# Background

libjpeg-turbo is a JPEG image codec that uses SIMD instructions to accelerate baseline JPEG compression and decompression on x86, x86-64, Arm, PowerPC, and MIPS systems, as well as progressive JPEG compression on x86, x86-64, and Arm systems. On such systems, libjpeg-turbo is generally 2-6x as fast as libjpeg, all else being equal. On other types of systems, libjpeg-turbo can still outperform libjpeg by a significant amount, by virtue of its highly-optimized Huffman coding routines. In many cases, the performance of libjpeg-turbo rivals that of proprietary high-speed JPEG codecs.

libjpeg-turbo implements both the traditional libjpeg API as well as the less powerful but more straightforward TurboJPEG API. libjpeg-turbo also features colorspace extensions that allow it to compress from/decompress to 32-bit and big-endian pixel buffers (RGBX, XBGR, etc.), as well as a full-featured Java interface.

libjpeg-turbo was originally based on libjpeg/SIMD, an MMX-accelerated derivative of libjpeg v6b developed by Miyasaka Masaru. The TigerVNC and VirtualGL projects made numerous enhancements to the codec in 2009, and in early 2010, libjpeg-turbo spun off into an independent project, with the goal of making high-speed JPEG compression/decompression technology available to a broader range of users and developers.

# License

libjpeg-turbo is covered by three compatible BSD-style open source licenses. Refer to LICENSE.md for a roll-up of license terms.

# Building libjpeg-turbo

Refer to BUILDING.md for complete instructions.

# Using libjpeg-turbo

libjpeg-turbo includes two APIs that can be used to compress and decompress JPEG images:

- **TurboJPEG API** This API provides an easy-to-use interface for compressing and decompressing JPEG images in memory. It also provides some functionality that would not be straightforward to achieve using the underlying libjpeg API, such as generating planar YUV images and performing multiple simultaneous lossless transforms on an image. The Java interface for libjpeg-turbo is written on top of the TurboJPEG API. The TurboJPEG API is recommended for first-time users of libjpeg-turbo. Refer to tjexample.c and TJExample.java for examples of its usage and

to http://libjpeg-turbo.org/Documentation/Documentation for API documentation.

- **libjpeg API** This is the de facto industry-standard API for compressing and decompressing JPEG images. It is more difficult to use than the TurboJPEG API but also more powerful. The libjpeg API implementation in libjpeg-turbo is both API/ABI-compatible and mathematically compatible with libjpeg v6b. It can also optionally be configured to be API/ABI-compatible with libjpeg v7 and v8 (see below.) Refer to cjpeg.c and djpeg.c for examples of its usage and to libjpeg.txt for API documentation.

There is no significant performance advantage to either API when both are used to perform similar operations.

## Colorspace Extensions

libjpeg-turbo includes extensions that allow JPEG images to be compressed directly from (and decompressed directly to) buffers that use BGR, BGRX, RGBX, XBGR, and XRGB pixel ordering. This is implemented with ten new colorspace constants:

```
JCS_EXT_RGB   /* red/green/blue */
JCS_EXT_RGBX  /* red/green/blue/x */
JCS_EXT_BGR   /* blue/green/red */
JCS_EXT_BGRX  /* blue/green/red/x */
JCS_EXT_XBGR  /* x/blue/green/red */
JCS_EXT_XRGB  /* x/red/green/blue */
JCS_EXT_RGBA  /* red/green/blue/alpha */
JCS_EXT_BGRA  /* blue/green/red/alpha */
JCS_EXT_ABGR  /* alpha/blue/green/red */
JCS_EXT_ARGB  /* alpha/red/green/blue */
```

Setting `cinfo.in_color_space` (compression) or `cinfo.out_color_space` (decompression) to one of these values will cause libjpeg-turbo to read the red, green, and blue values from (or write them to) the appropriate position in the pixel when compressing from/decompressing to an RGB buffer.

