Package ‘zuyaml’

November 20, 2026

Title Parse and Emit YAML 1.2

Version 0.1.1

Description Converts between YAML 1.2 (<https://yaml.org/spec/1.2.2/>) and ordinary R objects using a bundled copy of the 'cyaml' C11 parser and emitter (<https://github.com/andrewmd5/cyaml>), so there is no system dependency and no runtime dependency beyond R itself. Ambiguous YAML features are handled strictly and predictably: duplicate keys are refused by default, the YAML 1.2 core schema is followed so that yes and no resolve as strings, and integers beyond double precision are preserved rather than silently rounded. A stream of documents and a sequence are different things, and the interface keeps them apart. Input size, nesting depth and the number of values materialised are all bounded, which makes the parser usable on untrusted input.

License MIT + file LICENSE

Copyright file inst/COPYRIGHTS

URL https://github.com/pedrobtz/zuyaml, https://pedrobtz.github.io/zuyaml/

BugReports https://github.com/pedrobtz/zuyaml/issues

Encoding UTF-8

Language en-GB

LinkingTo zufast

Suggests jsonlite, knitr, rmarkdown, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

NeedsCompilation yes

Author Pedro Baltazar [aut, cre, cph], Andrew Sampson [ctb, cph] (author of the bundled cyaml library)

Maintainer Pedro Baltazar <pedrobtz@gmail.com>

Repository CRAN

Date/Publication 2026-11-20 10:50:02 UTC

Contents

zuyaml-package 2
yaml_emit 3
yaml_parse 4
yaml_read 6
zuyaml_bigint 7
zuyaml_map 8

Index **9**

`zuyaml-package` *zuyaml: Parse and Emit 'YAML' 1.2*

Description

Converts between 'YAML' 1.2 (https://yaml.org/spec/1.2.2/) and ordinary R objects using a bundled copy of the 'cyaml' C11 parser and emitter (https://github.com/andrewmd5/cyaml), so there is no system dependency and no runtime dependency beyond R itself. Ambiguous 'YAML' features are handled strictly and predictably: duplicate keys are refused by default, the 'YAML' 1.2 core schema is followed so that yes and no resolve as strings, and integers beyond double precision are preserved rather than silently rounded. A stream of documents and a sequence are different things, and the interface keeps them apart. Input size, nesting depth and the number of values materialised are all bounded, which makes the parser usable on untrusted input.

Author(s)

Maintainer: Pedro Baltazar <pedrobtz@gmail.com> [copyright holder]

Authors:

- Pedro Baltazar <pedrobtz@gmail.com> [copyright holder]

Other contributors:

- Andrew Sampson (author of the bundled cyaml library) [contributor, copyright holder]

See Also

Useful links:

- https://github.com/pedrobtz/zuyaml
- https://pedrobtz.github.io/zuyaml/
- Report bugs at https://github.com/pedrobtz/zuyaml/issues

`yaml_emit` *Emit YAML*

Description

`yaml_emit()` converts one R object to a YAML document. `yaml_emit_all()` converts a list of objects to a YAML *stream*.

Usage

```
yaml_emit(
  x,
  indent = 2L,
  width = 80L,
  document_start = FALSE,
  document_end = FALSE
)

yaml_emit_all(
  x,
  indent = 2L,
  width = 80L,
  document_start = FALSE,
  document_end = FALSE
)
```

Arguments

`x`	An R object. `yaml_emit_all()` takes a list, one element per document.
`indent`	Spaces per indentation level, 1 to 255.
`width`	Line width before wrapping, 0 to 255. `0` disables wrapping.
`document_start`	Emit a leading `---`.
`document_end`	Emit a trailing `...`.

Details

Emission favours readable YAML over reproducing any particular source formatting. Comments, quoting style and flow-versus-block choices are not preserved; see `vignette("zuyaml")` for the conversions that do not round trip.

