@push.rocks/gulp-function

A Gulp plugin to execute functions within a Gulp task pipeline.

readme.md for @push.rocks/gulp-function

@push.rocks/gulp-function is a Gulp plugin designed to execute custom functions within a Gulp pipeline. It accepts a function call as a parameter and allows for flexible task executions, including asynchronous tasks, with easy integration into the Gulp workflow.

Install

To use @push.rocks/gulp-function in your project, you'll first need to install it via npm. You can do this by running the following command in your project's root directory:

```
npm install @push.rocks/gulp-function --save-dev
```

This command will add `@push.rocks/gulp-function` as a development dependency to your project, making it available for use in your Gulp tasks.

Usage

The primary use case for `@push.rocks/gulp-function` is to incorporate custom function executions into your Gulp pipelines. This can be particularly useful for tasks that require conditional processing, external API calls, or any other logic that extends beyond the capabilities of existing Gulp plugins.

Here's how you can use `@push.rocks/gulp-function` within your Gulp setup:

Basic Example

Let's start with a simple scenario where you want to log the path of files processed by Gulp.

```
import gulp from 'gulp';
import gulpFunction, { forEach } from '@push.rocks/gulp-function';
```

```
const logFilePath = async (file: Buffer, enc: string) => {
  console.log(`Processing file: ${file.path}`);
};

gulp.task('log', () => {
  return gulp.src('./src/**/*.ts')
    .pipe(forEach(logFilePath))
    .pipe(gulp.dest('./dist'));
});
```

In this example, we're using the `forEach` method exported by `@push.rocks/gulp-function` to execute `logFilePath` for each file matched by `gulp.src`. The file's path is logged to the console during the build process.

Advanced Use Case: Asynchronous Function Execution

`@push.rocks/gulp-function` truly shines when you need to perform asynchronous operations within your Gulp tasks. Here's an example that demonstrates making an asynchronous API call for each file:

```
import gulp from 'gulp';
import gulpFunction, { forEach } from '@push.rocks/gulp-function';
import axios from 'axios';

const uploadFileToServer = async (file: Buffer, enc: string) => {
  const formData = new FormData();
  formData.append('file', file.contents, file.relative);

  await axios.post('https://example.com/upload', formData, {
    headers: {
      'Content-Type': 'multipart/form-data'
    }
  });

  console.log(`Uploaded file: ${file.path}`);
};

gulp.task('uploadFiles', () => {
```

```
  return gulp.src('./uploads/**/*')
    .pipe(forEach(uploadFileToServer))
    .pipe(gulp.dest('./dist/uploads'));
});
```

In this more advanced example, we use the `forEach` method to upload each file to a remote server using `axios`. The function `uploadFileToServer` is executed for each file, where an HTTP POST request is made to upload the file. This shows how you can integrate asynchronous operations seamlessly into your Gulp pipeline.

Execution Modes

`@push.rocks/gulp-function` supports three execution modes for your functions: `forEach`, `forFirst`, and `atEnd`.

- **forEach**: Executes the provided function for each file in the stream.
- **forFirst**: Executes the provided function only for the first file that passes through the stream.
- **atEnd**: Executes the provided function once after all files have passed through the stream.

These modes provide flexibility in determining when your custom functions should be triggered during the build process.

Conclusion

`@push.rocks/gulp-function` is a powerful tool for integrating custom processing logic into Gulp workflows. Whether you need to execute simple synchronous tasks or complex asynchronous operations, `@push.rocks/gulp-function` offers the flexibility and ease of use to enhance your Gulp builds significantly.

Remember, the examples provided are starting points. The real power of `@push.rocks/gulp-function` lies in its ability to adapt to your specific use cases, enabling you to automate and streamline your build processes as never before.

License and Legal Information

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.