Package 'yaml'

July 26, 2024

Type Package

Title Methods to Convert R Data to YAML and Back

Date 2024-07-22

Version 2.3.10

Suggests RUnit

Author Shawn P Garbett [aut], Jeremy Stephens [aut, cre], Kirill Simonov [aut], Yihui Xie [ctb], Zhuoer Dong [ctb], Hadley Wickham [ctb], Jeffrey Horner [ctb], reikoch [ctb], Will Beasley [ctb], Brendan O'Connor [ctb], Gregory R. Warnes [ctb], Michael Quinn [ctb], Zhian N. Kamvar [ctb], Charlie Gao [ctb]

Maintainer Shawn Garbett <shawn.garbett@vumc.org>

Description Implements the 'libyaml' 'YAML' 1.1 parser and emitter (<https://pyyaml.org/wiki/LibYAML>) for R.

URL https://github.com/vubiostat/r-yaml/

BugReports https://github.com/vubiostat/r-yaml/issues

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-07-26 15:10:02 UTC

Contents

`as.yaml` *Convert an R object into a YAML string*

Description

Convert an R object into a YAML string

Usage

```
as.yaml(x, line.sep = c("\n", "\r\n", "\r"), indent = 2, omap = FALSE,
        column.major = TRUE, unicode = TRUE, precision = getOption('digits'),
        indent.mapping.sequence = FALSE, handlers = NULL)
```

Arguments

`x`	the object to be converted
`line.sep`	the line separator character(s) to use
`indent`	the number of spaces to use for indenting
`omap`	determines whether or not to convert a list to a YAML omap; see Details
`column.major`	determines how to convert a data.frame; see Details
`unicode`	determines whether or not to allow unescaped unicode characters in output
`precision`	number of significant digits to use when formatting numeric values
`indent.mapping.sequence`	determines whether or not to indent sequences in mapping context
`handlers`	named list of custom handler functions for R objects; see Details

Details

If you set the `omap` option to TRUE, as.yaml will create ordered maps (or omaps) instead of normal maps.

The `column.major` option determines how a data frame is converted. If TRUE, the data frame is converted into a map of sequences where the name of each column is a key. If FALSE, the data frame is converted into a sequence of maps, where each element in the sequence is a row. You'll probably almost always want to leave this as TRUE (which is the default), because using `yaml.load` on the resulting string returns an object which is much more easily converted into a data frame via `as.data.frame`.

You can specify custom handler functions via the `handlers` argument. This argument must be a named list of functions, where the names are R object class names (i.e., 'numeric', 'data.frame', 'list', etc). The function(s) you provide will be passed one argument (the R object) and can return any R object. The returned object will be emitted normally.

Character vectors that have a class of 'verbatim' will not be quoted in the output YAML document except when the YAML specification requires it. This means that you cannot do anything that would result in an invalid YAML document, but you can emit strings that would otherwise be quoted. This is useful for changing how logical vectors are emitted (see below for example).

Character vectors that have an attribute of 'quoted' will be wrapped in double quotes (see below for example).

You can specify YAML tags for R objects by setting the 'tag' attribute to a character vector of length 1. If you set a tag for a vector, the tag will be applied to the YAML sequence as a whole, unless the vector has only 1 element. If you wish to tag individual elements, you must use a list of 1-length vectors, each with a tag attribute. Likewise, if you set a tag for an object that would be emitted as a YAML mapping (like a data frame or a named list), it will be applied to the mapping as a whole. Tags can be used in conjunction with YAML deserialization functions like `yaml.load` via custom handlers, however, if you set an internal tag on an incompatible data type (like "!seq 1.0"), errors will occur when you try to deserialize the document.

Value

Returns a YAML string which can be loaded using `yaml.load` or copied into a file for external use.

Author(s)

Jeremy Stephens <jeremy.f.stephens@vumc.org>

References

YAML: http://yaml.org

YAML omap type: http://yaml.org/type/omap.html

See Also

`yaml.load`

Examples

```
as.yaml(1:10)
as.yaml(list(foo=1:10, bar=c("test1", "test2")))
as.yaml(list(foo=1:10, bar=c("test1", "test2")), indent=3)
as.yaml(list(foo=1:10, bar=c("test1", "test2")), indent.mapping.sequence=TRUE)
as.yaml(data.frame(a=1:10, b=letters[1:10], c=11:20))
as.yaml(list(a=1:2, b=3:4), omap=TRUE)
as.yaml("multi\nline\nstring")
as.yaml(function(x) x + 1)
as.yaml(list(foo=list(list(x = 1, y = 2), list(x = 3, y = 4))))

# custom handler
as.yaml(Sys.time(), handlers = list(
  POSIXct = function(x) format(x, "%Y-%m-%d")
))

# custom handler with verbatim output to change how logical vectors are
# emitted
as.yaml(c(TRUE, FALSE), handlers = list(
  logical = verbatim_logical))

# force quotes around a string
```

```
port_def <- "80:80"
attr(port_def, "quoted") <- TRUE
x <- list(ports = list(port_def))
as.yaml(x)

# custom tag for scalar
x <- "thing"
attr(x, "tag") <- "!thing"
as.yaml(x)

# custom tag for sequence
x <- 1:10
attr(x, "tag") <- "!thing"
as.yaml(x)

# custom tag for mapping
x <- data.frame(a = letters[1:5], b = letters[6:10])
attr(x, "tag") <- "!thing"
as.yaml(x)

# custom tag for each element in a list
x <- list(1, 2, 3)
attr(x[[1]], "tag") <- "!a"
attr(x[[2]], "tag") <- "!b"
attr(x[[3]], "tag") <- "!c"
as.yaml(x)
```

