readme.md for @push.rocks/smartjson

□□ Typed JSON handling for modern Node.js and TypeScript applications

A powerful library for working with JSON in TypeScript, providing type-safe serialization, advanced buffer handling, deep object comparison, and support for complex class instances. Perfect for applications that need reliable JSON manipulation with full TypeScript support.

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

Installation

```
# Using npm
npm install @push.rocks/smartjson --save

# Using yarn
yarn add @push.rocks/smartjson

# Using pnpm (recommended)
pnpm add @push.rocks/smartjson
```

Features

 Type-Safe JSON Operations - Full TypeScript support with proper typing
 Class Instance Serialization - Fold and unfold class instances to/from JSON
 Buffer & Binary Support - Seamless handling of Buffers and Typed Arrays
 JSON Lines Support - Parse, stringify and compare JSONL data streams
 Pretty Printing - Beautiful formatted JSON output
⚡ **Stable Stringification** - Consistent key ordering for reliable comparisons
♻ **Circular Reference Handling** - Safe one-way stringify for objects with cycles
 Deep Equality Checks - Compare complex objects and JSON structures
 Base64 Encoding - Built-in base64 JSON encoding/decoding

Quick Start

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

// Parse JSON with automatic buffer handling
const parsed = smartjson.parse('{"name":"example","data":{"type":"Buffer","data":[1,2,3]}}');

// Stringify with stable key ordering
const json = smartjson.stringify({ z: 1, a: 2, m: 3 });
// Result: '{"a":2,"m":3,"z":1}'

// Pretty print for human readability
const pretty = smartjson.stringifyPretty({ hello: 'world', count: 42 });
// Result:
// {
//   "hello": "world",
//   "count": 42
// }
```

Core Functions

JSON Parsing and Stringification

```
// Standard parsing with automatic Buffer detection
const obj = smartjson.parse('{"hello":"world"}');
```

```
// Stable stringification (consistent key ordering)
const jsonStr = smartjson.stringify({ name: 'test', id: 1 });

// Pretty printing for debugging
const prettyJson = smartjson.stringifyPretty({
  nested: {
    data: 'value'
  }
});
```

Safe One-Way Stringification (New in v5.1.0)

Handle circular references and unserializable values safely for hashing and comparisons:

```
// Create objects with circular references
const objA = { name: 'A' };
const objB = { name: 'B', ref: objA };
objA['ref'] = objB; // Circular reference!

// Normal stringify would throw, but stableOneWayStringify handles it
const safeJson = smartjson.stableOneWayStringify(objA);
// Circular references are replaced with "__cycle__"

// Perfect for object hashing
const hash1 = crypto.createHash('sha256')
  .update(smartjson.stableOneWayStringify(complexObject))
  .digest('hex');

// Handles unserializable values gracefully
const objWithGetter = {
  normal: 'value',
  get throwing() { throw new Error('Cannot serialize!'); }
};
const safe = smartjson.stableOneWayStringify(objWithGetter);
// Throwing getters are replaced with "__unserializable__"

// Custom key ordering for consistent hashes
```

```
const ordered = smartjson.stableOneWayStringify(
  { z: 1, a: 2 },
  ['a', 'z'] // Specify key order
);
```

Base64 JSON Encoding

Encode JSON data as base64 for safe transmission:

```
const myData = {
  message: 'Hello World',
  timestamp: Date.now()
};

// Encode to base64
const encoded = smartjson.stringifyBase64(myData);
console.log(encoded); // Base64 string

// Decode back to object
const decoded = smartjson.parseBase64(encoded);
console.log(decoded); // Original object
```

JSON Lines (JSONL) Support

Perfect for streaming data and log processing:

```
// Parse JSON Lines format
const jsonLines = `{"event":"start","time":1234}
{"event":"data","value":42}
{"event":"end","time":5678}`;

const events = smartjson.parseJsonL(jsonLines);
// Result: Array of parsed objects

