

@push.rocks/smartarchive

A library for working with archive files, providing utilities for compressing and decompressing data.

- [readme.md for @push.rocks/smartarchive](#)
- [changelog.md for @push.rocks/smartarchive](#)











readme.md for @push.rocks/smartarchive

A powerful, streaming-first archive manipulation library with a fluent builder API. Works seamlessly in **Node.js**, **Deno**, and **browsers**.

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.

Features

-  **Multi-format support** - Handle `.zip`, `.tar`, `.tar.gz`, `.tgz`, `.gz`, and `.bz2` archives
-  **Streaming-first architecture** - Process large archives without memory constraints
-  **Fluent builder API** - Chain methods for readable, expressive code
-  **Smart detection** - Automatically identifies archive types via magic bytes
-  **High performance** - Built on `modern-tar` and `fflate` for speed
-  **Flexible I/O** - Work with files, URLs, streams, and buffers seamlessly
-  **Modern TypeScript** - Full type safety and excellent IDE support
-  **Dual-mode operation** - Extract existing archives OR create new ones
-  **Cross-runtime** - Works in Node.js, Deno, and browsers
-  **Browser-ready** - Dedicated browser bundle with zero Node.js dependencies

Installation

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

```
# Using npm
npm install @push.rocks/smartarchive

# Using yarn
yarn add @push.rocks/smartarchive
```

Quick Start

Extract an archive from URL

```
import { SmartArchive } from '@push.rocks/smartarchive';

// Extract a .tar.gz archive from a URL directly to the filesystem
await SmartArchive.create()
  .url('https://registry.npmjs.org/some-package/-/some-package-1.0.0.tgz')
  .extract('./extracted');
```

Create an archive from entries

```
import { SmartArchive } from '@push.rocks/smartarchive';

// Create a tar.gz archive with files
await SmartArchive.create()
  .format('tar.gz')
  .compression(6)
  .entry('config.json', JSON.stringify({ name: 'myapp' }))
  .entry('readme.txt', 'Hello World!')
  .toFile('./backup.tar.gz');
```

Extract with filtering and path manipulation

```
import { SmartArchive } from '@push.rocks/smartarchive';

// Extract only JSON files, stripping the first path component
await SmartArchive.create()
  .url('https://registry.npmjs.org/lodash/-/lodash-4.17.21.tgz')
  .stripComponents(1)           // Remove 'package/' prefix
  .include(/\.json$/)          // Only extract JSON files
  .extract('./node_modules/lodash');
```

Browser Usage

smartarchive provides a dedicated browser-compatible bundle with no Node.js dependencies:

```
// Import from the /web subpath for browser environments
import { TarTools, ZipTools, GzipTools, Bzip2Tools } from '@push.rocks/smartarchive/web';

// Create a TAR archive in the browser
const tarTools = new TarTools();
const tarBuffer = await tarTools.packFiles([
  { archivePath: 'hello.txt', content: 'Hello from the browser!' },
  { archivePath: 'data.json', content: JSON.stringify({ browser: true }) }
]);

// Create a TAR.GZ archive
const tgzBuffer = await tarTools.packFilesToTarGz([
  { archivePath: 'file.txt', content: 'Compressed!' }
], 6);

// Extract a TAR archive
const entries = await tarTools.extractTar(tarBuffer);
for (const entry of entries) {
  console.log(`${entry.path}: ${entry.content.length} bytes`);
}

// Work with ZIP files
const zipTools = new ZipTools();
const zipBuffer = await zipTools.createZip([
  { archivePath: 'doc.txt', content: 'Document content' }
]);
```

```
], 6);

const zipEntries = await zipTools.extractZip(zipBuffer);

// GZIP compression
const gzipTools = new GzipTools();
const compressed = gzipTools.compressSync(new TextEncoder().encode('Hello World'), 6);
const decompressed = gzipTools.decompressSync(compressed);
```

Browser Bundle Exports

The `/web` subpath exports these browser-compatible tools:

Export	Description
<code>TarTools</code>	Create and extract TAR and TAR.GZ archives
<code>ZipTools</code>	Create and extract ZIP archives
<code>GzipTools</code>	GZIP compression and decompression
<code>Bzip2Tools</code>	BZIP2 decompression (extraction only)

📌 **Note:** The browser bundle does **not** include `SmartArchive` (which requires filesystem access). Use the individual tool classes for browser applications.

