readme.md for @push.rocks/smartjimp

Lightning-fast image processing for TypeScript, powered by Sharp and Jimp

npm version TypeScript License: MIT

Why SmartJimp? □□

SmartJimp bridges the gap between two powerful image processing libraries:

- **Sharp** □□ - Blazing fast, native C++ bindings, perfect for production
- **Jimp** □□ - Pure JavaScript, runs anywhere Node.js does

Choose your engine based on your needs - same API, different performance characteristics. Plus, you get automatic caching out of the box!

Installation □□

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

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

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

Quick Start □□♂

```
import { SmartJimp } from '@push.rocks/smartjimp';

// Choose your engine
const imageProcessor = new SmartJimp({ mode: 'sharp' }); // or 'jimp'

// That's it! You're ready to process images □□
```

Features □

□□ Resize Images

Resize images while maintaining aspect ratio or set exact dimensions:

```
import { SmartJimp } from '@push.rocks/smartjimp';
import { SmartFile } from '@push.rocks/smartfile';

const processor = new SmartJimp({ mode: 'sharp' });

// Load an image
const imageFile = await SmartFile.fromUrl('https://example.com/image.jpg');

// Resize to 500px width (height auto-calculated to maintain aspect ratio)
const resized = await processor.getFromSmartfile(imageFile, {
  width: 500
});

// Or resize to exact dimensions
const exactSize = await processor.getFromSmartfile(imageFile, {
  width: 800,
  height: 600
});

// Save the result
await SmartFile.fromBuffer('./resized.jpg', resized).write();
```

□□ Format Conversion

Convert between PNG, JPEG, WebP, and AVIF formats with ease:

```
// Convert any image to PNG
const pngBuffer = await processor.getFromSmartfile(imageFile, {
  format: 'png'
});

// Convert to standard JPEG
const jpegBuffer = await processor.getFromSmartfile(imageFile, {
  format: 'jpeg'
});

// Convert to WebP for smaller file sizes
const webpBuffer = await processor.getFromSmartfile(imageFile, {
  format: 'webp',
  width: 1200 // Resize while converting!
});

// Convert to AVIF for next-gen compression (Sharp only)
const avifBuffer = await processor.getFromSmartfile(imageFile, {
  format: 'avif'
});
```

□□ Quality Control

Fine-tune image compression quality for the perfect balance between file size and visual fidelity:

```
// High quality JPEG (larger file size)
const highQualityJpeg = await processor.getFromSmartfile(imageFile, {
  format: 'jpeg',
  quality: 90  // 1-100, higher = better quality
});

// Optimized for web (smaller file size)
const webOptimized = await processor.getFromSmartfile(imageFile, {
  format: 'jpeg',
  quality: 75,  // Good balance for web
  progressive: true
});
```

```
// Ultra-compressed WebP
const tinyWebP = await processor.getFromSmartfile(imageFile, {
  format: 'webp',
  quality: 60  // WebP handles lower quality better than JPEG
});

// Quality works with all lossy formats
const qualityAvif = await processor.getFromSmartfile(imageFile, {
  format: 'avif',
  quality: 80  // AVIF provides excellent quality even at lower values
});
```

> □□ **Pro tip**: Different formats handle quality settings differently. AVIF and WebP generally look better than JPEG at the same quality level.

□□ Progressive JPEG Support

Create progressive JPEGs that load in multiple passes for better perceived performance:

```
// Create a progressive JPEG (Sharp mode only)
const progressiveJpeg = await processor.getFromSmartfile(imageFile, {
  format: 'jpeg',
  progressive: true,
  width: 800
});

// Convert PNG to progressive JPEG
const pngFile = await SmartFile.fromUrl('https://example.com/image.png');
const progressiveFromPng = await processor.getFromSmartfile(pngFile, {
  format: 'jpeg',
  progressive: true
});

// Standard JPEG (non-progressive)
const standardJpeg = await processor.getFromSmartfile(imageFile, {
  format: 'jpeg',
  progressive: false // default
```

```
});
```

> □□ **Note**: Progressive JPEG encoding is only supported in Sharp mode. Jimp will create standard JPEGs regardless of the progressive setting.

□□ Image Effects

Apply visual effects to your images:

```
// Invert colors (currently jimp-only)
const inverted = await processor.getFromSmartfile(imageFile, {
  invert: true
});
```

⚡ Direct Buffer Processing

Work directly with buffers for maximum flexibility:

```
// Process raw image buffers
const processedBuffer = await processor.computeAssetVariation(imageBuffer, {
  width: 300,
  format: 'webp',
  quality: 85,
  invert: true
});

// Create progressive JPEG from buffer
const progressiveBuffer = await processor.computeAssetVariation(imageBuffer, {
  format: 'jpeg',
  progressive: true,
  quality: 88
});

// Special AVIF creation method (sharp mode only)
const avifBuffer = await processor.createAvifImageFromBuffer(imageBuffer);
```

□□ Built-in Caching

SmartJimp automatically caches processed images for lightning-fast repeated operations:

```
// First call: processes the image
const result1 = await processor.getFromSmartfile(imageFile, { width: 500 });

// Second call: returns from cache instantly! ⚡
const result2 = await processor.getFromSmartfile(imageFile, { width: 500 });
```

Cache features:

- Automatic cache key generation based on source and parameters
- 10-minute default TTL
- Memory + disk hybrid caching with LevelCache

Sharp vs Jimp: Choose Your Fighter □□

Use Sharp when:

- □□♂ **Performance is critical** - Up to 10x faster than Jimp
- □□ **You need AVIF support** - Sharp exclusive feature
- □□ **You need progressive JPEGs** - Sharp exclusive feature
- □□ **Running on servers** - Where native dependencies are OK
- □□ **Processing many images** - Batch operations benefit from speed

Use Jimp when:

- □□ **Cross-platform compatibility is key** - Pure JavaScript, no build steps
- ☁ **Deploying to serverless** - Some platforms don't support native modules
- □□ **You need pixel-level manipulation** - Jimp has more pixel manipulation features
- □□ **Simpler deployment** - No need to worry about sharp's native dependencies

Advanced Examples □□

Batch Processing with Async Operations

```
import { SmartJimp } from '@push.rocks/smartjimp';
import { SmartFile } from '@push.rocks/smartfile';

const processor = new SmartJimp({ mode: 'sharp' });

// Process multiple images concurrently
const imageUrls = [
  'https://example.com/photo1.jpg',
  'https://example.com/photo2.jpg',
  'https://example.com/photo3.jpg'
];

const processedImages = await Promise.all(
  imageUrls.map(async (url) => {
    const file = await SmartFile.fromUrl(url);
    return processor.getFromSmartfile(file, {
      width: 800,
      format: 'jpeg',
      progressive: true // Web-optimized progressive JPEGs
    });
  })
);

// Save all processed images
await Promise.all(
  processedImages.map((buffer, index) =>
    SmartFile.fromBuffer(`./output/image-${index}.jpg`, buffer).write()
  )
);
```

Creating Responsive Image Sets

```
// Generate multiple sizes for responsive images
const sizes = [320, 640, 1024, 1920];
const formats = ['webp', 'jpeg'] as const;
```

```
for (const size of sizes) {
  for (const format of formats) {
    const processed = await processor.getFromSmartfile(originalImage, {
      width: size,
      format,
      // Use progressive for JPEGs
      progressive: format === 'jpeg'
    });

    await SmartFile.fromBuffer(
      `./responsive/image-${size}.${format}`,
      processed
    ).write();
  }
}
```

Web-Optimized Image Pipeline

```
// Create a complete image optimization pipeline
async function optimizeForWeb(sourceImage: SmartFile) {
  const processor = new SmartJimp({ mode: 'sharp' });

  // Create multiple formats for browser compatibility
  const formats = {
    // Modern browsers: AVIF
    avif: await processor.getFromSmartfile(sourceImage, {
      format: 'avif',
      width: 1200,
      quality: 85  // AVIF excels at lower quality settings
    }),

    // Good browser support: WebP
    webp: await processor.getFromSmartfile(sourceImage, {
      format: 'webp',
      width: 1200,
      quality: 82  // WebP sweet spot for quality/size
    }),
```

```
    // Universal fallback: Progressive JPEG
    jpeg: await processor.getFromSmartfile(sourceImage, {
      format: 'jpeg',
      progressive: true,
      width: 1200,
      quality: 80  // Standard web quality
    })
  };

  return formats;
}
```

API Reference □□

Constructor Options

```
interface ISmartJimpOptions {
  mode: 'sharp' | 'jimp'; // Choose your processing engine
}
```

Processing Options

```
interface IAssetVariation {
  format?: 'avif' | 'webp' | 'png' | 'jpeg';  // Output format
  width?: number;                              // Target width (maintains aspect ratio if height not set)
  height?: number;                             // Target height (maintains aspect ratio if width not set)
  invert?: boolean;                            // Invert colors (jimp only currently)
  progressive?: boolean;                       // Create progressive JPEG (sharp only, jpeg format only)
  quality?: number;                            // Compression quality 1-100 (lossy formats only)
}
```

Main Methods

`getFromSmartfile(smartfile, options?)`

Process a SmartFile instance with caching enabled.

`computeAssetVariation(buffer, options)`

Process a raw buffer without caching.

`createAvifImageFromBuffer(buffer)`

Create an AVIF image from buffer (sharp mode only).

Performance Tips □□

1. **Use Sharp mode in production** - It's significantly faster
2. **Leverage the built-in cache** - Process identical images only once
3. **Batch operations** - Use `Promise.all()` for concurrent processing
4. **Choose appropriate formats**:
   - AVIF: Best compression, limited support
   - WebP: Good compression, good support
   - Progressive JPEG: Universal support, good perceived performance
   - PNG: Lossless, larger files

Troubleshooting □□

Sharp installation issues

If you encounter issues with Sharp:

```
# Rebuild sharp from source
npm rebuild sharp

# Or install with specific platform
npm install sharp --platform=linux --arch=x64
```

Memory usage

For processing many large images:

```
// Process in batches to control memory usage
const batchSize = 10;
for (let i = 0; i < images.length; i += batchSize) {
  const batch = images.slice(i, i + batchSize);
  await processBatch(batch);
}
```

Progressive JPEG verification

To verify if a JPEG is progressive:

```
# Using ImageMagick
identify -verbose image.jpg | grep "Interlace"
# Output: "Interlace: JPEG" means progressive

# Using exiftool
exiftool -EncodingProcess image.jpg
```

Contributing □□

We welcome contributions! Please feel free to submit a Pull Request.

License and Legal Information

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

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 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, and any usage must be approved in writing by Task Venture Capital GmbH.

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.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #2
Created 2026-03-28 11:10:43 UTC by foss.global Team
Updated 2026-03-28 12:17:32 UTC by foss.global Team