`read_yaml` *Read a YAML file*

Description

Read a YAML document from a file and create an R object from it

Usage

```
read_yaml(file, fileEncoding = "UTF-8", text, error.label, readLines.warn=TRUE, ...)
```

Arguments

`file`	either a character string naming a file or a connection open for writing
`fileEncoding`	character string: if non-empty declares the encoding used on a file (not a connection) so the character data can be re-encoded. See `file`.
`text`	character string: if `file` is not supplied and this is, then data are read from the value of `text` via a text connection. Notice that a literal string can be used to include (small) data sets within R code.
`error.label`	a label to prepend to error messages (see Details).
`readLines.warn`	logical (default:TRUE) suppress warnings from readLines used inside read_yaml
`...`	arguments to pass to `yaml.load`

Details

This function is a convenient wrapper for `yaml.load` and is a nicer alternative to `yaml.load_file`.

You can specify a label to be prepended to error messages via the `error.label` argument. If `error.label` is missing, `read_yaml` will make an educated guess for the value of `error.label` by either using the specified filename (when `file` is a character vector) or using the description of the supplied connection object (via the `summary` function). If `text` is used, the default value of `error.label` will be `NULL`.

Value

If the root YAML object is a map, a named list or list with an attribute of 'keys' is returned. If the root object is a sequence, a list or vector is returned, depending on the contents of the sequence. A vector of length 1 is returned for single objects.

Author(s)

Jeremy Stephens <jeremy.f.stephens@vumc.org>

References

YAML: http://yaml.org

libyaml: https://pyyaml.org/wiki/LibYAML

See Also

`yaml.load`, `write_yaml`, `yaml.load_file`

Examples

```
## Not run:
  # reading from a file connection
  filename <- tempfile()
  cat("test: data\n", file = filename)
  con <- file(filename, "r")
  read_yaml(con)
  close(con)

  # using a filename to specify input file
  read_yaml(filename)

## End(Not run)

  # reading from a character vector
  read_yaml(text="- hey\n- hi\n- hello")
```

`verbatim_logical` *Alternative logical handler*

Description

A yaml handler function that causes logical vectors to emit `true`/`false` instead of `yes`/`no` values.

Usage

```
verbatim_logical(x)
```

Arguments

`x`	logical vector to convert to `true`/`false`.

Details

Pass this function to `as.yaml()` as part of the `handler` argument list like `list(logical = verbatim_logical)`.

Value

Returns a vector of strings of either `true` or `false` of class `verbatim`.

Author(s)

Charles Dupont and James Goldie (jimjam-slam)

See Also

`as.yaml`

Examples

```
vector <- c(TRUE, FALSE, TRUE)

as.yaml(vector, handlers=list(logical=verbatim_logical))
```

write_yaml *Write a YAML file*

Description

Write the YAML representation of an R object to a file

Usage

```
write_yaml(x, file, fileEncoding = "UTF-8", ...)
```

Arguments

x	the object to be converted
file	either a character string naming a file or a connection open for writing
fileEncoding	character string: if non-empty declares the encoding to be used on a file (not a connection) so the character data can be re-encoded as they are written. See file.
...	arguments to as.yaml

Details

If file is a non-open connection, an attempt is made to open it and then close it after use.

This function is a convenient wrapper around as.yaml.

Author(s)

Jeremy Stephens <jeremy.f.stephens@vumc.org>

See Also

as.yaml, read_yaml, yaml.load_file

Examples

```
## Not run:
  # writing to a file connection
  filename <- tempfile()
  con <- file(filename, "w")
  write_yaml(data.frame(a=1:10, b=letters[1:10], c=11:20), con)
  close(con)

  # using a filename to specify output file
  write_yaml(data.frame(a=1:10, b=letters[1:10], c=11:20), filename)

## End(Not run)
```

yaml.load *Convert a YAML string into R objects*

Description

Parse a YAML string and return R objects.

