

Package ‘frontmatter’

May 8, 2026

Title Parse Front Matter from Documents

Version 0.2.0

Description Extracts and parses structured metadata ('YAML' or 'TOML') from the beginning of text documents. Front matter is a common pattern in 'Quarto' documents, 'R Markdown' documents, static site generators, documentation systems, content management tools and even 'Python' and 'R' scripts where metadata is placed at the top of a document, separated from the main content by delimiter fences.

License MIT + file LICENSE

URL <https://github.com/posit-dev/frontmatter>,
<https://posit-dev.github.io/frontmatter/>

BugReports <https://github.com/posit-dev/frontmatter/issues>

Imports cpp11, rlang, tomleditor, yaml12

Suggests testthat (>= 3.0.0), withr, yaml

LinkingTo cpp11

Config/Needs/website brand.yml

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation yes

Author Garrick Aden-Buie [aut, cre] (ORCID:
<<https://orcid.org/0000-0002-7111-0077>>),
Posit Software, PBC [cph, fnd] (ROR: <<https://ror.org/03wc8by49>>)

Maintainer Garrick Aden-Buie <garrick@posit.co>

Repository CRAN

Date/Publication 2026-02-09 19:50:02 UTC

Contents

format_front_matter	2
parse_front_matter	5

format_front_matter *Format and Write YAML or TOML Front Matter*

Description

Serialize R data as YAML or TOML front matter and combine it with document content. `format_front_matter()` returns the formatted document as a string, while `write_front_matter()` writes it to a file or prints to the console. These functions are the inverse of `parse_front_matter()` and `read_front_matter()`.

Usage

```
format_front_matter(
  x,
  delimiter = "yaml",
  format = "auto",
  format_yaml = NULL,
  format_toml = NULL
)
```

```
write_front_matter(
  x,
  path = NULL,
  delimiter = "yaml",
  ...,
  format = "auto",
  format_yaml = NULL,
  format_toml = NULL
)
```

Arguments

- | | |
|---|---|
| <code>x</code> | A list with data and body elements, typically as returned by <code>parse_front_matter()</code> or <code>read_front_matter()</code> . The data element contains the metadata to serialize (can be NULL to write body only), and body contains the document content (can be NULL or empty). |
| <code>delimiter</code> | A character string specifying the fence style, or a character vector for custom delimiters. See Delimiter Formats for available options. |
| <code>format</code> | The serialization format: "auto" (detect from delimiter), "yaml", or "toml". Usually auto-detection works well. |
| <code>format_yaml</code> , <code>format_toml</code> | Custom formatter functions, or NULL to use defaults. Each function should accept an R object and return a character string. |
| <code>path</code> | File path to write to, or NULL to print to the console |
| <code>...</code> | Additional arguments passed to <code>writeBin()</code> when writing to a file (e.g., <code>useBytes</code>). |

Value

- `format_front_matter()`: A character string containing the formatted document with front matter.
- `write_front_matter()`: Called for its side effect; returns NULL invisibly.

Functions

- `format_front_matter()`: Format front matter as a string
- `write_front_matter()`: Write front matter to a file or console

Delimiter Formats

The `delimiter` argument controls the fence style used to wrap the front matter. You can use these built-in shortcuts:

Shortcut	Format	Opening	Closing	Use Case
"yaml"	YAML	-	-	Markdown, R Markdown, Quarto
"toml"	TOML	+++	+++	Hugo, some static site generators
"yaml_comment"	YAML	# -	# -	R scripts, Python scripts
"toml_comment"	TOML	# +++	# +++	R scripts, Python scripts
"yaml_roxy"	YAML	#' -	#' -	Roxygen2 documentation
"toml_roxy"	TOML	#' +++	#' +++	Roxygen2 documentation
"toml_pep723"	TOML	# /// script	# ///	Python PEP 723 inline metadata

For custom delimiters, pass a character vector of length 1, 2, or 3:

- **Length 1**: Used as both opener and closer, with no line prefix
- **Length 2**: `c(opener, prefix)` where opener is also used as closer
- **Length 3**: `c(opener, prefix, closer)` for full control

Custom Formatters

By default, the package uses `yaml12::format_yaml()` for YAML and `tomledit::to_toml()` for TOML. You can provide custom formatter functions via `format_yaml` and `format_toml` to override these defaults.

Custom formatters must accept an R object and return a character string containing the serialized content.

YAML Specification Version

The default YAML formatter uses YAML 1.2 via `yaml12::format_yaml()`. To use YAML 1.1 formatting instead (via `yaml::as_yaml()`), set either:

- The R option `frontmatter.serialize_yaml.spec` to "1.1"
- The environment variable `FRONTMATTER_SERIALIZE_YAML_SPEC` to "1.1"

The option takes precedence over the environment variable. Valid values are "1.1" and "1.2" (the default).