Core Concepts 📖

Fluent Builder Pattern

`SmartArchive` uses a fluent builder pattern where you chain methods to configure the operation:

```
SmartArchive.create() // Start a new builder
  .source(...)        // Configure source (extraction mode)
  .options(...)        // Set options
  .terminal()          // Execute the operation
```

Two Operating Modes

Extraction Mode - Load an existing archive and extract/analyze it:

```
SmartArchive.create()  
  .url('...') // or .file(), .stream(), .buffer()  
  .extract('./out') // or .toSmartFiles(), .list(), etc.
```

Creation Mode - Build a new archive from entries:

```
SmartArchive.create()  
  .format('tar.gz') // Set output format  
  .entry(...) // Add files  
  .toFile('./out.tar.gz') // or .toBuffer(), .toStream()
```

⚠ **Note:** You cannot mix extraction and creation methods in the same chain.

API Reference

Source Methods (Extraction Mode)

Method	Description
<code>.url(url)</code>	Load archive from a URL
<code>.file(path)</code>	Load archive from local filesystem
<code>.stream(readable)</code>	Load archive from any Node.js readable stream
<code>.buffer(buffer)</code>	Load archive from an in-memory Buffer

Creation Methods (Creation Mode)

Method	Description
<code>.format(fmt)</code>	Set output format: <code>'tar'</code> , <code>'tar.gz'</code> , <code>'tgz'</code> , <code>'zip'</code> , <code>'gz'</code>
<code>.compression(level)</code>	Set compression level (0-9, default: 6)

Method	Description
<code>.entry(path, content)</code>	Add a file entry (string or Buffer content)
<code>.entries(array)</code>	Add multiple entries at once
<code>.directory(path, archiveBase?)</code>	Add entire directory contents
<code>.addSmartFile(file, path?)</code>	Add a SmartFile instance
<code>.addStreamFile(file, path?)</code>	Add a StreamFile instance

Filter Methods (Both Modes)

Method	Description
<code>.filter(predicate)</code>	Filter entries with custom function
<code>.include(pattern)</code>	Only include entries matching regex/string pattern
<code>.exclude(pattern)</code>	Exclude entries matching regex/string pattern

Extraction Options

Method	Description
<code>.stripComponents(n)</code>	Strip N leading path components
<code>.overwrite(bool)</code>	Overwrite existing files (default: false)
<code>.fileName(name)</code>	Set output filename for single-file archives (gz, bz2)

Terminal Methods (Extraction)

Method	Returns	Description
<code>.extract(targetDir)</code>	<code>Promise<void></code>	Extract to filesystem directory
<code>.toStreamFiles()</code>	<code>Promise<StreamIntake<StreamFile>></code>	Get stream of StreamFile objects
<code>.toSmartFiles()</code>	<code>Promise<SmartFile[]></code>	Get in-memory SmartFile array
<code>.extractFile(path)</code>	<code>Promise<SmartFile null></code>	Extract single file by path
<code>.list()</code>	<code>Promise<IArchiveEntryInfo[]></code>	List all entries
<code>.analyze()</code>	<code>Promise<IArchiveInfo></code>	Get archive metadata
<code>.hasFile(path)</code>	<code>Promise<boolean></code>	Check if file exists

Terminal Methods (Creation)

Method	Returns	Description
<code>.build()</code>	<code>Promise<SmartArchive></code>	Build the archive (implicit in other terminals)
<code>.toBuffer()</code>	<code>Promise<Buffer></code>	Get archive as Buffer
<code>.toFile(path)</code>	<code>Promise<void></code>	Write archive to disk
<code>.toStream()</code>	<code>Promise<Readable></code>	Get raw archive stream

