TOML cheatsheet

This is a quick reference cheat sheet to the TOML format configuration file syntax.

Getting Started

Introduction

TOML is a minimal configuration file format that's easy to read due to obvious semantics.

- Document (toml.io)
- Learn X in Y minutes (learnxinyminutes.com)

Examples

```
bool = true
date = 2006-05-27T07:32:00Z
string = "hello"
number = 42
float = 3.14
scientificNotation = 1e+12
```

Comments

```
# A single line comment example

# block level comment example
# comment line 1
# comment line 2
# comment line 3
```

Integer

```
int1 = +42
int2 = 0
int3 = -21
integerRange = 64
```

Float

```
float2 = 3.1415
float4 = 5e+22
float7 = 6.626e-34
```

Boolean

```
bool1 = true
bool2 = false
boolMustBeLowercase = true
```

Datetime

```
date1 = 1989-05-27T07:32:00Z
date2 = 1989-05-26T15:32:00-07:00
date3 = 1989-05-27T07:32:00
date4 = 1989-05-27
time1 = 07:32:00
time2 = 00:32:00.999999
```

String

```
str1 = "I'm a string."
str2 = "You can \"quote\" me."
str3 = "Name\tJos\u00E9\nLoc\tSF."
```

See: Strings

Table

```
[owner]
name = "Tom Preston-Werner"
dob = 1979-05-27T07:32:00-08:00
```

See: Tables

Array

```
array1 = [1, 2, 3]
array2 = ["Commas", "are", "delimiter"]
array3 = [8001, 8001, 8002]
```

Friendly Array

```
array1 = [ "Don't mix", "different", "types" ]
array2 = [ [ 1.2, 2.4 ], ["all", 'strings', """are the same""", '''type'''] ]
array3 = [
  "Whitespace", "is",
  "ignored"
]
```

TOML Strings

Multiline String

```
multiLineString = """
Multi-line basic strings are surrounded
by three quotation marks on each side
and allow newlines.
"""
```

Literal String

```
path = 'C:\Users\nodejs\templates'
path2 = '\\User\admin$\system32'
quoted = 'Tom "Dubs" Preston-Werner'
regex = '<\i\c*\s*>'
```

Surrounded by single quotes. Escaping are not allowed.

MultiLine Literal String

```
re = '''\d{2} apps is t[wo]o many'''
lines = '''
The first newline is
trimmed in raw strings.
All other whitespace
is preserved.
'''
```

TOML Tables

Basic

```
[name]
foo = 1
bar = 2
```

`foo` and `bar` are keys in the table called `name`

Nested

```
[table1]
    foo = "bar"

[table1.nested_table]
    baz = "bat"
```

Array-like

```
[[comments]]
author = "Nate"
text = "Great Article!"

[[comments]]
author = "Anonymous"
text = "Love it!"
```

↓ Equivalent JSON

```
{
    "comments" : [
        {
            "author" : "Nate",
            "text" : "Great Article!"
```

Dot separated

```
[dog."tater.man"]
type = "pug"
```

↓ Equivalent JSON

```
{
```

Multi-nested

```
[foo.bar.baz]
bat = "hi"
```

↓ Equivalent JSON

```
{
```

```
    "tater.man": {
      "type": "pug"
    }
  }
}
```

```
      "bar" : {
        "baz" : {
          "bat" : "hi"
        }
      }
    }
}
```

```
    },
    {
      "author" : "Anonymous",
      "text" : "Love It!"
    }
  ]
}
```

Ignore whitespace

```
[a.b.c]          # this is best practice
[ d.e.f ]        # same as [d.e.f]
[ g .  h  .i ]   # same as [g.h.i]
[ j . "ʞ" .'l' ] # same as [j."ʞ".'l']
```

Inline Table

```
name = { first = "Tom", last = "Preston-Werner" }
point = { x = 1, y = 2 }
animal = { type.name = "pug" }
```

Related Cheatsheet

- **ES6** Cheatsheet Quick Reference
- **JSON** Cheatsheet Quick Reference
- **Kubernetes** Cheatsheet Quick Reference
- **YAML** Cheatsheet Quick Reference

Recent Cheatsheet

- **Google Search** Cheatshee Quick Reference
- **Kubernetes** Cheatsheet Quick Reference
- **ES6** Cheatsheet Quick Reference
- **ASCII Code** Cheatsheet Quick Reference

QuickRef.ME

Share quick reference and cheat sheet for developers.

中文版 #Notes