@push.rocks/smartfile

Provides comprehensive tools for efficient file management in Node.js using TypeScript, including handling streams, virtual directories, and various file operations.

- readme.md for @push.rocks/smartfile
- changelog.md for @push.rocks/smartfile

readme.md for @push.rocks/smartfile

> **High-level file representation classes for Node.js**

□□ What is smartfile?

`@push.rocks/smartfile` provides powerful **in-memory file representations** for Node.js applications. It offers clean, TypeScript-first classes for working with files (`SmartFile`), streams (`StreamFile`), and virtual file collections (`VirtualDirectory`).

Think of it as your go-to solution for **content manipulation**, **file transformations**, and **in-memory file operations** - all while seamlessly integrating with @push.rocks/smartfs for actual filesystem operations.

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 want to sign a contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

□□ Installation

```
pnpm install @push.rocks/smartfile
# Optional: Install smartfs for filesystem operations
pnpm install @push.rocks/smartfs
```

□ Key Features

- □□ **Factory Pattern** - Clean, consistent API for creating file instances
- □□ **Streaming Support** - Handle massive files efficiently with `StreamFile`
- □□ **Virtual Directories** - Work with in-memory file collections
- □□ **URL Support** - Directly fetch files from URLs
- □□ **Content Manipulation** - Edit, transform, and parse file content
- ⚡ **TypeScript First** - Full type safety and IntelliSense support
- □□ **Comprehensive Collection API** - Filter, map, find files in virtual directories

□□ Quick Start

Using the Factory

```
import { SmartFileFactory } from '@push.rocks/smartfile';

// Create factory (uses Node.js filesystem by default)
const factory = SmartFileFactory.nodeFs();

// Load a file into memory
const file = await factory.fromFilePath('./config.json');

// Edit content
await file.editContentAsString(async (content) => {
  return content.toUpperCase();
});

// Save back to disk
await file.write();
```

With SmartFs Integration

```
import { SmartFileFactory } from '@push.rocks/smartfile';
import { SmartFs, SmartFsProviderNode } from '@push.rocks/smartfs';
```

```
// Create SmartFs instance with provider
const smartFs = new SmartFs(new SmartFsProviderNode());

// Create factory bound to this filesystem
const factory = new SmartFileFactory(smartFs);

// Now all file operations use the smartfs instance
const file = await factory.fromFilePath('./data.json');
await file.write(); // Uses smartfs under the hood
```

□□ Core Components

SmartFileFactory

The factory is your entry point for creating all file instances:

```
import { SmartFileFactory } from '@push.rocks/smartfile';

const factory = SmartFileFactory.nodeFs();

// Create from various sources
const fileFromPath = await factory.fromFilePath('./data.json');
const fileFromUrl = await factory.fromUrl('https://example.com/config.json');
const fileFromBuffer = factory.fromBuffer('./file.txt', Buffer.from('content'));
const fileFromString = factory.fromString('./file.txt', 'Hello World', 'utf8');

// Create StreamFile instances
const stream = await factory.streamFromPath('./large-file.zip');
const streamFromUrl = await factory.streamFromUrl('https://example.com/video.mp4');

// Create VirtualDirectory instances
const vdir = await factory.virtualDirectoryFromPath('./src');
const emptyVdir = factory.virtualDirectoryEmpty();
```

SmartFile Class

Represents a single file loaded in memory:

```
// Created via factory
const file = await factory.fromFilePath('./data.json');

// Content access
const asString = file.parseContentAsString();
const asBuffer = file.parseContentAsBuffer();

// Content manipulation
await file.editContentAsString(async (content) => {
  const data = JSON.parse(content);
  data.updated = new Date().toISOString();
  return JSON.stringify(data, null, 2);
});

// File operations
await file.write();                          // Save to original location
await file.writeToDiskAtPath('./output.json'); // Save to specific path
await file.writeToDir('./dist');            // Save to directory
await file.read();                           // Reload from disk
await file.delete();                         // Delete from disk

// Metadata
const size = await file.getSize();           // File size in bytes
const hash = await file.getHash('content');  // SHA256 hash
const stream = file.getStream();             // Get as Node.js stream

// Path information
console.log(file.path);                      // Relative path
console.log(file.absolutePath);              // Absolute path
console.log(file.parsedPath);                // Parsed path components
```