There is no depth limit: `x` is an R object, already in memory. A list nested deeper than the C stack allows raises R's `stackOverflowError`, and output longer than an R string can hold (2^31 - 1 bytes) is an error with code `"limit_size"`.

Value

A length-one UTF-8 character vector.

Examples

```
cat(yaml_emit(list(host = "localhost", port = 8080L)))
cat(yaml_emit_all(list(list(a = 1L), list(b = 2L))))

# Strings that look like other types stay strings.
cat(yaml_emit(list(version = "42")))
```

`yaml_parse` *Parse YAML*

Description

`yaml_parse()` parses a single YAML document. `yaml_parse_all()` parses a YAML *stream* and returns one element per document.

Usage

```
yaml_parse(
  x,
  simplify = FALSE,
  aliases = c("resolve", "error"),
  big_integers = c("bigint", "double", "error"),
  tags = c("ignore", "error"),
  duplicate_keys = FALSE,
  max_depth = 128L,
  max_size = 64 * 1024^2,
  max_nodes = 1e+06,
  path = NULL
)

yaml_parse_all(
  x,
  simplify = FALSE,
  aliases = c("resolve", "error"),
  big_integers = c("bigint", "double", "error"),
  tags = c("ignore", "error"),
  duplicate_keys = FALSE,
  max_depth = 128L,
  max_size = 64 * 1024^2,
  max_nodes = 1e+06,
  path = NULL
)
```

Arguments

`x`	A length-one character vector containing YAML, or a raw vector of UTF-8 YAML bytes.

`simplify` If `TRUE`, sequences whose elements are all scalars of the same type collapse to an atomic vector. The default is `FALSE`, so the shape of the result never depends on the contents of the document.

`aliases` How to treat YAML aliases. `"resolve"` (the default) replaces each alias with the value of its target; node identity is not preserved. `"error"` rejects any document containing an alias.

`big_integers` How to represent integers too large for an R numeric type to hold exactly (beyond 2^53). `"bigint"` (the default) returns a zuyaml_bigint character vector holding the decimal value, `"double"` is an explicit opt-in to lossy conversion, and `"error"` refuses the document. The package never loses integer precision silently.

`tags` How to treat an application tag such as `!duration`. Core schema tags (`!!str`, `!!int`, `!!float`, `!!bool`, `!!null`) always override resolution, so `!!str 12` is the string `"12"`. `"ignore"` (the default) converts a tagged value as though it were untagged; `"error"` rejects the document. Ignoring is the default because rejecting would refuse a great deal of ordinary YAML.

`duplicate_keys` If `FALSE` (the default), a mapping with duplicate keys is an error. If `TRUE`, duplicates become duplicate names in the resulting list.

`max_depth` Maximum nesting depth, or `0` for unlimited. It bounds parse time as well as depth, because the time to scan flow collections (`[...]`, `{...}`) grows with the square of their nesting depth, so set it to `0` only for trusted input. A document nested deeper than the C stack allows raises R's `stackOverflowError` rather than crashing.

`max_size` Maximum input size in bytes, or `0` for unlimited.

`max_nodes` Maximum number of R values materialised, or `0` for unlimited. This is the limit that bounds alias expansion: a billion-laughs document is small and shallow, so neither `max_size` nor `max_depth` constrains it. The default leaves roughly fifty times the headroom a very large document needs.

`path` Optional file path, used only to make error messages more informative. `yaml_read()` and its companions set it for you.

Details

Both functions parse the input as a stream. `yaml_parse()` then requires it to contain exactly one document, so trailing documents are never silently discarded:

documents	`yaml_parse()`	`yaml_parse_all()`
0	error	`list()`
1	the object	list of length 1
more	error	list of that length

A zero-document stream is an error rather than `NULL`, because `NULL` is the legitimate result of parsing a document whose content is `null`. Comment-only input is a zero-document stream.

Value

`yaml_parse()` returns an R object. `yaml_parse_all()` returns a list with one element per document.

