Commits	Contributor
1	caesar
1	ianstormtaylor
1	qm3ster
1	zachwhaley

Author

Jon Schlinkert

- LinkedIn Profile
- GitHub Profile
- Twitter Profile

License

This file was generated by verb-generate-readme, v0.6.0, on April 01, 2018.

gray-matter

Parse front-matter from a string or file. Fast, reliable and easy to use. Parses YAML front matter by default, but also has support for YAML, JSON, TOML or Coffee Front-Matter, with options to set custom delimiters. Used by metalsmith, assemble, verb and many other projects.

Please consider following this project's author, Jon Schlinkert, and consider starring the project to show your :heart: and support.

Install

Install with npm:

Heads up!

Please see the changelog to learn about breaking changes that were made in v3.0.

What does this do?

Run this example

Add the HTML in the following example to `example.html`, then add the following code to `example.js` and run `$ node example` (without the `$`):

```
const fs = require('fs');
const matter = require('gray-matter');
const str =
        fs.readFileSync('example.html',
        'utf8');
console.log(matter(str));
```

Converts a string with front-matter, like this:

```
---
title: Hello
slug: home
---
<h1>Hello world!</h1>
```

Into an object like this:


```
$ npm install -g verbose/verb#dev verb-
        generate-readme && verb
```

Related projects

You might also be interested in these projects:

- assemble: Get the rocks out of your socks! Assemble makes you fast at creating web projects… more | homepage
- metalsmith: An extremely simple, pluggable static site generator. | homepage
- verb: Documentation generator for GitHub projects. Verb is extremely powerful, easy to use, and is used… more | homepage
- gray-matter-loader: A webpack loader for gray-matter. homepage

Contributors

Commits	Contributor
174	jonschlinkert
7	RobLoach
5	heymind
4	doowb
3	aljopro
2	reccanti
2	onokumus
2	moozzyk
1	Ajedi32

options.delims

Decrecated, please use options.delimiters instead.

options.parsers

Decrecated, please use options.engines instead.

About

Contributing

Pull requests and stars are always welcome. For bugs and feature requests, please create an issue.

Running Tests

Running and reviewing unit tests is a great way to get familiarized with a library and its API. You can install dependencies and run tests with the following command:

```
$ npm install && npm test
```

Building docs

(This project's readme.md is generated by verb, please don't edit the readme directly. Any changes to the readme must be made in the .verb.md readme template.)

To generate the readme, run the following command:


```
{
  content: '<h1>Hello world!</h1>',
  data: {
    title: 'Hello',
    slug: 'home'
  }
}
```

Why use gray-matter?

- **simple**: main function takes a string and returns an object
- **accurate**: better at catching and handling edge cases than front-matter parsers that rely on regex for parsing
- **fast**: faster than other front-matter parsers that use regex for parsing
- **flexible**: By default, gray-matter is capable of parsing YAML, JSON and JavaScript front-matter. But other engines may be added.
- **extensible**: Use custom delimiters, or add support for any language, like TOML, CoffeeScript, or CSON
- **battle-tested**: used by assemble, metalsmith, phenomic, verb, generate, update and many others.

Rationale

Why did we create gray-matter in the first place?

We created gray-matter after trying out other libraries that failed to meet our standards and requirements.

Some libraries met most of the requirements, but *none met all of them*.

Here are the most important:

- Be usable, if not simple
- Use a dependable and well-supported library for parsing YAML
- Support other languages besides YAML
- Support stringifying back to YAML or another language
- Don't fail when no content exists
- Don't fail when no front matter exists
- Don't use regex for parsing. This is a relatively simple parsing operation, and regex is the slowest and most error-prone way to do it.
- Have no problem reading YAML files directly
- Have no problem with complex content, including **non-front-matter** fenced code blocks that contain examples of YAML front matter. Other parsers fail on this.
- Support stringifying back to front-matter. This is useful for linting, updating properties, etc.
- Allow custom delimiters, when it's necessary for avoiding delimiter collision.
- Should return an object with at least these three properties:
  - `data`: the parsed YAML front matter, as a JSON object
  - `content`: the contents as a string, without the front matter
  - `orig`: the "original" content (for debugging)

Usage

Using Node's `require()` system:

Default: `---`

Open and close delimiters can be passed in as an array of strings.