Usage Examples

Download and extract npm packages

```
import { SmartArchive } from '@push.rocks/smartarchive';

const pkg = await SmartArchive.create()
  .url('https://registry.npmjs.org/lodash/-/lodash-4.17.21.tgz');

// Quick inspection of package.json
const pkgJson = await pkg.extractFile('package/package.json');
if (pkgJson) {
  const metadata = JSON.parse(pkgJson.contents.toString());
  console.log(`Package: ${metadata.name}@${metadata.version}`);
}

// Full extraction with path normalization
await SmartArchive.create()
  .url('https://registry.npmjs.org/lodash/-/lodash-4.17.21.tgz')
  .stripComponents(1)
  .extract('./node_modules/lodash');
```

Create ZIP archive

```
import { SmartArchive } from '@push.rocks/smartarchive';

await SmartArchive.create()
  .format('zip')
  .compression(9)
  .entry('report.txt', 'Monthly sales report...')
  .entry('data/figures.json', JSON.stringify({ revenue: 10000 }))
  .entry('images/logo.png', pngBuffer)
  .toFile('./report-bundle.zip');
```

Create TAR.GZ from directory

```
import { SmartArchive } from '@push.rocks/smartarchive';

await SmartArchive.create()
  .format('tar.gz')
  .compression(9)
  .directory('./src', 'source') // Archive ./src as 'source/' in archive
  .toFile('./project-backup.tar.gz');
```

Stream-based extraction

```
import { SmartArchive } from '@push.rocks/smartarchive';

const fileStream = await SmartArchive.create()
  .file('./large-archive.tar.gz')
  .toStreamFiles();

fileStream.on('data', async (streamFile) => {
  console.log(`Processing: ${streamFile.relativeFilePath}`);

  if (streamFile.relativeFilePath.endsWith('.json')) {
    const content = await streamFile.getContentAsBuffer();
    const data = JSON.parse(content.toString());
    // Process JSON data...
  }
});
```

```
fileStream.on('end', () => {
  console.log('Extraction complete');
});
```

Filter specific file types

```
import { SmartArchive } from '@push.rocks/smartarchive';

// Extract only TypeScript files
const tsFiles = await SmartArchive.create()
  .url('https://example.com/project.tar.gz')
  .include(/\.ts$/)
  .exclude(/node_modules/)
  .toSmartFiles();

for (const file of tsFiles) {
  console.log(`${file.relative}: ${file.contents.length} bytes`);
}
```

Analyze archive without extraction

```
import { SmartArchive } from '@push.rocks/smartarchive';

const archive = SmartArchive.create()
  .file('./unknown-archive.tar.gz');

// Get format info
const info = await archive.analyze();
console.log(`Format: ${info.format}`);
console.log(`Compressed: ${info.isCompressed}`);

// List contents
const entries = await archive.list();
for (const entry of entries) {
  console.log(`${entry.path} (${entry.isDirectory ? 'dir' : 'file'})`);
}
```

```
// Check for specific file
if (await archive.hasFile('package.json')) {
  const pkgFile = await archive.extractFile('package.json');
  console.log(pkgFile?.contents.toString());
}
```

Working with GZIP files

```
import { SmartArchive, GzipTools } from '@push.rocks/smartarchive';

// Decompress a .gz file
await SmartArchive.create()
  .file('./data.json.gz')
  .fileName('data.json') // Specify output name (gzip doesn't store filename)
  .extract('./decompressed');

// Use GzipTools directly for compression/decompression
const gzipTools = new GzipTools();

// Compress a buffer (sync and async available)
const input = new TextEncoder().encode('Hello World');
const compressed = gzipTools.compressSync(input, 9);
const decompressed = gzipTools.decompressSync(compressed);

// Async versions (internally use sync for cross-runtime compatibility)
const compressedAsync = await gzipTools.compress(input, 6);
const decompressedAsync = await gzipTools.decompress(compressedAsync);
```

