Building on Unix Platforms (including Cygwin)

Build Requirements

-- autoconf 2.56 or later
-- automake 1.7 or later
-- libtool 1.4 or later

-- NASM (if building x86 or x86-64 SIMD extensions)

- 0.98, or 2.01 or later is required for a 32-bit build
- NASM 2.00 or later is required for a 64-bit build
- NASM 2.07 or later is required for a 64-bit build on OS X. This can be obtained from MacPorts (http://www.macports.org/).

The binary RPMs released by the NASM project do not work on older Linux systems, such as Red Hat Enterprise Linux 4. On such systems, you can easily build and install NASM from a source RPM by downloading one of the SRPMs from

http://www.nasm.us/pub/nasm/releasebuilds

and executing the following as root:

```
ARCH=`uname -m`
rpmbuild --rebuild nasm-{version}.src.rpm
rpm -Uvh /usr/src/redhat/RPMS/$ARCH/nasm-{version}.$ARCH.rpm
```

NOTE: the NASM build will fail if texinfo is not installed.

-- GCC v4.1 or later recommended for best performance

-- If building the TurboJPEG/OSS Java wrapper, JDK or OpenJDK 1.5 or later is required. Some systems, such as OS X 10.4, Solaris 10 and later, and Red Hat Enterprise Linux 5 and later, have this pre-installed. On OS X 10.5 and later, it will be necessary to install the Java Developer Package, which can be downloaded from http://connect.apple.com. For systems that do not have a JDK installed, you can obtain the Oracle Java Development Kit from http://www.java.com.

Out-of-Tree Builds

Binary objects, libraries, and executables are generated in the same directory from which configure was executed (the "binary directory"), and this directory need not necessarily be the same as the libjpeg-turbo source directory. You can create multiple independent binary directories, in which different versions of libjpeg-turbo can be built from the same source tree using different compilers or settings. In the sections below, {build_directory} refers to the binary directory, whereas {source_directory} refers to the libjpeg-turbo source directory. For in-tree builds, these directories are the same.

Building libjpeg-turbo

The following procedure will build libjpeg-turbo on Linux, FreeBSD, 32-bit OS X, Cygwin, and Solaris/x86 systems (on Solaris, this generates a 32-bit library. See below for 64-bit build instructions.)

```
  cd {source_directory}
  autoreconf -fiv
  cd {build_directory}
  sh {source_directory}/configure [additional configure flags]
  make
```

NOTE: Running autoreconf in the source directory is only necessary if building libjpeg-turbo from the SVN repository.

This will generate the following files under .libs/

```
  libjpeg.a
      Static link library for libjpeg-turbo

  libjpeg.so.{version} (Linux, Solaris)
  libjpeg.{version}.dylib (OS X)
  cygjpeg-{version}.dll (Cygwin)
      Shared library for libjpeg-turbo

  libjpeg.so (Linux, Solaris)
  libjpeg.dylib (OS X)
  libjpeg.dll.a (Cygwin)
      Development stub for libjpeg-turbo shared library

  libturbojpeg.a
      Static link library for TurboJPEG/OSS

  libturbojpeg.so (Linux, Solaris)
  libturbojpeg.dylib (OS X)
      Shared library and development stub for TurboJPEG/OSS

  cygturbojpeg.dll (Cygwin)
      Shared library for TurboJPEG/OSS

  libturbojpeg.dll.a (Cygwin)
      Development stub for TurboJPEG/OSS shared library
```

{version} is 62.0.0, 7.0.0, or 8.0.2, depending on whether libjpeg v6b (default), v7, or v8 emulation is enabled. If using Cygwin, {version} is 62, 7, or 8.

libjpeg v7 or v8 API/ABI Emulation

Add --with-jpeg7 to the configure command line to build a version of libjpeg-turbo that is API/ABI-compatible with libjpeg v7. Add --with-jpeg8 to the configure command to build a version of libjpeg-turbo that is API/ABI-compatible with libjpeg v8. See README-turbo.txt for more information on libjpeg v7 and v8 emulation.

Arithmetic Coding Support

Since the patent on arithmetic coding has expired, this functionality has been included in this release of libjpeg-turbo. libjpeg-turbo's implementation is based on the implementation in libjpeg v8, but it works when emulating libjpeg v7 or v6b as well. The default is to enable both arithmetic encoding and decoding, but those who have philosophical objections to arithmetic coding can add --without-arith-enc or --without-arith-dec to the configure command line to disable encoding or decoding (respectively.)

