@push.rocks/smartffmpeg

smart and performant cross-platform format conversion

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

readme.md for @push.rocks/smartffmpeg

A powerful, modern Node.js wrapper for FFmpeg with a fluent builder API, memory stream support, and zero filesystem dependencies for in-memory operations. □□

Features

- □□ **Fluent Builder API** – Chain methods for clean, readable code
- □□ **Memory Streams** – Process media directly from/to Buffers without touching the filesystem
- □□ **Progress Tracking** – Real-time progress callbacks with percentage, fps, bitrate, and speed
- □□ **TypeScript First** – Full type safety with comprehensive interfaces
- □□ **Bundled Binaries** – Ships with `ffmpeg-static` and `ffprobe-static` for zero-config setup
- □□ **Web Streams Support** – Native `ReadableStream` and `WritableStream` compatibility
- □□ **Dual API** – Modern builder API + legacy methods for backward compatibility

Install

```
pnpm install @push.rocks/smartffmpeg
# or
npm install @push.rocks/smartffmpeg
```

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.

Usage

Quick Start

```
import { SmartFfmpeg } from '@push.rocks/smartffmpeg';

const ffmpeg = new SmartFfmpeg();

// Convert a video with the fluent API
await ffmpeg.input('/path/to/input.mp4')
  .videoCodec('libx264')
  .audioCodec('aac')
  .videoBitrate('2M')
  .audioBitrate('128k')
  .size(1920, 1080)
  .crf(23)
  .preset('fast')
  .output('/path/to/output.mp4')
  .run();
```

Builder API (Recommended) □□

The modern builder API provides a fluent, chainable interface for constructing FFmpeg commands:

```
const ffmpeg = new SmartFfmpeg();

// File-to-file conversion with progress
await ffmpeg.input('/path/to/input.mp4')
  .videoCodec('libx264')
  .audioBitrate('128k')
  .crf(23)
  .onProgress(progress => {
    console.log(`□ Progress: ${progress.percent?.toFixed(1)}%`);
    console.log(`   Speed: ${progress.speed}, FPS: ${progress.fps}`);
  })
  .output('/path/to/output.mp4')
```

```
  .run();
```

Memory Stream Support □□

Process media entirely in memory – perfect for serverless functions, APIs, and pipelines:

```
import fs from 'fs/promises';

// Buffer → Buffer conversion
const inputBuffer = await fs.readFile('input.mp4');
const outputBuffer = await ffmpeg.input(inputBuffer, { format: 'mp4' })
  .videoCodec('libx264')
  .toBuffer('webm');

// File → Buffer
const buffer = await ffmpeg.input('/path/to/input.mp4')
  .videoCodec('libx264')
  .toBuffer('mp4');

// Buffer → File
await ffmpeg.input(inputBuffer, { format: 'mp4' })
  .videoCodec('libx264')
  .output('/path/to/output.mp4')
  .run();

// Get a Web ReadableStream
const stream = ffmpeg.input('/path/to/input.mp4')
  .videoCodec('libx264')
  .toStream('mp4');

// Pipe to a Web WritableStream
await ffmpeg.input('/path/to/input.mp4')
  .videoCodec('libx264')
  .pipe(writableStream, 'mp4');

// Web ReadableStream input (e.g., from fetch response)
const response = await fetch('https://example.com/video.mp4');
const webStream = response.body; // ReadableStream<Uint8Array>
const convertedBuffer = await ffmpeg.input(webStream, { format: 'mp4' })
```

```
  .videoCodec('libx264')
  .toBuffer('webm');

// Uint8Array input
const uint8Data = new Uint8Array(videoBytes);
const output = await ffmpeg.input(uint8Data, { format: 'mp4' })
  .videoCodec('libx264')
  .toBuffer('webm');
```

Video Operations □□

```
// Scale/resize video
await ffmpeg.input('input.mp4')
  .scale(1280, 720)
  .output('720p.mp4')
  .run();

// Crop video (width, height, x, y)
await ffmpeg.input('input.mp4')
  .crop(640, 480, 100, 50)
  .output('cropped.mp4')
  .run();

// Flip video
await ffmpeg.input('input.mp4')
  .flipHorizontal()
  .flipVertical()
  .output('flipped.mp4')
  .run();

// Change frame rate
await ffmpeg.input('input.mp4')
  .fps(30)
  .output('30fps.mp4')
  .run();

// Add padding
await ffmpeg.input('input.mp4')
```

```
  .pad(1920, 1080, 0, 0, 'black')
  .output('padded.mp4')
  .run();

// Rotate video
await ffmpeg.input('input.mp4')
  .rotate('PI/2')  // 90 degrees
  .output('rotated.mp4')
  .run();

// Custom video filter
await ffmpeg.input('input.mp4')
  .videoFilter('eq=brightness=0.1:saturation=1.5')
  .output('adjusted.mp4')
  .run();
```