StreamFile Class

Perfect for handling large files without memory overhead:

```
// Created via factory
const streamFile = await factory.streamFromPath('./bigfile.zip');

// Or from URL
```

```
const urlStream = await factory.streamFromUrl('https://example.com/large.mp4');

// Or from buffer
const bufferStream = factory.streamFromBuffer(Buffer.from('content'));

// Write to disk (streams the content)
await streamFile.writeToDisk('./output/bigfile.zip');
await streamFile.writeToDir('./output');

// Get content (loads into memory - use carefully!)
const buffer = await streamFile.getContentAsBuffer();
const string = await streamFile.getContentAsString('utf8');

// Get as Node.js stream for piping
const readStream = await streamFile.createReadStream();

// Convert to SmartFile (loads into memory)
const smartFile = await streamFile.toSmartFile();

// Get file size
const size = await streamFile.getSize();
```

VirtualDirectory Class

Manage collections of SmartFiles in memory:

```
// Created via factory
const vdir = await factory.virtualDirectoryFromPath('./src');

// Or create empty
const emptyVdir = factory.virtualDirectoryEmpty();

// Or from file array
const files = [file1, file2, file3];
const vdirFromFiles = factory.virtualDirectoryFromFileArray(files);

// ==========================================
// Collection Queries (in-memory operations)
```

```
// ============================================

// Check existence in collection
if (vdir.exists('components/Button.tsx')) {
  console.log('File exists in virtual directory');
}

// Get file from collection
const file = await vdir.getFileByPath('utils/helpers.ts');

// List all files
const allFiles = vdir.listFiles();

// List directory paths represented in collection
const dirs = vdir.listDirectories();

// Filter files
const tsFiles = vdir.filter(f => f.path.endsWith('.ts'));
const largeFiles = vdir.filter(f => f.contentBuffer.length > 10000);

// Map/transform files
const uppercased = vdir.map(f => {
  f.contentBuffer = Buffer.from(f.parseContentAsString().toUpperCase());
  return f;
});

// Find specific file
const configFile = vdir.find(f => f.path.includes('config'));

// Collection info
const fileCount = vdir.size();
const empty = vdir.isEmpty();

// ============================================
// Collection Mutations
// ============================================

// Add files
vdir.addSmartfile(newFile);
```

```
vdir.addSmartfiles([file1, file2, file3]);

// Remove file
vdir.removeByPath('old-file.ts');

// Clear all files
vdir.clear();

// Merge another virtual directory
vdir.merge(otherVirtualDir);

// ============================================
// Load/Save (filesystem bridge operations)
// ============================================

// Save all files to disk
await vdir.saveToDisk('./dist');

// Reload from disk
await vdir.loadFromDisk('./src');

// Work with subdirectories
const subVdir = await vdir.shiftToSubdirectory('components');
await vdir.addVirtualDirectory(otherVdir, 'lib');
```

□□ Integration with SmartFs

For filesystem operations beyond loading/saving content, use @push.rocks/smartfs:

```
import { SmartFileFactory } from '@push.rocks/smartfile';
import { SmartFs, SmartFsProviderNode } from '@push.rocks/smartfs';

const smartFs = new SmartFs(new SmartFsProviderNode());
const factory = new SmartFileFactory(smartFs);

// Use smartfile for content manipulation
const file = await factory.fromFilePath('./config.json');
```

```
await file.editContentAsString(async (s) => s.toUpperCase());
await file.write();

// Use smartfs for filesystem operations
const exists = await smartFs.file('./config.json').exists();
await smartFs.file('./config.json').copy('./config.backup.json');
const stats = await smartFs.file('./config.json').stat();

// List directory with smartfs
const entries = await smartFs.directory('./src').list();
```