// Produce JSONL from objects
const jsonlOut = smartjson.stringifyJsonL([
  { event: 'start', time: 1234 },
  { event: 'data', value: 42 },
```

```
  { event: 'end', time: 5678 }
]);

// Compare JSON Lines data
const jsonL1 = `{"id":1}\n{"id":2}`;
const jsonL2 = `{"id":1}\n{"id":2}`;
const isEqual = smartjson.deepEqualJsonLStrings(jsonL1, jsonL2); // true
```

Advanced Class Serialization

Creating Serializable Classes

Transform class instances to JSON and back while preserving type safety:

```
import { Smartjson, foldDec } from '@push.rocks/smartjson';

class User extends Smartjson {
  @foldDec() accessor username: string;
  @foldDec() accessor email: string;
  @foldDec() accessor settings: UserSettings;

  // Properties without @foldDec won't be serialized
  private internalId: string;

  constructor(username: string, email: string) {
    super();
    this.username = username;
    this.email = email;
    this.settings = new UserSettings();
    this.internalId = Math.random().toString();
  }
}

class UserSettings extends Smartjson {
  @foldDec() accessor theme: 'light' | 'dark' = 'light';
  @foldDec() accessor notifications: boolean = true;
}
```

```
// Create and serialize
const user = new User('john_doe', 'john@example.com');
user.settings.theme = 'dark';

// Convert to JSON
const jsonString = user.foldToJson();
console.log(jsonString);
//
{"username":"john_doe","email":"john@example.com","settings":{"theme":"dark","notifications":t
rue}}

// Restore from JSON with correct typing
const restoredUser = User.enfoldFromJson(jsonString);
console.log(restoredUser instanceof User); // true
console.log(restoredUser.settings instanceof UserSettings); // true
```

Working with Nested Objects

```
class Company extends Smartjson {
  @foldDec() accessor name: string;
  @foldDec() accessor employees: Employee[] = [];

  addEmployee(employee: Employee) {
    this.employees.push(employee);
  }
}

class Employee extends Smartjson {
  @foldDec() accessor name: string;
  @foldDec() accessor role: string;
  @foldDec() accessor salary: number;

  constructor(name: string, role: string, salary: number) {
    super();
    this.name = name;
    this.role = role;
    this.salary = salary;
```

```
  }
}

const company = new Company();
company.name = 'TechCorp';
company.addEmployee(new Employee('Alice', 'Developer', 100000));
company.addEmployee(new Employee('Bob', 'Designer', 90000));

// Serialize entire object graph
const json = company.foldToJson();

// Deserialize with all nested objects properly instantiated
const restored = Company.enfoldFromJson(json);
```

Buffer and Binary Data Handling

SmartJson seamlessly handles binary data in JSON:

```
// Automatic Buffer handling
const dataWithBuffer = {
  name: 'BinaryData',
  buffer: Buffer.from('Hello World'),
  typedArray: new Uint8Array([1, 2, 3, 4, 5])
};

// Stringify (buffers are automatically encoded)
const jsonStr = smartjson.stringify(dataWithBuffer);

// Parse (buffers are automatically restored)
const restored = smartjson.parse(jsonStr);
console.log(restored.buffer); // Buffer
console.log(restored.typedArray); // Uint8Array
```

Deep Comparison

Compare complex objects with automatic normalization:

```
// Deep object comparison
const obj1 = {
  nested: {
    array: [1, 2, { deep: 'value' }],
    flag: true
  },
  name: 'test'
};

const obj2 = {
  name: 'test',  // Different order
  nested: {
    flag: true,   // Different order
    array: [1, 2, { deep: 'value' }]
  }
};

const isEqual = smartjson.deepEqualObjects(obj1, obj2); // true
```

Real-World Examples

API Response Caching

