

@push.rocks/smarthbs

A library that enhances handlebars with better file system support, templates compilation, and partials registration.

- [readme.md for @push.rocks/smarthbs](#)
- [changelog.md for @push.rocks/smarthbs](#)

readme.md for @push.rocks/smarthbs

A library that enhances handlebars with better file system support, templates compilation, and partials registration.

Install

To install `@push.rocks/smarthbs`, run the following command in your terminal:

```
npm install @push.rocks/smarthbs --save
```

This will add `@push.rocks/smarthbs` as a dependency to your project, allowing you to leverage its enhanced Handlebars templating capabilities.

Usage

The `@push.rocks/smarthbs` library extends Handlebars functionality by introducing better file system interaction, template compilation, and an easier way to manage partials. The following sections walk you through the various features and how you can utilize them for creating dynamic and organized templates.

Getting Started

First, import the `@push.rocks/smarthbs` module using ECMAScript Module (ESM) syntax in a TypeScript file:

```
import * as smarthbs from '@push.rocks/smarthbs';
```

Managing Partial

Handlebars' partials allow for embedding templates within other templates, making it simple to manage reusable pieces of template code. With `@push.rocks/smarthbs`, you can efficiently register an entire directory of partials.

Registering a Directory of Partials

To register all `.hbs` files in a directory, including those in subdirectories, as partials:

```
await smarthbs.registerPartialDir('./path/to/partial');
```

Each `.hbs` file in the specified directory becomes available as a partial. Partials are identified by their paths relative to the specified directory.

Compiling Templates

Compiling directories of Handlebars templates is seamless with `@push.rocks/smarthbs`. This feature reads templates from a source directory, compiles them using a specified data context, and writes the rendered output to a destination directory.

Compile a Directory

```
await smarthbs.compileDirectory('./source/templates', './destination/html', 'data.json');
```

Here, every `.hbs` file in `./source/templates` is compiled with data from `data.json`. The rendered outputs are saved as `.html` files in `./destination/html`.

Working with Variables

When handling complex templates, you might want to analyze which variables are used, verify their satisfaction, and ensure data completeness.

Finding Variables in Templates

To extract all variables used within a Handlebars template string:

```
const templateVariables = await smarthbs.findVarsInHbsString("Your template {{example}} here.");
console.log(templateVariables); // Outputs an array of variable names: ['example']
```

Ensuring Variables Satisfaction

To check if a given data object satisfies all required variables in a template:

```
const missingVars = await smarthbs.checkVarsSatisfaction("Your template {{example}} here.", {
  anotherVar: "some value" });
if(missingVars.length) {
  console.error('Missing variables:', missingVars);
}
```

This function ensures the provided data object contains all variables used in the template. Otherwise, it returns an array with the names of the missing variables.

Rendering Templates

You can use [@push.rocks/smarthbs](https://github.com/pushrocks/smarthbs) to compile Handlebars templates directly from strings or files:

From a String

```
const stringTemplate = await smarthbs.getTemplateForString("Hello, {{name}}!");
const renderedString = stringTemplate({ name: "World" });
console.log(renderedString); // "Hello, World!"
```

From a File

```
const fileTemplate = await smarthbs.getTemplateForFile('./path/to/template.hbs');
const renderedFileString = fileTemplate({ key: "value" });
console.log(renderedFileString); // Outputs the processed template with provided data
```

Ensuring Safe Syntax

If your Handlebars templates go through multiple processing stages, you might need to protect and restore the syntax:

```
const processedString = await smarthbs.postprocess("This is {{-safe}} syntax.");
console.log(processedString); // Restores to "This is {{safe}} syntax."
```

This approach allows you to keep placeholders intact during various stages, converting `{{- ... }}-` syntax back to `{{ ... }}`.

Advanced Features and Helpers

Custom Helpers

Extend Handlebars with custom helpers to introduce new functionalities or debug existing templates. For instance:

- **Analyze Helper:** Displays partials and their details.
- **Log Registered Partials:** Logs all registered partials, aiding in debugging.
- **Runtime Compilation:** Compile templates dynamically using the `__compile` helper.

Example: Building an HTML Page

Suppose you are building a simple HTML page. First, define a partial for the header and a general layout:

Create a new header partial:

```
<!-- ./partials/header.hbs -->
<header>
  <h1>{{title}}</h1>
</header>
```

Define a base layout that includes the header and a body:

```
<!-- ./layouts/main.hbs -->
{{> header title=pageTitle}}
<section>
  <h2>{{subtitle}}</h2>
  <p>{{content}}</p>
</section>
```

In your script, register partials and compile the layout:

```
import * as smarthbs from '@push.rocks/smarthbs';

// Register partials
await smarthbs.registerPartialDir('./partials');

// Prepare data for compilation
const data = {
  pageTitle: "My Awesome Page",
  subtitle: "Welcome to the world of dynamic templates!",
  content: "Handlebars makes creating reusable templates easy."
};
```

```
// Compile and render the layout
const mainTemplate = await smarthbs.getTemplateForFile('./layouts/main.hbs');
const renderedHtml = mainTemplate(data);

console.log(renderedHtml);

// Outputs the full HTML replacing variables in the layout with data
```

Conclusion

The [@push.rocks/smarthbs](#) library enhances the already powerful Handlebars templating engine with capabilities that are crucial for modern development workflows, especially those involving complex template management and dynamic content generation. Whether managing large-scale projects with numerous reusable components or simply wanting a better way to handle templates and partials, this tool provides a robust solution to enhance your projects and improve productivity.

License and Legal Information

This repository contains open-source code that is licensed under the MIT License. A copy of the MIT License can be found in the [license](#) file within this repository.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH and are not included within the scope of the MIT license granted herein. Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines, and any usage must be approved in writing by Task Venture Capital GmbH.

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.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

changelog.md for @push.rocks/smarthbs

2025-01-25 - 3.0.4 - fix(metadata)

Removed CI configuration file .gitlab-ci.yml and updated project metadata

- Deleted the .gitlab-ci.yml file to potentially switch or update CI/CD configurations.
- Updated the `description` and `keywords` in package.json and npmextra.json for better project documentation and SEO.

2024-04-01 to 2024-05-29 - 3.0.3 - maintenance

Maintenance updates and minor configuration changes.

- Updated the project description.
- Updated TypeScript configuration (`tsconfig`).
- Modified `npmextra.json` file for `githost`.

2023-06-25 to 2023-07-10 - 3.0.3 - organization

Organizational changes and fixes in core components.

- Transitioned to a new organization scheme.

2023-03-06 to 2023-06-25 - 3.0.1 to 3.0.2 - fixes

Core updates and critical fixes.

- Updated core components with fixes.

2022-07-24 - 2.0.9 - breaking change

Significant shifts in module support.

- **BREAKING CHANGE:** Switched to ECMAScript Module (ESM) format.

2018-08-27 - 1.0.16 - breaking change

Major updates in project scope.

- **BREAKING CHANGE:** Updated project scope to `@pushrocks`.

2017-05-26 - 1.0.15 - functionality

Enhancements in processing capabilities.

- Added post-processing functionality.

2017-05-12 - 1.0.13 - documentation improvements

Enhanced documentation.

- Added new documentation materials.

2017-05-01 - 1.0.8 - variable handling

Improvements in variable utilization.

- Introduced extended variable checking capabilities.