□□ Common Use Cases

Configuration File Management

```
const factory = SmartFileFactory.nodeFs();

// Load, modify, and save config
const config = await factory.fromFilePath('./package.json');
await config.editContentAsString(async (content) => {
  const pkg = JSON.parse(content);
  pkg.version = '2.0.0';
  return JSON.stringify(pkg, null, 2);
});
await config.write();
```

Batch File Processing

```
const factory = SmartFileFactory.nodeFs();

// Load directory into virtual collection
const vdir = await factory.virtualDirectoryFromPath('./content');

// Process all markdown files
const mdFiles = vdir.filter(f => f.path.endsWith('.md'));
```

```
for (const file of mdFiles.listFiles()) {
  await file.editContentAsString(async (content) => {
    // Add frontmatter, transform links, etc.
    return `---\nprocessed: true\n---\n\n${content}`;
  });
}

// Save processed files
await vdir.saveToDisk('./dist/content');
```

Download and Process Remote Files

```
const factory = SmartFileFactory.nodeFs();

// Fetch from URL
const remoteFile = await factory.fromUrl('https://api.example.com/data.json');

// Process content
await remoteFile.editContentAsString(async (content) => {
  const data = JSON.parse(content);
  // Transform data
  return JSON.stringify(data.results, null, 2);
});

// Save locally
await remoteFile.writeToDiskAtPath('./cache/data.json');
```

Large File Streaming

```
const factory = SmartFileFactory.nodeFs();

// Download large file as stream
const largeFile = await factory.streamFromUrl('https://example.com/large-dataset.csv');

// Save to disk (streams, doesn't load all into memory)
await largeFile.writeToDisk('./data/dataset.csv');
```

```
// Or get size without downloading entire file
const size = await largeFile.getSize();
console.log(`File size: ${size} bytes`);
```

Virtual File System for Testing

```
import { SmartFileFactory } from '@push.rocks/smartfile';
import { SmartFs, SmartFsProviderMemory } from '@push.rocks/smartfs';

// Use in-memory filesystem for tests
const memoryFs = new SmartFs(new SmartFsProviderMemory());
const factory = new SmartFileFactory(memoryFs);

// Create virtual files
const testFile = factory.fromString('test.txt', 'test content');
await testFile.write(); // Writes to in-memory filesystem

// Test your code without touching real filesystem
```

□□ Architecture

Responsibility Split

@push.rocks/smartfile (this package):

- □ In-memory file representations (SmartFile, StreamFile, VirtualDirectory)
- □ Content manipulation and transformation
- □ Loading content FROM sources (disk, URL, buffer, string)
- □ Saving content TO destinations (disk, stream)
- □ Collection operations (filter, map, find on VirtualDirectory)

@push.rocks/smartfs:

- □ Low-level filesystem operations (exists, stat, copy, move, delete)
- □ Directory operations (list, create, remove)
- □ Provider abstraction (Node.js fs, in-memory, S3, etc.)
- □ Streaming (readStream, writeStream)
- □ Transactions and file watching

API Reference

SmartFileFactory

Method	Description
`SmartFileFactory.nodeFs()`	Create factory with Node.js filesystem provider
`new SmartFileFactory(smartFs)`	Create factory with custom SmartFs instance
`factory.fromFilePath(path, base?)`	Load file from disk into SmartFile
`factory.fromUrl(url)`	Fetch file from URL into SmartFile
`factory.fromBuffer(path, buffer, base?)`	Create SmartFile from Buffer
`factory.fromString(path, content, encoding, base?)`	Create SmartFile from string
`factory.streamFromPath(path)`	Create StreamFile from disk
`factory.streamFromUrl(url)`	Create StreamFile from URL
`factory.streamFromBuffer(buffer, path?)`	Create StreamFile from Buffer
`factory.virtualDirectoryFromPath(path)`	Load directory into VirtualDirectory
`factory.virtualDirectoryEmpty()`	Create empty VirtualDirectory
`factory.virtualDirectoryFromFileArray(files)`	Create VirtualDirectory from SmartFiles

