Package: frontmatter (via r-universe)

July 3, 2026

Title Parse Front Matter from Documents

Version 0.3.0

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, tomledit, yaml12

Suggests testthat (>= 3.0.0), withr, yaml

LinkingTo cpp11

Config/Needs/website brand.yml

Config/testthat/edition 3

Encoding UTF-8

Config/roxygen2/version 8.0.0

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 https://cran.r-universe.dev

Date/Publication 2026-07-03 03:10:02 UTC

RemoteUrl https://github.com/cran/frontmatter

RemoteRef HEAD

RemoteSha 03a720a969df2e3896b00fc79c0aca0aa5181d46

Contents

`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 = NULL,
  format = "auto",
  format_yaml = NULL,
  format_toml = NULL
)

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

Arguments

`x`	A list with `data` and `body` elements, typically as returned by `parse_front_matter()` or `read_front_matter()`. 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).
`delimiter`	A character string specifying the fence style, or a character vector for custom delimiters. See **Delimiter Formats** for available options. When `NULL` (the default), the delimiter is inferred automatically: if `x` was returned by `parse_front_matter()` or `read_front_matter()`, the original fence style is preserved; otherwise `write_front_matter()` falls back to the file extension of `path`, and finally to `"yaml"`.

`format` The serialization format: `"auto"` (detect from delimiter), `"yaml"`, or `"toml"`. Usually auto-detection works well.

`format_yaml`, `format_toml` Custom formatter functions, or `NULL` to use defaults. Each function should accept an R object and return a character string.

`path` File path to write to, or `NULL` to print to the console

`...` Additional arguments passed to `writeBin()` when writing to a file (e.g., `useBytes`).

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
`"yaml_sql_line"`	YAML	`-- ---`	`-- ---`	SQL scripts (line comments)
`"toml_sql_line"`	TOML	`-- +++`	`-- +++`	SQL scripts (line comments)
`"yaml_sql_block_compact"`	YAML	`/* ---`	`--- */`	SQL scripts (block comments)
`"toml_sql_block_compact"`	TOML	`/* +++`	`+++ */`	SQL scripts (block comments)
`"yaml_sql_block_expanded"`	YAML	`/* + ---`	`--- + */`	SQL scripts (block comments)
`"toml_sql_block_expanded"`	TOML	`/* + +++`	`+++ + */`	SQL scripts (block comments)

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`, `yaml_roxy`, or `yaml_sql_line`), 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.

When `delimiter` is `NULL` (the default), the delimiter is inferred automatically, making roundtrips seamless:

- `format_front_matter()` uses the `fence_type` attribute preserved by `parse_front_matter()` and `read_front_matter()`, so the output uses the same fence style as the original document.
- `write_front_matter()` additionally falls back to a built-in extension-to-delimiter map when the input has no `fence_type` attribute:

Extension	Default delimiter
`.sql`	`"yaml_sql_block_compact"`
`.py`	`"toml_pep723"`
`.R`	`"yaml_roxy"`
`.md`, `.qmd`, `.Rmd`	`"yaml"`

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 (delimiter inferred from fence_type attr, falls back to yaml)
format_front_matter(doc)

# Write to a file (delimiter inferred from .md extension -> yaml)
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 explicitly
format_front_matter(doc, delimiter = "toml")

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

# Write to an R file: delimiter inferred from .R extension -> yaml_roxy
tmp_r <- tempfile(fileext = ".R")
write_front_matter(r_script, tmp_r)
readLines(tmp_r)

# Roundtrip: delimiter is automatically preserved from the original format
original <- "# ---
# title: Original
# ---
# R code here"

doc <- parse_front_matter(original)
doc$data$title <- "Modified"
format_front_matter(doc) # uses yaml_comment, matching the source
```

`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). Comment-wrapped formats (`#`, `#'`, `--`) and SQL block comments (`/* */`) are also supported. `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 `readLines()`).
`parse_yaml`, `parse_toml`	A function that takes a string and returns a parsed R object, or `NULL` to use the default parser. Use `identity` 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)

read_front_matter(tmpfile)
```

Index