TurboJPEG/OSS Java Wrapper

Add --with-java to the configure command line to incorporate an optional Java Native Interface wrapper into the TurboJPEG/OSS dynamic library and build the Java front-end classes to support it. This allows the TurboJPEG/OSS dynamic library to be used directly from Java applications. See java/README for more details.

You can set the JAVAC, JAR, and JAVA configure variables to specify alternate commands for javac, jar, and java (respectively.) You can also set the JAVACFLAGS configure variable to specify arguments that should be passed to the Java compiler when building the front-end classes, and JNI_CFLAGS to specify arguments that should be passed to the C compiler when building the JNI wrapper. Run 'configure --help' for more details.

Installing libjpeg-turbo

If you intend to install these libraries and the associated header files, then replace 'make' in the instructions above with

```
  make install prefix={base dir} libdir={library directory}
```

For example,

```
  make install prefix=/usr/local libdir=/usr/local/lib64
```

will install the header files in /usr/local/include and the library files in /usr/local/lib64. If 'prefix' and 'libdir' are not specified, then the default is to install the header files in /opt/libjpeg-turbo/include and the library files in /opt/libjpeg-turbo/lib.

NOTE: You can specify a prefix of /usr and a libdir of, for instance, /usr/lib64 to overwrite the system's version of libjpeg. If you do this, however, then be sure to BACK UP YOUR SYSTEM'S INSTALLATION OF LIBJPEG before overwriting it. It is recommended that you instead install libjpeg-turbo into a non-system directory and manipulate the LD_LIBRARY_PATH or create sym links to force applications to use libjpeg-turbo instead of libjpeg. See README-turbo.txt for more information.

Build Recipes

32-bit Library Build on 64-bit Linux

Add

```
  --host i686-pc-linux-gnu CFLAGS='-O3 -m32' LDFLAGS=-m32
```

to the configure command line.

64-bit Library Build on 64-bit OS X

Add

```
  --host x86_64-apple-darwin NASM=/opt/local/bin/nasm
```

to the configure command line. NASM 2.07 or later from MacPorts must be installed.

32-bit Library Build on 64-bit OS X

Add

```
  --host i686-apple-darwin CFLAGS='-O3 -m32' LDFLAGS=-m32
```

to the configure command line.

64-bit Backward-Compatible Library Build on 64-bit OS X

Add

```
  --host x86_64-apple-darwin NASM=/opt/local/bin/nasm \
  CFLAGS='-isysroot /Developer/SDKs/MacOSX10.4u.sdk \
    -mmacosx-version-min=10.4 -O3' \
    LDFLAGS='-isysroot /Developer/SDKs/MacOSX10.4u.sdk \
    -mmacosx-version-min=10.4'
```

to the configure command line. The OS X 10.4 SDK, and NASM 2.07 or later from MacPorts, must be installed.

32-bit Backward-Compatible Library Build on OS X

Add

```
  --host i686-apple-darwin \
    CFLAGS='-isysroot /Developer/SDKs/MacOSX10.4u.sdk \
    -mmacosx-version-min=10.4 -O3 -m32' \
    LDFLAGS='-isysroot /Developer/SDKs/MacOSX10.4u.sdk \
    -mmacosx-version-min=10.4 -m32'
```

to the configure command line. The OS X 10.4 SDK must be installed.

64-bit Library Build on 64-bit Solaris

Add

```
  --host x86_64-pc-solaris CFLAGS='-O3 -m64' LDFLAGS=-m64
```

to the configure command line.

32-bit Library Build on 64-bit FreeBSD

Add

```
  --host i386-unknown-freebsd CC='gcc -B /usr/lib32' CFLAGS='-O3 -m32' \
    LDFLAGS='-B/usr/lib32'
```

to the configure command line. NASM 2.07 or later from FreeBSD ports must be installed.

Oracle Solaris Studio

Add

```
  CC=cc
```

to the configure command line. libjpeg-turbo will automatically be built with the maximum optimization level (-xO5) unless you override CFLAGS.