SmartFile Instance Methods

Method	Description
`file.write()`	Save to original location
`file.writeToDiskAtPath(path)`	Save to specific path
`file.writeToDir(dir)`	Save to directory (preserves relative path)
`file.read()`	Reload content from disk
`file.delete()`	Delete file from disk
`file.editContentAsString(fn)`	Transform content as string
`file.parseContentAsString(encoding?)`	Get content as string
`file.parseContentAsBuffer()`	Get content as Buffer
`file.getHash(type?)`	Get SHA256 hash ('path', 'content', 'all')
`file.getSize()`	Get content size in bytes

Method	Description
`file.getStream()`	Get content as Node.js Readable stream

StreamFile Instance Methods

Method	Description
`stream.writeToDisk(path)`	Stream content to disk
`stream.writeToDir(dir)`	Stream to directory
`stream.createReadStream()`	Get as Node.js Readable stream
`stream.getContentAsBuffer()`	Load entire content into Buffer
`stream.getContentAsString(encoding?)`	Load entire content as string
`stream.getSize()`	Get content size in bytes
`stream.toSmartFile()`	Convert to SmartFile (loads into memory)

VirtualDirectory Instance Methods

Collection Queries:

Method	Description
`vdir.exists(path)`	Check if file exists in collection
`vdir.has(path)`	Alias for exists()
`vdir.getFileByPath(path)`	Get SmartFile by path
`vdir.listFiles()`	Get all SmartFiles
`vdir.listDirectories()`	Get all directory paths
`vdir.filter(predicate)`	Filter files, returns new VirtualDirectory
`vdir.map(fn)`	Transform files, returns new VirtualDirectory
`vdir.find(predicate)`	Find first matching file
`vdir.size()`	Get file count
`vdir.isEmpty()`	Check if empty

Collection Mutations:

Method	Description
`vdir.addSmartfile(file)`	Add single file

Method	Description
`vdir.addSmartfiles(files)`	Add multiple files
`vdir.removeByPath(path)`	Remove file by path
`vdir.clear()`	Remove all files
`vdir.merge(otherVdir)`	Merge another VirtualDirectory

Load/Save:

Method	Description
`vdir.saveToDisk(dir)`	Write all files to disk
`vdir.loadFromDisk(dir)`	Load files from disk (replaces collection)

□□ TypeScript Support

Full TypeScript support with comprehensive type definitions:

```
import type { SmartFile, StreamFile, VirtualDirectory, SmartFileFactory } from
'@push.rocks/smartfile';

const processFile = async (file: SmartFile): Promise<void> => {
  const content = file.parseContentAsString();
  // TypeScript knows content is string
};
```

□□ Backward Compatibility

Version 12.0.0 introduces the factory pattern. Legacy exports are deprecated but still functional:

```
// ⚠□ Deprecated (still works, but will be removed)
import * as smartfile from '@push.rocks/smartfile';
const file = await smartfile.SmartFile.fromFilePath('./file.txt');
await smartfile.fs.copy('./a.txt', './b.txt');

// □ Recommended (new factory pattern)
import { SmartFileFactory } from '@push.rocks/smartfile';
```

```
const factory = SmartFileFactory.nodeFs();
const file = await factory.fromFilePath('./file.txt');

// For filesystem operations, use smartfs:
import { SmartFs, SmartFsProviderNode } from '@push.rocks/smartfs';
const smartFs = new SmartFs(new SmartFsProviderNode());
await smartFs.file('./a.txt').copy('./b.txt');
```

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.