Example:

```
// format delims as a string
matter.read('file.md', {delims: '~~~'});
// or an array (open/close)
matter.read('file.md', {delims: ['~~~',
        '~~~']});
```

would parse:

```
~~~
title: Home
~~~
This is the {{title}} page.
```

Deprecated options

options.lang

Decrecated, please use options.language instead.

```
---
title = "TOML"
description = "Front matter"
categories = "front matter toml"
---
This is content
```

Results in:

```
{ content: 'This is content',
  excerpt: '',
  data:
   { title: 'TOML',
     description: 'Front matter',
     categories: 'front matter toml' } }
```

Dynamic language detection

Instead of defining the language on the options, gray-matter will automatically detect the language defined after the first delimiter and select the correct engine to use for parsing.

```
---toml
title = "TOML"
description = "Front matter"
categories = "front matter toml"
---
This is content
```

options.delimiters

Type: `String`


```
const matter = require('gray-matter');
```

Or with typescript

```
import matter = require('gray-matter');
// OR
import * as matter from 'gray-matter';
```

Pass a string and options to gray-matter:

```
console.log(matter('---\ntitle: Front
        Matter\n---\nThis is
        content.'));
```

Returns:

```
{
  content: '\nThis is content.',
  data: {
    title: 'Front Matter'
  }
}
```

More about the returned object in the following section.

Returned object

gray-matter returns a `file` object with the following properties.

Enumerable

- `file.data` **{Object}**: the object created by parsing front-matter
- `file.content` **{String}**: the input string, with `matter` stripped
- `file.excerpt` **{String}**: an excerpt, if defined on the options
- `file.empty` **{String}**: when the front-matter is "empty" (either all whitespace, nothing at all, or just comments and no data), the original string is set on this property. See #65 for details regarding use case.
- `file.isEmpty` **{Boolean}**: true if front-matter is empty.

Non-enumerable

In addition, the following non-enumberable properties are added to the object to help with debugging.

- `file.orig` **{Buffer}**: the original input string (or buffer)
- `file.language` **{String}**: the front-matter language that was parsed. `yaml` is the default
- `file.matter` **{String}**: the *raw*, un-parsed front-matter string
- `file.stringify` **{Function}**: stringify the file by converting `file.data` to a string in the given language, wrapping it in delimiters and prepending it to `file.content`.


```
    toml: {
      parse: toml.parse.bind(toml),

      // example of throwing an error to
         let users know stringifying is
      // not supported (a TOML
         stringifier might exist, this is
         just an example)
      stringify: function() {
        throw new Error('cannot
        stringify to TOML');
      }
    }
  }
});

console.log(file);
```

options.language

Type: `String`

Default: `yaml`

Define the engine to use for parsing front-matter.

```
console.log(matter(string, {language:
        'toml'}));
```

Example

The following HTML string:

options.engines

Define custom engines for parsing and/or stringifying front-matter.

Type: `Object` Object of engines

Default: `JSON`, `YAML` and `JavaScript` are already handled by default.

Engine format

Engines may either be an object with `parse` and (optionally) `stringify` methods, or a function that will be used for parsing only.

Examples

```
const toml = require('toml');

/**
 * defined as a function
 */

const file = matter(str, {
  engines: {
    toml: toml.parse.bind(toml),
  }
});

/**
 * Or as an object
 */

const file = matter(str, {
  engines: {
```


Run the examples

If you'd like to test-drive the examples, first clone gray-matter into `my-project` (or wherever you want):

```
$ git clone https://github.com/
        jonschlinkert/gray-matter my-
        project
```

CD into `my-project` and install dependencies:

```
$ cd my-project && npm install
```

Then run any of the examples to see how gray-matter works:

```
$ node examples/<example_name>
```

Links to examples

- coffee
- excerpt-separator
- excerpt-stringify
- excerpt
- javascript
- json-stringify
- json
- restore-empty
- sections-excerpt
- sections
- toml
- yaml-stringify

- yaml

API

matter

Takes a string or object with `content` property, extracts and parses front-matter from the string, then returns an object with `data`, `content` and other useful properties.