Your application can check for the existence of these extensions at compile time with:

```
#ifdef JCS_EXTENSIONS
```

At run time, attempting to use these extensions with a libjpeg implementation that does not support them will result in a "Bogus input colorspace" error. Applications can trap this error in order to test whether run-time support is available for the colorspace extensions.

When using the RGBX, BGRX, XBGR, and XRGB colorspaces during decompression, the X byte is undefined, and in order to ensure the best performance, libjpeg-turbo can set that byte to whatever value it wishes. If an application

expects the X byte to be used as an alpha channel, then it should specify `JCS_EXT_RGBA`, `JCS_EXT_BGRA`, `JCS_EXT_ABGR`, or `JCS_EXT_ARGB`. When these colorspace constants are used, the X byte is guaranteed to be 0xFF, which is interpreted as opaque.

Your application can check for the existence of the alpha channel colorspace extensions at compile time with:

```
#ifdef JCS_ALPHA_EXTENSIONS
```

jcstest.c, located in the libjpeg-turbo source tree, demonstrates how to check for the existence of the colorspace extensions at compile time and run time.

## libjpeg v7 and v8 API/ABI Emulation

With libjpeg v7 and v8, new features were added that necessitated extending the compression and decompression structures. Unfortunately, due to the exposed nature of those structures, extending them also necessitated breaking backward ABI compatibility with previous libjpeg releases. Thus, programs that were built to use libjpeg v7 or v8 did not work with libjpeg-turbo, since it is based on the libjpeg v6b code base. Although libjpeg v7 and v8 are not as widely used as v6b, enough programs (including a few Linux distros) made the switch that there was a demand to emulate the libjpeg v7 and v8 ABIs in libjpeg-turbo. It should be noted, however, that this feature was added primarily so that applications that had already been compiled to use libjpeg v7+ could take advantage of accelerated baseline JPEG encoding/decoding without recompiling. libjpeg-turbo does not claim to support all of the libjpeg v7+ features, nor to produce identical output to libjpeg v7+ in all cases (see below.)

By passing an argument of `-DWITH_JPEG7=1` or `-DWITH_JPEG8=1` to `cmake`, you can build a version of libjpeg-turbo that emulates the libjpeg v7 or v8 ABI, so that programs that are built against libjpeg v7 or v8 can be run with libjpeg-turbo. The following section describes which libjpeg v7+ features are supported and which aren't.

### Support for libjpeg v7 and v8 Features

#### Fully supported

- **libjpeg API: IDCT scaling extensions in decompressor** libjpeg-turbo supports IDCT scaling with scaling factors of 1/8, 1/4, 3/8, 1/2, 5/8, 3/4, 7/8, 9/8, 5/4, 11/8, 3/2, 13/8, 7/4, 15/8, and 2/1 (only 1/4 and 1/2 are SIMD-accelerated.)
- **libjpeg API: Arithmetic coding**
- **libjpeg API: In-memory source and destination managers** See notes below.

- **cjpeg: Separate quality settings for luminance and chrominance** Note that the libpjeg v7+ API was extended to accommodate this feature only for convenience purposes. It has always been possible to implement this feature with libjpeg v6b (see rdswitch.c for an example.)
- **cjpeg: 32-bit BMP support**
- **cjpeg: `-rgb` option**
- **jpegtran: Lossless cropping**
- **jpegtran: `-perfect` option**
- **jpegtran: Forcing width/height when performing lossless crop**
- **rdjpgcom: `-raw` option**
- **rdjpgcom: Locale awareness**

**Not supported** NOTE: As of this writing, extensive research has been conducted into the usefulness of DCT scaling as a means of data reduction and SmartScale as a means of quality improvement. Readers are invited to peruse the research at http://www.libjpeg-turbo.org/About/SmartScale and draw their own conclusions, but it is the general belief of our project that these features have not demonstrated sufficient usefulness to justify inclusion in libjpeg-turbo.