changelog.md for @push.rocks/smartfile

2025-11-29 - 13.1.0 - feat(smartfs)

Integrate @push.rocks/smartfs and wire SmartFileFactory.nodeFs to return a real SmartFs instance

- Add runtime dependency @push.rocks/smartfs@^1.1.0 to package.json (previous peerDependency removed).
- Export smartfs from ts/plugins.ts so internal modules can access SmartFs and providers.
- Implement SmartFileFactory.nodeFs() to instantiate plugins.smartfs.SmartFs with SmartFsProviderNode and return a factory bound to that SmartFs.
- Enable factory-created SmartFile/StreamFile/VirtualDirectory instances to use the bound SmartFs for filesystem operations (read/write/streams).

2025-11-22 - 13.0.1 - fix(smartfile)

Stream and filesystem integrations: remove fs-extra, switch to fs/promises.rename, add Web WritableStream compatibility, and use smartFs recursive directory methods; update plugins exports and README.

- Remove fs-extra dependency and its types from package.json and stop exporting fsExtra from plugins.
- Replace fs-extra rename with fs/promises.rename in SmartFile.rename to use native promises API.
- Add Web WritableStream handling in StreamFile.writeToDisk (convert to Node.js Writable via Writable.fromWeb) and ensure Web ReadableStream conversion in createReadStream.
- Use smartFs.directory(...).recursive().create() when creating directories and smartFs.directory(path).recursive().list() when listing to ensure recursive behavior.
- Update README to add Issue Reporting and Security section pointing to community.foss.global for bug/vulnerability reports.

2025-11-22 - 13.0.0 - BREAKING CHANGE(SmartFileFactory)

Refactor to in-memory file API and introduce SmartFileFactory; delegate filesystem operations to @push.rocks/smartfs; bump to 12.0.0

- Introduce SmartFileFactory as the canonical entry point for creating SmartFile, StreamFile and VirtualDirectory instances.
- Refactor SmartFile, StreamFile and VirtualDirectory to be in-memory representations and accept an optional SmartFs instance for filesystem operations.
- Delegate low-level filesystem operations to @push.rocks/smartfs (added as a peerDependency); legacy fs/memory/fsStream/interpreter namespace exports removed/deprecated.
- Add StreamFile.toSmartFile(), VirtualDirectory.loadFromDisk(), and other convenience methods to work with the new factory/smartFs integration.
- Update tests to use a MockSmartFs and the factory API; add test assets.
- Documentation and readme updated with migration instructions and examples showing SmartFileFactory.nodeFs() and SmartFs usage.
- Bumped package version to 12.0.0 — this is a breaking change; consumers should migrate from legacy namespace exports to the factory + @push.rocks/smartfs workflow.

2025-08-18 - 11.2.7 - fix(ci)

Remove .npmrc containing hard-coded npm registry configuration

- Removed .npmrc which contained 'registry=https://registry.npmjs.org/'
- Avoids committing environment-specific npm registry configuration; rely on user or CI environment settings instead

2025-08-18 - 11.2.6 - fix(fs)

Improve fs and stream handling, enhance SmartFile/StreamFile, update tests and CI configs

- Fix listFileTree to correctly handle '**/' patterns by running both root and nested patterns and deduplicating results.
- Enhance waitForFileToBeReady: support directory paths (wait for first file) and improved file-stability checks with timeouts and retries.