Working with TAR archives directly

```
import { TarTools } from '@push.rocks/smartarchive';

const tarTools = new TarTools();

// Create a TAR archive from entries (buffer-based, good for small files)
const tarBuffer = await tarTools.packFiles([
```

```

    { archivePath: 'hello.txt', content: 'Hello, World!' },
    { archivePath: 'data.json', content: JSON.stringify({ foo: 'bar' }) }
  ]);

// Create a TAR.GZ archive
const tgzBuffer = await tarTools.packFilesToTarGz([
  { archivePath: 'file.txt', content: 'Compressed content' }
], 6);

// Extract a TAR archive
const entries = await tarTools.extractTar(tarBuffer);
for (const entry of entries) {
  console.log(`${entry.path}: ${entry.isDirectory ? 'dir' : 'file'}`);
}

// Extract a TAR.GZ archive
const tgzEntries = await tarTools.extractTarGz(tgzBuffer);

// Node.js only: Pack a directory (buffer-based)
const dirBuffer = await tarTools.packDirectory('./src');
const dirTgzBuffer = await tarTools.packDirectoryToTarGz('./src', 9);

```

Streaming TAR for Large Files (Node.js only)

For large files that don't fit in memory, use the streaming APIs:

```

import { TarTools } from '@push.rocks/smartarchive';
import * as fs from 'fs';

const tarTools = new TarTools();

// ===== STREAMING PACK =====
// Create a TAR pack stream - files are processed one at a time
const pack = tarTools.getPackStream();

// Add files with streaming content (requires size for streams)
await tarTools.addFileToPack(pack, {

```

```
    fileName: 'small.txt',
    content: 'Hello World' // Strings and buffers auto-detect size
  });

await tarTools.addFileToPack(pack, {
  fileName: 'large-video.mp4',
  content: fs.createReadStream('./video.mp4'),
  size: fs.statSync('./video.mp4').size // Size required for streams
});

pack.finalize();
pack.pipe(fs.createWriteStream('output.tar'));

// ===== STREAMING DIRECTORY PACK =====
// Pack entire directory with true streaming (no buffering)
const tarStream = await tarTools.getDirectoryPackStream('./large-folder');
tarStream.pipe(fs.createWriteStream('backup.tar'));

// With GZIP compression
const tgzStream = await tarTools.getDirectoryPackStreamGz('./large-folder', 6);
tgzStream.pipe(fs.createWriteStream('backup.tar.gz'));

// ===== STREAMING EXTRACT =====
// Extract large archives without loading into memory
const extract = tarTools.getExtractStream();

extract.on('entry', (header, stream, next) => {
  console.log(`Extracting: ${header.name} (${header.size} bytes)`);

  const writeStream = fs.createWriteStream(`./out/${header.name}`);
  stream.pipe(writeStream);
  writeStream.on('finish', next);
});

extract.on('finish', () => console.log('Extraction complete'));

fs.createReadStream('large-archive.tar').pipe(extract);

// Or use the convenient directory extraction
```

```
await tarTools.extractToDirectory(  
  fs.createReadStream('archive.tar'),  
  './output-folder'  
);
```

Working with ZIP archives directly

```
import { ZipTools } from '@push.rocks/smartarchive';  
  
const zipTools = new ZipTools();  
  
// Create a ZIP archive from entries  
const zipBuffer = await zipTools.createZip([  
  { archivePath: 'readme.txt', content: 'Hello!' },  
  { archivePath: 'data.bin', content: new Uint8Array([0x00, 0x01, 0x02]) }  
], 6);  
  
// Extract a ZIP buffer  
const entries = await zipTools.extractZip(zipBuffer);  
for (const entry of entries) {  
  console.log(`${entry.path}: ${entry.content.length} bytes`);  
}
```