To build a 64-bit version of libjpeg-turbo using Oracle Solaris Studio, add

```
  --host x86_64-pc-solaris CC=cc CFLAGS='-xO5 -m64' LDFLAGS=-m64
```

to the configure command line.

MinGW Build on Cygwin

Use CMake (see recipes below)

ARM Support

This release of libjpeg-turbo can use ARM NEON SIMD instructions to accelerate JPEG compression/decompression by approximately 2-4x on ARMv7 and later platforms. If libjpeg-turbo is configured on an ARM Linux platform, then the build system will automatically include the NEON SIMD routines, if they are supported.

Building libjpeg-turbo for iOS

iOS platforms, such as the iPhone and iPad, also use ARM processors, some of which support NEON instructions. Additional steps are required to build libjpeg-turbo for these platforms. The steps below assume iOS SDK v4.3. If you are using a different SDK version, then you will need to modify the examples accordingly.

Additional build requirements:

  gas-preprocessor.pl (https://github.com/yuvi/gas-preprocessor) should be installed in your PATH.

Set the following shell variables for simplicity:

```
  IOS_PLATFORMDIR="/Developer/Platforms/iPhoneOS.platform"
  IOS_SYSROOT="$IOS_PLATFORMDIR/Developer/SDKs/iPhoneOS4.3.sdk"
  IOS_GCC="$IOS_PLATFORMDIR/Developer/usr/bin/arm-apple-darwin10-llvm-gcc-4.2"

  ARM v6 only (up to and including iPhone 3G):
  IOS_CFLAGS="-march=armv6 -mcpu=arm1176jzf-s -mfpu=vfp"

  ARM v7 only (iPhone 3GS and newer, iPad):
  IOS_CFLAGS="-march=armv7 -mcpu=cortex-a8 -mtune=cortex-a8 -mfpu=neon"
```

Follow the procedure under "Building libjpeg-turbo" above, adding

```
  --host arm-apple-darwin10 --enable-static --disable-shared \
    CC="$IOS_GCC" LD="$IOS_GCC" \
    CFLAGS="-mfloat-abi=softfp -isysroot $IOS_SYSROOT -O3 $IOS_CFLAGS" \
    LDFLAGS="-mfloat-abi=softfp -isysroot $IOS_SYSROOT $IOS_CFLAGS"
```

to the configure command line.

Once built, lipo can be used to combine the ARM v6 and v7 variants into a universal library.

Building on Windows (Visual C++ or MinGW)

Build Requirements

-- CMake (http://www.cmake.org) v2.6 or later

-- Microsoft Visual C++ 2005 or later

   If you don't already have Visual C++, then the easiest way to get it is by installing the Windows SDK:

   http://msdn.microsoft.com/en-us/windows/bb980924.aspx

   The Windows SDK includes both 32-bit and 64-bit Visual C++ compilers and everything necessary to build libjpeg-turbo.

* For 32-bit builds, you can also use Microsoft Visual C++ Express Edition. Visual C++ Express Edition is a free download.
* If you intend to build libjpeg-turbo from the command line, then add the appropriate compiler and SDK directories to the INCLUDE, LIB, and PATH environment variables. This is generally accomplished by executing vcvars32.bat or vcvars64.bat and SetEnv.cmd. vcvars32.bat and vcvars64.bat are part of Visual C++ and are located in the same directory as the compiler. SetEnv.cmd is part of the Windows SDK. You can pass optional arguments to SetEnv.cmd to specify a 32-bit or 64-bit build environment.

... OR ...

-- MinGW

GCC v4.1 or later recommended for best performance

-- NASM (http://www.nasm.us/) 0.98 or later (NASM 2.05 or later is required for a 64-bit build)

-- If building the TurboJPEG/OSS Java wrapper, JDK 1.5 or later is required. This can be downloaded from http://www.java.com.

Out-of-Tree Builds

Binary objects, libraries, and executables are generated in the same directory from which cmake was executed (the "binary directory"), and this directory need not necessarily be the same as the libjpeg-turbo source directory. You can create multiple independent binary directories, in which different versions of libjpeg-turbo can be built from the same source tree using different compilers or settings. In the sections below, {build_directory} refers to the binary directory, whereas {source_directory} refers to the libjpeg-turbo source directory. For in-tree builds, these directories are the same.