- StreamFile improvements: support Web ReadableStream -> Node Readable conversion, better multi-use buffering, more robust fromUrl/fromBuffer/fromStream implementations, and accurate byte-length computation.
- SmartFile updates: switch fromUrl to SmartRequest, robust rename (optional on-disk rename), safer read/write paths and consistent Buffer handling for hashing and content edits.
- fs module tweaks: copy/copySync gained replaceTargetDir option, improved toObjectSync error messages, toReadStream now validates existence, and various synchronous/async API consistency fixes.
- memory module: consistent async/sync write APIs and smartfileArrayToFs formatting/behavior fixes.
- fsstream: improved stream processing and SmartReadStream robustness (error handling, read logic and watcher improvements).
- Tests: reformatted and strengthened tests (more explicit assertions, added tests for '**/*.ts**' **and** '/*' edge cases, updated imports to tapbundle usage).
- CI/workflows: updated IMAGE and NPMCI_COMPUTED_REPOURL values and switched npmci package install to @ship.zone/npmci in workflow files.
- package.json: bumped various dependencies, updated test script (timeout, logfile), added typings/main fields, homepage fix, pnpm overrides and minor metadata fixes.
- .gitignore: added entries for AI tool folders (.claude/, .serena/).
- Docs/readme: expanded README with clearer examples, features and TypeScript usage; updated readme hints for listFileTree behavior.

2025-05-26 - 11.2.5 - fix(dev)

Update dev dependencies and add local permission settings

- Bump @git.zone/tsbuild from 2.5.2 to 2.6.4
- Bump @git.zone/tstest from 1.9.0 to 2.2.5
- Add .claude/settings.local.json to configure permissions for Bash(pnpm test:*)

2025-05-24 - 11.2.4 - fix(config)

Add local permissions configuration for pnpm test commands in .claude/settings.local.json

- Introduced .claude/settings.local.json to allow Bash(pnpm test:*) permissions
- Ensured local testing commands have proper execution rights

2025-05-21 - 11.2.2 - fix(tests/settings)

Improve test assertions and update local settings permissions

- Refactor StreamFile tests to assert content string type using toBeTypeofString
- Update file existence tests to use resolves.toBeTrue and resolves.toBeFalse for cleaner promise handling
- Add .claude/settings.local.json to allow specific Bash permissions for pnpm test commands

2025-05-21 - 11.2.1 - fix(fs)

Fix inconsistent glob matching in listFileTree and update test imports and dependency versions for enhanced stability.

- Enhanced listFileTree to support **/ patterns by using dual patterns (root and nested) with deduplication.
- Updated test imports to use '@git.zone/tstest/tapbundle' for consistency across test files.
- Bumped dependency versions (@push.rocks/lik, smartpromise, smartrequest, glob) in package.json.
- Added npm configuration (.npmrc) and local settings for improved test verbosity.

2025-01-29 - 11.2.0 - feat(fs)

Enhanced copy method with optional replaceTargetDir option for directory replacement

- Added optional 'replaceTargetDir' option to 'copy' and 'copySync' methods in 'fs.ts'.
- The 'replaceTargetDir' option allows replacing the target directory if both source and target are directories.

2025-01-29 - 11.1.9 - fix(fs)

Fix directory handling in copy and copySync functions

- Ensured existing directories at destination are removed before copying over them in async copy.
- Added a similar check and handling for synchronous copySync when destination is a directory.

2025-01-29 - 11.1.8 - fix(fs)

Fixed copy and copySync functions to ensure they always overwrite files.

- Fixed bug in copy function where files were not being overwritten when they already existed at the destination.
- Fixed bug in copySync function to ensure files are overwritten to match the async function's behavior.

2025-01-29 - 11.1.7 - fix(fs)

Refactor copy and copySync functions to simplify return type

- Changed the return type of fs.copy and fs.copySync from boolean to void.
- Removed unnecessary promise handling in fs.copy.

2025-01-29 - 11.1.6 - fix(fs)

Fix issues with fs file copy functions.

- Updated dependencies in package.json.
- Corrected comments for asynchronous and synchronous file copy functions in fs.ts.

2025-01-07 - 11.1.5 - fix(fs)

Improve waitForFileToBeReady function to handle directories and file stabilization

- Enhanced the waitForFileToBeReady to handle directory paths by checking for file existence within directories and waiting for stabilization.
- Modified the watcher logic to cater to changes when monitoring directories for file appearance.