In-memory round-trip

```
import { SmartArchive } from '@push.rocks/smartarchive';  
  
// Create archive in memory  
const archive = await SmartArchive.create()  
  .format('tar.gz')  
  .entry('config.json', JSON.stringify({ version: '1.0.0' })))  
  .build();  
  
const buffer = await archive.toBuffer();  
  
// Extract from buffer  
const files = await SmartArchive.create()
```

```
.buffer(buffer)
.toSmartFiles();

for (const file of files) {
  console.log(`${file.relative}: ${file.contents.toString()}`);
}
```

Real-World Use Cases

CI/CD: Download & Extract Build Artifacts

```
const artifacts = await SmartArchive.create()
  .url(`${CI_SERVER}/artifacts/build-${BUILD_ID}.zip`)
  .stripComponents(1)
  .extract('./dist');
```

Backup System

```
// Create backup
await SmartArchive.create()
  .format('tar.gz')
  .compression(9)
  .directory('./data')
  .toFile(`./backups/backup-${Date.now()}.tar.gz`);

// Restore backup
await SmartArchive.create()
  .file('./backups/backup-latest.tar.gz')
  .extract('/restore/location');
```

Bundle files for HTTP download

```
import { SmartArchive } from '@push.rocks/smartarchive';
```

```
// Express/Fastify handler
app.get('/download-bundle', async (req, res) => {
  const buffer = await SmartArchive.create()
    .format('zip')
    .entry('report.pdf', pdfBuffer)
    .entry('data.xlsx', excelBuffer)
    .entry('images/chart.png', chartBuffer)
    .toBuffer();

  res.setHeader('Content-Type', 'application/zip');
  res.setHeader('Content-Disposition', 'attachment; filename=report-bundle.zip');
  res.send(buffer);
});
```

Data Pipeline: Process Compressed Datasets

```
const fileStream = await SmartArchive.create()
  .url('https://data.source/dataset.tar.gz')
  .toStreamFiles();

fileStream.on('data', async (file) => {
  if (file.relativeFilePath.endsWith('.csv')) {
    const content = await file.getContentAsBuffer();
    // Stream CSV processing...
  }
});
```

Supported Formats

Format	Extension(s)	Extract	Create	Browser
TAR	<code>.tar</code>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
TAR.GZ / TGZ	<code>.tar.gz</code> , <code>.tgz</code>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ZIP	<code>.zip</code>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Format	Extension(s)	Extract	Create	Browser
GZIP	.gz	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
BZIP2	.bz2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Type Definitions

```
// Supported archive formats
type TArchiveFormat = 'tar' | 'tar.gz' | 'tgz' | 'zip' | 'gz' | 'bz2';

// Compression level (0 = none, 9 = maximum)
type TCompressionLevel = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9;

// Entry for creating archives
interface IArchiveEntry {
  archivePath: string;
  content: string | Buffer | Uint8Array | SmartFile | StreamFile;
  size?: number;
  mode?: number;
  mtime?: Date;
}

// Information about an archive entry
interface IArchiveEntryInfo {
  path: string;
  size: number;
  isDirectory: boolean;
  isFile: boolean;
  mtime?: Date;
  mode?: number;
}

// Archive analysis result
interface IArchiveInfo {
  format: TArchiveFormat | null;
  isCompressed: boolean;
  isArchive: boolean;
  entries?: IArchiveEntryInfo[];
```

```
}
```

Performance Tips

1. **Use streaming for large files** - `.toStreamFiles()` processes entries one at a time without loading the entire archive
2. **Choose appropriate compression** - Use 1-3 for speed, 6 (default) for balance, 9 for maximum compression
3. **Filter early** - Use `.include()` / `.exclude()` to skip unwanted entries before processing
4. **Use Uint8Array in browsers** - The browser bundle works with `Uint8Array` for optimal performance

Error Handling

```
import { SmartArchive } from '@push.rocks/smartarchive';

try {
  await SmartArchive.create()
    .url('https://example.com/file.zip')
    .extract('./output');
} catch (error) {
  if (error.message.includes('No source configured')) {
    console.error('Forgot to specify source');
  } else if (error.message.includes('No format specified')) {
    console.error('Forgot to set format for creation');
  } else if (error.message.includes('extraction mode')) {
    console.error('Cannot mix extraction and creation methods');
  } else {
    console.error('Archive operation failed:', error.message);
  }
}
```