Roundtrip Support

Documents formatted with these functions can be read back with `parse_front_matter()` or `read_front_matter()`. For comment-prefixed formats (like `yaml_comment` or `yaml_roxy`), a separator line is automatically inserted between the closing fence and the body when the body starts with the same comment prefix, ensuring clean roundtrip behavior.

See Also

`parse_front_matter()` and `read_front_matter()` for the inverse operations.

Examples

```
# Create a document with YAML front matter
doc <- list(
  data = list(title = "My Document", author = "Jane Doe"),
  body = "Document content goes here."
)

# Format as a string
format_front_matter(doc)

# Write to a file
tmp <- tempfile(fileext = ".md")
write_front_matter(doc, tmp)
readLines(tmp)

# Print to console (when path is NULL)
write_front_matter(doc)

# Use TOML format
format_front_matter(doc, delimiter = "toml")

# Use comment-wrapped format for R scripts
r_script <- list(
  data = list(title = "Analysis Script"),
  body = "# Load libraries\nlibrary(dplyr)"
)
format_front_matter(r_script, delimiter = "yaml_comment")

# Roundtrip example: read, modify, write
original <- "---
title: Original
---
Content here"

doc <- parse_front_matter(original)
doc$data$title <- "Modified"
format_front_matter(doc)
```

parse_front_matter *Parse YAML or TOML Front Matter*

Description

Extract and parse YAML or TOML front matter from a file or a text string. Front matter is structured metadata at the beginning of a document, delimited by fences (--- for YAML, +++ for TOML). `parse_front_matter()` processes a character string, while `read_front_matter()` reads from a file. Both functions return a list with the parsed front matter and the document body.

Usage

```
parse_front_matter(text, parse_yaml = NULL, parse_toml = NULL)
```

```
read_front_matter(path, parse_yaml = NULL, parse_toml = NULL)
```

Arguments

text	A character string or vector containing the document text. If a vector with multiple elements, they are joined with newlines (as from <code>readLines()</code>).
parse_yaml, parse_toml	A function that takes a string and returns a parsed R object, or NULL to use the default parser. Use <code>identity</code> to return the raw string without parsing.
path	A character string specifying the path to a file. The file is assumed to be UTF-8 encoded. A UTF-8 BOM (byte order mark) at the start of the file is automatically stripped if present.

Value

A named list with two elements:

- `data`: The parsed front matter as an R object, or NULL if no valid front matter was found.
- `body`: The document content after the front matter, with leading empty lines removed. If no front matter is found, this is the original text.

Functions

- `parse_front_matter()`: Parse front matter from text
- `read_front_matter()`: Parse front matter from a file.

Custom Parsers

By default, the package uses `yaml12::parse_yaml()` for YAML and `tomledit::parse_toml()` for TOML. You can provide custom parser functions via `parse_yaml` and `parse_toml` to override these defaults.

Use `identity` to return the raw YAML or TOML string without parsing.

YAML Specification Version

The default YAML parser uses YAML 1.2 via `yaml12::parse_yaml()`. To use YAML 1.1 parsing instead (via `yaml::yaml.load()`), set either:

- The R option `frontmatter.parse_yaml.spec` to `"1.1"`
- The environment variable `FRONTMATTER_PARSE_YAML_SPEC` to `"1.1"`

The option takes precedence over the environment variable. Valid values are `"1.1"` and `"1.2"` (the default).

YAML 1.1 differs from YAML 1.2 in several ways, most notably in how it handles boolean values (e.g., `yes/no` are booleans in 1.1 but strings in 1.2).

Examples

```
# Parse YAML front matter
text <- "---
title: My Document
date: 2024-01-01
---
Document content here"

result <- parse_front_matter(text)
result$data$title # "My Document"
result$body      # "Document content here"

# Parse TOML front matter
text <- "+++
title = 'My Document'
date = 2024-01-01
+++
Document content"

result <- parse_front_matter(text)

# Get raw YAML without parsing
result <- parse_front_matter(text, parse_yaml = identity)

# Use a custom parser that adds metadata
result <- parse_front_matter(
  text,
  parse_yaml = function(x) {
    data <- yaml12::parse_yaml(x)
    data$parsed_at <- Sys.time()
    data
  }
)

# Or read from a file
tmpfile <- tempfile(fileext = ".md")
writeLines(text, tmpfile)
```

parse_front_matter

7

`read_front_matter(tmpfile)`

Index

`format_front_matter`, 2

`parse_front_matter`, 5
`parse_front_matter()`, 2, 4

`read_front_matter (parse_front_matter)`,
5
`read_front_matter()`, 2, 4

`tomledit::parse_toml()`, 5
`tomledit::to_toml()`, 3

`write_front_matter`
(`format_front_matter`), 2
`writeBin()`, 2

`yaml12::format_yaml()`, 3
`yaml12::parse_yaml()`, 5, 6
`yaml::as.yaml()`, 3
`yaml::yaml.load()`, 6