- Introduced a helper function to ensure paths exist and another to resolve the first file in directories.
- Corrected logic for polling and stabilizing files within directories.

2025-01-07 - 11.1.4 - fix(fs)

Fix file existence check in waitForFileToBeReady method.

- Ensured that the directory and file exist before setting up the watcher in waitForFileToBeReady.
- Changed ensureDirectoryExists to ensureFileExists for correct file path verification.
- Handled ENOENT errors correctly to retry file existence checks until timeout is reached.

2025-01-07 - 11.1.3 - fix(fs)

Fix TypeScript type issue in fs module

- Corrected a TypeScript type in the fs module's checkFileStability function.

2025-01-07 - 11.1.2 - fix(fs)

Fix issues in file stability check and directory existence verification in fs module

- Removed unused variable 'isFileAvailable' in 'waitForFileToBeReady'.
- Fixed logic for ensuring the target directory exists before setting up file stability watcher.
- Refactored directory existence logic into 'ensureDirectoryExists' function.

2025-01-07 - 11.1.1 - fix(fs)

Improve waitForFileToBeReady function for file stability detection

- Enhanced error handling and file stability checks in waitForFileToBeReady function
- Added timeout feature for file readiness check
- Improved directory access check before file availability check

2025-01-07 - 11.1.0 - feat(SmartFile)

Add rename functionality to SmartFile class

- Implemented a new method to rename a file within the SmartFile class.
- The rename method updates the file path and optionally writes the renamed file to the disk.

2024-12-15 - 11.0.23 - fix(fs)

Handle errors in toObjectSync method

- Added error handling in toObjectSync function to capture and provide more informative error messages.

2024-06-23 - 11.0.22 - fix(core)

Update dependencies and changelog

- Updated @push.rocks/smartstream to ^3.0.44
- Updated glob to ^10.4.2
- Updated @types/node to ^20.14.8

2024-06-23 - 11.0.21 - fix(dependencies)

Update dependencies to latest versions

- Updated @push.rocks/smartpromise to ^4.0.4
- Updated @push.rocks/smartstream to ^3.0.44
- Updated glob to ^10.4.2
- Updated @types/node to ^20.14.8