Building libjpeg-turbo

Visual C++ (Command Line)

```
  cd {build_directory}
  cmake -G "NMake Makefiles" -DCMAKE_BUILD_TYPE=Release {source_directory}
  nmake
```

This will build either a 32-bit or a 64-bit version of libjpeg-turbo, depending on which version of cl.exe is in the PATH.

The following files will be generated under {build_directory}:

```
  jpeg-static.lib
      Static link library for libjpeg-turbo
  sharedlib/jpeg{version}.dll
```

```
      DLL for libjpeg-turbo
  sharedlib/jpeg.lib
      Import library for libjpeg-turbo DLL
  turbojpeg-static.lib
      Static link library for TurboJPEG/OSS
  turbojpeg.dll
      DLL for TurboJPEG/OSS
  turbojpeg.lib
      Import library for TurboJPEG/OSS DLL
```

{version} is 62, 7, or 8, depending on whether libjpeg v6b (default), v7, or v8 emulation is enabled.

Visual C++ (IDE)

Choose the appropriate CMake generator option for your version of Visual Studio (run "cmake" with no arguments for a list of available generators.) For instance:

```
  cd {build_directory}
  cmake -G "Visual Studio 9 2008" {source_directory}
```

You can then open ALL_BUILD.vcproj in Visual Studio and build one of the configurations in that project ("Debug", "Release", etc.) to generate a full build of libjpeg-turbo.

This will generate the following files under {build_directory}:

```
  {configuration}/jpeg-static.lib
      Static link library for libjpeg-turbo
  sharedlib/{configuration}/jpeg{version}.dll
      DLL for libjpeg-turbo
  sharedlib/{configuration}/jpeg.lib
      Import library for libjpeg-turbo DLL
  {configuration}/turbojpeg-static.lib
      Static link library for TurboJPEG/OSS
  {configuration}/turbojpeg.dll
      DLL for TurboJPEG/OSS
  {configuration}/turbojpeg.lib
      Import library for TurboJPEG/OSS DLL
```

{configuration} is Debug, Release, RelWithDebInfo, or MinSizeRel, depending on the configuration you built in the IDE, and {version} is 62, 7, or 8, depending on whether libjpeg v6b (default), v7, or v8 emulation is enabled.

MinGW

```
  cd {build_directory}
  cmake -G "MSYS Makefiles" {source_directory}
  make
```

This will generate the following files under {build_directory}

```
  libjpeg.a
      Static link library for libjpeg-turbo
  sharedlib/libjpeg-{version}.dll
      DLL for libjpeg-turbo
  sharedlib/libjpeg.dll.a
      Import library for libjpeg-turbo DLL
  libturbojpeg.a
      Static link library for TurboJPEG/OSS
  libturbojpeg.dll
      DLL for TurboJPEG/OSS
  libturbojpeg.dll.a
      Import library for TurboJPEG/OSS DLL
```

{version} is 62, 7, or 8, depending on whether libjpeg v6b (default), v7, or v8 emulation is enabled.

Debug Build

Add "-DCMAKE_BUILD_TYPE=Debug" to the cmake command line. Or, if building with NMake, remove "-DCMAKE_BUILD_TYPE=Release" (Debug builds are the default with NMake.)

libjpeg v7 or v8 API/ABI Emulation

Add "-DWITH_JPEG7=1" to the cmake command line to build a version of libjpeg-turbo that is API/ABI-compatible with libjpeg v7. Add "-DWITH_JPEG8=1" to the cmake command to build a version of libjpeg-turbo that is API/ABI-compatible with libjpeg v8. See README-turbo.txt for more information on libjpeg v7 and v8 emulation.

Arithmetic Coding Support

Since the patent on arithmetic coding has expired, this functionality has been included in this release of libjpeg-turbo. libjpeg-turbo's implementation is based on the implementation in libjpeg v8, but it works when emulating libjpeg v7 or v6b as well. The default is to enable both arithmetic encoding and decoding, but those who have philosophical objections to arithmetic coding can add "-DWITH_ARITH_ENC=0" or "-DWITH_ARITH_DEC=0" to the cmake command line to disable encoding or decoding (respectively.)