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.

changelog.md for @push.rocks/smartarchive

2026-01-01 - 5.2.0 - feat(tartools)

add streaming TAR support (tar-stream), Node.js streaming APIs for TarTools, and browser / web bundle docs

- Add tar-stream runtime dependency and @types/tar-stream devDependency
- Introduce streaming TarTools APIs: getPackStream, addFileToPack, getExtractStream, extractToDirectory, getDirectoryPackStream, getDirectoryPackStreamGz
- Switch SmartArchive TAR extraction to use tar-stream extract for true streaming ingestion of entries
- Export tarStream in plugins and export ITarPackFileOptions from the Node.js entrypoint
- Update packDirectory/packDirectoryToTarGz to handle files safely and use fflate.gzipSync for buffer-based gzipping
- README updates: document /web browser bundle, browser usage examples, Uint8Array guidance, updated feature table and streaming examples

2026-01-01 - 5.1.0 - feat(archive)

introduce ts_shared browser-compatible layer, refactor Node-specific tools to wrap/shared implementations, and modernize archive handling

- Split code into ts_shared (browser-compatible) and ts_web entrypoint; node-specific wrappers remain under ts/
- Switched TAR implementation from tar-stream to modern-tar and replaced stream-based TAR handling with Uint8Array-based pack/unpack
- Normalized shared APIs to use Uint8Array (instead of Buffer) for browser compatibility — callers may need to adapt Buffer/Uint8Array usage (breaking)
- Moved BZIP2, GZIP, ZIP, TAR logic into ts_shared and updated plugins to re-export shared plugins for web builds
- Adjusted classes.smartarchive to consume shared tools and convert between Buffer and Uint8Array where needed

- Added package.json exports for "." and "./web", bumped several dependency/devDependency versions, and added modern-tar and related changes
- Updated npmextra.json with new scoped configuration and release registries
- Removed pnpm-workspace.yaml entries and deleted several legacy Node-only files in favor of shared implementations

2025-11-25 - 5.0.1 - fix(ziptools,gziptools)

Use fflate synchronous APIs for ZIP and GZIP operations for Deno compatibility; add TEntryFilter type and small docs/tests cleanup

- Replace fflate async APIs (zip, unzip, gzip, gunzip with callbacks) with synchronous counterparts (zipSync, unzipSync, gzipSync, gunzipSync) to avoid Web Worker issues in Deno
- ZipCompressionStream.finalize now uses fflate.zipSync and emits compressed Buffer synchronously
- GzipTools.compress / decompress now delegate to compressSync / decompressSync for cross-runtime compatibility
- ZipTools.createZip and ZipTools.extractZip now use zipSync/unzipSync and return Buffers
- Add TEntryFilter type to ts/interfaces.ts for fluent API entry filtering
- Minor readme.hints.md updates and small whitespace tidy in tests

2025-11-25 - 5.0.0 - BREAKING CHANGE(SmartArchive)

Refactor public API: rename factory/extraction methods, introduce typed interfaces and improved compression tools

- Renamed SmartArchive factory methods: fromArchiveUrl -> fromUrl, fromArchiveFile -> fromFile, fromArchiveStream -> fromStream; added fromBuffer helper.
- Renamed extraction APIs: exportToFs -> extractToDirectory and exportToStreamOfStreamFiles -> extractToStream; stream-based helpers updated accordingly.
- Export surface reorganized (ts/index.ts): core interfaces and errors are exported and new modules (bzip2tools, archiveanalyzer) are publicly available.

- Introduced strong TypeScript types (ts/interfaces.ts) and centralized error types (ts/errors.ts) including Bzip2Error and BZIP2_ERROR_CODES.
- Refactored format implementations and stream transforms: GzipTools/GzipCompressionTransform/GzipDecompressionTransform, ZipTools (ZipCompressionStream, ZipDecompressionTransform), TarTools improvements.
- BZIP2 implementation improvements: new bit iterator (IBitReader), clearer error handling and streaming unbzip2 transform.
- Updated tests to use the new APIs and method names.
- Breaking change: public API method names and some class/transform names have changed — this requires code updates for consumers.

