JS-YAML - YAML 1.2 parser / writer for JavaScript

Online Demo

This is an implementation of YAML, a human-friendly data serialization language. Started as PyYAML port, it was completely rewritten from scratch. Now it's very fast, and supports 1.2 spec.

Installation

YAML module for node.js

```
npm install js-yaml
```

CLI executable

If you want to inspect your YAML files from CLI, install js-yaml globally:

```
npm install -g js-yaml
```

Usage

```
usage: js-yaml [-h] [-v] [-c] [-t] file

Positional arguments:
  file           File with YAML
document(s)

Optional arguments:
  -h, --help     Show this help message
and exit.
  -v, --version  Show program's version
number and exit.
  -c, --compact  Display errors in
compact mode
  -t, --trace    Show stack trace on
error
```

API

Here we cover the most 'useful' methods. If you need advanced details (creating your own tags), see examples for more info.

```
const yaml = require('js-yaml');
const fs   = require('fs');

// Get document, or throw exception on
        error
try {
```

```
  const doc =
        yaml.load(fs.readFileSync('/
        home/ixti/example.yml',
        'utf8'));
  console.log(doc);
} catch (e) {
  console.log(e);
}
```

load (string [, options])

Parses `string` as single YAML document. Returns either a plain object, a string, a number, `null` or `undefined`, or throws `YAMLException` on error. By default, does not support regexps, functions and undefined.

options:

- `filename` *(default: null)* - string to be used as a file path in error/warning messages.
- `onWarning` *(default: null)* - function to call on warning messages. Loader will call this function with an instance of `YAMLException` for each warning.
- `schema` *(default:* `DEFAULT_SCHEMA`*)* - specifies a schema to use.
  - `FAILSAFE_SCHEMA` - only strings, arrays and plain objects: http://www.yaml.org/spec/1.2/spec.html#id2802346
  - `JSON_SCHEMA` - all JSON-supported types: http://www.yaml.org/spec/1.2/spec.html#id2803231

- `CORE_SCHEMA` - same as `JSON_SCHEMA`: http://www.yaml.org/spec/1.2/spec.html#id2804923
- `DEFAULT_SCHEMA` - all supported YAML types.

- `json` *(default: false)* - compatibility with JSON.parse behaviour. If true, then duplicate keys in a mapping will override values rather than throwing an error.

  NOTE: This function **does not** understand multi-document sources, it throws exception on those.

  NOTE: JS-YAML **does not** support schema-specific tag resolution restrictions. So, the JSON schema is not as strictly defined in the YAML specification. It allows numbers in any notation, use `Null` and `NULL` as `null`, etc. The core schema also has no such restrictions. It allows binary notation for integers.

loadAll (string [, iterator] [, options])

Same as `load()`, but understands multi-document sources. Applies `iterator` to each document if specified, or returns array of documents.

```
const yaml = require('js-yaml');

yaml.loadAll(data, function (doc) {
  console.log(doc);
});
```


Also, reading of properties on implicit block mapping keys is not supported yet. So, the following YAML document cannot be loaded.

```
&anchor foo:
  foo: bar
  *anchor: duplicate key
  baz: bat
  *anchor: duplicate key
```

js-yaml for enterprise

Available as part of the Tidelift Subscription

The maintainers of js-yaml and thousands of other packages are working with Tidelift to deliver commercial support and maintenance for the open source dependencies you use to build your applications. Save time, reduce risk, and improve code health, while paying the maintainers of the exact dependencies you use. Learn more.

```
!!float '3.14...'           # number
!!binary '...base64...'     # buffer
!!timestamp 'YYYY-...'      # date
!!omap [ ... ]              # array of
key-value pairs
!!pairs [ ... ]             # array or
array pairs
!!set { ... }               # array of
objects with given keys and null values
!!str '...'                 # string
!!seq [ ... ]               # array
!!map { ... }               # object
```

JavaScript-specific tags

See [js-yaml-js-types] for extra types.

Caveats

Note, that you use arrays or objects as key in JS-YAML. JS does not allow objects or arrays as keys, and stringifies (by calling `toString()` method) them at the moment of adding them.

```
---
? [ foo, bar ]
: - baz
? { foo: bar }
: - baz
  - baz

{ "foo,bar": ["baz"], "[object
        Object]": ["baz", "baz"] }
```


dump (object [, options])

Serializes `object` as a YAML document. Uses `DEFAULT_SCHEMA`, so it will throw an exception if you try to dump regexps or functions. However, you can disable exceptions by setting the `skipInvalid` option to `true`.

options:

- `indent` *(default: 2)* - indentation width to use (in spaces).
- `noArrayIndent` *(default: false)* - when true, will not add an indentation level to array elements
- `skipInvalid` *(default: false)* - do not throw on invalid types (like function in the safe schema) and skip pairs and single values with such types.
- `flowLevel` *(default: -1)* - specifies level of nesting, when to switch from block to flow style for collections. -1 means block style everwhere
- `styles` - “tag” => “style” map. Each tag may have own set of styles.
- `schema` *(default:* `DEFAULT_SCHEMA`*)* specifies a schema to use.
- `sortKeys` *(default:* `false`*)* - if `true`, sort keys when dumping YAML. If a function, use the function to sort the keys.
- `lineWidth` *(default:* `80`*)* - set max line width. Set `-1` for unlimited width.
- `noRefs` *(default:* `false`*)* - if `true`, don’t convert duplicate objects into references

- `noCompatMode` *(default:* `false`*)* - if `true` don't try to be compatible with older yaml versions. Currently: don't quote "yes", "no" and so on, as required for YAML 1.1
- `condenseFlow` *(default:* `false`*)* - if `true` flow sequences will be condensed, omitting the space between `a, b`. Eg. `'[a,b]'`, and omitting the space between `key: value` and quoting the key. Eg. `'{"a":b}'` Can be useful when using yaml for pretty URL query params as spaces are %-encoded.
- `quotingType` *(' or ", default: ')* - strings will be quoted using this quoting style. If you specify single quotes, double quotes will still be used for non-printable characters.
- `forceQuotes` *(default:* `false`*)* - if `true`, all non-key strings will be quoted even if they normally don't need to.
- `replacer` - callback `function (key, value)` called recursively on each key/value in source object (see `replacer` docs for `JSON.stringify`).

The following table show availlable styles (e.g. "canonical", "binary"…) available for each tag (.e.g. !!null, !!int …). Yaml output is shown on the right side after `=>` (default setting) or `->`:

```
!!null
  "canonical"   -> "~"
  "lowercase"   => "null"
  "uppercase"   -> "NULL"
  "camelcase"   -> "Null"

!!int
  "binary"      -> "0b1", "0b101010",
"0b11100011110l0"
  "octal"       -> "0o1", "0o52",
"0o16172"
  "decimal"     => "1", "42", "7290"
```



```
  "hexadecimal" -> "0x1", "0x2A",
"0x1C7A"

!!bool
  "lowercase"   => "true", "false"
  "uppercase"   -> "TRUE", "FALSE"
  "camelcase"   -> "True", "False"

!!float
  "lowercase"   => ".nan", '.inf'
  "uppercase"   -> ".NAN", '.INF'
  "camelcase"   -> ".NaN", '.Inf'
```

Example:

```
dump(object, {
  'styles': {
    '!!null': 'canonical' // dump null
        as ~
  },
  'sortKeys':
        true        // sort object keys
});
```

Supported YAML types

The list of standard YAML tags and corresponding JavaScript types. See also YAML tag discussion and YAML types repository.

```
!!null ''                   # null
!!bool 'yes'                # bool
!!int '3...'                # number
```