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
`TarTools`	Create and extract TAR and TAR.GZ archives
`ZipTools`	Create and extract ZIP archives
`GzipTools`	GZIP compression and decompression
`Bzip2Tools`	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
`.url(url)`	Load archive from a URL
`.file(path)`	Load archive from local filesystem
`.stream(readable)`	Load archive from any Node.js readable stream
`.buffer(buffer)`	Load archive from an in-memory Buffer

Creation Methods (Creation Mode)

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

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

Filter Methods (Both Modes)

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

Extraction Options

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

Terminal Methods (Extraction)

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

Terminal Methods (Creation)

Method	Returns	Description
`.build()`	`Promise<SmartArchive>`	Build the archive (implicit in other terminals)
`.toBuffer()`	`Promise<Buffer>`	Get archive as Buffer
`.toFile(path)`	`Promise<void>`	Write archive to disk
`.toStream()`	`Promise<Readable>`	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	`.tar`	□	□	□
TAR.GZ / TGZ	`.tar.gz`, `.tgz`	□	□	□
ZIP	`.zip`	□	□	□

Format	Extension(s)	Extract	Create	Browser
GZIP	`.gz`	✅	✅	✅
BZIP2	`.bz2`	✅	✅	✅

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.

Revision #3
Created 2026-03-28 11:09:58 UTC by foss.global Team
Updated 2026-03-28 12:16:36 UTC by foss.global Team