Audio Operations □□

```
// Extract audio from video
await ffmpeg.input('video.mp4')
  .noVideo()
  .audioCodec('libmp3lame')
  .audioBitrate('320k')
  .format('mp3')
  .output('audio.mp3')
  .run();

// Remove audio from video
await ffmpeg.input('video.mp4')
  .noAudio()
  .output('silent.mp4')
  .run();

// Adjust volume
await ffmpeg.input('input.mp4')
  .volume(1.5)  // 150% volume
  .output('louder.mp4')
  .run();
```

```
// Normalize audio (loudnorm filter)
await ffmpeg.input('input.mp4')
  .normalize()
  .output('normalized.mp4')
  .run();

// Custom audio filter
await ffmpeg.input('input.mp4')
  .audioFilter('aecho=0.8:0.88:60:0.4')
  .output('echo.mp4')
  .run();

// Set sample rate and channels
await ffmpeg.input('input.mp4')
  .sampleRate(44100)
  .audioChannels(2)  // Stereo
  .output('resampled.mp4')
  .run();
```

Trimming and Seeking ✂

```
// Seek to position and set duration
await ffmpeg.input('input.mp4')
  .seek(10)       // Start at 10 seconds
  .duration(30)   // Output 30 seconds
  .output('clip.mp4')
  .run();

// Trim helper (start to end)
await ffmpeg.input('input.mp4')
  .trim(10, 40)   // From 10s to 40s
  .output('clip.mp4')
  .run();

// Fast seek (before input decoding)
await ffmpeg.input('input.mp4')
  .seekInput(60)  // Fast seek to 60s
```

```
  .duration(10)
  .output('clip.mp4')
  .run();
```

Complex Filters □□

```
// Custom complex filter graph (e.g., for high-quality GIFs)
await ffmpeg.input('input.mp4')
  .complexFilter('[0:v]split[s0][s1];[s0]palettegen[p];[s1][p]paletteuse')
  .output('output.gif')
  .run();
```

Debug: Get Generated Arguments □□

Inspect the FFmpeg command that would be generated:

```
const args = ffmpeg.input('input.mp4')
  .videoCodec('libx264')
  .crf(23)
  .output('output.mp4')
  .getArgs('output.mp4');

console.log(args);
// ['-y', '-i', 'input.mp4', '-c:v', 'libx264', '-crf', '23', 'output.mp4']
```

Legacy API (Still Supported) □□

The original API remains available for backward compatibility:

```
const ffmpeg = new SmartFfmpeg();

// Get media info
const info = await ffmpeg.getMediaInfo('input.mp4');
console.log(`Duration: ${info.format.duration}s`);
console.log(`Streams: ${info.streams.length}`);

// Convert with options
```

```
await ffmpeg.convert('input.mp4', 'output.webm', {
  videoCodec: 'libvpx-vp9',
  audioCodec: 'libopus',
  videoBitrate: '1M',
  audioBitrate: '128k',
});

// Convert with progress
await ffmpeg.convertWithProgress('input.mp4', 'output.mp4', {
  videoCodec: 'libx264',
}, progress => {
  console.log(`${progress.percent?.toFixed(1)}%`);
});

// Extract audio
await ffmpeg.extractAudio('video.mp4', 'audio.mp3', {
  audioCodec: 'libmp3lame',
  audioBitrate: '320k',
});

// Screenshot at specific time
await ffmpeg.screenshot('video.mp4', 'thumb.png', {
  time: 10,
  width: 1280,
});

// Generate multiple thumbnails
const thumbs = await ffmpeg.generateThumbnails('video.mp4', './thumbs', {
  count: 5,
  width: 320,
});

// Resize video
await ffmpeg.resize('input.mp4', 'output.mp4', 1920, 1080);

// Trim video
await ffmpeg.trim('input.mp4', 'clip.mp4', 10, 30);

// Convert to GIF
```

```
await ffmpeg.toGif('video.mp4', 'animation.gif', {
  width: 480,
  fps: 15,
  startTime: 5,
  duration: 3,
});

// Concatenate files
await ffmpeg.concat(['part1.mp4', 'part2.mp4'], 'combined.mp4');

// Add audio to video
await ffmpeg.addAudio('video.mp4', 'music.mp3', 'output.mp4');

// Get FFmpeg capabilities
const encoders = await ffmpeg.getEncoders();
const decoders = await ffmpeg.getDecoders();
const formats = await ffmpeg.getFormats();

// Run raw FFmpeg command
await ffmpeg.runRaw(['-i', 'input.mp4', '-vf', 'hflip', 'output.mp4']);
```