Usage

```
yaml.load(string, as.named.list = TRUE, handlers = NULL, error.label = NULL,
          eval.expr = getOption("yaml.eval.expr", FALSE),
          merge.precedence = c("order", "override"), merge.warning = FALSE)
yaml.load_file(input, error.label, readLines.warn=TRUE, ...)
```

Arguments

string	the YAML string to be parsed
as.named.list	whether or not to return a named list for maps (TRUE by default)
handlers	named list of custom handler functions for YAML types (see Details)
input	a filename or connection; if input is a filename, that file must be encoded in UTF-8
error.label	a label to prepend to error messages (see Details)
eval.expr	whether or not to evaluate expressions found in the YAML document (see Details)
merge.precedence	behavior of precedence during map merges (see Details)
merge.warning	whether or not to warn about ignored key/value pairs during map merges
readLines.warn	logical (default:TRUE) suppress warnings from readLines used inside read_yaml
...	arguments to pass to yaml.load

Details

Use yaml.load to load a YAML string. For files and connections, use yaml.load_file, which calls yaml.load with the contents of the specified file or connection.

Sequences of uniform data (e.g. a sequence of integers) are converted into vectors. If the sequence is not uniform, it's returned as a list. Maps are converted into named lists by default, and all the keys in the map are converted to strings. If you don't want the keys to be coerced into strings, set as.named.list to FALSE. When it's FALSE, a list will be returned with an additional attribute named 'keys', which is a list of the un-coerced keys in the map (in the same order as the main list).

You can specify custom handler functions via the handlers argument. This argument must be a named list of functions, where the names are the YAML types (i.e., 'int', 'float', 'seq', etc). The functions you provide will be passed one argument. Custom handler functions for string types (all types except sequence and map) will receive a character vector of length 1. Custom sequence functions will be passed a list of objects. Custom map functions will be passed the object that the

internal map handler creates, which is either a named list or a list with a 'keys' attribute (depending on `as.named.list`). ALL functions you provide must return an object. See the examples for custom handler use.

You can specify a label to be prepended to error messages via the `error.label` argument. When using `yaml.load_file`, you can either set the `error.label` argument explicitly or leave it missing. If missing, `yaml.load_file` will make an educated guess for the value of `error.label` by either using the specified filename (when `input` is a character vector) or using the description of the supplied connection object (via the `summary` function). You can explicity set `error.label` to `NULL` if you don't want to use this functionality.

There is a built-in handler that will evaluate expressions that are tagged with the '!expr' tag. Currently this handler is disabled by default for security reasons. If a '!expr' tag exists and this is set to FALSE a warning will occur. Alternately, you can set the option named 'yaml.eval.expr' via the `options` function to turn on evaluation.

The `merge.precedence` parameter controls how merge keys are handled. The YAML merge key specification is not specific about how key/value conflicts are resolved during map merges. As a result, various YAML library implementations vary in merge key behavior (notably Python and Ruby). This package's default behavior (when `merge.precedence` is 'order') is to give precedence to key/value pairs that appear first. If you set `merge.precedence` to 'override', natural map key/value pairs will override any duplicate keys found in merged maps, regardless of order. This is the default behavior in Python's YAML library.

This function uses the YAML parser provided by libyaml, which conforms to the YAML 1.1 specification.

Value

If the root YAML object is a map, a named list or list with an attribute of 'keys' is returned. If the root object is a sequence, a list or vector is returned, depending on the contents of the sequence. A vector of length 1 is returned for single objects.

Author(s)

Jeremy Stephens <jeremy.f.stephens@vumc.org>

References

YAML: http://yaml.org

libyaml: https://pyyaml.org/wiki/LibYAML

YAML merge specification: http://yaml.org/type/merge.html

See Also

`as.yaml`

Examples

```
yaml.load("- hey\n- hi\n- hello")
yaml.load("foo: 123\nbar: 456")
yaml.load("- foo\n- bar\n- 3.14")
```

```
  yaml.load("foo: bar\n123: 456", as.named.list = FALSE)

## Not run:
  # reading from a file (uses readLines internally)
  filename <- tempfile()
  cat("foo: 123", file=filename, sep="\n")
  yaml.load_file(filename)

## End(Not run)

  # custom scalar handler
  my.float.handler <- function(x) { as.numeric(x) + 123 }
  yaml.load("123.456", handlers=list("float#fix"=my.float.handler))

  # custom sequence handler
  yaml.load("- 1\n- 2\n- 3", handlers=list(seq=function(x) { as.integer(x) + 3 }))

  # custom map handler
  yaml.load("foo: 123", handlers=list(map=function(x) { x$foo <- x$foo + 123; x }))

  # handling custom types
  yaml.load("!sqrt 555", handlers=list(sqrt=function(x) { sqrt(as.integer(x)) }))
  yaml.load("!foo\n- 1\n- 2", handlers=list(foo=function(x) { as.integer(x) + 1 }))
  yaml.load("!bar\none: 1\ntwo: 2", handlers=list(bar=function(x) { x$one <- "one"; x }))

  # loading R expressions
  # NOTE: this will not be done by default in the near future
  doc <- yaml.load("inc: !expr function(x) x + 1", eval.expr=TRUE)
  doc$inc(1)

  # adding a label to error messages
  try(yaml.load("*", error.label = "foo"))
```

Index