@push.rocks/smartfm

Documentation for @push.rocks/smartfm

- readme.md for @push.rocks/smartfm
- changelog.md for @push.rocks/smartfm

readme.md for @push.rocks/smartfm

The `@push.rocks/smartfm` module handles frontmatter data in markdown and other text files. The module allows easy parsing, stringification, and extraction of frontmatter data, using YAML or JSON format. It leverages the `gray-matter` library for advanced functionality with a simple API.

Install

To use `@push.rocks/smartfm`, you first need to install it via npm. Make sure you have Node.js installed, and run the following command in your terminal:

```
npm install @push.rocks/smartfm
```

This will add the module as a dependency to your project.

Usage

Once the module is installed, you can start using its capabilities by importing and initializing the `Smartfm` class in your TypeScript or JavaScript projects. The following sections provide comprehensive examples of the module's functionality:

Importing the Module

To get started with the module, use the following import statement:

```
import { Smartfm } from '@push.rocks/smartfm';
```

The module exports the `Smartfm` class, which you will use to manipulate frontmatter in your files.

Initializing the `Smartfm` Class

The `Smartfm` class requires an options object to be passed during initialization. The options object specifies the type of frontmatter format you are working with, which can either be `'yaml'` or `'json'`. Here's how you can initialize the class:

```
// Initialize with YAML frontmatter
let smartfm = new Smartfm({ fmType: 'yaml' });

// Initialize with JSON frontmatter
let smartfmJSON = new Smartfm({ fmType: 'json' });
```

The initialized objects can then be used to parse and stringify frontmatter in strings and text files.

Parsing Frontmatter from Strings

To parse frontmatter data from a string that includes frontmatter, use the `parse` method. This function returns an object containing the parsed frontmatter data and the rest of the string:

```
const sampleString = `---
title: Test Document
author: John Doe
---
This is the body content of the file.`;

let parsedContent = smartfm.parse(sampleString);
console.log(parsedContent.data); // Outputs the frontmatter data as a JavaScript object
console.log(parsedContent.content); // Outputs "This is the body content of the file."
```

The `parse` method will recognize and extract the frontmatter block, returning it as a JavaScript object while preserving the original content of the string.

Stringifying a String with Frontmatter

If you have a string and frontmatter data that you want to combine into a correctly formatted frontmatter block in a string, you can use the `stringify` method:

```
const bodyContent = "This is some markdown content.";
const frontmatterData = {
  date: "2023-01-01",
  tags: ["example", "frontmatter"]
};
```

```
let combinedString = smartfm.stringify(bodyContent, frontmatterData);
console.log(combinedString);
// Outputs a string with the frontmatter serialized at the top
```

This method appends the serialized frontmatter to the top of the given body content, making it ready for use in typical markdown or frontmatter-supported documents.

Parsing Frontmatter from Commented Sections

Sometimes frontmatter data might be commented out. The `parseFromComments` method allows you to parse such data by providing a comment prefix:

```
const commentedFile = `# ---
# title: Commented Example
# date: 2023-10-15
# ---
Actual file content here...`;

let parsedCommentedContent = smartfm.parseFromComments('# ', commentedFile);
console.log(parsedCommentedContent.data); // Outputs { title: "Commented Example", date:
"2023-10-15" }
```

The `parseFromComments` method strips the comment prefixes before parsing the frontmatter, enabling easy extraction of data listed in commented sections.

Using the Module for Complex Scenarios

Beyond these basic operations, the `@push.rocks/smartfm` module can be a crucial part of more complex workflows. For instance, integrating frontmatter management with static site generators, content management systems (CMS), or any situation where content metadata is necessary.

Example Scenario: A Static Site Generator

Imagine a static site generator that uses markdown files with frontmatter as data sources for pages. `@push.rocks/smartfm` can help read, extract, and manipulate these files programmatically:

```
import fs from 'fs';
import path from 'path';

const contentDir = path.join(process.cwd(), 'content');

fs.readdir(contentDir, (err, files) => {
  if (err) throw err;

  files.forEach(file => {
    const filePath = path.join(contentDir, file);
    fs.readFile(filePath, 'utf8', (err, data) => {
      if (err) throw err;

      const smartfmInstance = new Smartfm({ fmType: 'yaml' });
      const parsedPage = smartfmInstance.parse(data);

      console.log('Page Title:', parsedPage.data.title);
      console.log('Markdown Content:', parsedPage.content);

      // Optionally, modify and update frontmatter
      parsedPage.data.title = 'Updated Title';
      const updatedContent = smartfmInstance.stringify(parsedPage.content, parsedPage.data);

      fs.writeFile(filePath, updatedContent, (err) => {
        if (err) throw err;
        console.log('File updated successfully.');
      });
    });
  });
});
```

In this example, we read all markdown files from a directory, parse them to extract frontmatter data, optionally modify the frontmatter, and write the updated content back to the files.

