [eE] [-+]? [0-9]+ )?

Here we introduce the Boolean type, matched only by true and false.

In this schema, the !!str tag actually gets a meaning, although it's mostly theoretical:

- true          *# boolean*  
- "true"        *# string, because it's quoted*  
- !!str true    *# string, because of !!str*  
- !!bool "true" *# boolean, because of !!bool*

The current YAML Team considers this Schema as the best, so this will probably make its way into YAML 1.3 as a default.

When reviewing this article, [@mohawk2++](https://github.com/mohawk2) found a mistake with the regex for floats. The JSON [specification](http://json.org/) requires at least one digit after a dot, while the above regex does not, making 3. valid. We should fix this in YAML 1.3, and I made an [RFC](https://github.com/yaml/summit.yaml.io/wiki/RFC-025) for it.

**Core Schema**

[Core](http://yaml.org/spec/1.2/spec.html#id2804923)

The Core Schema has the same tags, but the matching rules match more:

* All JSON Tags, plus the following additional or overriden ones:
* !!null: null. Match: null | Null | NULL | ~ | empty scalar
* !!bool: Boolean. Match: true | True | TRUE | false | False | FALSE
* !!int: Integer Base 10. Match: [-+]? [0-9]+
* !!int: Octal. Match: 0o [0-7]+
* !!int: Hex. Match: 0x [0-9a-fA-F]+
* !!float: Float. Match: [-+]? ( \. [0-9]+ | [0-9]+ ( \. [0-9]\* )? ) ( [eE] [-+]? [0-9]+ )?
* !!float: Infinity. Match: [-+]? ( \.inf | \.Inf | \.INF )
* !!float: Not a Number. Match: \.nan | \.NaN | \.NAN

The Boolean rule accepts a little bit more. And the ~ for the null value is introduced, which you might know from YAML 1.1.

Note that in this Schema, also the empty scalar is resolved as null. In the JSON Schema, it was resolved as the empty string.

**YAML 1.1**

In YAML 1.1, there is no concept of a schema (at least the word is not mentioned), it's rather called Types, but it's quite similar.

Most of the Types are optional for YAML processors to implement. The optional types aren't really official, but many of them have been implemented in a handful of processors.

They are *not* part of YAML 1.2 anymore, but some Types have survived as part of a schema.

Here is how it works in YAML 1.1:

A Type describes a tagname and an associated datatype and matching rules.

There are basically three standard, mandatory Tags/Types, which you know from the YAML 1.2 Failsafe Schema:

* !!str
* !!map
* !!seq

Some optional types are:

* !!int [int](http://yaml.org/type/int.html)
* !!float [float](http://yaml.org/type/float.html)
* !!null [null](http://yaml.org/type/null.html)
* !!bool [bool](http://yaml.org/type/bool.html)
* !!binary [binary](http://yaml.org/type/binary.html)
* !!timestamp [timestamp](http://yaml.org/type/timestamp.html)

Let's look at the [Boolean definition](http://yaml.org/type/bool.html)

* Shorthand: !!bool
* Canonical: y|n
* Regexp:

y|Y|yes|Yes|YES|n|N|no|No|NO |true|True|TRUE|false|False|FALSE |on|On|ON|off|Off|OFF # Yeah, right, crazy

It was considered a bad decision to allow so many values for booleans, that's why in YAML 1.2 it was significantly reduced.

**How to write compatible YAML**

Unlike JSON, which is commonly used for data exchange between different systems, YAML is often used with a specific processor. Still, it's a good idea to write YAML as compatible as possible.

If you don't need the extra types the Core schema offers, and you follow the YAML 1.2 JSON Schema, your YAML should work for most YAML 1.1/1.2 processors.

For YAML 1.3, at this point there only exist RFCs, but we consider the JSON schema as a default. One reason for that is the compatibility with JSON itself. Note that YAML is actually not an *exact* superset of JSON, although this was a goal of 1.2.

**Booleans**

As you learned, only the plain scalars true and false are recognized as booleans in all versions/schemas (except Failsafe), at least if the Framework you are using did implement booleans.

**Null values**

If you are not sure, always use the plain string null. Some frameworks load the empty scalar as an empty string, and this is probably because the spec is not very clear about this.

For the same reason, don't use the tilde ~, as it's only provided by the Core Schema.

**Integers, Floats**

Also for these types, you should look at the YAML 1.2 JSON Schema to be compatible. +23 is not recognized as an integer (only in the Core schema); Also NaN, Inf, octal and hexadecimal aren't supported.

**Other types of Schemas**

**Language specific schemas**

A YAML Processor often also implements a language specific Schema.

Let's say we want to serialize a Dice object, that is a sequence of integers.

This is how serializing of generic objects looks like in several languages:

---  
*# Ruby Psych*  
dice: !ruby/Object:Dice [3, 6]  
  
---  
*# perl YAML::XS, YAML.pm, YAML::Syck (Dump and Load)*  
dice: !!perl/array:Dice [3, 6]  
  
---  
*# perl YAML.pm, YAML::Syck (Load)*  
dice: !perl/array:Dice [3, 6]  
  
---  
*# Pyyaml*  
dice: !!python/object/new:\_\_main\_\_.Dice  
  - !!python/tuple [3, 6]

The 1.2 Spec suggests a single ! for those tags. YAML 1.1 doesn't say anything about that. That's why we currently have this mixture. Some use !, some !!, some Dump with !! but also support ! when Loading.

We haven't decided yet what YAML 1.3 will recommend. Since this is language specific, at least processors in the same language should be compatible to each other.

**User defined schemas with local tags**

YAML also allows you to do fancy things like loading objects from specific local tags (a local tag starts with one !), or nodes matching a certain regular expression.

*# local tag !Dice -> Dice object*  
- !Dice [3,6]  
  
  *# matched from `digit 'd' digit` -> Dice object*  
- 3d6  
  
  *# local tag !template -> process string and replace template syntax with*  
  *# variables*  
- !template "The home directory is {{ env.HOME }}"

**User defined schemas with URI tags**

As mentioned above, the standard tags like !!str are shorthand for !<tag:yaml.org,2002:str>. You can also use your own Schema, for example !<tag:clarkevans.com,2002:invoice>. Because this is quite verbose, you can create your own shorthand for a schema:

%TAG !foo! tag:clarkevans.com,2002:  
---  
invoice1: !foo!invoice  
  ...  
invoice2: !<tag:clarkevans.com,2002:invoice>  
  ...

In fact, you can overwrite the standard schema like this:

%TAG !! tag:example.org,2017:  
---  
- !!int foo *# !<tag:example.org,2017:int>*

A tag shorthand is valid for the next document and not global for the complete YAML stream.

**YAML 1.3**

If you want to join the YAML Team, visit us on Freenode IRC. Currently we're hanging out in the *#yaml* and *#libyaml* channels.