

# @push.rocks/smarts

## pawn

A node module for smart subprocess handling with support for promises and streamlined subprocess communication.

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

# readme.md for @push.rocks/smartspawn

Given the provided files and their contents, the comprehensive documentation for using `@push.rocks/smartspawn` with TypeScript and in a manner that provides thorough examples and descriptions of features and scenarios is extensive. However, I'll outline a concise and informative usage guide that can serve as a helpful starting point. For an in-depth understanding and a complete reference to all features, further reading, exploration, and experimentation with the library are recommended.

## @push.rocks/smartspawn

smart subprocess handling

## Install

To install `@push.rocks/smartspawn`, open your terminal and run the following command:

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

This command fetches the package from npm and adds it to your project's dependencies.

## Usage

This guide uses TypeScript and ESM syntax to demonstrate how to utilize `@push.rocks/smartspawn` for smart subprocess handling in your application.

First, ensure that your TypeScript environment is set up to support ECMAScript modules (ESM). Your `tsconfig.json` should include:

```
{  
  "compilerOptions": {
```

```
"module": "ESNext",  
"target": "ESNext",  
"moduleResolution": "node"  
}  
}
```

## Starting and Stopping a Simple Thread

`ThreadSimple` class allows you to manage subprocesses effectively. Here's how to start and stop a simple thread:

```
// Import the necessary components from the library  
import { ThreadSimple } from '@push.rocks/smarts spawn';  
  
// Define a function that creates, starts, and stops a thread  
async function demonstrateThreadSimple() {  
  // Create an instance of ThreadSimple  
  const mySimpleThread = new ThreadSimple('./path/to/worker.js');  
  
  // Start the thread  
  await mySimpleThread.start();  
  console.log('Thread has started.');  
  // Stop the thread  
  await mySimpleThread.stop();  
  console.log('Thread has stopped.');}  
  
// Call the function to demonstrate its functionality  
demonstrateThreadSimple();
```

The `./path/to/worker.js` should be the path to your worker script that you want to execute in a separate process.

## Wrapping Processes

`smarts spawn` also provides functionality to wrap subprocesses, allowing you to modify or set up an environment before execution. Here's how to wrap and unwrap subprocesses:

```
import { startSpawnWrap, endSpawnWrap } from '@push.rocks/smartspawn';

// Start a wrap
startSpawnWrap('path/to/script', ['arg1', 'arg2'], { ENV_VAR: 'value' });
console.log('Subprocess wrapped.');
```

  

```
// End the wrap when it's no longer needed
endSpawnWrap();
console.log('Subprocess unwrapped.');
```

## Advanced Thread Management

`@push.rocks/smartspawn` integrates with the `threads` package to provide advanced threading capabilities. Here's a brief example showing how you might utilize it for more complex scenarios:

```
import { Thread, spawn, Worker } from 'threads';

async function advancedThreadExample() {
  const thread = await spawn(new Worker('./worker'));

  const result = await thread.doWork();
  console.log(`Result from thread: ${result}`);

  await Thread.terminate(thread);
}

advancedThreadExample();
```

Ensure you explore the `threads` documentation for a more detailed understanding of creating workers and communicating between the main process and threads.

## Conclusion

The `@push.rocks/smartspawn` package simplifies managing subprocesses in your Node.js applications, offering straightforward APIs for starting, stopping, and communicating with child processes. Whether you need to execute a simple script in the background or leverage sophisticated multi-threading capabilities, `smartspawn` provides the tools necessary to implement these features efficiently and robustly.

Remember, the examples above are starting points. Review the source code, tests, and type declarations for a comprehensive understanding of everything `@push.rocks/smarts spawn` and its integrated packages can do. Experiment with the library in your projects to best learn how to utilize its full potential in real-world applications.

# 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](mailto: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.