- **libjpeg API: DCT scaling in compressor** `cinfo.scale_num` and `cinfo.scale_denom` are silently ignored. There is no technical reason why DCT scaling could not be supported when emulating the libjpeg v7+ API/ABI, but without the SmartScale extension (see below), only scaling factors of 1/2, 8/15, 4/7, 8/13, 2/3, 8/11, 4/5, and 8/9 would be available, which is of limited usefulness.
- **libjpeg API: SmartScale** `cinfo.block_size` is silently ignored. SmartScale is an extension to the JPEG format that allows for DCT block sizes other than 8x8. Providing support for this new format would be feasible (particularly without full acceleration.) However, until/unless the format becomes either an official industry standard or, at minimum, an accepted solution in the community, we are hesitant to implement it, as there is no sense of whether or how it might change in the future. It is our belief that SmartScale has not demonstrated sufficient usefulness as a lossless format nor as a means of quality enhancement, and thus our primary interest in providing this feature would be as a means of supporting additional DCT scaling factors.
- **libjpeg API: Fancy downsampling in compressor** `cinfo.do_fancy_downsampling` is silently ignored. This requires the DCT scaling feature, which is not supported.
- **jpegtran: Scaling** This requires both the DCT scaling and SmartScale features, which are not supported.

- **Lossless RGB JPEG files** This requires the SmartScale feature, which is not supported.

#### What About libjpeg v9?

libjpeg v9 introduced yet another field to the JPEG compression structure (`color_transform`), thus making the ABI backward incompatible with that of libjpeg v8. This new field was introduced solely for the purpose of supporting lossless SmartScale encoding. Furthermore, there was actually no reason to extend the API in this manner, as the color transform could have just as easily been activated by way of a new JPEG colorspace constant, thus preserving backward ABI compatibility.

Our research (see link above) has shown that lossless SmartScale does not generally accomplish anything that can't already be accomplished better with existing, standard lossless formats. Therefore, at this time it is our belief that there is not sufficient technical justification for software projects to upgrade from libjpeg v8 to libjpeg v9, and thus there is not sufficient technical justification for us to emulate the libjpeg v9 ABI.

### In-Memory Source/Destination Managers

By default, libjpeg-turbo 1.3 and later includes the `jpeg_mem_src()` and `jpeg_mem_dest()` functions, even when not emulating the libjpeg v8 API/ABI. Previously, it was necessary to build libjpeg-turbo from source with libjpeg v8 API/ABI emulation in order to use the in-memory source/destination managers, but several projects requested that those functions be included when emulating the libjpeg v6b API/ABI as well. This allows the use of those functions by programs that need them, without breaking ABI compatibility for programs that don't, and it allows those functions to be provided in the "official" libjpeg-turbo binaries.

Those who are concerned about maintaining strict conformance with the libjpeg v6b or v7 API can pass an argument of `-DWITH_MEM_SRCDST=0` to `cmake` prior to building libjpeg-turbo. This will restore the pre-1.3 behavior, in which `jpeg_mem_src()` and `jpeg_mem_dest()` are only included when emulating the libjpeg v8 API/ABI.

On Un*x systems, including the in-memory source/destination managers changes the dynamic library version from 62.2.0 to 62.3.0 if using libjpeg v6b API/ABI emulation and from 7.2.0 to 7.3.0 if using libjpeg v7 API/ABI emulation.

Note that, on most Un*x systems, the dynamic linker will not look for a function in a library until that function is actually used. Thus, if a program is built against libjpeg-turbo 1.3+ and uses `jpeg_mem_src()` or `jpeg_mem_dest()`, that program will not fail if run against an older version of libjpeg-turbo or against libjpeg v7- until the program actually tries to call `jpeg_mem_src()` or `jpeg_mem_dest()`. Such is not the case on Windows. If a program is built against

the libjpeg-turbo 1.3+ DLL and uses `jpeg_mem_src()` or `jpeg_mem_dest()`, then it must use the libjpeg-turbo 1.3+ DLL at run time.