2024-06-07 - 11.0.20 - Changelog

11.0.20

2024-06-07 - 11.0.19 - fix(core): update

11.0.19

- fix(core): update

2024-06-07 - 11.0.18 - fix(core): update

11.0.18

- fix(core): update

2024-06-06 - 11.0.17 - fix(core): update

11.0.17

- fix(core): update

2024-06-06 - 11.0.16 - fix(core): update

11.0.16

- fix(core): update

2024-05-29 - 11.0.16 - update description

11.0.16

- update description

2024-05-17 - 11.0.15 - fix(core): update

11.0.15

- fix(core): update

2024-04-14 - 11.0.14 - update tsconfig

11.0.14

- update tsconfig

2024-04-12 - 11.0.13 - fix(core): update

11.0.13

- fix(core): update

2024-04-12 - 11.0.12 - fix(core): update

11.0.12

- fix(core): update

2024-04-12 - 11.0.11 - fix(core): update

11.0.11

- fix(core): update

2024-04-03 - 11.0.10 - fix(core): update

11.0.10

- fix(core): update

2024-04-03 - 11.0.9 - fix(core): update

11.0.9

- fix(core): update

2024-04-02 - 11.0.8 - fix(core): update

11.0.8

- fix(core): update

2024-04-02 - 11.0.7 - fix(core): update

11.0.7

- fix(core): update

2024-04-02 - 11.0.6 - fix(core): update

11.0.6

- fix(core): update

2024-04-01 - 11.0.5 - update npmextra.json

11.0.5

- update npmextra.json: githost

2024-04-01 - 11.0.4 - fix(core): update

11.0.4

- fix(core): update

2023-11-24 - 11.0.3 - fix(core): update

11.0.3

- fix(core): update

2023-11-07 - 11.0.2 - fix(core): update

11.0.2

- fix(core): update

2023-11-07 - 11.0.1 - fix(core): update

11.0.1

- fix(core): update

2023-11-06 - 11.0.0 - fix(core): update

11.0.0

- fix(core): update

2023-11-06 - 10.0.40 - BREAKING CHANGE(core): update

10.0.40

- BREAKING CHANGE(core): update

2023-11-04 - 10.0.39 - fix(core): update

10.0.39

- fix(core): update

2023-11-04 - 10.0.38 - fix(core): update

10.0.38

- fix(core): update

2023-11-04 - 10.0.37 - fix(core): update

10.0.37

- fix(core): update

2023-11-03 - 10.0.36 - fix(core): update

10.0.36

- fix(core): update

2023-11-03 - 10.0.35 - fix(core): update

10.0.35

- fix(core): update

2023-11-03 - 10.0.34 - fix(core): update

10.0.34

- fix(core): update

2023-11-03 - 10.0.33 - fix(core): update

10.0.33

- fix(core): update

2023-10-12 - 10.0.32 - fix(core): update

10.0.32

- fix(core): update

2023-09-22 - 10.0.31 - fix(core): update

10.0.31

- fix(core): update

2023-08-31 - 10.0.30 - fix(core): update

10.0.30

- fix(core): update

2023-08-23 - 10.0.29 - fix(core): update

10.0.29

- fix(core): update

2023-07-12 - 10.0.28 - fix(core): update

10.0.28

- fix(core): update

2023-07-10 - 10.0.27 - fix(core): update

10.0.27

- fix(core): update

2023-07-10 - 10.0.26 - fix(core): update

10.0.26

- fix(core): update

2023-07-08 - 10.0.25 - fix(core): update

10.0.25

- fix(core): update

2023-06-25 - 10.0.24 to 10.0.14 - Series of Fixes

10.0.24 to 10.0.14

- Series of fixes in the core module

2023-01-09 - 10.0.13 to 10.0.6 - Series of Fixes

10.0.13 to 10.0.6

- Series of fixes in the core module

2022-09-05 - 10.0.5 to 10.0.3 - Series of Fixes

10.0.5 to 10.0.3

- Series of fixes in the core module

2022-06-07 - 10.0.2 to 10.0.1 - Series of Fixes

10.0.2 to 10.0.1

- Series of fixes in the core module

2022-06-07 - 9.0.7 - BREAKING CHANGE(core): switch to esm

9.0.7

- BREAKING CHANGE(core): switch to esm

2022-03-11 - 9.0.6 to 9.0.2 - Series of Fixes

9.0.6 to 9.0.2

- Series of fixes in the core module

2021-12-01 - 9.0.1 - fix(core): update

9.0.1

- fix(core): update

2021-12-01 - 9.0.0 - fix(absolute pathing)

9.0.0

- add functions for easily getting absolute paths

2021-11-30 - 8.0.11 - BREAKING CHANGE(relative pathing)

8.0.11

- improved relative pathing

2020-08-10 - 8.0.10 to 7.0.12 - Series of Fixes and Updates

8.0.10 to 7.0.12

- Series of fixes in the core module
- BREAKING CHANGE(Smartfile class): switch to a Buffer-only approach

2019-02-17 - 7.0.0 - fix(core): update dependencies

7.0.0

- fix(core): update dependencies

2019-01-27 - 6.0.12 - BREAKING CHANGE(smartfile.fs.fileExists)

6.0.12

- now returns a Promise

2018-08-19 - 6.0.11 to 6.0.6 - Series of Fixes

6.0.11 to 6.0.6

- Series of fixes in core and dependencies

2018-07-03 - 6.0.5 to 5.0.0 - Series of Fixes

6.0.5 to 5.0.0

- Series of fixes in core and dependencies

2018-02-16 - 4.2.28 - BREAKING CHANGE(scope)

4.2.28

- switch to pushrocks scope