readme.md for @push.rocks/smartpath

@push.rocks/smartpath offers smart ways to handle paths.

Install

To install `@push.rocks/smartpath`, you'll need to run the following command using npm. Ensure you have Node.js and npm installed before you start.

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

Usage

The `@push.rocks/smartpath` module provides a sophisticated yet straightforward approach to handle, manipulate, and evaluate file and directory paths within your TypeScript projects. By incorporating features like path normalization, transformation, and validation, `smartpath` facilitates powerful and efficient file system and URL path manipulation. This guide dives deep into the module's capabilities, showcasing a broad spectrum of use cases and demonstrating its versatility in handling paths.

Getting Started

To begin utilizing `smartpath` in your project, start by importing it in your TypeScript file:

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

Isomorphic Path Module

For cross-platform path operations that work in any JavaScript environment (Node.js, browsers, Deno, etc.), use the isomorphic module:

```
import * as isoPath from '@push.rocks/smartpath/iso';

// Join paths with automatic platform detection
const joinedPath = isoPath.join('/home/user', 'documents', 'file.txt');
// Unix: /home/user/documents/file.txt
// Windows: C:\Users\documents\file.txt

// Convert file:// URLs to system paths
const systemPath = isoPath.fileUrlToPath('file:///home/user/file.txt');
// Unix: /home/user/file.txt
// Windows: C:\home\user\file.txt

// Convert system paths to file:// URLs
const fileUrl = isoPath.pathToFileUrl('/home/user/file.txt');
// Result: file:///home/user/file.txt

// Get directory from path or file URL
const dir = isoPath.dirname('/home/user/documents/file.txt');
// Result: /home/user/documents
```

The isomorphic module automatically detects the path style (Windows vs POSIX) and handles:

- file:// URL conversions
- Mixed path separators
- Cross-platform compatibility
- Proper handling of Windows drive letters and UNC paths

Creating a Smartpath Instance

Instantiating a `Smartpath` object allows for the enrichment of path strings with additional context and manipulation capabilities:

```
const mySmartpath = new smartpath.Smartpath('/some/path/to/some.file');
console.log(mySmartpath);
```

Path Validation

Determining whether a path points to a file or directory is a common requirement. Here's how you can achieve this:

```
if (smartpath.check.isFile('./path/to/file.txt')) {
  console.log('This is a file.');
}

if (smartpath.check.isDir('./path/to/directory')) {
  console.log('This is a directory.');
}
```

Absolute Path Conversion

Converting a relative path to an absolute one is a frequent operation, especially in dynamic file handling scenarios:

```
const absolutePath = smartpath.transform.makeAbsolute('./relative/path/to/file.txt');
console.log(`Absolute path: ${absolutePath}`);
```

Handling Multiple Paths

`smartpath` shines when dealing with multiple paths, offering efficient bulk operations:

```
const paths = ['./path/to/file1.txt', './another/path/to/file2.txt'];
const absolutePaths = smartpath.transform.toAbsolute(paths);
console.log(absolutePaths);
```

Understanding Path Types

Identifying whether a path represents a local file system path or a URL is straightforward:

```
const pathType = smartpath.get.type('https://example.com/resource');
console.log(`Path type: ${pathType}`); // 'url'
```

Home Directory Paths

Easily manage paths relative to the user's home directory:

```
const homePath = smartpath.get.home('~/path/to/resource');
console.log(`Home directory path: ${homePath}`);
```

Analyzing Path Components

Breaking down a path into its components allows for detailed path analysis and manipulation:

```
const pathLevels = smartpath.get.pathLevels('/path/to/resource');
console.log(pathLevels); // ['path', 'to', 'resource']
```

Path Manipulation and More

Beyond the basics, `smartpath` offers a comprehensive set of tools for robust path manipulation, including normalizing paths across different operating systems, working with URL paths, and handling special path constructs like ".." and ".".

For more advanced use cases, such as transforming path lists, deriving relative paths, or integrating path handling into larger application workflows, `smartpath` provides both utility functions and object-oriented interfaces that streamline these operations.

By leveraging the full spectrum of `smartpath`'s features, developers can handle virtually any path-related task with ease, efficiency, and reliability. Whether you're building a complex file system utility, managing web application assets, or simply need reliable path manipulation in your TypeScript projects, `smartpath` offers the functionality and flexibility required to do the job right.

License and Legal Information

Trademarks

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.

Revision #3
Created 2026-03-28 11:11:38 UTC by foss.global Team
Updated 2026-03-28 12:18:30 UTC by foss.global Team