Both cjpeg and djpeg have been extended to allow testing the in-memory source/destination manager functions. See their respective man pages for more details.

## Mathematical Compatibility

For the most part, libjpeg-turbo should produce identical output to libjpeg v6b. The one exception to this is when using the floating point DCT/IDCT, in which case the outputs of libjpeg v6b and libjpeg-turbo can differ for the following reasons:

- The SSE/SSE2 floating point DCT implementation in libjpeg-turbo is ever so slightly more accurate than the implementation in libjpeg v6b, but not by any amount perceptible to human vision (generally in the range of 0.01 to 0.08 dB gain in PNSR.)
- When not using the SIMD extensions, libjpeg-turbo uses the more accurate (and slightly faster) floating point IDCT algorithm introduced in libjpeg v8a as opposed to the algorithm used in libjpeg v6b. It should be noted, however, that this algorithm basically brings the accuracy of the floating point IDCT in line with the accuracy of the accurate integer IDCT. The floating point DCT/IDCT algorithms are mainly a legacy feature, and they do not produce significantly more accuracy than the accurate integer algorithms (to put numbers on this, the typical difference in PNSR between the two algorithms is less than 0.10 dB, whereas changing the quality level by 1 in the upper range of the quality scale is typically more like a 1.0 dB difference.)
- If the floating point algorithms in libjpeg-turbo are not implemented using SIMD instructions on a particular platform, then the accuracy of the floating point DCT/IDCT can depend on the compiler settings.

While libjpeg-turbo does emulate the libjpeg v8 API/ABI, under the hood it is still using the same algorithms as libjpeg v6b, so there are several specific cases in which libjpeg-turbo cannot be expected to produce the same output as libjpeg v8:

- When decompressing using scaling factors of 1/2 and 1/4, because libjpeg v8 implements those scaling algorithms differently than libjpeg v6b does, and libjpeg-turbo's SIMD extensions are based on the libjpeg v6b behavior.
- When using chrominance subsampling, because libjpeg v8 implements this with its DCT/IDCT scaling algorithms rather than with a separate downsampling/upsampling algorithm. In our testing, the subsampled/upsampled output of libjpeg v8 is less accurate than that of libjpeg

v6b for this reason.

- When decompressing using a scaling factor > 1 and merged (AKA "non-fancy" or "non-smooth") chrominance upsampling, because libjpeg v8 does not support merged upsampling with scaling factors > 1.

# Performance Pitfalls

## Restart Markers

The optimized Huffman decoder in libjpeg-turbo does not handle restart markers in a way that makes the rest of the libjpeg infrastructure happy, so it is necessary to use the slow Huffman decoder when decompressing a JPEG image that has restart markers. This can cause the decompression performance to drop by as much as 20%, but the performance will still be much greater than that of libjpeg. Many consumer packages, such as Photoshop, use restart markers when generating JPEG images, so images generated by those programs will experience this issue.

## Fast Integer Forward DCT at High Quality Levels

The algorithm used by the SIMD-accelerated quantization function cannot produce correct results whenever the fast integer forward DCT is used along with a JPEG quality of 98-100. Thus, libjpeg-turbo must use the non-SIMD quantization function in those cases. This causes performance to drop by as much as 40%. It is therefore strongly advised that you use the accurate integer forward DCT whenever encoding images with a JPEG quality of 98 or higher.

# Memory Debugger Pitfalls

Valgrind and Memory Sanitizer (MSan) can generate false positives (specifically, incorrect reports of uninitialized memory accesses) when used with libjpeg-turbo's SIMD extensions. It is generally recommended that the SIMD extensions be disabled, either by passing an argument of `-DWITH_SIMD=0` to `cmake` when configuring the build or by setting the environment variable `JSIMD_FORCENONE` to `1` at run time, when testing libjpeg-turbo with Valgrind, MSan, or other memory debuggers.