# @push.rocks/smartobject

A library to simplify and enhance object manipulation and traversal.

# readme.md for @push.rocks/smartobject

work with objects

## Install

To install `@push.rocks/smartobject`, use the following npm command. It's recommended to install this package in a TypeScript project to make full use of its functionalities and type definitions.

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

This command will add `@push.rocks/smartobject` to your project's `package.json` file and download the package to the `node_modules` directory.

## Usage

This module provides a convenient way to work with JavaScript objects, offering utilities for deep object manipulation, comparison, and more. Below are detailed examples of how to utilize `@push.rocks/smartobject` in a variety of use cases. Always use ESM syntax in combination with TypeScript for the best developer experience.

## Importing the Module

Before you can use the functionalities provided by `smartobject`, you need to import the required classes or functions into your TypeScript file.

```
import {
  SmartObject,
  fastDeepEqual,
  smartGet,
  smartAdd,
```

```
  compareObjects,
  forEachMinimatch
} from '@push.rocks/smartobject';
```

# Working with SmartObject

The `SmartObject` class is a wrapper around a standard JavaScript object that provides additional capabilities for manipulating and querying the object.

## Creating a SmartObject

```
// Original JavaScript object
const originalObject = {
  level1: {
    level2: 'value',
  },
};

// Creating a SmartObject
const smartObj = new SmartObject(originalObject);
```

## Adding and Retrieving Values

```
// Adding a new value
smartObj.addValueAtFlatPathString('level1.newKey', 'newValue');

// Retrieving a value
const value = smartObj.getValueAtFlatPathString('level1.level2');
console.log(value); // Output: 'value'
```

## Converting to a Flat Object

```
const flatObject = smartObj.toFlatObject();
console.log(flatObject); // Output: { 'level1.level2': 'value', 'level1.newKey': 'newValue' }
```

# Deep Object Comparison

`compareObjects` allows you to compare two objects and get detailed information about their differences.

```
const obj1 = {name: 'John', age: 30};
const obj2 = {name: 'John', age: 31};

const comparisonResult = compareObjects(obj1, obj2);
console.log(comparisonResult);
```

# Deep Object Manipulation

`smartAdd` and `smartGet` enable deep manipulation and querying of objects using dot notation paths.

```
// Using smartAdd to add a deep-nested property
const obj = {};
smartAdd(obj, 'user.info.name', 'Jane Doe');

// Using smartGet to retrieve a deep-nested property
const name = smartGet(obj, 'user.info.name');
console.log(name); // Output: Jane Doe
```

# Iterating over Object Properties with Minimatch Patterns

`forEachMinimatch` provides a way to execute a function for each object property that matches a given minimatch pattern.

```
const userObj = {
  'user.name': 'John Doe',
  'user.age': 30,
  'config.theme': 'dark',
};

forEachMinimatch(userObj, 'user.*', (value, key) => {
  console.log(key, value);
});
```

This module leverages the power of TypeScript to offer robust typing and intelligent auto-completion, making it easier to work with complex objects and avoid common pitfalls. Whether you are manipulating deeply nested properties, comparing object differences, or iterating over specific

object keys, `@push.rocks/smartobject` provides a versatile toolkit to streamline your object handling logic.

For a complete list of features and functionalities, including additional utility functions not covered in these examples, please refer to the module's documentation and type definitions.

# 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.