TurboJPEG/OSS Java Wrapper

Add "-DWITH_JAVA=1" to the cmake command line to incorporate an optional Java Native Interface wrapper into the TurboJPEG/OSS dynamic library and build the Java front-end classes to support it. This allows the TurboJPEG/OSS dynamic library to be used directly from Java applications. See java/README for more details.

If you are using CMake 2.8, you can set the Java_JAVAC_EXECUTABLE, Java_JAVA_EXECUTABLE, and Java_JAR_EXECUTABLE CMake variables to specify alternate commands or locations for javac, jar, and java (respectively.) If you are using CMake 2.6, set JAVA_COMPILE, JAVA_RUNTIME, and JAVA_ARCHIVE instead. You can also set the JAVACFLAGS CMake variable to specify arguments that should be passed to the Java compiler when building the front-end classes.

Installing libjpeg-turbo

You can use the build system to install libjpeg-turbo into a directory of your choosing (as opposed to creating an installer.) To do this, add:

```
  -DCMAKE_INSTALL_PREFIX={install_directory}
```

to the cmake command line.

For example,

```
  cmake -G "NMake Makefiles" -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_INSTALL_PREFIX=c:\libjpeg-turbo {source_directory}
  nmake install
```

will install the header files in c:\libjpeg-turbo\include, the library files in c:\libjpeg-turbo\lib, the DLL's in c:\libjpeg-turbo\bin, and the documentation in c:\libjpeg-turbo\doc.

Build Recipes

64-bit MinGW Build on Cygwin

```
  cd {build_directory}
  CC=/usr/bin/x86_64-w64-mingw32-gcc \
    cmake -G "Unix Makefiles" -DCMAKE_SYSTEM_NAME=Windows \
    -DCMAKE_AR=/usr/bin/x86_64-w64-mingw32-ar \
    -DCMAKE_RANLIB=/usr/bin/x86_64-w64-mingw32-ranlib {source_directory}
  make
```

This produces a 64-bit build of libjpeg-turbo that does not depend on cygwin1.dll or other Cygwin DLL's. The mingw64-x86_64-gcc-core and mingw64-x86_64-gcc-g++ packages (and their dependencies) must be installed.

32-bit MinGW Build on Cygwin

```
  cd {build_directory}
  CC=/usr/bin/i686-w64-mingw32-gcc \
    cmake -G "Unix Makefiles" -DCMAKE_SYSTEM_NAME=Windows \
    -DDCMAKE_AR=/usr/bin/i686-w64-mingw32-ar \
    -DCMAKE_RANLIB=/usr/bin/i686-w64-mingw32-ranlib {source_directory}
  make
```

This produces a 32-bit build of libjpeg-turbo that does not depend on cygwin1.dll or other Cygwin DLL's. The mingw64-i686-gcc-core and mingw64-i686-gcc-g++ packages (and their dependencies) must be installed.

MinGW-w64 Build on Windows

This produces a 64-bit build of libjpeg-turbo using the "native" MinGW-w64 toolchain (which is faster than the Cygwin version):

```
  cd {build_directory}
  CC={mingw-w64_binary_path}/x86_64-w64-mingw32-gcc \
    cmake -G "MSYS Makefiles" \
    -DCMAKE_AR={mingw-w64_binary_path}/x86_64-w64-mingw32-ar \
    -DCMAKE_RANLIB={mingw-w64_binary_path}/x86_64-w64-mingw32-ranlib \
    {source_directory}
  make
```

MinGW Build on Linux

```
  cd {build_directory}
  CC={mingw_binary_path}/i386-mingw32-gcc \
    cmake -G "Unix Makefiles" -DCMAKE_SYSTEM_NAME=Windows \
    -DCMAKE_AR={mingw_binary_path}/i386-mingw32-ar \
    -DCMAKE_RANLIB={mingw_binary_path}/i386-mingw32-ranlib \
    {source_directory}
  make
```

Creating Release Packages

The following commands can be used to create various types of release packages:

Unix

```
make rpm
```

Create Red Hat-style binary RPM package. Requires RPM v4 or later.

```
make srpm
```

This runs 'make dist' to create a pristine source tarball, then creates a Red Hat-style source RPM package from the tarball. Requires RPM v4 or later.

```
make deb
```

Create Debian-style binary package. Requires dpkg.

```
make dmg
```

Create Macintosh package/disk image. This requires the PackageMaker application, which must be installed in /Developer/Applications/Utilities.

```
make udmg [BUILDDIR32={32-bit build directory}]
```