Params

- `input` **{Object|String}**: String, or object with `content` string
- `options` **{Object}**
- `returns` **{Object}**

Example

```
const matter = require('gray-matter');
console.log(matter('---\ntitle:
        Home\n---\nOther stuff'));
//=> { data: { title: 'Home'}, content:
        'Other stuff' }
```


options.excerpt_separator

Type: `String`

Default: `undefined`

Define a custom separator to use for excerpts.

```
console.log(matter(string,
        {excerpt_separator: '<!-- end --
        >'}));
```

Example

The following HTML string:

```
---
title: Blog
---
My awesome blog.
<!-- end -->
<h1>Hello world</h1>
```

Results in:

```
{
  data: { title: 'Blog'},
  excerpt: 'My awesome blog.',
  content: 'My awesome blog.\n<!-- end
        -->\n<h1>Hello world</h1>'
}
```

parameters, so you can control how the excerpt is extracted from the content.

Example

```
// returns the first 4 lines of the
        contents
function firstFourLines(file, options) {
  file.excerpt =
        file.content.split('\n').slice(0,
        4).join(' ');
}

const file =  matter([
  '---',
  'foo: bar',
  '---',
  'Only this',
  'will be',
  'in the',
  'excerpt',
  'but not this...'
].join('\n'), {excerpt:
        firstFourLines});
```

Results in:

```
{
  content: 'Only this\nwill be\nin
        the\nexcerpt\nbut not this...',
  data: { foo: 'bar' },
  excerpt: 'Only this will be in the
        excerpt'
}
```


.stringify

Stringify an object to YAML or the specified language, and append it to the given string. By default, only YAML and JSON can be stringified. See the engines section to learn how to stringify other languages.

Params

- `file` **{String|Object}**: The content string to append to stringified front-matter, or a file object with `file.content` string.
- `data` **{Object}**: Front matter to stringify.
- `options` **{Object}**: Options to pass to gray-matter and js-yaml.
- `returns` **{String}**: Returns a string created by wrapping stringified yaml with delimiters, and appending that to the given string.

Example

```
console.log(matter.stringify('foo bar
        baz', {title: 'Home'}));
// results in:
// ---
// title: Home
// ---
// foo bar baz
```

.read

Synchronously read a file from the file system and parse front matter. Returns the same object as the main function.

Params

- `filepath` **{String}**: file path of the file to read.
- `options` **{Object}**: Options to pass to gray-matter.
- `returns` **{Object}**: Returns an object with `data` and `content`

Example

```
const file = matter.read('./content/
        blog-post.md');
```

.test

Returns true if the given `string` has front matter.

Params

- `string` **{String}**
- `options` **{Object}**
- `returns` **{Boolean}**: True if front matter exists.

Options

options.excerpt

Type: `Boolean|Function`

Default: `undefined`

Extract an excerpt that directly follows front-matter, or is the first thing in the string if no front-matter exists.

If set to `excerpt: true`, it will look for the frontmatter delimiter, `---` by default and grab everything leading up to it.

Example

```
const str = '---\nfoo: bar\n---
        \nThis is an excerpt.\n---
        \nThis is content';
const file = matter(str, { excerpt:
        true });
```

Results in:

```
{
  content: 'This is an excerpt.\n---
        \nThis is content',
  data: { foo: 'bar' },
  excerpt: 'This is an excerpt.\n'
}
```

You can also set `excerpt` to a function. This function uses the ‘file’ and ‘options’ that were initially passed to gray-matter as