2025-11-25 - 4.2.4 - fix(plugings)

Migrate filesystem usage to Node fs/fsPromises and upgrade smartfile to v13; add listFileTree helper and update tests

- Bumped dependency @push.rocks/smartfile to ^13.0.0 and removed unused dependency `through`
- Replaced usages of smartfile.fs and smartfile.fsStream with Node native fs and fs/promises (createReadStream/createWriteStream, mkdir({recursive:true}), stat, readFile)
- Added plugins.listFileTree helper (recursive directory lister) and used it in TarTools.packDirectory and tests
- Updated SmartArchive.exportToFs to use plugins.fs and plugins.fsPromises for directory creation and file writes
- Updated TarTools to use plugins.fs.createReadStream and plugins.fsPromises.stat when packing directories
- Converted/updated tests to a Node/Deno-friendly test file (test.node+deno.ts) and switched test helpers to use fsPromises
- Added readme.hints.md with migration notes for Smartfile v13 and architecture/dependency notes

2025-11-25 - 4.2.3 - fix(build)

Upgrade dev tooling: bump @git.zone/tsbuild, @git.zone/tsrun and @git.zone/tstest versions

- Bump @git.zone/tsbuild from ^2.6.6 to ^3.1.0
- Bump @git.zone/tsrun from ^1.3.3 to ^2.0.0
- Bump @git.zone/tstest from ^2.3.4 to ^3.1.3

2025-08-18 - 4.2.2 - fix(smartarchive)

Improve tar entry streaming handling and add in-memory gzip/tgz tests

- Fix tar entry handling: properly consume directory entries (resume stream) and wait for entry end before continuing to next header
- Wrap tar file entries with a PassThrough so extracted StreamFile instances can be consumed while the tar extractor continues
- Handle nested archives correctly by piping resultStream -> decompressionStream -> analyzer -> unpacker, avoiding premature end signals
- Add and expand tests in test/test.gzip.ts: verify package.json and TS/license files after extraction, add in-memory gzip extraction test, and add real tgz-in-memory download+extraction test
- Minor logging improvements for tar extraction flow

2025-08-18 - 4.2.1 - fix(gzip)

Improve gzip streaming decompression, archive analysis and unpacking; add gzip tests

- Add a streaming DecompressGunzipTransform using fflate.Gunzip with proper _flush handling to support chunked gzip input and avoid buffering issues.
- Refactor ArchiveAnalyzer: introduce IAnalyzedResult, getAnalyzedStream(), and getDecompressionStream() to better detect mime types and wire appropriate decompression streams (gzip, zip, bzip2, tar).
- Use SmartRequest response streams converted via stream.Readable.fromWeb for URL sources in SmartArchive.getArchiveStream() to improve remote archive handling.
- Improve nested archive unpacking and SmartArchive export pipeline: more robust tar/zip handling, consistent SmartDuplex usage and backpressure handling.
- Enhance exportToFs: ensure directories, improved logging for relative paths, and safer write-stream wiring.
- Add comprehensive gzip-focused tests (test/test.gzip.ts) covering file extraction, stream extraction, header filename handling, large files, and a real-world tgz-from-URL extraction scenario.