On 64-bit OS X systems, this creates a Macintosh package and disk image that contains universal i386/x86-64 binaries. You should first configure a 32-bit out-of-tree build of libjpeg-turbo, then configure a 64-bit out-of-tree build, then run 'make udmg' from the 64-bit build directory. The build system will look for the 32-bit build under {source_directory}/osxx86 by default, but you can override this by setting the BUILDDIR32 variable on the make command line as shown above.

```
make iosdmg [BUILDDIR32={32-bit build directory}] \
  [BUILDDIRARMV6={ARM v6 build directory}] \
  [BUILDDIRARMV7={ARM v7 build directory}] \
```

On OS X systems, this creates a Macintosh package and disk image in which the libjpeg-turbo static libraries contain ARM architectures necessary to build iOS applications. If building on an x86-64 system, the binaries will also contain the i386 architecture, as with 'make udmg' above. You should first configure ARM v6 and ARM v7 out-of-tree builds of libjpeg-turbo (see "Building libjpeg-turbo for iOS" above.) If you are building an x86-64 version of libjpeg-turbo, you should configure a 32-bit out-of-tree build as well. Next, build libjpeg-turbo as you would normally, using an out-of-tree build. When it is built, run 'make iosdmg' from the build directory. The build system will look for the ARM v6 build under {source_directory}/iosarmv6 by default, the ARM v7 build under {source_directory}/iosarmv7 by default, and (if applicable) the 32-bit build under {source_directory}/osxx86 by default, but you can override this by setting the BUILDDIR32, BUILDDIRARMV6, and/or BUILDDIRARMV7 variables on the make command line as shown above.

```
make sunpkg
```

Build a Solaris package. This requires pkgmk, pkgtrans, and bzip2.

```
make csunpkg [BUILDDIR32={32-bit build directory}]
```

On 64-bit Solaris systems, this creates a combined package that contains both 32-bit and 64-bit libraries. You should first configure a 32-bit out-of-tree build of libjpeg-turbo, then configure a 64-bit out-of-tree build, then run 'make csunpkg' from the 64-bit build directory. The build system will look for the 32-bit build under {source_directory}/solx86 by default, but you can override this by setting the BUILDDIR32 variable on the make command line as shown above.

```
make cygwinpkg
```

Build a Cygwin binary package.

Windows

If using NMake:

```
  cd {build_directory}
  nmake installer
```

If using MinGW:

```
  cd {build_directory}
  make installer
```

If using the Visual Studio IDE, build the "installer" project.

The installer package (libjpeg-turbo[-gcc][64].exe) will be located under {build_directory}. If building using the Visual Studio IDE, then the installer package will be located in a subdirectory with the same name as the configuration you built (such as {build_directory}\Debug\ or {build_directory}\Release\).

Building a Windows installer requires the Nullsoft Install System (http://nsis.sourceforge.net/.) makensis.exe should be in your PATH.

```
*******************************************************************************
**     Regression testing
*******************************************************************************
```

The most common way to test libjpeg-turbo is by invoking 'make test' on Unix/Linux platforms or 'ctest' on Windows platforms, once the build has completed. This runs a series of tests to ensure that mathematical compatibility has been maintained between libjpeg-turbo and libjpeg v6b. This also invokes the TurboJPEG unit tests, which ensure that the colorspace extensions, YUV encoding, decompression scaling, and other features of the TurboJPEG C and Java APIs are working properly (and, by extension, that the equivalent features of the underlying libjpeg API are also working.)

Invoking 'make testclean' or 'nmake testclean' (if using NMake) or building the 'testclean' target (if using the Visual Studio IDE) will clean up the output images generated by 'make test'.

On Unix/Linux platforms, more extensive tests of the TurboJPEG/OSS C and Java wrappers can be run by invoking 'make tjtest'. These extended TurboJPEG tests essentially iterate through all of the available features of the TurboJPEG APIs that are not covered by the TurboJPEG unit tests (this includes the lossless transform options) and compare the images generated by each feature to images generated using the equivalent feature in the libjpeg API. The extended TurboJPEG tests are meant to test for regressions in the TurboJPEG wrappers, not in the underlying libjpeg-turbo library.