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 Partials

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/partials');
```

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` 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

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.

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.

Revision #3
Created 2026-03-28 11:10:41 UTC by foss.global Team
Updated 2026-03-28 12:17:29 UTC by foss.global Team