API Reference

Builder API Methods

Input Methods

Method	Description
`input(source, options?)`	Set input (file path, Buffer, Uint8Array, or ReadableStream)
`seekInput(time)`	Seek before input (fast seek)
`inputArgs(...args)`	Add custom input arguments

Video Methods

Method	Description

`videoCodec(codec)`	Set video codec (`libx264`, `libx265`, `libvpx-vp9`, `libaom-av1`, etc.)
`videoBitrate(bitrate)`	Set video bitrate (e.g., `'1M'`, `'2000k'`)
`size(width, height?)`	Set output dimensions
`fps(rate)`	Set frame rate
`crf(value)`	Set Constant Rate Factor (0-51, lower = better quality)
`preset(value)`	Set encoding preset (`ultrafast` → `veryslow`)
`noVideo()`	Remove video stream
`videoFilter(filter)`	Add custom video filter
`scale(w, h)`	Scale video
`crop(w, h, x, y)`	Crop video
`rotate(angle)`	Rotate video
`flipHorizontal()`	Flip horizontally
`flipVertical()`	Flip vertically
`pad(w, h, x, y, color)`	Add padding

Audio Methods

Method	Description
`audioCodec(codec)`	Set audio codec (`aac`, `libmp3lame`, `libopus`, etc.)
`audioBitrate(bitrate)`	Set audio bitrate (e.g., `'128k'`, `'320k'`)
`sampleRate(rate)`	Set sample rate in Hz
`audioChannels(count)`	Set channel count (1=mono, 2=stereo)
`noAudio()`	Remove audio stream
`audioFilter(filter)`	Add custom audio filter
`volume(level)`	Set volume multiplier
`normalize()`	Apply loudnorm filter

Timing Methods

Method	Description
`seek(time)`	Seek to position
`duration(time)`	Set output duration
`trim(start, end)`	Trim to time range

Output Methods

Method	Description
`format(fmt)`	Set output format (`mp4`, `webm`, `mkv`, `mp3`, etc.)
`output(path)`	Set output file path
`outputArgs(...args)`	Add custom output arguments
`overwrite(bool)`	Overwrite existing file (default: `true`)
`complexFilter(graph)`	Set complex filter graph

Execution Methods

Method	Description
`run()`	Execute and write to file
`toBuffer(format?)`	Execute and return Buffer
`toStream(format?)`	Execute and return Web ReadableStream
`pipe(writable, format?)`	Pipe to Web WritableStream
`getArgs(outputPath?)`	Get FFmpeg arguments (debugging)

Callbacks

Method	Description
`onProgress(callback)`	Set progress callback

Supported Codecs

Video Codecs

- `libx264` – H.264 (most compatible)
- `libx265` – H.265/HEVC (better compression)
- `libvpx` – VP8
- `libvpx-vp9` – VP9 (web-friendly)
- `libaom-av1` – AV1 (best compression, slower)
- `copy` – Copy without re-encoding

Audio Codecs

- `aac` – AAC (most compatible)
- `libmp3lame` – MP3

- `libopus` – Opus (excellent quality)
- `libvorbis` – Vorbis
- `flac` – Lossless
- `copy` – Copy without re-encoding

Encoding Presets

Speed/quality tradeoff for x264/x265:

- `ultrafast` → `superfast` → `veryfast` → `faster` → `fast` → `medium` → `slow` → `slower` → `veryslow`

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/smartffmpeg

2025-12-11 - 1.1.1 - fix(smartffmpeg)

Bump patch version to 1.1.1

- No source changes detected in the working tree — prepare a patch release (1.1.1) for packaging/release purposes.

2025-12-11 - 1.1.0 - feat(core)

Initial release: add SmartFfmpeg fluent builder API with in-memory streams, progress tracking, and bundled ffmpeg/ffprobe

- Introduce SmartFfmpeg main class and FfmpegCommand fluent builder API for chainable ffmpeg commands
- Support in-memory inputs/outputs: Buffer, Uint8Array, Node and Web Readable/Writable streams (pipe(), toBuffer(), toStream())
- Progress reporting via onProgress callbacks and ffmpeg progress parsing
- Bundled binaries using ffmpeg-static and ffprobe-static (paths exposed via plugins)
- Legacy API preserved (convert, convertWithProgress, getMediaInfo, generateThumbnails, screenshot, concat, addAudio, etc.)
- Utilities for probing encoders/decoders/formats and helper methods (screenshot, thumbnails, GIF conversion, concat)

2025-12-11 - 1.0.0 - initial release

Initial release of the project.

- Repository initialized.

- Basic project scaffold and initial files added to establish the foundation for future development.