Advanced Use Cases

`@push.rocks/smartfm` is flexible enough to support advanced operations like validating frontmatter, applying transformations, and more. Here are a few ideas to explore:

Frontmatter Validation

You might want to validate the structure of frontmatter data against a schema before processing. Here's a conceptual idea using `@push.rocks/smartfm`:

```
import Ajv from 'ajv';

const ajv = new Ajv();
const schema = {
  type: 'object',
  properties: {
    title: { type: 'string' },
    date: { type: 'string', format: 'date' }
  },
  required: ['title', 'date']
};

const validate = ajv.compile(schema);

const frontmatter = smartfm.parse(sampleString).data;
if (validate(frontmatter)) {
  console.log('Frontmatter is valid.');
} else {
  console.error('Frontmatter validation failed:', validate.errors);
}
```

Using a validation library like Ajv, you can ensure that the parsed frontmatter adheres to a required schema or format, enhancing data integrity in larger applications.

Transformed Output

Suppose you want to output the parsed markdown content in a transformed format, such as HTML. Here’s how you could integrate a markdown parsing library:

```
import marked from 'marked';

const parsedResult = smartfm.parse(sampleString);
const htmlContent = marked(parsedResult.content);

console.log('HTML Content:', htmlContent);
```

By combining `@push.rocks/smartfm` with a markdown parsing library like `marked`, you can convert markdown content into other formats while handling the associated frontmatter.

With these examples, you can see how `@push.rocks/smartfm` fits into various workflows involving frontmatter and text processing. The module's simplicity and effectiveness make it a suitable choice for projects that require clean handling of document metadata.

Feel free to explore and extend these examples based on your particular use case. Whether you are building a simple script, a web application, or an extensive content management system, `@push.rocks/smartfm` provides a robust foundation for frontmatter operations. Happy coding!

License and Legal Information

Trademarks

Company Information

Task Venture Capital GmbH
Registered at District court Bremen HRB 35230 HB, Germany

For any legal inquiries or if you require further information, please contact us via email at hello@task.vc.

changelog.md for @push.rocks/smartfm

2025-01-24 - 2.2.2 - fix(package)

Fix export path in package.json

- Corrected the export path from './dist/index.js' to './dist_ts/index.js' in package.json

2025-01-24 - 2.2.1 - fix(documentation)

Remove unnecessary markdown syntax from the README.

- Fixed minor formatting issue in the README file by removing extraneous markdown syntax.

2025-01-24 - 2.2.0 - feat(core)

Initial release of smartfm module for handling frontmatter

- Added Smartfm class to handle YAML and JSON frontmatter
- Implemented methods for parsing and stringifying frontmatter
- Included support for parseFromComments to handle commented out frontmatter

2025-01-24 - 2.1.2 - fix(documentation)

Updated readme.md to add comprehensive examples and advanced use cases.

- Added examples for initializing the Smartfm class with YAML and JSON frontmatter.
- Included detailed usage examples for parsing and stringifying frontmatter.
- Added complex scenarios and advanced use cases like using the module with static site generators and frontmatter validation.

2025-01-24 - 2.1.1 - fix(documentation)

Improved and updated README with detailed usage instructions

- Replaced placeholder text in README with complete usage guide
- Added comprehensive examples for parsing and stringifying frontmatter data
- Included detailed instructions on how to install and use the Smartfm class

2025-01-23 - 2.1.0 - feat(ci)

Introduce new CI workflows for Gitea and remove GitLab CI

- Added .gitea/workflows/default_nottags.yaml and .gitea/workflows/default_tags.yaml for Gitea CI workflow.
- Removed .gitlab-ci.yml indicating a shift from GitLab CI to Gitea CI.
- Updated package.json to use updated repository URLs conforming to the new hosting strategy.
- Updated README.md and npmextra.json with new npm package name and repository details.

2024-04-01 - 2.0.4 - Maintenance

Switch to a new organizational scheme and multiple updates to `npmextra.json`.

- Updated `npmextra.json` multiple times for consistency with `githost`.
- Switched to a new organization scheme.

2019-09-04 - 2.0.1 to 2.0.4 - Core Updates

Continual core updates to enhance stability.

- Core updates implemented in versions 2.0.1, 2.0.2, and 2.0.3.

2018-08-27 - 2.0.0 - Dependencies

Significant updates in dependencies management.

- Removed obsolete import of `typings-global`.
- Introduction of the 2.x series with dependency fixes.

2018-08-27 - 1.0.5 - BREAKING CHANGE

Transition to a new organizational scope.

- Switched to `@pushrocks` scope, marking a major change in project organization.

2017-05-27 - 1.0.3 - 1.0.4 - Configuration Updates

Enhancements in configuration and standards.

- Added `npmextra.json`.
- Updated CI YAML file.
- Adopted latest project standards.

2016-11-14 - 1.0.0 to 1.0.2 - Initial Release & Improvements

Initial project release with subsequent improvements.

- Initial working release.
- Improved Readme for better user guidance.
- Continuous updates to align with project standards.