@push.rocks/smartdelay

A TypeScript library providing enhanced timeout functions compatible with async/await patterns.

- readme.md for @push.rocks/smartdelay

readme.md for @push.rocks/smartdelay

@push.rocks/smartdelay is a modern library designed to simplify working with timeouts in the async/await era, all while being fully written in TypeScript. This tool offers a range of functionalities that streamline the process of implementing delays and timeouts in your asynchronous JavaScript code, making it more readable and maintainable.

Install

To integrate @push.rocks/smartdelay into your project, you can install it via npm. Run the following command in your project directory:

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

This command adds the package to your project's dependencies, ensuring that you can easily import and utilize smartdelay's functions in your TypeScript files.

Usage

@push.rocks/smartdelay simplifies the handling of timeouts within async functions, offering methods to introduce specified delays or randomized time intervals. Below are detailed examples to demonstrate its usage. These examples are crafted using ECMAScript Modules (ESM) syntax and TypeScript.

Basic Delay

To introduce a basic delay in your asynchronous function, use the `delayFor` function. This method halts the execution for a specified number of milliseconds.

```
import { delayFor } from '@push.rocks/smartdelay';

async function basicDelayExample() {
```

```
  console.log('Delay start');
  await delayFor(3000); // Execution will pause here for 3 seconds
  console.log('Delay ended');
}

basicDelayExample();
```

In the above example, the program prints "Delay start", waits for 3 seconds due to `delayFor`, and then prints "Delay ended".

Delay with Randomization

For scenarios where you need a delay within a random time range, `delayForRandom` can be utilized. This introduces a non-deterministic delay duration, making it ideal for simulating real-world scenarios or for testing purposes.

```
import { delayForRandom } from '@push.rocks/smartdelay';

async function randomDelayExample() {
  console.log('Random delay start');
  await delayForRandom(2000, 5000); // Delay execution for a random duration between 2 and 5
seconds
  console.log('Random delay ended');
}

randomDelayExample();
```

This function takes two parameters: the minimum and maximum bounds (in milliseconds) for the random delay.

Passing Through Values

Both `delayFor` and `delayForRandom` can be used to pass through values after the delay. This feature can be particularly useful when chaining asynchronous operations.

```
import { delayFor } from '@push.rocks/smartdelay';

async function passThroughExample() {
  const result = await delayFor(3000, 'Hello after delay');
  console.log(result); // Outputs: Hello after delay
```

```
}

passThroughExample();
```

Advanced Usage: Timeout Class

@push.rocks/smartdelay provides a `Timeout` class for more granular control over timeouts, including support for cancellation.

```
import { Timeout } from '@push.rocks/smartdelay';

async function timeoutExample() {
  const timeout = new Timeout<string>(5000, 'Result after 5 seconds');
  // Cancel the timeout if needed
  // timeout.cancel();

  try {
    const result = await timeout.promise;
    console.log(result); // Result after 5 seconds (if not cancelled)
  } catch (error) {
    console.error('Timeout was cancelled', error);
  }
}

timeoutExample();
```

This class allows you to programmatically cancel the timeout before it completes, providing flexibility for dynamic timeout management situations.

Conclusion

@push.rocks/smartdelay offers a TypeScript-friendly, easy-to-use solution for managing timeouts and delays in asynchronous JavaScript. By leveraging this module, developers can write cleaner, more readable async code with minimal boilerplate. Whether you're implementing a simple delay, a random delay, or need finer control over your timeout logic, smartdelay provides the tools you need to get the job done effectively.

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.