```
class CachedAPIResponse extends Smartjson {
  @foldDec() accessor data: any;
  @foldDec() accessor timestamp: number;
  @foldDec() accessor endpoint: string;

  isExpired(maxAge: number = 3600000): boolean {
    return Date.now() - this.timestamp > maxAge;
  }

  static fromAPICall(endpoint: string, data: any): CachedAPIResponse {
    const response = new CachedAPIResponse();
    response.endpoint = endpoint;
    response.data = data;
```

```
    response.timestamp = Date.now();
    return response;
  }
}

// Store API response
const apiData = await fetch('/api/users');
const cached = CachedAPIResponse.fromAPICall('/api/users', await apiData.json());
localStorage.setItem('cached_users', cached.foldToJson());

// Retrieve and check
const stored = localStorage.getItem('cached_users');
if (stored) {
  const cached = CachedAPIResponse.enfoldFromJson(stored);
  if (!cached.isExpired()) {
    return cached.data; // Use cached data
  }
}
```

Configuration Management

```
class AppConfig extends Smartjson {
  @foldDec() accessor apiUrl: string;
  @foldDec() accessor features: Map<string, boolean> = new Map();
  @foldDec() accessor limits: {
    maxUploadSize: number;
    maxConcurrentRequests: number;
  };

  enableFeature(name: string) {
    this.features.set(name, true);
  }

  save() {
    fs.writeFileSync('config.json', this.foldToJson());
  }

  static load(): AppConfig {
```

```
    const json = fs.readFileSync('config.json', 'utf-8');
    return AppConfig.enfoldFromJson(json);
  }
}
```

API Reference

Core Functions

- `parse(jsonString: string): any` - Parse JSON with automatic buffer handling
- `stringify(obj: any, simpleOrderArray?: string[], options?: Options): string` - Convert to JSON with stable ordering
- `stableOneWayStringify(obj: any, simpleOrderArray?: string[], options?: Options): string` - Safe stringify with circular reference handling (one-way, for hashing/comparison)
- `stringifyPretty(obj: any): string` - Pretty print JSON with 2-space indentation
- `stringifyBase64(obj: any): string` - Encode JSON as base64
- `parseBase64(base64String: string): any` - Decode base64 JSON
- `parseJsonL(jsonLinesString: string): any[]` - Parse JSON Lines format
- `stringifyJsonL(items: any[]): string` - Stringify array to JSON Lines
- `deepEqualObjects(obj1: any, obj2: any): boolean` - Deep comparison of objects
- `deepEqualJsonLStrings(jsonL1: string, jsonL2: string): boolean` - Compare JSON Lines strings

Smartjson Class

- `Smartjson.enfoldFromObject<T>(obj: any): T` - Create instance from plain object
- `Smartjson.enfoldFromJson<T>(json: string): T` - Create instance from JSON string
- `instance.foldToObject(): any` - Convert instance to plain object
- `instance.foldToJson(): string` - Convert instance to JSON string

Decorators

- `@foldDec()` - Mark class property for serialization (use with `accessor` keyword)

Performance Tips

1. **Use stable stringification** for consistent hashing and comparison
2. **Enable pretty printing** only for debugging (it's slower)
3. **Cache base64 encodings** when repeatedly sending the same data
4. **Use JSON Lines** for streaming large datasets
5. **Use stableOneWayStringify** for objects with circular references (for hashing/caching)
6. **Prefer regular stringify** for round-trip serialization without cycles

Migration Guide

If you're migrating from native JSON:

```
// Before
JSON.parse(jsonString);
JSON.stringify(object);

// After
smartjson.parse(jsonString);  // Adds buffer support
smartjson.stringify(object);  // Adds stable ordering & buffer support
```

Browser Support

This library supports modern browsers and Node.js environments. For older browsers, ensure you have appropriate polyfills for `TextEncoder` and `TextDecoder`.

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

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 or third parties, 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 or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or 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 #2
Created 2026-03-28 11:40:18 UTC by foss.global Team
Updated 2026-03-28 12:18:15 UTC by foss.global Team