2025-08-18 - 4.2.0 -

feat(classes.smartarchive)

Support URL streams, recursive archive unpacking and filesystem export; improve ZIP/GZIP/BZIP2 robustness; CI and package metadata updates

- Add `exportToFs(targetDir, fileName?)` to write extracted `StreamFile` objects to the filesystem (ensures directories, logs relative paths, waits for write completion).
- Implement `exportToStreamOfStreamFiles` with recursive unpacking pipeline that handles `application/x-tar` (tar-stream Extract), `application/zip` (fflate Unzip), nested archives and `StreamIntake` for `StreamFile` results.
- Enhance `getArchiveStream()` to support URL/web streams (`SmartRequest`) and return Node Readable streams for remote archives.
- Make ZIP decompression more robust: accept `ArrayBuffer`-like chunks, coerce to `Buffer` before pushing to `fflate.Unzip`, and ensure `SmartDuplex` handling of results.
- Fixes and improvements to `bzip2/gzip/tar` tool implementations (various bug/formatting fixes, safer CRC and stream handling).
- Update CI workflows to use new registry image and adjust `npmci` install path; minor `.gitignore` additions.
- Package metadata tweaks: bugs URL and homepage updated, `packageManager/pnpm` fields adjusted.
- Documentation/readme expanded and polished with quick start, examples and API reference updates.
- Small test and plugin export cleanups (formatting and trailing commas removed/added).
- TypeScript/formatting fixes across many files (consistent casing, trailing commas, typings, `tsconfig` additions).

2025-08-18 - 4.1.0 -

feat(classes.smartarchive)

Support URL web streams, add recursive archive unpacking and filesystem export, and improve ZIP decompression robustness

- `ts/classes.smartarchive.ts`: add `exportToFs(targetDir, fileName?)` to write extracted `StreamFile` objects to the filesystem (ensures directories, logs relative paths, waits for write completion).
- `ts/classes.smartarchive.ts`: implement `exportToStreamOfStreamFiles` with recursive unpacking pipeline that handles `application/x-tar` (tar-stream Extract), `application/zip`

(fflate unzip), nested archives and StreamIntake for StreamFile results.

- ts/classes.smartarchive.ts: improve getArchiveStream() for URL sources by using SmartRequest.create().url(...).get() and converting the returned Web stream into a Node Readable stream.
- ts/classes.ziptools.ts: make ZIP decompression writeFunction more robust — accept non-Buffer chunks, coerce to Buffer before pushing to fflate.Unzip, and loosen the writeFunction typing to handle incoming ArrayBuffer-like data.

2024-10-13 - 4.0.39 - fix(core)

Fix dependencies and update documentation.

- Ensure package uses the latest dependencies
- Reviewed and grouped imports in TypeScript files
- Updated readme with advanced usage examples

2024-10-13 - 4.0.38 - fix(dependencies)

Update dependencies to latest versions

- Updated @push.rocks/smartfile to version 11.0.21
- Updated @push.rocks/smartpromise to version 4.0.4
- Updated @push.rocks/smartstream to version 3.0.46
- Updated @push.rocks/smarturl to version 3.1.0
- Updated file-type to version 19.5.0
- Updated @git.zone/tsbuild to version 2.1.84
- Updated @git.zone/tsrun to version 1.2.49
- Updated @push.rocks/tapbundle to version 5.3.0

2024-06-08 - 4.0.24 to 4.0.37 - Fixes and Updates

Core updates and bug fixes were implemented in versions 4.0.24 through 4.0.37.

- Repeated core updates and fixes applied consistently across multiple versions.

2024-06-06 - 4.0.22 to 4.0.23 - Descriptions and Fixes Updates

Efforts to update documentation and core features.

- "update description" in 4.0.22
- Updates to `tsconfig` and `npmextra.json` were performed.
- Ongoing core fixes.

2023-11-06 - 4.0.0 - Major Update with Breaking Changes

Introduction of significant updates and breaking changes.

- Transition to new version 4.0.0 with core updates.
- Break in compatibility due to major structural changes with core functionalities.

2023-07-11 - 3.0.6 - Organizational Changes

Structural reorganization and updates to the organization schema.

- Switch to new organizational schema implemented.

2022-04-04 - 3.0.0 - Build Updates and Breaking Changes

Major build update introducing breaking changes.

- Introduction of ESM structure with breaking changes.

2016-01-18 - 0.0.0 to 1.0.0 - Initial Development and Launch

Initial software development and establishment of core features.

- Project set-up including Travis CI integration.
- Launch of the first full version with code restructuring.
- Added callback support.