Examples

```
yaml_parse("host: localhost\nport: 8080\ntls: true\n")

# The core schema, not YAML 1.1: `yes` is a string, and a quoted number
# stays a string.
str(yaml_parse("answer: yes\nversion: \"42\"\n"))

# A stream is not a sequence: one element per document.
yaml_parse_all("---\nfirst\n---\nsecond\n")
```

`yaml_read` *Read and write YAML files*

Description

`yaml_read()` reads one YAML document from a file; `yaml_read_all()` reads a stream. `yaml_write()` and `yaml_write_all()` are their counterparts.

Usage

```
yaml_read(path, ...)

yaml_read_all(path, ...)

yaml_write(x, path, ...)

yaml_write_all(x, path, ...)
```

Arguments

`path`	Path to a file.
`...`	Passed to `yaml_parse()` or `yaml_emit()`.
`x`	An R object. `yaml_write_all()` takes a list, one element per document.

Details

These work on bytes, not text: files are read and written with `readBin()` and `writeBin()`, so nothing depends on the session's locale, and the file path is carried into parse errors to make them locatable.

Value

`yaml_read()` returns an R object and `yaml_read_all()` a list of them. The writers return `path` invisibly.

Examples

```
path <- tempfile(fileext = ".yml")
yaml_write(list(host = "localhost", port = 8080L), path)
yaml_read(path)
unlink(path)
```

`zuyaml_bigint` *Integers too large for R's numeric types*

Description

A character vector holding the decimal representation of integers that R cannot store exactly: beyond 2^53 a double silently loses precision, and R has no native 64-bit integer scalar.

Usage

```
zuyaml_bigint(x)
```

Arguments

`x`	A character vector of decimal integers, or an object to coerce.

Details

`yaml_parse()` returns one of these when it meets such a value and `big_integers = "bigint"` (the default). It carries the *decimal* normalisation of the value, not the source text, so `0x1FFFFFFFFFFFFFFF` and its decimal spelling compare equal.

This is a marker for values whose precision must be preserved, not a big-integer arithmetic type: there is no arithmetic. Use `as.numeric()` to accept the precision loss deliberately, or a package such as `bit64` for real 64-bit arithmetic.

Value

`zuyaml_bigint()` returns a character vector of class `"zuyaml_bigint"`.

Examples

```
yaml_parse("9223372036854775807")
as.numeric(yaml_parse("9223372036854775807")) # lossy, on purpose
```

zuyaml_map *Mappings whose keys are not scalars*

Description

YAML mapping keys can be sequences or mappings, which cannot become R names without destroying structure. `yaml_parse()` represents such a mapping as a `zuyaml_map`: two parallel lists, `keys` and `values`.

Usage

```
## S3 method for class 'zuyaml_map'
print(x, ...)
```

Arguments

`x`	A `zuyaml_map`.
`...`	Ignored.

Details

This is parse-only. The vendored cyaml builder can only attach string keys, so a `zuyaml_map` cannot be emitted; see `vignette("zuyaml")` for the full list of conversions that do not round-trip.

Value

`print()` returns `x` invisibly.

Examples

```
yaml_parse("? [one, two]\n: value\n")
```

Index

∗ **package**
zuyaml-package, 2

print.zuyaml_map *(*zuyaml_map*)*, 8

readBin(), *6*

writeBin(), *6*

yaml_emit, 3
yaml_emit(), *6*
yaml_emit_all *(*yaml_emit*)*, 3
yaml_parse, 4
yaml_parse(), *6–8*
yaml_parse_all *(*yaml_parse*)*, 4
yaml_read, 6
yaml_read(), *5*
yaml_read_all *(*yaml_read*)*, 6
yaml_write *(*yaml_read*)*, 6
yaml_write_all *(*yaml_read*)*, 6

zuyaml *(*zuyaml-package*)*, 2
zuyaml-package, 2
zuyaml_bigint, *5*, 7
zuyaml_map, 8